INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
(Approved by AICTE | NAAC Accreditation with 'A' Grade | Accredited by NBA | Affiliated to JNTUH)
Dundigal, Hyderabad - 500 043, Telangana

OUTCOME BASED EDUCATION
WITH
CHOICE BASED CREDIT SYSTEM

BACHELOR OF TECHNOLOGY
CSE (CYBER SECURITY)

ACADEMIC REGULATIONS, COURSE STRUCTURE AND SYLLABI
UG.20

B.Tech Regular Four Year Degree Program
(for the batches admitted from the academic year 2020 - 2021)
&

B.Tech (Lateral Entry Scheme)
(for the batches admitted from the academic year 2021 - 2022)

These rules and regulations may be altered/changed from time to time by the academic council
FAILURE TO READ AND UNDERSTAND THE RULES IS NOT AN EXCUSE

VISION

To bring forth professionally competent and socially sensitive engineers, capable of working across cultures meeting the global standards ethically.

MISSION

To provide students with an extensive and exceptional education that prepares them to excel in their profession, guided by dynamic intellectual community and be able to face the technically complex world with creative leadership qualities.

Further, be instrumental in emanating new knowledge through innovative research that emboldens entrepreneurship and economic development for the benefit of wide spread community.

QUALITY POLICY

Our policy is to nurture and build diligent and dedicated community of engineers providing a professional and unprejudiced environment, thus justifying the purpose of teaching and satisfying the stake holders.

A team of well qualified and experienced professionals ensure quality education with its practical application in all areas of the Institute.

PROGRAM OUTCOMES (PO's)

Engineering Graduates will be able to:

PO1: **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: **Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: **Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: **Conduct Investigations of Complex Problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: **Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: **Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: **Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: **Life-Long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

CONTENTS

"Take up one idea.
Make that one idea your life-think of it, dream of it, live on that idea.
Let the brain muscles, nerves, every part of your body be full of that idea and just leave every other idea alone. This is the way to success"

Swami Vivekananda

PRELIMINARY DEFINITIONS AND NOMENCLATURES

AICTE: Means All India Council for Technical Education, New Delhi.

Autonomous Institute: Means an institute designated as Autonomous by University Grants Commission (UGC), New Delhi in concurrence with affiliating University (Jawaharlal Nehru Technological University, Hyderabad) and State Government.

Academic Autonomy: Means freedom to an institute in all aspects of conducting its academic programs, granted by UGC for Promoting Excellence.

Academic Council: The Academic Council is the highest academic body of the institute and is responsible for the maintenance of standards of instruction, education and examination within the institute. Academic Council is an authority as per UGC regulations and it has the right to take decisions on all academic matters including academic research.

Academic Year: It is the period necessary to complete an actual course of study within a year. It comprises two main semesters i.e., (one odd + one even) and one supplementary semester.

Branch: Means specialization in a program like B.Tech degree program in Aeronautical Engineering, B.Tech degree program in Computer Science and Engineering etc.

Board of Studies (BOS): BOS is an authority as defined in UGC regulations, constituted by Head of the Organization for each of the departments separately. They are responsible for curriculum design and updation in respect of all the programs offered by a department.

Backlog Course: A course is considered to be a backlog course, if the student has obtained a failure grade (F) in that course.

Basic Sciences: The courses offered in the areas of Mathematics, Physics, Chemistry etc., are considered to be foundational in nature.

Betterment: Betterment is a way that contributes towards improvement of the students' grade in any course(s). It can be done by either (a) re-appearing or (b) re-registering for the course.

Commission: Means University Grants Commission (UGC), New Delhi.

Choice Based Credit System: The credit based semester system is one which provides flexibility in designing curriculum and assigning credits based on the course content and hours of teaching along with provision of choice for the student in the course selection.

Certificate Course: It is a course that makes a student to have hands-on expertise and skills required for holistic development in a specific area/field.

Compulsory course: Course required to be undertaken for the award of the degree as per the program.

Continuous Internal Examination: It is an examination conducted towards sessional assessment.

Core: The courses that are essential constituents of each engineering discipline are categorized as professional core courses for that discipline.

Course: A course is a subject offered by a department for learning in a particular semester.

Course Outcomes: The essential skills that need to be acquired by every student through a course.

Credit: A credit is a unit that gives weight to the value, level or time requirements of an academic course. The number of 'Contact Hours' in a week of a particular course determines its credit value. One credit is equivalent to one lecture/tutorial hour per week.

Credit point: It is the product of grade point and number of credits for a course.

Cumulative Grade Point Average (CGPA): It is a measure of cumulative performance of a student over all the completed semesters. The CGPA is the ratio of total credit points secured by a student in various courses in all semesters and the sum of the total credits of all courses in all the semesters. It is expressed up to two decimal places.

Curriculum: Curriculum incorporates the planned interaction of students with instructional content, materials, resources, and processes for evaluating the attainment of Program Educational Objectives.

Department: An academic entity that conducts relevant curricular and co-curricular activities, involving both teaching and non-teaching staff, and other resources in the process of study for a degree.

Detention in a Course: Student who does not obtain minimum prescribed attendance in a course shall be detained in that particular course.

Dropping from Semester: Student who doesn't want to register for any semester can apply in writing in prescribed format before the commencement of that semester.

Elective Course: A course that can be chosen from a set of courses. An elective can be Professional Elective and / or Open Elective.

Evaluation: Evaluation is the process of judging the academic performance of the student in her/his courses. It is done through a combination of continuous internal assessment and semester end examinations.

Experiential Engineering Education (ExEEd): Engineering entrepreneurship requires strong technical skills in engineering design and computation with key business skills from marketing to business model generation. Our students require sufficient skills to innovate in existing companies or create their own.

Grade: It is an index of the performance of the students in a said course. Grades are indicated by alphabets.

Grade Point: It is a numerical weight allotted to each letter grade on a 10 - point scale.

Honours: An Honours degree typically refers to a higher level of academic achievement at an undergraduate level.

Institute: Means Institute of Aeronautical Engineering, Hyderabad unless indicated otherwise by the context.

Massive Open Online Courses (MOOC): MOOC courses inculcate the habit of self learning. MOOC courses would be additional choices in all the elective group courses.

Minor: Minor are coherent sequences of courses which may be taken in addition to the courses required for the B.Tech degree.

Pre-requisite: A specific course or subject, the knowledge of which is required to complete before student register another course at the next grade level.

Professional Elective: It indicates a course that is discipline centric. An appropriate choice of minimum number of such electives as specified in the program will lead to a degree with specialization.

Program: Means, UG degree program: Bachelor of Technology (B.Tech); PG degree program: Master of Technology (M.Tech) / Master of Business Administration (MBA).

Program Educational Objectives: The broad career, professional and personal goals that every student will achieve through a strategic and sequential action plan.

Project work: It is a design or research based work to be taken up by a student during his/her final year to achieve a particular aim. It is a credit based course and is to be planned carefully by the student.

Re-Appearing: A student can reappear only in the semester end examination for theory component of a course, subject to the regulations contained herein.

Registration: Process of enrolling into a set of courses in a semester of a program.

Regulations: The regulations, common to all B.Tech programs offered by Institute, are designated as "IARE Regulations – R20" and are binding on all the stakeholders.

Semester: It is a period of study consisting of 15 to 18 weeks of academic work equivalent to normally 90 working days. Odd semester commences usually in July and even semester in December of every year.

Semester End Examinations: It is an examination conducted for all courses offered in a semester at the end of the semester.

S/he: Means "she" and "he" both.

Student Outcomes: The essential skill sets that need to be acquired by every student during her/his program of study. These skill sets are in the areas of employability, entrepreneurial, social and behavioral.

University: Means Jawaharlal Nehru Technological University Hyderabad (JNTUH), Hyderabad, is an affiliating University.

Withdraw from a Course: Withdrawing from a course means that a student can drop from a course within the first two weeks of odd or even semester (deadlines are different for summer sessions). However, s/he can choose a substitute course in place of it, by exercising the option within 5 working days from the date of withdrawal.

FOREWORD

The autonomy is conferred to Institute of Aeronautical Engineering (IARE), Hyderabad by University Grants Commission (UGC), New Delhi based on its performance as well as future commitment and competency to impart quality education. It is a mark of its ability to function independently in accordance with the set norms of the monitoring bodies including J N T University Hyderabad (JNTUH), Hyderabad and AICTE, New Delhi. It reflects the confidence of the affiliating University in the autonomous institution to uphold and maintain standards it expects to deliver on its own behalf. Thus, an autonomous institution is given the freedom to have its own **curriculum, examination system** and **monitoring mechanism**, independent of the affiliating University but under its observance.

IARE is proud to win the credence of all the above bodies monitoring the quality in education and has gladly accepted the responsibility of sustaining, if not improving upon the standards and ethics for which it has been striving for more than a decade in reaching its present standing in the arena of contemporary technical education. As a follow up, statutory bodies such as Academic Council and Board of Studies (BOS) are constituted with the guidance of the Governing Body of the institute and recommendations of the JNTUH to frame the regulations, course structure, and syllabi under autonomous status.

The autonomous regulations, course structure, and syllabi have been prepared after prolonged and detailed interaction with several expertise solicited from academics, industry and research, in accordance with the vision and mission of the institute in order to produce a quality engineering graduate to the society.

All the faculty, parents, and students are requested to go through all the rules and regulations carefully. Any clarifications needed are to be sought at appropriate time and from the principal of the institute, without presumptions, to avoid unwanted subsequent inconveniences and embarrassments. The cooperation of all the stake holders is requested for the successful implementation of the autonomous system in the larger interests of the institute and brighter prospects of engineering graduates.

PRINCIPAL

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

ACADEMIC REGULATIONS – UG.20

B.Tech. Regular Four Year Degree Program
(for the batches admitted from the academic year 2020 - 2021)
&
B.Tech. (Lateral Entry Scheme)
(for the batches admitted from the academic year 2021 - 2022)

For pursuing four year undergraduate Bachelor of Technology (B.Tech) degree program of study in engineering offered by Institute of Aeronautical Engineering under Autonomous status.

A student shall undergo the prescribed courses as given in the program curriculum to obtain his/her degree in major in which he/she is admitted with 160 credits in the entire program of 4 years. Additional 20 credits can be acquired for the degree of B.Tech with **Honours or additional Minor in Engineering**. These additional 20 credits will have to be acquired with massive open online courses (MOOCs), to tap the zeal and excitement of learning beyond the classrooms. This creates an excellent opportunity for students to acquire the necessary skill set for employability through massive open online courses where the rare expertise of world famous experts from academics and industry are available.

Separate certificate will be issued in addition to major degree program mentioning that the student has cleared Honours / Minor specialization in respective courses.

1. CHOICE BASED CREDIT SYSTEM

The credit based semester system provides flexibility in designing program curriculum and assigning credits based on the course content and hours of teaching. The Choice Based Credit System (CBCS) provides a 'cafeteria' type approach in which the students can take courses of their choice, learn at their own pace, undergo additional courses and acquire more than the required credits, and adopt an interdisciplinary approach to learning.

A course defines learning objectives and learning outcomes and comprises lectures / tutorials / laboratory work / field work / project work / comprehensive examination / seminars / assignments / MOOCs / alternative assessment tools / presentations / self-study etc., or a combination of some of these. Under the CBCS, the requirement for awarding a degree is prescribed in terms of number of credits to be completed by the students.

2. MEDIUM OF INSTRUCTION

The medium of instruction shall be **English** for all courses, examinations, seminar presentations and project work. The program curriculum will comprise courses of study as given in course structure, in accordance with the prescribed syllabi.

3. PROGRAMS OFFERED

Presently, the institute is offering Bachelor of Technology (B.Tech) degree programs in eleven disciplines. The various programs and their two-letter unique codes are given in Table 1.

Table 1: B.Tech Programs offered

S. No	Name of the Program	Title	Code
1	Aeronautical Engineering	AE	07
2	Computer Science and Engineering	CS	05
3	Computer Science and Engineering (AI & ML)	CA	34
4	Computer Science and Engineering (Data Science)	CD	35
5	Computer Science and Engineering (Cyber Security)	CC	36
6	Computer Science and Information Technology	CI	37
7	Information Technology	IT	06
8	Electronics and Communication Engineering	EC	04
9	Electrical and Electronics Engineering	EE	02
10	Mechanical Engineering	ME	03
11	Civil Engineering	CE	01

4. SEMESTER STRUCTURE

Each academic year is divided into three semesters, TWO being **MAIN SEMESTERS** (one odd + one even) and ONE being a **SUPPLEMENTARY SEMESTER**. Main semesters are for regular class work. Supplementary Semester is primarily for failed students i.e. registration for a course for the first time is generally not permitted in the supplementary semester.

4.1 Each main semester shall be of 21 weeks (Table 1) duration and this period includes time for registration of courses, course work, examination preparation, and conduct of examinations.

4.2 Each main semester shall have a minimum of 90 working days.

4.3 The supplementary semester shall be a fast track semester consisting of eight weeks and this period includes time for registration of courses, course work, and examination preparation, conduct of examinations, assessment, and declaration of final results.

4.4 All subjects may not be offered in the supplementary semester. The student has to pay a stipulated fee prescribed by the institute to register for a course in the supplementary semester. The supplementary semester is provided to help the student in not losing an academic year. It is optional for a student to make use of supplementary semester. **Supplementary semester is a special semester and the student cannot demand it as a matter of right** and will be offered based on availability of faculty and other institute resources.

4.5 The institute may use **supplementary semester** to arrange add-on courses for regular students and / or for deputing them for practical training / FSI model. A student can register for a maximum number of 15 credits during a supplementary semester.

The registration for the supplementary semester (during May – July, every year) provides an opportunity to students to clear their backlogs ('F' grade) or who are prevented from appearing for SEE examinations due to shortage of attendance less than 65% in each course ('SA' Grade) in the earlier semesters or the courses which he / she could not register (Drop / Withdraw) due to any reason.

Students will not be permitted to register for more than 15 credits (both I and II semester) in the supplementary semester. Students required to register for supplementary semester courses are to pay a nominal fee within the stipulated time. A separate circular shall be issued at the time of supplementary semester.

It will be optional for a student to get registered in the course(s) of supplementary semester; otherwise, he / she can opt to appear directly in supplementary examination. However, if a student gets registered in a course of supplementary semester, then it will be compulsory for a student to fulfill attendance criterion (≥90%) of supplementary semester and he / she will lose option to appear in immediate supplementary examination.

The students who have earlier taken SEE examination and register afresh for the supplementary semester may revoke the CIA marks secured by them in their regular/earlier attempts in the same course. Once revoked, the students shall not seek restoration of the CIA marks.

Supplementary semester will be at an accelerated pace e.g. one credit of a course shall require two hours/week so that the total number of contact hours can be maintained same as in normal semester.

Instructions and guidelines for the supplementary semester course:

- A minimum of 36 to 40 hours will be taught by the faculty for every course.
- Only the students registered and having sufficient percentage of attendance for the course will be permitted to write the examination.
- The assessment procedure in a supplementary semester course will be similar to the procedure for a regular semester course.
- Student shall register for the supplementary semester as per the schedule given in academic calendar.
- Once registered, students will not be allowed to withdraw from supplementary semester.

4.6 The academic calendar shown in Table 2 is declared at the beginning of the academic year.

Table 2: Academic Calendar

FIRST SEMESTER (21 weeks)	I Spell Instruction Period	8 weeks	19 weeks
	I Continuous Internal Assessment Examinations (Mid-term)	1 week	
	II Spell Instruction Period	8 weeks	
	II Continuous Internal Assessment Examinations (Mid-term)	1 week	
	Preparation and Practical Examinations	1 week	
	Semester End Examinations		2 weeks
Semester Break and Supplementary Exams			2 weeks
SECOND SEMESTER (21 weeks)	I Spell Instruction Period	8 weeks	19 weeks
	I Continuous Internal Assessment Examinations (Mid-term)	1 week	
	II Spell Instruction Period	8 weeks	
	II Continuous Internal Assessment Examinations (Mid-term)	1 week	
	Preparation & Practical Examinations	1 week	
	Semester End Examinations		2 weeks
Summer Vacation, Supplementary Semester and Remedial Exams			8 weeks

4.7 Students admitted on transfer from JNTUH affiliated institutes, Universities and other institutes in the subjects in which they are required to earn credits so as to be on par with regular students as prescribed by concerned 'Board of Studies'.

5.0 REGISTRATION / DROPPING / WITHDRAWAL

The academic calendar includes important academic activities to assist the students and the faculty. These include, dates assigned for registration of courses, dropping of courses and withdrawal from courses. This enables the students to be well prepared and take full advantage of the flexibility provided by the credit system.

5.1. Each student has to compulsorily register for course work at the beginning of each semester as per the schedule mentioned in the Academic Calendar. It is compulsory for the student to register for courses in time. The registration will be organized departmentally under the supervision of the Head of the Department.

5.2. In ABSENTIA, registration will not be permitted under any circumstances.

5.3. At the time of registration, students should have cleared all the dues of Institute and Hostel for the previous semesters, paid the prescribed fees for the current semester and not been debarred from the institute for a specified period on disciplinary or any other ground.

5.4. In the first two semesters, the prescribed course load per semester is fixed and is mandated to registered all courses. Withdrawal / dropping of courses in the first and second semester is not allowed.

5.5. In higher semesters, the average load is 22 credits / semester, with its minimum and maximum limits being set at 16 and 28 credits. This flexibility enables students (**from IV semester onwards**) to cope-up with the course work considering the academic strength and capability of student.

5.6. **Dropping of Courses:**
Within one week after the last date of first internal assessment test or by the date notified in the academic calendar, the student may in consultation with his / her faculty mentor/adviser, drop one or more courses without prejudice to the minimum number of credits as specified in clause 5.4. The dropped courses are not recorded in the memorandum of grades. Student must complete the dropped subject by registering in the supplementary semester / forthcoming semester in order to earn the required credits. Student must complete the dropped subject by registering in the supplementary semester / forthcoming semester in order to earn the required credits.

5.7. **Withdrawal from Courses:**
A student is permitted to withdraw from a course by the date notified in the academic calendar. Such withdrawals will be permitted without prejudice to the minimum number of credits as specified in clause 5.4. A student cannot withdraw a course more than once and withdrawal of reregistered courses is not permitted.

6.0 CREDIT SYSTEM

The B.Tech Program shall consist of a number of courses and each course shall be assigned with credits. The curriculum shall comprise Theory Courses, Elective Courses, Laboratory Courses, Value Added Courses, Mandatory Courses, Experiential Engineering Education (ExEEd), Internship and Project work.

Depending on the complexity and volume of the course, the number of contact periods per week will be assigned. Each theory and laboratory course carries credits based on the number of hours / week.

- Contact classes (Theory): 1 credit per lecture hour per week, 1 credit per tutorial hour per week.
- Laboratory hours (Practical): 1 credit for 2 practical hours per week.
- Project work: 1 credit for 2 hours of project work per week.
- Mandatory Courses: No credit is awarded.
- Value Added Courses: No credit is awarded.
- Experiential Engineering Education (ExEEd): 1 credit for two per hours.

Credit distribution for courses offered is given in Table 5.

Table 5: Credit distribution

S. No	Course	Hours	Credits
1	Theory Course	2 / 3 / 4	2 / 3 / 4
2	Elective Courses	3	3
3	Laboratory Courses	2 / 3 / 4	1 / 1.5 / 2

4	Mandatory Course / Value Added Course	-	0
5	Project Work	-	10
6	Full Semester Internship (FSI) Project work	-	10

Major benefits of adopting the credit system are listed below:

- Quantification and uniformity in the listing of courses for all programs at College, like core, electives and project work.
- Ease of allocation of courses under different heads by using their credits to meet national /international practices in technical education.
- Convenience to specify the minimum / maximum limits of course load and its average per semester in the form of credits to be earned by a student.
- Flexibility in program duration for students by enabling them to pace their course load within minimum/maximum limits based on their preparation and capabilities.
- Wider choice of courses available from any department of the same College or even from other similar Colleges, either for credit or for audit.
- Improved facility for students to optimize their learning by availing of transfer of credits earned by them from one College to another.

7.0 CURRICULAR COMPONENTS

Courses in a curriculum may be of three kinds: **Foundation / Skill, Core and Elective Courses.**

Foundation / Skill Course:

Foundation courses are the courses based upon the content leads to enhancement of skill and knowledge as well as value based and are aimed at man making education. Skill subjects are those areas in which one needs to develop a set of skills to learn anything at all. They are fundamental to learning any subject.

Professional Core Courses:

There may be a core course in every semester. This is the course which is to be compulsorily studied by a student as a core requirement to complete the requirement of a program in the said discipline of study.

Elective Course:

Electives provide breadth of experience in respective branch and application areas. Elective course is a course which can be chosen from a pool of courses. It may be:

- Supportive to the discipline of study
- Providing an expanded scope
- Enabling an exposure to some other discipline / domain
- Nurturing student's proficiency / skill.

An elective may be Professional Elective, is a discipline centric focusing on those courses which add generic proficiency to the students or may be Open Elective, chosen from unrelated disciplines.

There are six professional elective tracks; students can choose not more than two courses from each track. Overall, students can opt for six professional elective courses which suit their project work in consultation with the faculty advisor/mentor. Nevertheless, one course from each of the four open electives has to be selected. A student may also opt for more elective courses in his/her area of interest.

Every course of the B.Tech program will be placed in one of the eight categories with minimum credits as listed in the Table 6.

Table 6: Category Wise Distribution of Credits

S. No	Category	Breakup of Credits
1	Humanities and Social Sciences (HSMC), including Management.	6
2	Basic Science Courses (BSC) including Mathematics, Physics and Chemistry.	18.5
3	Engineering Science Courses (ESC), including Workshop, Drawing, ExEEd, Basics of Electrical / Electronics / Mechanical / Computer Engineering.	20.5
4	Professional Core Courses (PCC), relevant to the chosen specialization / branch.	78
5	Professional Electives Courses (PEC), relevant to the chosen specialization / branch.	18
6	Open Elective Courses (OEC), from other technical and/or emerging subject areas.	09
7	Project work (PROJ) / Full Semester Internship (FSI) Project work	10
8	Mandatory Courses (MC) / Value Added Courses (VAC).	Non-Credit
TOTAL		**160**

Semester wise course break-up

Following are the **TWO** models of course structure out of which any student shall choose or will be allotted with one model based on their academic performance.

i. Full Semester Internship (FSI) Model and

ii. Non Full Semester Internship (NFSI) Model

In the FSI Model, out of the selected students - half of students shall undergo Full Semester Internship in VII semester and the remaining students in VIII semester. In the Non-FSI Model, all the selected students shall carry out the course work and Project work as specified in the course structure. A student who secures a minimum CGPA of 7.5 upto IV semester with **no current arrears** and maintains the CGPA of 7.5 till VI Semester shall be eligible to opt for FSI.

8. EVALUATION METHODOLOGY

Each theory course will be evaluated for a total of 100 marks, with 30 marks for Continuous Internal Assessment (CIA) and 70 marks for Semester End Examination (SEE). Student's performance in a course shall be judged by taking into account the results of CIA and SEE together. Table-7 shows the typical distribution of weightage for CIA and SEE.

Table 7: Assessment pattern for Theory Courses

Component		Marks	Total Marks
CIA	Continuous Internal Examination – 1 (Mid-term)	10	30
	Continuous Internal Examination – 2 (End-term)	10	
	Tech talk / Quiz – 1 and Quiz – 2	5	
	Concept video / Alternative Assessment Tool (AAT)	5	
SEE	Semester End Examination (SEE)	70	70
Total Marks			**100**

8.1. Semester End Examination (SEE):

The SEE is conducted for 70 marks of 3 hours duration. The syllabus for the theory courses is divided into FIVE modules and each modules carries equal weightage in terms of marks distribution. The question paper pattern is as follows.

Two full questions with 'either' 'or' choice will be drawn from each module. Each question carries 14 marks. There could be a maximum of two sub divisions in a question.

The emphasis on the questions is broadly based on the following criteria:

50 %	To test the objectiveness of the concept
50 %	To test the analytical skill of the concept OR to test the application skill of the concept

8.1. Continuous Internal Assessment (CIA):

For each theory course the CIA shall be conducted by the faculty / teacher handling the course. CIA is conducted for a total of 30 marks, with 20 marks for Continuous Internal Examination (CIE), 05 marks for Quiz and 05 marks for Alternative Assessment Tool (AAT). **Two CIE Tests are Compulsory** and sum of the two tests, along with the scores obtained in the quizzes (average of Quiz – 1 and Quiz – 2) / AAT shall be considered for computing the final CIA of a student in a given course.

The CIE Tests/quizzes/AAT shall be conducted by the course faculty with due approval from the HOD. Advance notification for the conduction of Quiz/AAT is mandatory and the responsibility lies with the concerned course faculty.

8.1.1. Continuous Internal Examination (CIE):

Two CIE exams shall be conducted at the end of the 8th and 16th week of the semester respectively for 10 marks each of 2 hours duration consisting of five descriptive type questions out of which four questions have to be answered. The valuation and verification of answer scripts of CIE exams shall be completed within a week after the conduct of the Examination.

8.1.2. Quiz – Online Examination

Two Quiz exams shall be conducted along with CIE in online mode for 5 marks each, consisting of 10 short answers questions (Definitions and Terminology) and 10 multiple choice questions (having each question to be answered by tick marking the correct answer from the choices (commonly four) given against it. Such a question paper shall be useful in testing of knowledge, skills, application, analysis, evaluation and understanding of the students. Average of two quiz examinations shall be considered.

8.1.3. Alternative Assessment Tool (AAT)

In order to encourage innovative methods while delivering a course, the faculty members are encouraged to use the Alternative Assessment Tool (AAT). This AAT enables faculty to design own assessment patterns during the CIA. The AAT enhances the autonomy (freedom and flexibility) of individual faculty and enables them to create innovative pedagogical practices. If properly applied, the AAT converts the classroom into an effective learning centre.

The AAT may include tech talk, tutorial hours/classes, seminars, assignments, term paper, open ended experiments, concept videos, partial reproduction of research work, oral presentation of research work, developing a generic tool-box for problem solving, report based on participation in create-a-thon, make-a-thon, code-a-thon, hack-a-thon conducted by reputed organizations / any other. etc.

However, it is mandatory for a faculty to obtain prior permission from the concerned HOD and spell out the teaching/assessment pattern of the AAT prior to commencement of the classes.

8.2 Laboratory Course

Each laboratory will be evaluated for a total of 100 marks consisting of 30 marks for internal assessment and 70 marks for semester end lab examination. Out of 30 marks of internal assessment, continuous lab assessment will be done for 20 marks for the day to day performance and 10 marks for the final internal lab assessment. The semester end laboratory examination for 70 marks shall be conducted internally by the respective department with at least two faculty members as examiners, both nominated by the Principal from the panel of experts recommended by the Chairman, BOS.

All the drawing related courses are evaluated in line with laboratory courses. The distribution shall be 30

marks for internal evaluation (20 marks for day–to–day work, and 10 marks for internal tests) and 70 marks for semester end laboratory examination. There shall be ONE internal test of 10 marks in each semester.

8.3 Audit Courses

In Addition, a student can register for courses for audit only with a view to supplement his/her knowledge and/or skills. Here also, the student's grades shall have to be reflected in the Memorandum of Grades. But, these shall not be taken into account in determining the student's academic performance in the semester. In view of this, it shall not be necessary for the institute to issue any separate transcript covering the audit courses to the registrants at these courses. Its result shall be declared as "Satisfactory" or "Not Satisfactory" performance.

8.4 Mandatory Courses (MC)

These courses are among the compulsory courses but will not carry any credits. However, a pass in each such course during the program shall be necessary requirement for the student to qualify for the award of Degree. Its result shall be declared as "Satisfactory" or "Not Satisfactory" performance.

8.5 Additional Mandatory Courses for lateral entry B.Tech students

In addition to the non-credit mandatory courses for regular B.Tech students, the lateral entry students shall take up the following three non-credit mandatory bridge courses (one in III semester, one in IV semester and one in V semester) as listed in Table 8. The student shall pass the following non-credit mandatory courses for the award of the degree and must clear these bridge courses before advancing to the VII semester of the program.

Table-8: Additional Mandatory Courses for lateral entry

S. No	Additional mandatory courses for lateral entry students
1	Dip-Mathematics
2	Dip-Programming for Problem Solving
3	Dip-English Communication Skills

8.6 Value Added Courses

The value added courses are audit courses offered through joint ventures with various organizations providing ample scope for the students as well as faculty to keep pace with the latest technologies pertaining to their chosen fields of study. A plenty of value added programs will be proposed by the departments one week before the commencement of class work. The students are given the option to choose the courses according to their desires and inclinations as they choose the desired items in a cafeteria. The expertise gained through the value added programs should enable them to face the formidable challenges of the future and also assist them in exploring new opportunities. Its result shall be declared with "Satisfactory" or "Not Satisfactory" performance.

8.7 Experiential Engineering Education (ExEED)

Engineering entrepreneurship requires strong technical skills in engineering design and computation with key business skills from marketing to business model generation. Students require sufficient skills to innovate in existing companies or create their own.

This course will be evaluated for a total of 100 marks consisting of 30 marks for internal assessment and 70 marks for semester end Examination. Out of 30 marks of internal assessment, The Student has to submit Innovative Idea in a team of four members in the given format. The semester end examination for 70 marks shall be conducted internally, students has to present the Innovative Idea and it will be evaluated by internal ExEEd faculty with at least one faculty member as examiner from the industry, both nominated by the Principal from the panel of experts recommended by the Dean-CLET.

8.8 Project Work / FSI Project Work

This gives students a platform to experience a research driven career in engineering, while developing a device / systems and publishing in reputed SCI / SCOPUS indexed journals and/or filing an **Intellectual Property** (IPR-Patent/Copyright) to aid communities around the world. Students should work individually as per the

guidelines issued by head of the department concerned. The benefits to students of this mode of learning include increased engagement, fostering of critical thinking and greater independence.

The topic should be so selected that the students are enabled to complete the work in the stipulated time with the available resources in the respective laboratories. The scope of the work be handling part of the consultancy work, maintenance of the existing equipment, development of new experiment setup or can be a prelude to the main project with a specific outcome.

Project report will be evaluated for 100 marks in total. Assessment will be done for 100 marks out of which, the supervisor / guide will evaluate for 30 marks based on the work and presentation / execution of the work. Subdivision for the remaining 70 marks is based on publication, report, presentation, execution and viva-voce. Evaluation shall be done by a committee comprising the supervisor, Head of the department and an examiner nominated by the Principal from the panel of experts recommended by Chairman, BOS in consultation with Head of the department.

8.8.1 Project work

The student's project activity is spread over in VII semester and in VIII semesters. A student shall carry out the project work under the supervision of a faculty member or in collaboration with an Industry, R&D organization or another academic institution/University where sufficient facilities exist to carry out the project work.

Project work (phase-I) starts in VII semester as it takes a vital role in campus hiring process. Students shall select project titles from their respective logins uploaded by the supervisors at the beginning of VII semester. Three reviews are conducted by department review committee (DRC) for 10 marks each. Student must submit a project report summarizing the work done up to design phase/prototype by the end of VII semester. The semester end examination for project work (phase-I) is evaluated based on the project report submitted and a viva-voce exam for 70 marks by a committee comprising the head of the department, the project supervisor and an external examiner nominated by the Principal.

Project Work (phase-II) starts in VIII semester, shall be evaluated for 100 marks out of which 30 marks towards continuous internal assessment and 70 marks for semester end examination. Three reviews are to be conducted by DRC on the progress of the project for 30 marks. The semester end examination shall be based on the final report submitted and a viva-voce exam for 70 marks by a committee comprising the head of the department, the project supervisor and an external examiner nominated by the Principal.

A minimum of 40% of maximum marks shall be obtained to earn the corresponding credits.

8.8.2 Full Semester Internship (FSI)

FSI is a full semester internship program carry 10 credits. The FSI shall be opted in VII semester or in VIII semester. During the FSI, student has to spend one full semester in an identified industry / firm / R&D organization or another academic institution/University where sufficient facilities exist to carry out the project work.

Following are the evaluation guidelines:

- Quizzes: 2 times
- Quiz #1 - About the industry profile, weightage: 5%
- Quiz #2 - Technical-project related, weightage: 5%
- Seminars - 2 times (once in six weeks), weightage: 7.5% + 7.5%
- Viva-voce: 2 times (once in six weeks), weightage: 7.5% + 7.5%
- Project Report, weightage: 15%
- Internship Diary, weightage: 5 %
- Final Presentation, weightage: 40%

FSI shall be open to all the branches with a ceiling of maximum 10% distributed in both semesters. The selection procedure is:

- Choice of the students.
- CGPA (> 7.5) upto IV semester having no credit arrears.
- Competency Mapping / Allotment.

It is recommended that the FSI Project work leads to a research publication in a reputed Journal/Conference or the filing of patent/design with the patent office, or, the start-up initiative with a sustainable and viable business model accepted by the incubation center of the institute together with the formal registration of the startup.

8.9 Plagiarism index for Project Report:

All project reports shall go through the plagiarism check and the plagiarism index has to be less than 20%. Project reports with plagiarism more than 20% and less than 60% shall be asked for resubmission within a stipulated period of six months. Project reports with plagiarism more than 60% shall be rejected.

9. MAKEUP EXAMINATION

The make-up examination facility shall be available to students who may have missed to attend **CIE/Quiz** of one or more courses in a semester for valid reasons. The CIE make-up examination shall have comprehensive online objective type questions for 20 marks and Quiz for 5 marks. The content for the make-up examination shall be on the whole syllabus. The Makeup examination shall be conducted at the end of the respective semester.

10. SUPPLEMENTARY EXAMINATIONS

In addition to the Regular Semester End Examinations held at the end of each semester, Supplementary Semester End Examinations will be conducted within three weeks of the commencement of the teaching of the next semester. Candidates taking the Regular / Supplementary examinations as Supplementary candidates may have to take more than one Semester End Examination per day. A student can appear for any number of supplementary examinations till he/she clears all courses which he/she could not clear in the first attempt. However the maximum stipulated period for the course shall not be relaxed under any circumstances.

11. ATTENDANCE REQUIREMENTS AND DETENTION POLICY

11.1 It is desirable for a candidate to have 100% attendance in each course. In every course (theory/laboratory), student has to maintain a minimum of 75% attendance including the days of attendance in sports, games, NCC and NSS activities to be eligible for appearing in Semester End Examination of the course.

11.2 In case of medical issues, deficiency of attendance in each course to the extent of 10% may be condoned by the College Academic Committee (CAC) on the recommendation of the Head of the Department if the attendance is between 75% and 65% in every course, subjected to the submission of medical certificates, medical case file, and other needful documents to the concerned departments.

11.3 The basis for the calculation of the attendance shall be the period prescribed by the institute by its calendar of events. For late admission, attendance is reckoned from the date of admission to the program. However, in case of a student having less than 65% attendance in any course, s/he shall be detained in the course and in no case such process will be relaxed.

11.4 A candidate shall put in a minimum required attendance in atleast 60% of (rounded to the next highest integer) theory courses for getting promoted to next higher class / semester. Otherwise, s/he shall be declared detained and has to repeat semester.

11.5 Students whose shortage of attendance is not condoned in any subject are not eligible to write their semester end examination of that courses and their registration shall stand cancelled.

11.6 A prescribed fee shall be payable towards condonation of shortage of attendance.

11.7 A student shall not be promoted to the next semester unless he satisfies the attendance requirement of the present semester, as applicable. They may seek readmission into that semester when offered next. If any candidate fails to fulfill the attendance requirement in the present semester, he shall not be eligible for readmission into the same class.

11.8 Any student against whom any disciplinary action by the institute is pending shall not be permitted to attend any SEE in that semester.

12. CONDUCT OF SEMESTER END EXAMINATIONS AND EVALUATION

12.1 Semester end examination shall be conducted by the Controller of Examinations (COE) by inviting Question Papers from the External Examiners.

12.2 Question papers may be moderated for the coverage of syllabus, pattern of questions by a Semester End Examination Committee chaired by Head of the Department one day before the commencement of semester end examinations. Internal Examiner shall prepare a detailed scheme of valuation.

12.3 The answer papers of semester end examination should be evaluated by the internal examiner immediately after the completion of exam and the award sheet should be submitted to COE in a sealed cover.

12.4 COE shall invite 3 - 9 internal/external examiners to evaluate all the semester end examination answer books on a prescribed date(s). Practical laboratory exams are conducted involving external examiners.

12.5 Examinations Control Committee shall consolidate the marks awarded by examiner/s and award grades.

13. SCHEME FOR THE AWARD OF GRADE

13.1 A student shall be deemed to have satisfied the minimum academic requirements and earn the credits for each theory course, if s/he secures

a) Not less than 35% marks for each theory course in the semester end examination, and

b) A minimum of 40% marks for each theory course considering Continuous Internal Assessment (CIA) and Semester End Examination (SEE).

13.2 A student shall be deemed to have satisfied the minimum academic requirements and earn the credits for each Laboratory / Project work / FSI Project work, if s/he secures

a) Not less than 40% marks for each Laboratory / Project work / FSI Project work course in the semester end examination,

b) A minimum of 40% marks for each Laboratory / Project work / FSI Project work course considering both internal and semester end examination.

13.3 If a candidate fails to secure a pass in a particular course, it is mandatory that s/he shall register and reappear for the examination in that course during the next semester when examination is conducted in that course. It is mandatory that s/he should continue to register and reappear for the examination till s/he secures a pass.

13.4 A student shall be declared successful or 'passed' in a semester, if he secures a Grade Point ≥ 5 ('C' grade or above) in every course in that semester (i.e. when the student gets an SGPA ≥ 5.0 at the end of that particular semester); and he shall be declared successful or 'passed' in the entire under graduate programme, only when gets a CGPA ≥ 5.0 for the award of the degree as required.

14. LETTER GRADES AND GRADE POINTS

14.1 Performances of students in each course are expressed in terms of marks as well as in Letter Grades based on absolute grading system. The UGC recommends a 10-point grading system with the following letter grades as given in the Table-9.

Table-9: Grade Points Scale (Absolute Grading)

Range of Marks	Grade Point	Letter Grade
100 – 90	10	S (Superior)
89 – 80	9	A+ (Excellent)
79 – 70	8	A (Very Good)
69 – 60	7	B+ (Good)
59 – 50	6	B (Average)
49 – 40	5	C (Pass)
Below 40	0	F (Fail)
Absent	0	AB (Absent)
Authorized Break of Study	0	ABS

14.2 A student is deemed to have passed and acquired to correspondent credits in particular course if s/he obtains any one of the following grades: "S", "A+", "A", "B+", "B", "C".

14.3 A student obtaining Grade F shall be considered Failed and will be required to reappear in the examination.

14.4 For non credit courses, 'Satisfactory' or "Not Satisfactory" is indicated instead of the letter grade and this will not be counted for the computation of SGPA/CGPA.

14.5 "SA" denotes shortage of attendance (as per item 11) and hence prevention from writing Semester End Examination.

14.6 "W" denotes **withdrawal** from the exam for the particular course.

14.7 At the end of each semester, the institute issues grade sheet indicating the SGPA and CGPA of the student. However, grade sheet will not be issued to the student if s/he has any outstanding dues.

14.8 **Award of Class:**

Sometimes, it is necessary to provide equivalence of these averages, viz., SGPA and CGPA with the percentages and/or Class awarded as in the conventional system of declaring the results of University examinations. This shall be done by Autonomous Colleges under the University only at one stage by prescribing certain specific thresholds in these averages for First Class with Distinction, First Class and Second Class, at the time of Degree Award. This provision given in Table-10 follows the approach of the Council for this purpose as reproduced from the AICTE Approval Process Handbook:

Table 10: Percentage Equivalence of Grade Points (for a 10 – Point Scale)

Grade Point	Percentage of Marks / Class
5.5	50
6.0	55
6.5	60
7.0	65
7.5	70
8.0	75

Note:

(1) The following Formula for Conversion of CGPA to percentage of marks to be used only after a student has successfully completed the program:
Percentage of Marks = $(CGPA - 0.5) \times 10$

(2) Class designation:
≥75% (First Class with Distinction),
≥ 60% and <75 % (First Class),
≥ 50 % and <60% (Second Class),

≥45% and <50% (Pass Class).

(3) The SGPA will be computed and printed on the Memorandum of Grades only if the candidate passes in all the courses offered and gets minimum B grade in all the courses.

(4) CGPA is calculated only when the candidate passes in all the courses offered in all the semesters.

15. COMPUTATION OF SGPA AND CGPA

The UGC recommends to compute the Semester Grade Point Average (SGPA) and Cumulative Grade Point Average (CGPA). The credit points earned by a student are used for calculating the Semester Grade Point Average (SGPA) and the Cumulative Grade Point Average (CGPA), both of which are important performance indices of the student. SGPA is equal to the sum of all the total points earned by the student in a given semester divided by the number of credits registered by the student in that semester. CGPA gives the sum of all the total points earned in all the previous semesters and the current semester divided by the number of credits registered in all these semesters. Thus,

$$SGPA = \sum_{i=1}^{n} (C_i G_i) / \sum_{i=1}^{n} C_i$$

Where, C_i is the number of credits of the i^{th} course and G_i is the grade point scored by the student in the i^{th} course and n represent the number of courses in which a student is registered in the concerned semester.

$$CGPA = \sum_{j=1}^{m} (C_j S_j) / \sum_{j=1}^{m} C_j$$

Where, S_j is the SGPA of the j^{th} semester and C_j is the total number of credits upto the semester and m represent the number of semesters completed in which a student registered upto the semester.

The SGPA and CGPA shall be rounded off to 2 decimal points and reported in the transcripts.

16. ILLUSTRATION OF COMPUTATION OF SGPA AND CGPA

16.1 Illustration for SGPA

Course Name	Course Credits	Grade letter	Grade point	Credit Point (Credit x Grade)
Course 1	3	A	8	3 x 8 = 24
Course 2	4	B+	7	4 x 7 = 28
Course 3	3	B	6	3 x 6 = 18
Course 4	3	S	10	3 x 10 = 30
Course 5	3	C	5	3 x 5 = 15
Course 6	4	B	6	4 x 6 = 24
	20			**139**

Thus, SGPA = 139 / 20 = 6.95

16.2 Illustration for CGPA

Semester 1	Semester 2	Semester 3	Semester 4
Credit: 20 SGPA: 6.9	Credit: 22 SGPA: 7.8	Credit: 25 SGPA: 5.6	Credit: 26 SGPA: 6.0
Semester 5	**Semester 6**		
Credit: 26 SGPA: 6.3	Credit: 25 SGPA: 8.0		

$$Thus,\ CGPA = \frac{20x6.9 + 22x7.8 + 25x5.6 + 26x6.0 + 26x6.3 + 25x8.0}{144} = 6.73$$

17. REVIEW OF SEE THEORY ANSWER BOOKS

Semester end examination answer books are made available online in CMS portal on the day of publication of results. A student, who is not satisfied with the assessment, is directed to apply for the review of his/her semester end examination answer book(s) in the theory course(s), within 2 working days from the publication of results in the prescribed format to the Controller of Examinations through the Head of the department with prescribed fee.

The Controller of Examinations shall appoint two examiners (chief examiner of original exam and a new examiner) for the review of the semester end examination (theory) answer book. Both examiners shall jointly review and marks awarded in the previous assessment shall be kept open.

The marks obtained by the candidate after the review shall be considered for grading, only if, the change in mark is more than or equal to 10% of total mark of semester end examination (theory). Marks obtained after re-evaluation shall stand final even if it is less than the original marks. Review is not permitted to the courses other than theory courses.

18. PROMOTION POLICIES

The following academic requirements have to be satisfied in addition to the attendance requirements mentioned in item no. 11.

18.1 For students admitted into B.Tech (Regular) program

18.1.1 A student will not be promoted from II semester to III semester unless s/he fulfills the academic requirement of securing 50% of the total credits (rounded to the next lowest integer) from I and II semester examinations, whether the candidate takes the examination(s) or not.

18.1.2 A student will not be promoted from IV semester to V semester unless s/he fulfills the academic requirement of securing 60% of the total credits (rounded to the next lowest integer) upto III semester **or** 60% of the total credits (rounded to the next lowest integer) up to IV semester, from all the examinations, whether the candidate takes the examination(s) or not.

18.1.3 A student shall be promoted from VI semester to VII semester only if s/he fulfills the academic requirements of securing 60% of the total credits (rounded to the next lowest integer) up to V semester **or** 60% of the total credits (rounded to the next lowest integer) up to VI semester from all the examinations, whether the candidate takes the examination(s) or not.

18.1.4 A student shall register for all the 160 credits and earn all the 160 credits. Marks obtained in all the 160 credits shall be considered for the award of the Grade.

18.2 For students admitted into B.Tech (lateral entry students)

18.2.1 A student will not be promoted from IV semester to V semester unless s/he fulfills the academic requirement of securing 60% of the total credits (rounded to the next lowest integer) up to IV semester, from all the examinations, whether the candidate takes the examination(s) or not.

18.2.2 A student shall be promoted from VI semester to VII semester only if s/he fulfills the academic requirements of securing 60% of the total credits (rounded to the next lowest integer) up to V semester **or** 60% of the total credits (rounded to the next lowest integer) up to VI semester from all the examinations, whether the candidate takes the examination(s) or not.

18.2.3 A student shall register for all the 126 credits and earn all the 126 credits. Marks obtained in all the 126 credits shall be considered for the award of the Grade.

19. GRADUATION REQUIREMENTS

The following academic requirements shall be met for the award of the B.Tech degree.

19.1 Student shall register and acquire minimum attendance in all courses and secure 160 credits (with minimum CGPA of 5.0), for regular program and 126 credits (with minimum CGPA of 5.0), for lateral entry program.

19.2 A student of a regular program, who fails to earn 160 credits within eight consecutive academic years from the year of his/her admission with a minimum CGPA of 5.0, shall forfeit his/her degree and his/her admission stands cancelled.

19.3 A student of a lateral entry program who fails to earn 126 credits within six consecutive academic years from the year of his/her admission with a minimum CGPA of 5.0, shall forfeit his/her degree and his/her admission stands cancelled.

20. BETTERMENT OF MARKS IN THE COURSES ALREADY PASSED

Students who clear all the courses in their first attempt and wish to improve their CGPA shall register and appear for betterment of marks for one course of any theory courses within a period of subsequent two semesters. The improved marks shall be considered for classification / distinction but not for ranking. If there is no improvement, there shall not be any change in the original marks already awarded.

21. AWARD OF DEGREE

21.1 Classification of degree will be as follows:

CGPA ≥ 8.0	CGPA ≥ 6.5 and < 8.0	CGPA ≥ 5.5 and < 6.5	CGPA ≥ 5.0 and < 5.5	CGPA < 5.0
First Class with Distinction	**First Class**	**Second Class**	**Pass Class**	**Fail**

21.2 A student with final CGPA (at the end of the under graduate programme) ≥8.00, and fulfilling the following conditions - shall be placed in '**first class with distinction'**. However,

(a) Should have passed all the courses in '**first appearance'** within the first 4 academic years (or 8 sequential semesters) from the date of commencement of first year first semester.

(b) Should have secured a CGPA ≥8.00, at the end of each of the 8 sequential semesters, starting from I year I semester onwards.

(c) Should not have been detained or prevented from writing the semester end examinations in any semester due to shortage of attendance or any other reason.

A student not fulfilling any of the above conditions with final CGPA >8 shall be placed in **'first class'.**

21.3 Students with final CGPA (at the end of the B.Tech program) ≥6.50 but <8.00 shall be placed in **'first class'.**

21.4 Students with final CGPA (at the end of the B.Tech program) ≥5.50 but <6.50, shall be placed in '**second class'**.

21.5 All other students who qualify for the award of the degree (as per item 19), with final CGPA (at the end of the B.Tech program) ≥5.0 but <5.50, shall be placed in '**pass class**'.

21.6 A student with final CGPA (at the end of the B.Tech program) < 5.00 will not be eligible for the award of the degree.

21.7 Students fulfilling the conditions listed under item 21.2 alone will be eligible for award of '**Gold Medal**'.

21.8. In order to extend the benefit to the students with one/two backlogs after either VI semester or VIII semester, GRAFTING option is provided to the students enabling their placements and fulfilling graduation requirements. Following are the guidelines for the Grafting:

(a) Grafting will be done among the courses within the semester shall draw a maximum of 7 marks from the any one of the cleared courses in the semester and will be grafted to the failed course in the same semester.

(b) Students shall be given a choice of grafting only once in the 4 years program, either after VI semester (Option #1) or after VIII semester (Option #2).

(c) Option#1: Applicable to students who have maximum of TWO theory courses in V and / or VI semesters.

Option#2: Applicable to students who have maximum of TWO theory courses in VII and / or VIII semesters.

(d) Eligibility for grafting:

i. Prior to the conduct of the supplementary examination after the declaration of VI or VIII semester results.

ii. S/he must appear in all regular or supplementary examinations as per the provisions laid down in regulations for the courses s/he appeals for grafting.

iii. The marks obtained by her/him in latest attempt shall be taken into account for grafting of marks in the failed course(s).

21.9 Student, who clears all the courses upto VII semester, shall have a chance to appear for Quick Supplementary Examination to clear the failed courses of VIII semester.

21.10 By the end of VI semester, all the students (regular and lateral entry students) shall complete one of the Value added course and mandatory course with acceptable performance.

21.11 In case, a student takes more than one attempt in clearing a course, the final marks secured shall be indicated by * mark in the grade sheet.

All the candidates who register for the semester end examination will be issued a memorandum of grades sheet by the institute. Apart from the semester wise memorandum of grades sheet, the institute will issue the provisional certificate and consolidated grades memorandum subject to the fulfillment of all the academic requirements.

22. B.TECH WITH HONOURS OR ADDITIONAL MINORS IN ENGINEERING

Students acquiring 160 credits are eligible to get B.Tech degree in Engineering. A student will be eligible to get B.Tech degree with Honours or additional Minors in Engineering, if s/he completes an additional 20 credits (3/4 credits per course). These could be acquired through MOOCs from SWAYAM / NPTEL / edX / Coursera / Udacity / PurdueNext / Khan Academy / QEEE etc. The list for MOOCs will be a dynamic one, as new courses are added from time to time. Few essential skill sets required for employability are also identified year wise. Students interested in doing MOOC courses shall register the course title at their department office at the start of the semester against the courses that are announced by the department. Any expense incurred for the MOOC course / summer program should be met by the students.

Only students having no credit arrears and a CGPA of 7.5 or above at the end of the fourth semester are eligible to register for B.Tech (Honours / Minor). After registering for the B.Tech (Honours / Minor) program, if a student fails in any course, s/he will not be eligible for B.Tech (Honours / Minor).

Every Department to develop and submit a Honours / Minors – courses list of 5 - 6 theory courses.

Honours Certificate for Vertical in his/her OWN Branch for Research orientation; Minor in any other branch for Improving Employability.

For the MOOCs platforms, where examination or assessment is absent (like SWAYAM) or where certification is costly (like Coursera or edX), faculty members of the institute prepare the examination question papers, for the courses undertaken by the students of respective Institutes, so that examinations Control Office (ECO) can conduct examination for the course. There shall be one Continuous Internal Examination (Quiz exam for 30 marks) after 8 weeks of the commencement of the course and semester end examination (Descriptive exam for 70 marks) shall be done along with the other regular courses.

A student can enroll for both Minor & Honours or for two Minors. The final grade sheet will only show the basic CGPA corresponding to the minimum requirement for the degree. The Minors/Honours will be indicated by a separate CGPA. The additional courses taken will also find separate mention in the grade sheet.

If a student drops (or terminated) from the Minor/Honours program, they cannot convert the earned credits into free or core electives; they will remain extra. These additional courses will find mention in the grade sheet (but not in the degree certificate). In such cases, the student may choose between the actual grade or a "Pass (P)" grade and also choose to omit the mention of the course as for the following:

- All the courses done under the dropped Minor/Honours will be shown in the grade sheet
- None of the courses done under the dropped Minor/Honours will be shown in the grade sheet.

Honours will be reflected in the degree certificate as "B.Tech (Honours) in XYZ Engineering". Similarly, Minor as "B.Tech in XYZ Engineering with Minor in ABC". If a student has done both Honours & Minor, it will be acknowledged as "B.Tech (Honours) in XYZ Engineering with Minor in ABC". And two minors will be reflected as "B.Tech in XYZ Engineering with Minor in ABC and Minor in DEF".

22.1. B.Tech with Honours

The total of 20 credits required to be attained for B.Tech Honours degree are distributed from V semester to VII semester in the following way:

For V semester	:	4 – 8 credits
For VI semester	:	4 – 8 credits
For VII semester	:	4 – 8 credits

Following are the details of such Honours which include some of the most interesting areas in the profession today:

S. No	Department	Honours scheme
1	Aeronautical Engineering	Aerospace Engineering / Space Science etc.
2	Computer Science and Engineering / Information Technology	Big data and Analytics / Cyber Physical Systems, Information Security / Cognitive Science / Artificial Intelligence/ Machine Learning / Data Science / Internet of Things (IoT) etc.
3	Electronics and Communication Engineering	Digital Communication / Signal Processing / Communication Networks / VLSI Design / Embedded Systems etc.
4	Electrical and Electronics Engineering	Renewable Energy systems / Energy and Sustainability / IoT Applications in Green Energy Systems etc.
5	Mechanical Engineering	Industrial Automation and Robotics / Manufacturing Sciences and Computation Techniques etc.
6	Civil Engineering	Structural Engineering / Environmental Engineering etc.

22.2 B.Tech with additional Minor in Engineering

Every department to develop and submit Minor courses list of 5 - 6 Theory courses. Student from any department is eligible to apply for Minor from any other department. The total of 20 credits to complete the B.Tech (Minor) program by registering for MOOC courses each having a minimum of 3/4 credits offered by reputed institutions / organization with the approval of the department. Registration of the student for B.Tech (Minor), is from V Semester to VII Semester of the program in the following way:

For V semester	:	4 – 8 credits
For VI semester	:	4 – 8 credits
For VII semester	:	4 – 8 credits

Only students having no credit arrears and a CGPA of 7.5 or above at the end of the fourth semester are eligible to register for B.Tech (Minor). After registering for the B.Tech (Minor) program, if a student fails in

any course, s/he will not be eligible for B.Tech (Minor).

Every student shall also have the option to do a minor in engineering. A major is a primary focus of study and a minor is a secondary focus of study. The minor has to be a subject offered by a department other than the department that offers the major of the student or it can be a different major offered by the same department. For example, a student with the declared major in Computer Science and Engineering (CSE) may opt to do a minor in Physics; in which case, the student shall receive the degree B.Tech, Computer Science and Engineering with a minor in Physics. A student can do Majors in chosen filed as per the career goal, and a minor may be chosen to enhance the major thus adding the diversity, breadth and enhanced skills in the field.

22.3 Advantages of Minor in Engineering:

The minors mentioned above are having lots of advantages and a few are listed below:

1. To apply the inter-disciplinary knowledge gained through a Major (Stream) + Minor.
2. To enable students to pursue allied academic interest in contemporary areas.
3. To provide an academic mechanism for fulfilling multidisciplinary demands of industries.
4. To provide effective yet flexible options for students to achieve basic to intermediate level competence in the Minor area.
5. Provides an opportunity to students to become entrepreneurs and leaders by taking business/ management minor.
6. Combination in the diverse fields of engineering e.g., CSE (Major) + Electronics (Minor) combination increases placement prospects in chip designing companies.
7. Provides an opportunity to Applicants to pursue higher studies in an inter-disciplinary field of study.
8. Provides opportunity to the Applicants to pursue interdisciplinary research.
9. To increase the overall scope of the undergraduate degrees.

22.4 Following are the details of such Minor / Honours which include some of the most interesting areas in the profession today:

1. Aerospace Engineering
2. Space Science
3. Industrial Automation and Robotics
4. Computer Science and Engineering
5. Data Analytics
6. Machine Learning
7. Data Science
8. Artificial Intelligence
9. Information Security
10. Internet of Things
11. Cyber Physical Systems
12. Electronic System Design
13. Renewable Energy Sources
14. Energy and Sustainability
15. Manufacturing Sciences and Computation Techniques
16. Structural Engineering
17. Environmental Engineering
18. Technological Entrepreneurship
19. Materials Engineering
20. Physics (Materials / Nuclear / Optical / Medical)
21. Mathematics (Combinatorics / Logic / Number theory / Dynamical systems and differential equations/ Mathematical physics / Statistics and Probability).

23.0 TEMPORARY BREAK OF STUDY FROM THE PROGRAM

23.1 A candidate is normally not permitted to take a break from the study. However, if a candidate intends to temporarily discontinue the program in the middle for valid reasons (such as accident or hospitalization due to prolonged ill health) and to rejoin the program in a later respective semester, s/he shall seek the

approval from the Principal in advance. Such application shall be submitted before the last date for payment of examination fee of the semester in question and forwarded through the Head of the Department stating the reasons for such withdrawal together with supporting documents and endorsement of his / her parent / guardian.

23.2 The institute shall examine such an application and if it finds the case to be genuine, it may permit the student to temporarily withdraw from the program. Such permission is accorded only to those who do not have any outstanding dues / demand at the College / University level including tuition fees, any other fees, library materials etc.

23.3 The candidate has to rejoin the program after the break from the commencement of the respective semester as and when it is offered.

23.4 The total period for completion of the program reckoned from the commencement of the semester to which the candidate was first admitted shall not exceed the maximum period specified in clause 19. The maximum period includes the break period.

23.5 If any candidate is detained for any reason, the period of detention shall not be considered as 'Break of Study'.

24. TERMINATION FROM THE PROGRAM

The admission of a student to the program may be terminated and the student is asked to leave the institute in the following circumstances:

a. The student fails to satisfy the requirements of the program within the maximum period stipulated for that program.
b. A student shall not be permitted to study any semester more than three times during the entire program of study.
c. The student fails to satisfy the norms of discipline specified by the institute from time to time.

25. TRANSCRIPT

The Transcript will be issued to the student as and when required and will contain a consolidated record of all the courses undergone by him/her, grades obtained and CGPA upto the date of issue of transcript. Only last letter grade obtained in a course by the student upto the date of issue of transcript will be shown in the Transcript.

26. WITH-HOLDING OF RESULTS

If the candidate has not paid any dues to the institute / if any case of indiscipline / malpractice is pending against him, the results and the degree of the candidate will be withheld.

27. GRADUATION DAY

The institute shall have its own annual Graduation Day for the award of degrees to the students completing the prescribed academic requirements in each case, in consultation with the University and by following the provisions in the Statute. The college shall institute prizes and medals to meritorious students and award them annually at the Graduation Day. This will greatly encourage the students to strive for excellence in their academic work.

28. DISCIPLINE

Every student is required to observe discipline and decorum both inside and outside the institute and are expected not to indulge in any activity which will tend to bring down the honour of the institute. If a student indulges in malpractice in any of the theory / practical examination, continuous assessment examinations, he/she shall be liable for punitive action as prescribed by the institute from time to time.

29. GRIEVANCE REDRESSAL COMMITTEE

The institute shall form a Grievance Redressal Committee for each course in each department with the Course Teacher and the HOD as the members. This Committee shall solve all grievances related to the course under consideration.

30. TRANSITORY REGULATIONS

A candidate, who is detained or has discontinued a semester, on readmission shall be required to do all the courses in the curriculum prescribed for the batch of students in which the student joins subsequently. However, exemption will be given to those candidates who have already passed such courses in the earlier semester(s) he was originally admitted into and substitute subjects are offered in place of them as decided by the Board of Studies. However, the decision of the Board of Studies will be final.

a) Four Year B.Tech Regular course:

A student who is following Jawaharlal Nehru Technological University (JNTUH) curriculum and detained due to the shortage of attendance at the end of the first semester shall join the autonomous batch of first semester. Such students shall study all the courses prescribed for the batch in which the student joins and considered on par with regular candidates of Autonomous stream and will be governed by the autonomous regulations.

A student who is following JNTUH curriculum, detained due to lack of credits or shortage of attendance at the end of the second semester or at the subsequent semesters shall join with the autonomous batch in the appropriate semester. Such candidates shall be required to pass in all the courses in the program prescribed by the Board of Studies concerned for that batch of students from that semester onwards to be eligible for the award of degree. However, exemption will be given in the courses of the semester(s) of the batch which he had passed earlier and substitute courses will be offered in place of them as decided by the Board of Studies. The student has to clear all his backlog courses up to previous semester by appearing for the supplementary examinations conducted by JNTUH for the award of degree. The total number of credits to be secured for the award of the degree will be sum of the credits up to previous semester under JNTUH regulations and the credits prescribed for the semester in which a candidate seeks readmission and subsequent semesters under the autonomous stream. The class will be awarded based on the academic performance of a student in the autonomous pattern.

b) Three Year B.Tech program under Lateral Entry Scheme:

A student who is following JNTUH curriculum and detained due to the shortage of attendance at the end of the first semester of second year shall join the autonomous batch of third semester. Such students shall study all the courses prescribed for the batch in which the student joins and considered on par with Lateral Entry regular candidates of Autonomous stream and will be governed by the autonomous regulations.

A student who is following JNTUH curriculum, if detained due to lack of credits or shortage of attendance at the end of the second semester of second year or at the subsequent semesters shall join with the autonomous batch in the appropriate semester. Such candidates shall be required to pass in all the courses in the program prescribed by the Board of Studies concerned for that batch of students from that semester onwards to be eligible for the award of degree. However, exemption will be given in the courses of the semester(s) of the batch which he had passed earlier and substitute courses are offered in place of them as decided by the Board of Studies. The student has to clear all his backlog courses up to previous semester by appearing for the supplementary examinations conducted by JNTUH for the award of degree. The total number of credits to be secured for the award of the degree will be sum of the credits up to previous semester under JNTUH regulations and the credits prescribed for the semester in which a candidate seeks readmission and subsequent semesters under the autonomous status. The class will be awarded based on the academic performance of a student in the autonomous pattern.

c) Transfer candidates (from non-autonomous college affiliated to JNTUH):

A student who is following JNTUH curriculum, transferred from other college to this institute in third semester or subsequent semesters shall join with the autonomous batch in the appropriate semester. Such candidates shall be required to pass in all the courses in the program prescribed by the Board of Studies concerned for that batch of students from that semester onwards to be eligible for the award of

degree. However, exemption will be given in the courses of the semester(s) of the batch which he had passed earlier and substitute courses are offered in their place as decided by the Board of Studies. The student has to clear all his backlog courses up to previous semester by appearing for the supplementary examinations conducted by JNTUH for the award of degree. The total number of credits to be secured for the award of the degree will be the sum of the credits up to the previous semester under JNTUH regulations and the credits prescribed for the semester in which a candidate joined after transfer and subsequent semesters under the autonomous status. The class will be awarded based on the academic performance of a student in the autonomous pattern.

d) Transfer candidates (from an autonomous college affiliated to JNTUH):

A student who has secured the required credits up to previous semesters as per the regulations of other autonomous institutions shall also be permitted to be transferred to this institute. A student who is transferred from the other autonomous colleges to this institute in third semester or subsequent semesters shall join with the autonomous batch in the appropriate semester. Such candidates shall be required to pass in all the courses in the program prescribed by the Board of Studies concerned for that batch of students from that semester onwards to be eligible for the award of degree. However, exemption will be given in the courses of the semester(s) of the batch which he had passed earlier and substitute subjects are offered in their place as decided by the Board of Studies. The total number of credits to be secured for the award of the degree will be the sum of the credits up to previous semester as per the regulations of the college from which he is transferred and the credits prescribed for the semester in which a candidate joined after transfer and subsequent semesters under the autonomous status. The class will be awarded based on the academic performance of a student in the autonomous pattern.

e) Readmission from IARE-R16/R18 to IARE-UG.20 regulations

A student took admission in IARE-R18 Regulations, detained due to lack of required number of credits or percentage of attendance at the end of any semester is permitted to take re-admission at appropriate level under any regulations prevailing in the institute subject to the following rules and regulations.

1. Student shall pass all the courses in the earlier scheme of regulations (IARE - R18). However, in case of having backlog courses, they shall be cleared by appearing for supplementary examinations conducted under IARE - R18 regulations from time to time.
2. After rejoining, the student is required to study the courses as prescribed in the new regulations for the re-admitted program at that level and thereafter.
3. If the student has already passed any course(s) of readmitted program in the earlier regulation / semester of study, such courses are exempted in the new scheme to appear for the course(s).
4. The courses that are not done in the earlier regulations / semester as compared with readmitted program need to be cleared after readmission by appearing for the examinations conducted time to time under the new regulations.
5. In general, after transition, course composition and number of credits / semester shall be balanced between earlier and new regulations on case to case basis.
6. In case, the students who do not have option of acquiring required credits with the existing courses offered as per the new curriculum, credit balance can be achieved by clearing the additional courses offered by the respective departments (approved in Academic Council meeting). The additional courses that are offered can be of theory or laboratory courses and shall be offered during semester.
7. Students re-joined in III semester shall be treated on par with “Lateral Entry” students for credits and graduation requirements. However, the student shall clear all the courses in B.Tech I Semester and B.Tech II Semester as per IARE-R18 regulations.

31. REVISION OF REGULATIONS AND CURRICULUM

The Institute from time to time may revise, amend or change the regulations, scheme of examinations and syllabi if found necessary and on approval by the Academic Council and the Governing Body and shall be binding on the students, faculty, staff, all authorities of the Institute and others concerned.

EQUENTLY ASKED QUESTIONS AND ANSWERS ABOUT AUTONOMY

1. Who grants Autonomy? UGC, Govt., AICTE or University

In case of Colleges affiliated to a university and where statutes for grant of autonomy are ready, it is the respective University that finally grants autonomy but only after concurrence from the respective state Government as well as UGC. The State Government has its own powers to grant autonomy directly to Govt. and Govt. aided Colleges.

2 Shall IARE award its own Degrees?

No. Degree will be awarded by Jawaharlal Nehru Technological University, Hyderabad with a mention of the name IARE on the Degree Certificate.

3 What is the difference between a Deemed University and an Autonomy College?

A Deemed University is fully autonomous to the extent of awarding its own Degree. A Deemed University is usually a Non-Affiliating version of a University and has similar responsibilities like any University. An Autonomous College enjoys Academic Autonomy alone. The University to which an autonomous college is affiliated will have checks on the performance of the autonomous college.

4 How will the Foreign Universities or other stake – holders know that we are an Autonomous College?

Autonomous status, once declared, shall be accepted by all the stake holders. The Govt. of Telangana mentions autonomous status during the First Year admission procedure. Foreign Universities and Indian Industries will know our status through our website.

5 What is the change of Status for Students and Teachers if we become Autonomous?

An autonomous college carries a prestigious image. Autonomy is actually earned out of our continued past efforts on academic performances, our capability of self- governance and the kind of quality education we offer.

6 Who will check whether the academic standard is maintained / improved after Autonomy? How will it be checked?

There is a built in mechanism in the autonomous working for this purpose. An Internal Committee called Academic Program Evaluation Committee, which will keep a watch on the academics and keep its reports and recommendations every year. In addition the highest academic council also supervises the academic matters. The standards of our question papers, the regularity of academic calendar, attendance of students, speed and transparency of result declaration and such other parameters are involved in this process.

7 Will the students of IARE as an Autonomous College qualify for University Medals and Prizes for academic excellence?

No. IARE has instituted its own awards, medals, etc. for the academic performance of the students. However for all other events like sports, cultural on co-curricular organized by the University the students shall qualify.

8 Can IARE have its own Convocation?

No. Since the University awards the Degree the Convocation will be that of the University, but there will be Graduation Day at IARE.

9 Can IARE give a provisional degree certificate?

Since the examinations are conducted by IARE and the results are also declared by IARE, the college sends a list of successful candidates with their final Grades and Grade Point Averages including CGPA to the University. Therefore with the prior permission of the University the college will be entitled to give the provisional certificate.

10 Will Academic Autonomy make a positive impact on the Placements or Employability?

Certainly. The number of students qualifying for placement interviews is expected to improve, due to rigorous and repetitive classroom teaching and continuous assessment. Also the autonomous status is more

responsive to the needs of the industry. As a result therefore, there will be a lot of scope for industry oriented skill development built-in into the system. The graduates from an autonomous college will therefore represent better employability.

11 What is the proportion of Internal and External Assessment as an Autonomous College?

Presently, it is 60% external and 40% internal. As the autonomy matures the internal assessment component shall be increased at the cost of external assessment.

12 Is it possible to have complete Internal Assessment for Theory or Practicals?

Yes indeed. We define our own system. We have the freedom to keep the proportion of external and internal assessment component to choose.

13 Why Credit based Grade System?

The credit based grade system is an accepted standard of academic performance the world over in all Universities. The acceptability of our graduates in the world market shall improve.

14 What exactly is a Credit based Grade System?

The credit based grade system defines a much better statistical way of judging the academic performance. One Lecture Hour per week of Teaching Learning process is assigned One Credit. One hour of laboratory work is assigned half credit. Letter Grades like A, B,C,D, etc. are assigned for a Range of Marks. (e.g. 91% and above is A+, 80 to 90 % could be A etc.) in Absolute Grading System while grades are awarded by statistical analysis in relative grading system. We thus dispense with sharp numerical boundaries. Secondly, the grades are associated with defined Grade Points in the scale of 1 to 10. Weighted Average of Grade Points is also defined Grade Points are weighted by Credits and averaged over total credits in a Semester. This process is repeated for all Semesters and a CGPA defines the Final Academic Performance

15 What are the norms for the number of Credits per Semester and total number of Credits for UG/PG program?

These norms are usually defined by UGC or AICTE. Usually around 25 Credits per semester is the accepted norm.

16 What is a Semester Grade Point Average (SGPA)?

The performance of a student in a semester is indicated by a number called SGPA. The SGPA is the weighted average of the grade points obtained in all the courses registered by the student during the semester.

$$SGPA = \sum_{i=1}^{n} (C_i G_i) / \sum_{i=1}^{n} C_i$$

Where, C_i is the number of credits of the i^{th} course and G_i is the grade point scored by the student in the i^{th} course and i represent the number of courses in which a student registered in the concerned semester. SGPA is rounded to two decimal places.

17 What is a Cumulative Grade Point Average (CGPA)?

An up-to-date assessment of overall performance of a student from the time of his first registration is obtained by calculating a number called CGPA, which is weighted average of the grade points obtained in all the courses registered by the students since he entered the Institute.

$$CGPA = \sum_{j=1}^{m} (C_j S_j) / \sum_{j=1}^{m} C_j$$

Where, S_j is the SGPA of the j^{th} semester and C_j is the total number of credits upto the semester and m represent the number of semesters completed in which a student registered upto the semester. CGPA is rounded to two decimal places.

18 Is there any Software available for calculating Grade point averages and converting the same into Grades?

Yes, The institute has its own MIS software for calculation of SGPA, CGPA, etc.

19 Will the teacher be required to do the job of calculating SGPAs etc. and convert the same into Grades?

No. The teacher has to give marks obtained out of whatever maximum marks as it is. Rest is all done by the computer.

20 Will there be any Revaluation or Re-Examination System?

No. There will double valuation of answer scripts. There will be a makeup Examination after a reasonable preparation time after the End Semester Examination for specific cases mentioned in the Rules and Regulations. In addition to this, there shall be a 'summer term' (compressed term) followed by the End Semester Exam, to save the precious time of students.

21 How fast Syllabi can be and should be changed?

Autonomy allows us the freedom to change the syllabi as often as we need.

22 Will the Degree be awarded on the basis of only final year performance?

No. The CGPA will reflect the average performance of all the semester taken together.

23 What are Statutory Academic Bodies?

Governing Body, Academic Council, Examination Committee and Board of Studies are the different statutory bodies. The participation of external members in everybody is compulsory. The institute has nominated professors from IIT, NIT, University (the officers of the rank of Pro-vice Chancellor, Deans and Controller of Examinations) and also the reputed industrialist and industry experts on these bodies.

24 Who takes Decisions on Academic matters?

The Governing Body of institute is the top academic body and is responsible for all the academic decisions. Many decisions are also taken at the lower level like Boards of Studies. Decisions taken at the Boared of Studies level are to be ratified at the Academic Council and Governing Body.

25 What is the role of Examination committee?

The Examinations Committee is responsible for the smooth conduct of internal, End Semester and make up Examinations. All matters involving the conduct of examinations spot valuations, tabulations preparation of Grade Sheet etc fall within the duties of the Examination Committee.

26 Is there any mechanism for Grievance Redressal?

The institute has grievance redressal committee, headed by Dean - Student affairs and Dean - IQAC.

27 How many attempts are permitted for obtaining a Degree?

All such matters are defined in Rules & Regulation

28 Who declares the result?

The result declaration process is also defined. After tabulation work wherein the SGPA, CGPA and final Grades are ready, the entire result is reviewed by the Moderation Committee. Any unusual deviations or gross level discrepancies are deliberated and removed. The entire result is discussed in the Examinations and Result Committee for its approval. The result is then declared on the institute notice boards as well put on the web site and Students Corner. It is eventually sent to the University.

29 Who will keep the Student Academic Records, University or IARE?

It is the responsibility of the Dean, Academics of the Autonomous College to keep and preserve all the records.

30 What is our relationship with the JNT University?

We remain an affiliated college of the JNT University. The University has the right to nominate its members on the academic bodies of the college.

31 Shall we require University approval if we want to start any New Courses?

Yes, It is expected that approvals or such other matters from an autonomous college will receive priority.

32 Shall we get autonomy for PG and Doctoral Programs also?

Yes, presently our PG programs also enjoying autonomous status.

MALPRACTICE RULES

DISCIPLINARY ACTION FOR / IMPROPER CONDUCT IN EXAMINATIONS

S. No	**Nature of Malpractices/Improper conduct**	**Punishment**
	If the candidate:	
1. (a)	Possesses or keeps accessible in examination hall, any paper, note book, programmable calculator, cell phone, pager, palm computer or any other form of material concerned with or related to the subject of the examination (theory or practical) in which he is appearing but has not made use of (material shall include any marks on the body of the candidate which can be used as an aid in the subject of the examination)	Expulsion from the examination hall and cancellation of the performance in that subject only.
(b)	Gives assistance or guidance or receives it from any other candidate orally or by any other body language methods or communicates through cell phones with any candidate or persons in or outside the exam hall in respect of any matter.	Expulsion from the examination hall and cancellation of the performance in that subject only of all the candidates involved. In case of an outsider, he will be handed over to the police and a case is registered against him.
2.	Has copied in the examination hall from any paper, book, programmable calculators, palm computers or any other form of material relevant to the subject of the examination (theory or practical) in which the candidate is appearing.	Expulsion from the examination hall and cancellation of the performance in that subject and all other subjects the candidate has already appeared including practical examinations and project work and shall not be permitted to appear for the remaining examinations of the subjects of that Semester/year. The Hall Ticket of the candidate is to be cancelled and sent to the Controller of Examinations.
3.	Impersonates any other candidate in connection with the examination.	The candidate who has impersonated shall be expelled from examination hall. The candidate is also debarred and forfeits the seat. The performance of the original candidate, who has been impersonated, shall be cancelled in all the subjects of the examination (including practicals and project work) already appeared and shall not be allowed to appear for examinations of the remaining subjects of that semester/year. The candidate is also debarred for two consecutive semesters from class work and all semester end examinations. The continuation of the course by the candidate is subject to the academic regulations in connection with forfeiture of seat. If the imposter is an outsider, he will be handed over to the police and a case is registered against him.
4.	Smuggles in the Answer book or additional sheet or takes out or arranges to send out the question paper during the examination or answer book or additional sheet, during or after the examination.	Expulsion from the examination hall and cancellation of performance in that subject and all the other subjects the candidate has already appeared including practical examinations and

		project work and shall not be permitted for the remaining examinations of the subjects of that semester/year. The candidate is also debarred for two consecutive semesters from class work and all semester end examinations. The continuation of the course by the candidate is subject to the academic regulations in connection with forfeiture of seat.
5.	Uses objectionable, abusive or offensive language in the answer paper or in letters to the examiners or writes to the examiner requesting him to award pass marks.	Cancellation of the performance in that subject.
6.	Refuses to obey the orders of the Controller of Examinations /Additional Controller of Examinations/any officer on duty or misbehaves or creates disturbance of any kind in and around the examination hall or organizes a walk out or instigates others to walk out, or threatens the COE or any person on duty in or outside the examination hall of any injury to his person or to any of his relations whether by words, either spoken or written or by signs or by visible representation, assaults the COE or any person on duty in or outside the examination hall or any of his relations, or indulges in any other act of misconduct or mischief which result in damage to or destruction of property in the examination hall or any part of the Institute premises or engages in any other act which in the opinion of the officer on duty amounts to use of unfair means or misconduct or has the tendency to disrupt the orderly conduct of the examination.	In case of students of the college, they shall be expelled from examination halls and cancellation of their performance in that subject and all other subjects the candidate(s) has (have) already appeared and shall not be permitted to appear for the remaining examinations of the subjects of that semester/year. The candidates also are debarred and forfeit their seats. In case of outsiders, they will be handed over to the police and a police case is registered against them.
7.	Leaves the exam hall taking away answer script or intentionally tears off the script or any part thereof inside or outside the examination hall.	Expulsion from the examination hall and cancellation of performance in that subject and all the other subjects the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the subjects of that semester/year. The candidate is also debarred for two consecutive semesters from class work and all semester end examinations. The continuation of the course by the candidate is subject to the academic regulations in connection with forfeiture of seat.
8.	Possess any lethal weapon or firearm in the examination hall.	Expulsion from the examination hall and cancellation of the performance in that subject and all other subjects the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the subjects of that semester/year. The candidate is also debarred and forfeits the seat.
9.	If student of the college, who is not a candidate for the particular examination or any person not	Student of the colleges expulsion from the examination hall and cancellation of the

	connected with the college indulges in any malpractice or improper conduct mentioned in clause 6 to 8.	performance in that subject and all other subjects the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the subjects of that semester/year. The candidate is also debarred and forfeits the seat. Person(s) who do not belong to the College will be handed over to police and, a police case will be registered against them.
10.	Comes in a drunken condition to the examination hall.	Expulsion from the examination hall and cancellation of the performance in that subject and all other subjects the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the subjects of that semester/year.
11.	Copying detected on the basis of internal evidence, such as, during valuation or during special scrutiny.	Cancellation of the performance in that subject and all other subjects the candidate has appeared including practical examinations and project work of that semester/year examinations.
12.	If any malpractice is detected which is not covered in the above clauses 1 to 11 shall be reported to the University for further action to award suitable punishment.	

FAILURE TO READ AND UNDERSTAND THE REGULATIONS IS NOT AN EXCUSE

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad – 500043

COURSE CATALOG

COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

I SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AHSC02	Linear Algebra and Calculus	BSC	Foundation	3	1	0	4	30	70	100
AHSC06	Chemistry	BSC	Foundation	2	0	0	2	30	70	100
AEEC01	Basic Electrical Engineering	ESC	Foundation	3	0	0	3	30	70	100
ACSC01	Python Programming	ESC	Foundation	3	0	0	3	30	70	100
ACSC06	Experiential Engineering Education (ExEEd) – Academic Sucess	ESC	Foundation	2	0	0	1	30	70	100
PRACTICAL										
AEEC04	Basic Electrical Engineering Laboratory	ESC	Foundation	0	0	3	1.5	30	70	100
ACSC02	Python Programming Laboratory	ESC	Foundation	0	0	3	1.5	30	70	100
AMEC04	Engineering Workshop Practice	ESC	Foundation	0	0	2	1	30	70	100
TOTAL				**13**	**01**	**08**	**17**	**240**	**560**	**800**

II SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AHSC01	English	HSMC	Foundation	3	0	0	3	30	70	100
AHSC08	Probability and Statistics	BSC	Foundation	3	1	0	4	30	70	100
AHSC09	Applied Physics	BSC	Foundation	3	0	0	3	30	70	100
ACSC04	Programming for Problem Solving using C	ESC	Foundation	3	0	0	3	30	70	100
PRACTICAL										
AHSC04	English Language and Communication Skills Laboratory	HSMC	Foundation	0	0	2	1	30	70	100
AHSC05	Physics Laboratory	BSC	Foundation	0	0	3	1.5	30	70	100
ACSC05	Programming for Problem Solving using C Laboratory	ESC	Foundation	0	0	3	1.5	30	70	100
TOTAL				**12**	**01**	**08**	**17**	**210**	**490**	**700**

III SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
ACSC08	Data Structures	PCC	Core	3	0	0	3	30	70	100
ACCC01	Mathematical Foundation for Cyber Security	PCC	Core	3	1	0	4	30	70	100
AECC08	Analog and Digital Electronics	ESC	Core	3	0	0	3	30	70	100
ACSC07	Computer Organization and Architecture	PCC	Core	3	0	0	3	30	70	100
ACSC12	Operating Systems	PCC	Core	3	0	0	3	30	70	100
ACSC09	ExEEd - Prototype / Design Building	ESC	Foundation	2	0	0	1	30	70	100
PRACTICALS										
ACSC10	Data Structures Laboratory	PCC	Core	0	0	3	1.5	30	70	100
AITC03	Programming with Objects Laboratory	PCC	Core	0	0	3	1.5	30	70	100
ACSC16	Linux Programming Laboratory	PCC	Core	0	1	2	2	30	70	100
MANDATORY / VALUE ADDED COURSES										
AHSC10	Essence of Indian Traditional Knowledge	MC	MC-1	Ref: 8.4 Academic Regulations, UG.20						
TOTAL				**17**	**02**	**08**	**22**	**270**	**630**	**900**

IV SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
ACCC02	Foundations of Cyber Security	PCC	Core	3	1	0	4	30	70	100
ACSC13	Design and Analysis of Algorithms	PCC	Core	3	0	0	3	30	70	100
AITC05	Database Management Systems	PCC	Core	3	0	0	3	30	70	100
AITC06	Computer Networks	PCC	Core	3	1	0	4	30	70	100
AHSC13	Business Economics and Financial Analysis	HSMC	Foundation	3	0	0	3	30	70	100
ACSC14	ExEEd - Fabrication / Model Development	ESC	Foundation	2	0	0	1	30	70	100
PRACTICALS										
AITC07	Database Management Systems Laboratory	PCC	Core	0	0	3	1.5	30	70	100
ACSC15	Design and Analysis of Algorithms Laboratory	PCC	Core	0	0	3	1.5	30	70	100
AITC12	Computer Networks Laboratory	PCC	Core	0	0	2	1	30	70	100
MANDATORY / VALUE ADDED COURSES										
AHSC14	Indian Constitution	MC	MC - 2	Ref: 8.4 Academic Regulations, UG.20						
	TOTAL			**17**	**02**	**08**	**22**	**270**	**630**	**900**

V SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AITC09	Web Application Development	PCC	Core	3	0	0	3	30	70	100
ACSC19	Object Oriented Software Engineering	PCC	Core	3	1	0	4	30	70	100
AITC04	Theory of Computation	PCC	Core	3	1	0	4	30	70	100
ACCC03	Network Security	PCC	Core	3	1	0	4	30	70	100
	Professional Elective – I	PEC	Elective	3	0	0	3	30	70	100
ACSC20	ExEEd - Project Based Learning	ESC	Foundation	2	0	0	1	30	70	100
PRACTICALS										
AITC10	Web Application Development Laboratory	PCC	Core	0	0	3	1.5	30	70	100
ACCC08	Network Security Laboratory	PCC	Core	0	0	3	1.5	30	70	100
MANDATORY / VALUE ADDED COURSES										
ACSC22	Competitive Programming using Graph Algorithms	VAC	Skill	Ref: 8.6, Academic Regulations-UG.20						
TOTAL				**17**	**03**	**06**	**22**	**240**	**560**	**800**

VI SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
ACCC09	Penetration Testing and Cyber Operations	PCC	Core	3	0	0	3	30	70	100
ACIC01	Data Mining and Knowledge Discovery	PCC	Core	3	1	0	4	30	70	100
ACSC40	Compiler Design	PCC	Core	3	1	0	4	30	70	100
	Professional Elective – II	PEC	Elective	3	0	0	3	30	70	100
	Open Elective – I	OEC	Elective	3	0	0	3	30	70	100
ACSC27	ExEEd - Research Based Learning	ESC	Foundation	2	0	0	1	30	70	100
PRACTICALS										
ACCC10	Penetration Testing and Cyber Operations Laboratory	PCC	Core	1	0	2	2	30	70	100
ACIC08	Data Mining and Knowledge Discovery Laboratory	PCC	Core	1	0	2	2	30	70	100
MANDATORY / VALUE ADDED COURSES										
ACSC28	Go Programming	VAC	Skill	Ref: 8.6, Academic Regulations-UG.20						
TOTAL				**19**	**02**	**04**	**22**	**240**	**560**	**800**

VII SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
ACCC11	Digital Forensics	PCC	Core	3	1	0	4	30	70	100
ACCC12	Cloud Security	PCC	Core	3	0	0	3	30	70	100
	Professional Elective – III	PEC	Elective	3	0	0	3	30	70	100
	Professional Elective – IV	PEC	Elective	3	0	0	3	30	70	100
	Open Elective – II	OEC	Elective	3	0	0	3	30	70	100
PRACTICALS										
ACCC17	Digital Forensics Laboratory	PCC	Core	0	0	3	1.5	30	70	100
ACCC18	Cloud Security Laboratory	PCC	Core	0	0	3	1.5	30	70	100
ACCC19	Project Work (Phase – 1)	PROJ	Project	0	0	4	2	30	70	100
TOTAL				**15**	**01**	**10**	**21**	**240**	**560**	**800**

VIII SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
	Professional Elective –V	PEC	Elective	3	0	0	3	30	70	100
	Professional Elective –VI	PEC	Elective	3	0	0	3	30	70	100
	Open Elective – III	OEC	Elective	3	0	0	3	30	70	100
PRACTICALS										
ACCC24	Project Work (Phase-2) / Full Semester Internship	PROJ	Project	0	0	16	8	30	70	100
TOTAL				**09**	**00**	**16**	**17**	**120**	**280**	**400**

PROFESSIONAL ELECTIVES

Course code	PE - I Networks and Security	Course code	PE - II Software Development	Course code	PE - III Applied Machine Learning
ACCC04	Database Security	ACIC04	Agile Software Development Approaches	AITC26	Principles of Artificial Intelligence
ACCC05	Network Programming and Management	ACIC05	Software Project Management	AITC27	Machine Learning
ACCC06	Software Defined Networks	ACIC06	Software Architecture and Design Patterns	AITC28	Data Handling and Visualization
ACCC07	Security Computing	ACIC07	Software Reliability	ACAC09	Advanced Social, Text and Media Analytics

Course code	PE - IV Cyber - Physical Systems	Course code	PE - V Computer Architecture and System Programming	Course code	PE - VI Models and Methods in Computing
ACCC13	Principles of Computer Security	ACSC36	Systems Programming	ACCC20	High Performance Computing
ACCC14	Cyber Security Techniques and Tools	ACDC12	Human Computer Interaction (UI & UX)	ACCC21	Quantum Computing
ACCC15	Ethical Hacking and Systems Defense	ACSC37	Internet Systems Programming	ACCC22	Ubiquitous Computing
ACCC16	Cyber Physical Systems	ACSC38	RUST Programming	ACCC23	Edge Computing

OPEN ELECTIVE – I

Course Code	Course Title
AHSC15	Soft Skills and Interpersonal Communication
AHSC16	Cyber Law and Ethics
AHSC17	Economic Policies in India
AHSC18	Global Warming and Climate Change
AHSC19	Intellectual Property Rights
AHSC20	Entrepreneurship

OPEN ELECTIVES – II

Course Code	Course Title
AEEC29	Industrial Automation and Control
AEEC30	Artificial Neural Networks
AEEC31	Renewable Energy Sources
AECC38	Basic Electronic Engineering
AECC39	Principles of Communications
AECC40	Embedded Systems

OPEN ELECTIVE - III

Course Code	Course Title
AAEC30	Flight Control Theory
AAEC31	Airframe Structural Design
AMEC34	Industrial Management
AMEC35	Elements of Mechanical Engineering
ACEC30	Modern Construction Materials
ACEC31	Disaster Management

VALUE ADDED COURSES / MANDATORY COURSES

Course Code	Course Title
AHSC10	Essence of Indian Traditional Knowledge (MC)
AHSC14	Indian Constitution (MC)
ACSC22	Competitive Programming using Graph Algorithms (VAC)
ACSC28	Go Programming (VAC)

SYLLABUS
(I - VIII SEMESTERS)

LINEAR ALGEBRA AND CALCULUS

I Semester: Common for All Branches

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC02	Foundation	3	1	-	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisite: Basic principles of algebra and calculus

I. COURSE OVERVIEW:

Linear algebra is a sub-field of mathematics concerned with vectors, matrices, and linear transforms. Calculus is the branch of mathematics which majorly deals with derivatives and integrals. Linear algebra is a key foundation to the field of machine learning. Matrices are used in computer animations, color image processing. Eigenvalues are used by engineers to discover new and better designs for the future. Differential equations have wide applications in various engineering and science disciplines as the laws of physics are generally written down as differential equations.The Fourier series has many applications in electrical engineering, image processing etc.The course includes types of Matrices, Rank, methods of finding rank, eigen values and eigen vectors, maxima and minima of functions of several variables, solutions of higher order ordinary differential equations and Fourier series.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The principles of Eigen value analysis and linear transformations, Matrix rank finding methods
II. The calculus of functions of several variables and the concept of maxima-minima for a three-dimensional surface.
III. The analytical methods for solving higher order differential equations with constant coefficients.
IV. Fourier series expansions in standard intervals as well as arbitrary intervals.

III. COURSE SYLLABUS:

MODULE-I: THEORY OF MATRICES (09)

Real matrices: Symmetric, skew-symmetric and orthogonal matrices; Complex matrices: Hermitian, Skew- Hermitian and unitary matrices; Elementary row and column transformations, finding rank of a matrix by reducing to Echelon form and Normal form; Finding the inverse of a matrix using Gauss-Jordan method;

MODULE –II: LINEAR TRANSFORMATIONS (09)

Cayley-Hamilton theorem: Statement, verification, finding inverse and powers of a matrix; Linear dependence and independence of vectors; Linear transformation; Eigen values and Eigen vectors of a matrix; Diagonalization of matrix by linear transformation.

MODULE –III: FUNCTIONS OF SINGLE AND SEVERAL VARIABLES (09)

Mean value theorems: Rolle's theorem, Lagrange's theorem, Cauchy's theorem-without proof.

Functions of several variables: Partial differentiation, Jacobian, functional dependence, maxima and minima of functions with two variables and three variables. Method of Lagrange multipliers.

MODULE –IV: HIGHER ORDER LINEAR DIFFERENTIAL EQUATIONS (09)

Linear differential equations of second and higher order with constant coefficients.

Non-homogeneous term of the type $f(x)=e^{ax}, \sin ax, \cos ax$ and $f(x)=x^n, e^{ax}v(x)$, Method of variation of parameters.

MODULE –V: FOURIER SERIES (09)

Fourier expansion of periodic function in a given interval of length 2π; Fourier series of even and odd functions; Fourier series in an arbitrary interval, Half- range Fourier sine and cosine expansions.

IV. TEXT BOOKS

1. B.S. Grewal, Higher Engineering Mathematics, Khanna Publishers, 36th Edition, 2010.
2. N.P. Bali and Manish Goyal, A text book of Engineering Mathematics, Laxmi Publications, Reprint, 2008.
3. Ramana B.V., Higher Engineering Mathematics, Tata McGraw Hill New Delhi, 11th Reprint 2010.

V. REFERENCE BOOKS:

1. Erwin Kreyszig, Advanced Engineering Mathematics, 9th Edition, John Wiley & Sons, 2006.
2. Veerarajan T., Engineering Mathematics for first year, Tata McGraw-Hill, New Delhi, 2008.
3. D. Poole, Linear Algebra: A Modern Introduction, 2nd Edition, Brooks/Cole, 2005.

VI. WEB REFERENCES:

1. http://www.efunda.com/math/math_home/math.cfm
2. http://www.ocw.mit.edu/resourcs/#Mathematics
3. http://www.sosmath.com
4. http://www.mathworld.wolfram.com

VII. E-TEXT BOOKS:

1. http://www.e-booksdirectory.com/details.php?ebook=10166
2. http://www.e-booksdirectory.com/details.php?ebook=7400re

CHEMISTRY

I Semester: CSE / CSE (AI&ML) / CSE (DS) / CSE (CS) / / CSIT / IT

II Semester: AE / ME / CE / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC06	Foundation	2	-	-	2	30	70	100
Contact Classes: 45	Tutorial Classes: 0	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There are no prerequisites to take this course.

I. COURSE OVERVIEW:

The concepts developed in this course involve elements and compounds and their applied industrial applications. It deals with topics such as batteries, corrosion and control of metallic materials, water and its treatment for different purposes, engineering materials such as plastics, elastomers and biodegradable polymers, their preparation, properties and applications, energy sources and environmental science. Sustainable chemistry that focuses on the design of the products and processes that minimize or eliminate the use and generation of hazardous substances is also included.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The concepts of electrochemical principles and causes of corrosion in the new developments and breakthroughs efficiently in engineering and technology.
II. The different parameters to remove causes of hardness of water and their reactions towards complexometric method.
III. The polymerization reactions with respect to mechanisms and its significance in industrial applications.
IV. The Significance of Green chemistry to reduce pollution in environment by using natural resources.

III. COURSE SYLLABUS

MODULE-I: ELECTROCHEMISTRY AND CORROSION (09)

Electro chemical cells: Electrode potential, standard electrode potential, Calomel electrode and Nernst equation; Electrochemical series and its applications; Numerical problems; Batteries: Primary (Dry cell) and secondary batteries (Lead-acid storage battery, Li-ion battery). Corrosion: Causes and effects of corrosion: Theories of chemical and electrochemical corrosion, mechanism of electrochemical corrosion; Corrosion control methods: Cathodic protection, sacrificial anode and impressed current Cathodic protection; Surface coatings: Metallic coatings- Methods of coating- Hot dipping- galvanization and tinning, electroplating.

MODULE –II: WATER TECHNOLOGY (09)

Introduction: Hardness of water, causes of hardness; types of hardness: temporary and permanent hardness, expression and units of hardness; estimation of hardness of water by complexometric method; potable water and its specifications, Steps involved in the treatment of water, disinfection of water by chlorination and ozonization; External treatment of water; Ion-exchange process; Desalination of water: Reverse osmosis, numerical problems.

MODULE-III: ENGINEERING MATERIALS (09)

Polymers-classification with examples, polymerization-addition, condensation and co-polymerization;
Plastics: Thermoplastics and thermosetting plastics; Compounding of plastics; Preparation, properties and applications of polyvinyl chloride, Teflon, Bakelite and Nylon-6, 6; Elastomers: Natural rubber, processing of natural rubber, vulcanization; Buna-s and Thiokol rubber; Biodegradable polymers.

Lubricants: characteristics of lubricants, mechanism of lubrication – thick film, thin film, extreme pressure lubrication, properties – flash and fire point, cloud and pour point, viscosity and oiliness of lubricants.

MODULE –IV: GREEN CHEMISTRY AND FUELS (09)

Introduction: Definition of green chemistry, methods of green synthesis: aqueous phase, microwave method, phase transfer catalyst and ultra sound method. Fuels: definition, classification of fuels ; Solid fuels: coal; analysis of coal: proximate and ultimate analysis; Liquid fuels: Petroleum and its refining; Gaseous fuels: Composition, characteristics and applications of LPG and CNG; Calorific value: Gross Calorific value(GCV) and Net Calorific value(NCV), numerical problems.

MODULE –V: NATURAL RESOURCES AND ENVIRONMENTAL POLLUTION (09)

Natural resources: Classification of resources, living and nonliving resources; Water resources: Use and over utilization of surface and ground water, floods and droughts, dams, benefits and problems; Land resources; Energy

resources: renewable and non-renewable energy sources, use of alternate energy source. Environmental pollution: Causes, effects and control measures of air pollution, water pollution, soil pollution and noise pollution.

IV. TEXT BOOKS:

1. P. C. Jain and Monica Jain, "Engineering Chemistry", DhanpatRai Publishing Company, 16th Edition, 2017.
2. ShashiChawla, "Text Book of Engineering Chemistry" DhanatRai and Company, 2017.
3. Prashanthrath, B.Rama Devi, Ch.VenkataRamana Reddy, Subhendu Chakroborty, Cengage Learning Publishers, 1st Edition, 2018.

V. REFERENCE BOOKS:

1. Bharathi Kumari, "Engineering Chemistry", VGS Book Links, 10th Edition, 2018.
2. B. Siva Shankar, "Engineering Chemistry", Tata McGraw Hill Publishing Limited, 3rd Edition, 2015.
3. S. S. Dara, Mukkanti, "Text of Engineering Chemistry", S. Chand & Co, New Delhi, 12th Edition, 2006.

VI. WEB REFERENCES:

1. Engineering chemistry (NPTEL Web-book), by B.L.Tembe, Kamaluddin and M.S.Krishnan. http://www.cdeep.iitb.ac.in/webpage_data/nptel/Core%20Science/Engineering%20Chemistry%201/About-Faculty.html
2. Polymer Science (NPTEL Web-book), by Prof. Dibakar Dhara https://onlinecourses.nptel.ac.in/noc20_cy21/preview
3. Environmental Chemistry and Analysis(NPTEL Web-book), by Prof. M.S.Subramanian https://nptel.ac.in/courses/122/106/122106030/

BASIC ELECTRICAL ENGINEERING

I Semester : CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

II Semester : AE / ME / CE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AEEC01	Foundation	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: Linear Algebra and Calculus

I. COURSE OVERVIEW:

The Basic Electrical Engineering enables knowledge on electrical quantities such as current, voltage, power, energy to know the impact of technology in global and societal context, provides knowledge on basic DC and AC circuits used in electrical and electronic devices, highlights the importance of transformers, electrical machines in generation, transmission and distribution of electric power, identify the types of electrical machines suitable for particular applications.

II. COURSE OBJECTIVES:

The students will try to learn:

Understand the basic electrical circuits and circuit laws to study the behavior AC and DC circuits.
Analyze electrical circuits with the help of network theorems.
Outline the concepts of network topology to reduce complexity of network and study its behavior.
Demonstrate the working principle of AC and DC machines.
Analyse single phase transformers circuits.

III.COURSE SYLLABUS:

MODULE – I: INTRODUCTION TO ELECTRICAL CIRCUITS (09)

Circuit concept: Ohm's law, Kirchhoff's laws, equivalent resistance of networks, Source transformation, Star to delta transformation, mesh and nodal analysis; Single phase AC circuits: Representation of alternating quantities, RMS, average, form and peak factor, concept of impedance and admittance.

MODULE – II: NETWORK THEOREMS AND NETWORK TOPOLOGY (09)

Network Theorems: Superposition, Reciprocity, Thevenin's, Norton's, Maximum power transfer for DC excitations circuits. Network Topology: Definitions, Graph, Tree, Incidence matrix, Basic Cut Set and Basic Tie Set Matrices for planar networks.

MODULE – III: DC MACHINES (09)

DC generators: Principle of operation, construction, EMF equation, types of DC generators. Losses and efficiency.

DC motors: Principle of operation, back EMF, torque equation, types of DC motors, Losses and efficiency, numerical problems.

MODULE –IV: SINGLE PHASE TRANSFORMERS (08)

Single Phase Transformers: Principle of operation, construction, types of transformers, EMF equation, operation of transformer under no load and on load, Phasor diagrams, equivalent circuit, efficiency, regulation and numerical problems.

MODULE – V: AC MACHINES (09)

Three Phase Induction motor: Principle of operation, slip, slip -torque characteristics, efficiency and applications; Alternators: Introduction, principle of operation, constructional features, calculation of regulation by synchronous impedance method and numerical problems.

IV. TEXT BOOKS:

1. A Chakrabarthy, "Electric Circuits", DhanipatRai& Sons, 6th Edition, 2010.
2. A Sudhakar, Shyammohan S Palli, "Circuits and Networks", Tata McGraw-Hill, 4th Edition, 2010.
3. A E Fitzgerald and C Kingsley, "Electric Machinery", McGraw Hill Education, 2013.
4. I JNagrath, DP Kothari, "Electrical Machines", Tata McGraw-Hill publication, 3rd Edition, 2010.

V. REFERENCE BOOKS:

1. John Bird, “Electrical Circuit Theory and Technology”, Newnes, 2nd Edition, 2003.
2. C L Wadhwa, “Electrical Circuit Analysis including Passive Network Synthesis”, International, 2nd Edition, 2009.
3. David A Bell, “Electric circuits”, Oxford University Press, 7th Edition, 2009.
4. PS Bimbra, “Electrical Machines”, Khanna Publishers, 2nd Edition, 2008.

VI. WEB REFERENCES:

1. https://www.igniteengineers.com
2. https://www.ocw.nthu.edu.tw
3. https://www.uotechnology.edu.iq
4. https://www.iare.ac.in

VII. E-TEXT BOOKS

1. https://www.bookboon.com/en/concepts-in-electric-circuits-ebook
2. https://www.www.jntubook.com
3. https://www.allaboutcircuits.com
4. https://www.freeengineeringbooks.com

PYTHON PROGRAMMING

I Semester: Common for all branches

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC01	Foundation	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: There are no prerequisites to take this course.

I. COURSE OVERVIEW:

This course introduces students to writing computer programs. This course presents the principles of structured programming using the Python language, one of the most increasingly preferred languages for programming today. Because of its ease of use, it is ideal as a first programming language and runs on both the PC and Macintosh platforms. However, the knowledge gained in the course can be applied later to other languages such as C and Java. The course uses iPython Notebook to afford a more interactive experience. Topics include fundamentals of computer programming in Python, object-oriented programming and graphical user interfaces.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Acquire programming skills in core Python.
II. Acquire Object-oriented programming skills in Python.
III. Develop the skill of designing graphical-user interfaces (GUI) in Python.
IV. Develop the ability to write database applications in Python.
V. Acquire Python programming skills to move into specific branches - Internet of Things (IoT), Data Science, Machine Learning (ML), Artificial Intelligence (AI) etc.

III. SYLLABUS:

MODULE – I: INTRODUCTION TO PYTHON (09)

Introduction to Python: Features of Python, History and Future of Python, Working with Python – interactive and script mode, Identifiers and Keywords, Comments, Indentation and Multi-lining, Data types – built-in data types, Operators and Expressions, Console Input/Output, Formatted printing, Built-in Functions, Library Functions.

MODULE – II: DECISION CONTROL STATEMENTS (09)

Selection/Conditional Branching Statements: if, if-else, nested if, if-elif-else statement(s), Basic Loop Structures/ Iterative Statements – while and for loop, Nested loops, break and continue statement, pass Statement, else Statement used with loops.

MODULE – III: CONTAINER DATA TYPES (09)

Lists: Accessing List elements, List operations, List methods, List comprehension; Tuples: Accessing Tuple elements, Tuple operations, Tuple methods, Tuple comprehension, Conversion of List comprehension to Tuple, Iterators and Iterables, zip() function.

Sets: Accessing Set elements, Set operations, Set functions, Set comprehension; Dictionaries: Accessing Dictionary elements, Dictionary operations, Dictionary Functions, Nested Dictionary, Dictionary comprehension.

MODULE - IV STRINGS AND FUNCTIONS (09)

Strings: Accessing String elements, String properties, String operations.
Functions: Communicating with functions, Variable Scope and lifetime, return statement, Types of arguments, Lambda functions, Recursive functions.

MODULE - V CLASSES AND OBJECTS (09)

Classes and Objects – Defining Classes, Creating Objects, Data Abstraction and Hiding through Classes, Class Method and self Argument, Class variables and Object variables, __init()__ and __del__() method, Public and private data members, Built-in Class Attributes, Garbage Collection. OOPs Features: Abstraction, Encapsulation, Inheritance, and Polymorphism.

IV. TEXT BOOKS:

1. Reema Thareja, “Python Programming - Using Problem Solving Approach”, Oxford Press, 1st Edition, 2017.
2. Dusty Philips, “Python 3 Object Oriented Programming”, PACKT Publishing, 2nd Edition, 2015.

V. REFERENCE BOOKS:

1. Yashavant Kanetkar, Aditya Kanetkar, “Let Us Python”, BPB Publications, 2nd Edition, 2019.
2. Martin C. Brown, “Python: The Complete Reference”, McGraw Hill, Indian Edition, 2018.
3. Michael H.Goldwasser, David Letscher, “Object Oriented Programming in Python”, Prentice Hall, 1st Edition, 2007.
4. Taneja Sheetal, Kumar Naveen, “Python Programming – A Modular Approach”, Pearson, 1st Edition, 2017.
   R Nageswar Rao, “Core Python Programming”, Dreamtech Press, 2018.

VI. WEB REFERENCES:

1. https://realPython.com/Python3-object-oriented-programming/
2. https://Python.swaroopch.com/oop.html
3. https://Python-textbok.readthedocs.io/en/1.0/Object_Oriented_Programming.html
4. https://www.programiz.com/Python-programming/

EXPERIENTIAL ENGINEERING EDUCATION (ExEEd) - ACADEMIC SUCCESS

I Semester: CSE / CSE (AI&ML) / CSE (DS) / CSE(CS) / IT / CSIT

II Semester: AE / ME / CE / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC06	Foundation	2	-	-	1	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

Prerequisite: There are no prerequisites to take this course

I.COURSE OVERVIEW:

The course aims to provide students with an understating of the different learning –coding platforms, role of the entrepreneur, innovation and technology in customer centric engineering.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The different ways in engaging continuous learning process through the interaction with peers in related topics.
II. The skills and potential opportunities using well know frameworks and analytical tools.
III. The attitudes, values, characteristics, behavior and processes with processing an entrepreneurial mindset.

III. COURSE OBJECTIVES:

WEEK – I
Introduction to ExEED - Dr. Ch. Srinivasulu

WEEK – II:
Skill Development - Dr. G Ramu

WEEK – III:
Skill Development - Dr. G Ramu

WEEK – IV:
Open Source platforms for Learning , Practice and Excel in their field - Dr. M MadhuBala

WEEK – V:
Opportunities and challenges - Respective Department HOD's

WEEK – VI:
Skill Development - Dr. G Ramu

WEEK – VII:
Skill Development - Dr. G Ramu

WEEK –VIII:
Entrepreneurial Mindset - Dr. J Sirisha Devi

WEEK – IX:
Entrepreneurial Mindset - Dr. J Sirisha Devi

WEEK – X:
Innovation Culture - Dr. M Pala Prasad Reddy

WEEK – XI:
Support & Funding from various organizations - Dr. M Pala Prasad Reddy

WEEK – XII:
Rapid Prototyping - Prof. V V S H Prasad

WEEK – XIII:
Intellectual Property Rights - Mr. K Aditya Nag

WEEK – XIV:
Story Telling by Students - Dr. Ch. Srinivasulu

BASIC ELECTRICAL ENGINEERING LABORATORY

I Semester : CSE /CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AEEC04	Foundation	-	-	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 42				Total Classes: 42		

Prerequisites: Linear Algebra and Calculus

I. COURSE OVERVIEW:

The basic electrical simulation lab enable the measurement of voltage, current, resistance in complex AC and DC circuits using digital simulation.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Implement different circuits and verify circuit concepts using digital Simulation.
II. Measure impedance of series RL, RC and RLC circuits.
III. Prove the various theorems used to reduce the complexity of electrical network.

III. COURSE SYLLABUS:

Expt. 1 : OHM'S LAW, KVL AND KCL
Verification of Ohm's, Verification of Kirchhoff's current law and Voltage law using hardware and digital simulation.

Expt. 2: **MESH ANALYSIS**
Determination of mesh currents using hardware and digital simulation.

Expt. 3: **NODAL ANALYSIS**
Measurement of nodal voltages using hardware and digital simulation.

Expt. 4: IMPEDANCE OF SERIES RL AND RC CIRCUIT
Examine the impedance of series RL and RC circuit using digital simulation.

Expt. 5: IMPEDANCE OF SERIES RLC CIRCUIT
Measure the impedance of series RLC Circuit using hardware and digital simulation.

Expt. 6: **SINGLE PHASE AC CIRCUITS**
Determination of average value, RMS value, form factor, peak factor of sinusoidal wave using digital simulation.

Expt. 7: **SUPERPOSITION AND MAXIMUM POWER TRANSFER THEOREM**
Verification of superposition and maximum power transfer theorem using hardware and digital simulation.

Expt. 8: **THEVENINS AND NORTON'S THEOREM**
Verification of Thevenin's and Norton's theorem using hardware and digital simulation.

Expt. 9: **SWINBURNE"S TEST**
Predetermination of efficiency of DC shunt machine.

Expt. 10: **MAGNITETIZATION CHARACTERISTICS**
Determine the critical field resistance from magnitetization characteristics of DC shunt generator.

Expt. 11: **BRAKE TEST ON DC SHUNT MOTOR**
Study the performance characteristics of DC shunt motor by brake test.

Expt. 12: **SPEED CONTROL OF DC SHUNT MOTOR**
Verify the armature and field control techniques of DC shunt motor.

Expt. 13: **OPEN CIRCUIT AND SHORT CIRCUIT TEST ON SINGLE PHASE TRANSFORMER**

Determination of losses and efficiency of single phase transformer.

Expt. 14: **SYNCHRONOUS IMPEDENCE METHOD**
Determine the regulation of alternator using synchronous impedance method.

REFERENCE BOOKS:

1. A Chakrabarti, "Circuit Theory", Dhanpat Rai Publications, 6th Edition, 2006.
2. William Hayt, Jack E Kemmerly S.M. Durbin, "Engineering Circuit Analysis", Tata McGraw Hill, 7th Edition, 2010.
3. K S Suresh Kumar, "Electric Circuit Analysis", Pearson Education, 1st Edition, 2013.
4. Etter, "Introduction to MATLAB 7", Pearson Education, 1st Edition, 2008.

WEB REFERENCES:

1. https://www.ee.iitkgp.ac.in
2. https://www.citchennai.edu.in
3. https://www.iare.ac.in

PYTHON PROGRAMMING LABORATORY

I Semester: **Common from all branches**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC02	Foundation	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes:36		

Prerequisite: **There are no prerequisites to take this course.**

I. COURSE OVERVIEW:

This course introduces students to writing computer programs. This course presents the principles of structured programming using the Python language, one of the most increasingly preferred languages for programming today. Because of its ease of use, it is ideal as a first programming language and runs on both the PC and Macintosh platforms. However, the knowledge gained in the course can be applied later to other languages such as C and Java. The course uses iPython Notebook to afford a more interactive experience. Topics include fundamentals of computer programming in Python, object-oriented programming and graphical user interfaces.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Acquire programming skills in core Python.
II. Acquire Object-oriented programming skills in Python.
III. Develop the skill of designing graphical-user interfaces (GUI) in Python.
IV. Develop the ability to write database applications in Python.
V. Acquire Python programming skills to move into specific branches - Internet of Things (IoT), Data Science, Machine Learning (ML), Artificial Intelligence (AI) etc.

III. COURSE SYLLABUS:

Week – 1: OPERATORS

a. Read a list of numbers and write a program to check whether a particular element is present or not using membership operators.
b. Read your name and age and write a program to display the year in which you will turn 100 years old.
c. Read radius and height of a cone and write a program to find the volume of a cone.
d. Write a program to compute distance between two points taking input from the user (Hint: use Pythagorean theorem)

Week – 2: CONTROL STRUCTURES

a. Read your email id and write a program to display the no of vowels, consonants, digits and white spaces in it using if…elif…else statement.
b. Write a program to create and display a dictionary by storing the antonyms of words. Find the antonym of a particular word given by the user from the dictionary using while loop.
c. Write a Program to find the sum of a Series 1/1! + 2/2! + 3/3! + 4/4! +…….+ n/n!. (Input :n = 5, Output : 2.70833)
d. In number theory, an abundant number or excessive number is a number for which the sum of its proper divisors is greater than the number itself. Write a program to find out, if the given number is abundant. (Input: 12, Sum of divisors of 12 = 1 + 2 + 3 + 4 + 6 = 16, sum of divisors 16 > original number 12)

Week – 3: LIST

a. Read a list of numbers and print the numbers divisible by x but not by y (Assume x = 4 and y = 5).
b. Read a list of numbers and print the sum of odd integers and even integers from the list.(Ex: [23, 10, 15, 14, 63], odd numbers sum = 101, even numbers sum = 24)
c. Read a list of numbers and print numbers present in odd index position. (Ex: [10, 25, 30, 47, 56, 84, 96], The numbers in odd index position: 25 47 84).
d. Read a list of numbers and remove the duplicate numbers from it. (Ex: Enter a list with duplicate elements: 10 20 40 10 50 30 20 10 80, The unique list is: [10, 20, 30, 40, 50, 80])

Week – 4: TUPLE

a. Given a list of tuples. Write a program to find tuples which have all elements divisible by K from a list of tuples. test_list = [(6, 24, 12), (60, 12, 6), (12, 18, 21)], K = 6, Output : [(6, 24, 12), (60, 12, 6)]
b. Given a list of tuples. Write a program to filter all uppercase characters tuples from given list of tuples. (Input: test_list = [("GFG", "IS", "BEST"), ("GFg", "AVERAGE"), ("GfG",), ("Gfg", "CS")], Output : [('GFG', 'IS', 'BEST')]).
c. Given a tuple and a list as input, write a program to count the occurrences of all items of the list in the tuple. (Input : tuple = ('a', 'a', 'c', 'b', 'd'), list = ['a', 'b'], Output : 3)

Week – 5: SET

a. Write a program to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).
b. Write a program to perform union, intersection and difference using Set A and Set B.
c. Write a program to count number of vowels using sets in given string (Input : "Hello World", Output: No. of vowels : 3)
d. Write a program to form concatenated string by taking uncommon characters from two strings using set concept (Input : S1 = "aacdb", S2 = "gafd", Output : "cbgf").

Week – 6: DICTIONARY

a. Write a program to do the following operations:
   i. Create a empty dictionary with dict() method
   ii. Add elements one at a time
   iii. Update existing key's value
   iv. Access an element using a key and also get() method
   v. Deleting a key value using del() method
b. Write a program to create a dictionary and apply the following methods:
   i. pop() method
   ii. popitem() method
   iii. clear() method
c. Given a dictionary, write a program to find the sum of all items in the dictionary.
d. Write a program to merge two dictionaries using update() method.

Week – 7: STRINGS

a. Given a string, write a program to check if the string is symmetrical and palindrome or not. A string is said to be symmetrical if both the halves of the string are the same and a string is said to be a palindrome string if one half of the string is the reverse of the other half or if a string appears same when read forward or backward.
b. Write a program to read a string and count the number of vowel letters and print all letters except 'e' and 's'.
c. Write a program to read a line of text and remove the initial word from given text. (Hint: Use split() method, Input : India is my country. Output : is my country)
d. Write a program to read a string and count how many times each letter appears. (Histogram).

Week – 8: USER DEFINED FUNCTIONS

a. A generator is a function that produces a sequence of results instead of a single value. Write a generator function for Fibonacci numbers up to n.
b. Write a function merge_dict(dict1, dict2) to merge two Python dictionaries.
c. Write a fact() function to compute the factorial of a given positive number.
d. Given a list of n elements, write a linear_search() function to search a given element x in a list.

Week – 9: BUILT-IN FUNCTIONS

a. Write a program to demonstrate the working of built-in statistical functions mean(), mode(), median() by importing statistics library.
b. Write a program to demonstrate the working of built-in trignometric functions sin(), cos(), tan(), hypot(), degrees(), radians() by importing math module.
c. Write a program to demonstrate the working of built-in Logarithmic and Power functions exp(), log(), log2(), log10(), pow() by importing math module.
d. Write a program to demonstrate the working of built-in numeric functions ceil(), floor(), fabs(), factorial(), gcd() by importing math module.

Week – 10: CLASS AND OBJECTS

a. Write a program to create a BankAccount class. Your class should support the following methods for
   i) Deposit
   ii) Withdraw
   iii) GetBalanace
   iv) PinChange
b. Create a SavingsAccount class that behaves just like a BankAccount, but also has an interest rate and a method that increases the balance by the appropriate amount of interest (Hint:use Inheritance).
c. Write a program to create an employee class and store the employee name, id, age, and salary using the constructor. Display the employee details by invoking employee_info() method and also using dictionary (__dict__).
d. Access modifiers in Python are used to modify the default scope of variables. Write a program to demonstrate the 3 types of access modifiers: public, private and protected.

Week – 11: MISCELLANEOUS PROGRAMS

a. Write a program to find the maximum and minimum K elements in Tuple using slicing and sorted() method (Input: test_tup = (3, 7, 1, 18, 9), k = 2, Output: (3, 1, 9, 18))
b. Write a program to find the size of a tuple using getsizeof() method from sys module and built-in __sizeof__() method.
c. Write a program to check if a substring is present in a given string or not.
d. Write a program to find the length of a string using various methods:
   i. Using len() method
   ii. Using for loop and in operator
   iii. Using while loop and slicing

Week – 12: ADDITIONAL PROGRAMS - FILE HANDLING

1. Write a program to read a filename from the user, open the file (say firstFile.txt) and then perform the following operations:
   i. Count the sentences in the file.
   ii. Count the words in the file.
   iii. Count the characters in the file.
2. Create a new file (Hello.txt) and copy the text to other file called target.txt. The target.txt file should store only lower case alphabets and display the number of lines copied.
3. Write a Python program to store N student's records containing name, roll number and branch. Print the given branch student's details only.

IV. REFERENCE BOOKS:

1. Michael H Goldwasser, David Letscher, "Object Oriented Programming in Python", Prentice Hall, 1st Edition, 2007.
2. Yashavant Kanetkar, Aditya Kanetkar, "Let us Python", BPB publication, 1st Edition, 2019.
3. Ashok Kamthane, Amit Kamthane, "Programming and Problem Solving with Python", McGraw Hill Education (India) Private Limited, 2018.
4. Taneja Sheetal, Kumar Naveen, "Python Programming – A modular approach", Pearson, 2017.
5. R Nageswara Rao, "Core Python Programming", Dreamtech press, 2017 Edition.

V. WEB REFERENCES:

1. https://realpython.com/python3-object-oriented-programming/
2. https://python.swaroopch.com/oop.html
3. https://python-textbok.readthedocs.io/en/1.0/Object_Oriented_Programming.html
4. https://www.programiz.com/python-programming/
5. https://www.geeksforgeeks.org/python-programming-language/

ENGINEERING WORKSHOP PRACTICE

I Semester: CSE / CSE (AI&ML) / CSE (DS) / CSE (CS) / CSIT / IT

II Semester: ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AMEC04	Foundation	-	-	2	1	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 28				Total Classes: 28		

Prerequisite: There are no prerequisites to take this course.

I. COURSE OVERVIEW:

The course is intended to provide the basic concepts about Engineering tools for cutting and measuring used in a workshop. The students will be benefited from hands on training process as well as knowledge to carry out a particular process for making a product. This course provides wider perspective of manufacturing, processes to learn and introduces major trades as well as digital manufacturing facilities.

II.COURSE OBJECTIVES:

The students will try to learn:

I. The application of jigs and fixtures, measuring, marking and cutting tools in various types of manufacturing processes.
II. The preparation of different joints in carpentry and fitting and also familiarizes wood working machinery.
III. The concepts of forming processes by forging, black-smithy and tin-smithy with an application extracts of Engineering Drawing.
IV. The standard electrical wiring practices for domestic and industrial appliances.
V. The current advancements in developing the prototype models through digital manufacturing facilities.

III. COURSE SYLLABUS:

Week-1: CARPENTRY-I

Batch I: Preparation of Tenon joint as per given dimensions.
Batch II: Preparation of Mortise joint as per given taperangle.

Week -2: CARPENTRY-II

Batch I: Preparation of dove tail joint as per given taper angle.
Batch II: Preparation of lap joint as per given dimensions.

Week-3: FITTING - I

Batch I: Make a straight fit for given dimensions.
Batch II: Make a square fit for given dimensions.

Week-4: FITTING - II

Batch I : Make a V fit for given dimensions
Batch II: Make a semicircular fit for given dimensions.

Week-5: BLACKSMITHY- I

Batch I: Prepare S-bend for given MS rod using open hearth furnace.
Batch II: Prepare J-bend for given MS rod using open hearth furnace.

Week-6: BLACKSMITHY- II

Batch I: Prepare Fan hook for given dimensions.
Batch II: Prepare Round to Square for given dimensions

Week-7: MOULD PREPARATION

Batch I: Prepare a wheel flange mould using a given wooden pattern.
Batch II: Prepare a bearing housing using an aluminum pattern.

Week-8: MOULD PREPARATION
Batch I: Prepare a bearing housing using an aluminum pattern.
Batch II: Prepare a wheel flange mould using a given wooden pattern.

Week-9: TINSMITHY- I
Batch I: Prepare the development of a surface and make a rectangular tray for given dimensions.
Batch II: Prepare the development of a surface and make a round tin for given dimensions.

Week-10: TINSMITHY- II
Batch I: Prepare the development of a surface and make a Square Tin, for given dimensions.
Batch II: Prepare the development of a surface and make a Conical Funnel for given dimensions.

Week-11: ELECTRICAL WIRING-I
Batch I: Make an electrical connection of two bulbs connected in series.
Batch II:Make an electrical connection of two bulbs connected in parallel

Week-12: ELECTRICAL WIRING-II
Batch I: Make an electrical connection of one bulb controlled by two switches connected.
Batch II: Make an electrical connection of tube light.

IV. REFERENCE BOOKS:

1. Hajra Choudhury S.K., Hajra Choudhury A.K. and Nirjhar Roy S.K., "Elements of Workshop Technology", Media promoters and publishers private limited, Mumbai, Vol. I 2008 and Vol. II 2010.
2. Kalpakjian S, Steven S. Schmid, "Manufacturing Engineering and Technology", Pearson Education India Edition, 4th Edition, 2002.
3. Gowri P. Hariharan, A. Suresh Babu, "Manufacturing Technology – I", Pearson Education, 2008.
4. Roy A. Lindberg, "Processes and Materials of Manufacture", Prentice Hall India, 4th Edition, 1998.
5. Rao P.N., "Manufacturing Technology", Vol. I and Vol. II, Tata McGraw-Hill House, 2017.

V. WEB REFERENCES:

http://www.iare.ac.in

ENGLISH

I Semester: AE / ECE / EEE / ME / CE

II Semester : CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC01	Foundation	2	-	-	2	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Standard applicability of vocabulary and grammer

I. COURSE OVERVIEW:

The sole aim of the course is to enhance the communication skills of upcoming engineering graduates to meet the requirements and challenges in a competitive global world. This course is designed to provide a well-rounded introduction to English language learning. Moreover, the course pays special attention to the typical problems and challenges confronted by the Indian learners of English like mispronunciation, spellings, and structures of English due to their mother tongue influence. This course includes General Introduction to Listening Skills, Speaking Skills, Vocabulary and Grammar, Reading Skills, and Writing Skills.

II. COURSE OBJECTIVES:

The Students will try to learn:

I. The theoretical and fundamental inputs to communicate intelligibly in English through standard Pronunciation.
II. The four language skills i.e., Listening, Speaking, Reading and Writing effectively and their application in real-life situations.
III. The Writing strategies of English using correct spelling, grammar, punctuation and appropriate vocabulary.
IV. Different mechanics of writing styles forms of writing emails, reports, formal and informal letters.

III. COURSE SYLLABUS:

MODULE-I: GENERAL INTRODUCTION AND LISTENING SKILLS (09)

Introduction to communication skills; Communication process; Elements of communication; Soft skills vs hard skills; Listening skills; Significance; Stages of listening; Barriers to listening and effectiveness of listening; Listening comprehension.

MODULE –II: SPEAKING SKILLS (09)

Significance; Essentials; Barriers and effectiveness of speaking; Verbal and non-verbal communication; Generating talks based on visual prompts; Public speaking; Exposure to structured talks; Addressing a small group or a large formal gathering; Oral presentation.

MODULE –III: VOCABULARY & GRAMMAR (09)

Vocabulary: The concept of Word Formation; Root words from foreign languages and their use in English; Acquaintance with prefixes and suffixes from foreign languages in English to form derivatives;
Idioms and phrases; One-word substitutes.

Grammar: Sentence structure; Uses of phrases and clauses; Punctuation; Subject verb agreement; Modifiers; Articles; Prepositions.

MODULE –IV: READING SKILLS (09)

Significance; Techniques of reading; Skimming-Reading for the gist of a text; Scanning - Reading for specific information; Intensive; Extensive reading; Reading comprehension; Reading for information transfer; Text to diagram; Diagram to text.

MODULE –V: WRITING SKILLS (09)

Significance; Effectiveness of writing; Organizing principles of Paragraphs in documents; Techniques for writing precisely; Letter writing; Formal and Informal letter writing; E-mail writing, Report Writing.

IV. TEXT BOOKS:

1. Handbook of English for Communication (Prepared by Faculty of English, IARE).

V. REFERENCE BOOKS:

1. Sanjay Kumar and Pushp Lata. “Communications Skills”. Oxford University Press. 2011.
2. Michael Swan. “Practical English Usage”, Oxford University Press, 1995.
3. F.T. Wood. “Remedial English Grammar”. Macmillan. 2007.
4. William Zinsser. “On Writing Well”. Harper Resource Book, 2001.
5. Raymond Murphy, “Essential English Grammar with Answers”, Cambridge University Press 2nd Edition, 2011.

VI. WEB REFERENCES:

1. www.edufind.com
2. www.myenglishpages.com
3. http:grammar.ccc.comment.edu
4. http:owl.english.prudue.edu

VII. E-TEXT BOOKS:

1. http://bookboon.com/en/communication-ebooks-zip
2. http://www.bloomsbury-international.com/images/ezone/ebook/writing-skills-pdf.pdf
3. https://americanenglish.state.gov/files/ae/resource_files/developing_writing.pdf
4. http://learningenglishvocabularygrammar.com/files/idiomsandphraseswithmeaningsandexamplespdf
5. http://www.robinwood.com/Democracy/GeneralEssays/CriticalThinking.pdf

PROBABILITY AND STATISTICS

II Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC08	Foundation	3	1	-	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisite: Fundamentals of statistics

I. COURSE OVERVIEW:

Probability theory is the branch of mathematics that deals with modelling uncertainty. Inferential Statistics and regression analysis together with random variate distributions are playing an exceptional role in designing data driven technology which is familiarly known as data centric engineering. They also have wide variety applications in telecommunications and other engineering disciplines. The course covers advanced topics of probability and statistics with applications. The course includes: random variables, probability distributions, hypothesis testing, confidence intervals, and linear regression. There is an emphasis placed on real-world applications to engineering problems.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The theory of random variables, basic random variate distributions and their applications.
II. The Methods and techniques for quantifying the degree of closeness among two or more variables and linear regression analysis.
III. The Estimation statistics and Hypothesis testing which play a vital role in the assessment of the quality of the materials, products and ensuring the standards of the engineering process.
IV. The statistical tools which are essential for translating an engineering problem into probability model.

III. COURSE SYLLABUS:

MODULE-I: RANDOM VARIABLES (09)

Random variables: Basic definitions, discrete and continuous random variables; Probability distribution: Probability mass function and probability density functions.

MODULE –II: PROBABILITY DISTRIBUTION (09)

Binomial distribution; Mean and variances of Binomial distribution, Poisson distribution: Poisson distribution as a limiting case of Binomial distribution, mean and variance of Poisson distribution, Normal distribution; Mean, Variance, Mode, Median of Normal distribution.

MODULE –III: CORRELATIONS AND REGRESSION (09)

Correlation: Karl Pearson's Coefficient of correlation, Rank correlation, Repeated Ranks.

Regression: Lines of regression, Regression coefficient, Angle between two lines of regression.

MODULE –IV: TEST OF HYPOTHESIS - I (09)

Sampling: Population, Sampling, standard error; Test of significance: Null hypothesis, alternate hypothesis; Large sample tests: Test of hypothesis for single mean, difference between means, single proportion and difference between proportions.

MODULE –V: TEST OF HYPOTHESIS - II (09)

Small sample tests: Student t-distribution, F-distribution and Chi-square distribution.

TEXT BOOKS:

1. Erwin Kreyszig, "Advanced Engineering Mathematics", John Wiley & Sons Publishers, 9th Edition, 2014.
2. B. S. Grewal, "Higher Engineering Mathematics", Khanna Publishers, 42nd Edition, 2012.

V. REFERENCE BOOKS:

1. S. C. Gupta, V. K. Kapoor, "Fundamentals of Mathematical Statistics", S. Chand & Co., 10th Edition, 2000.
2. N. P. Bali, "Engineering Mathematics", Laxmi Publications, 9th Edition, 2016.

3. Richard Arnold Johnson, Irwin Miller and John E. Freund, "Probability and Statistics for Engineers", Prentice Hall, 8th Edition, 2013.

VI. WEB REFERENCES:

1. http://www.efunda.com/math/math_home/math.cfm
2. http://www.ocw.mit.edu/resourcs/#Mathematics
3. http://www.sosmath.com
4. http://www.mathworld.wolfram.com

VII. E-TEXT BOOKS:

1. http://www.keralatechnologicaluniversity.blogspot.in/2015/06/erwin-kreyszig-advanced-engineering- mathematics-ktu-ebook-download.html
2. http://www.faadooengineers.com/threads/13449-Engineering-Maths-II-eBooks

APPLIED PHYSICS

II Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC09	Foundation	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite: Basic principles of semiconductors

I. COURSE OVERVIEW:

This course is structured specifically to make the students understand some of the core topics in physics essential for further studies in engineering. It focuses on illustrating and developing an understanding of the interplay between problem solving and their practical applications which include experimental techniques and modern equipment. The topics include quantum mechanics, semiconductors, opto-electronic devices, magnetism, dielectrics, lasers and fiber optics. At the end, this course helps students to appreciate the diverse real-time applications in technological fields in respective branches.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Basic formulations in wave mechanics for the evolution of energy levels and quantization of energies for a particle in a potential box with the help of mathematical description.
II. Fundamental properties of semiconductors including the band gap, charge carrier concentration, doping and transport mechanisms.
III. The metrics of optoelectronic components, lasers, optical fiber communication and be able to incorporate them into systems for optimal performance.
IV. The appropriate magnetic and dielectric materials required for various engineering applications.

III. COURSE SYLLABUS:

MODULE-I: QUANTUM MECHANICS (09)

Introduction to quantum physics, de-broglie's hypothesis, Wave-particle duality, Davisson and Germer experiment, Time-independent Schrodinger equation for wave function, Physical significance of the wave function, Schrodinger equation for one dimensional problems–particle in a box.

MODULE –II: INTRODUCTION TO SOLIDS AND SEMICONDUCTOTS (09)

Introduction to classical free electron theory and quantum theory, Bloch's theorem for particles in a periodic potential (Qualitative treatment), Kronig-Penney model (Qualitative treatment), classification: metals, semiconductors, and insulators. Intrinsic and extrinsic semiconductors, Carrier concentration, Dependence of Fermi level on carrier-concentration and temperature, Hall effect.

MODULE –III: SEMICONDUCTOR DEVICES (09)

Direct and indirect band gaps, p-n junction, V-I characteristics, Energy Band diagram, Biasing of a junction, Zener diode.

Construction and working of LED, Photo detectors, PIN, Avalanche photodiode, Solar cell.

MODULE –IV: ENGINEERED ELECTRIC AND MAGNETIC MATERIALS (09)

Polarisation, Permittivity, Dielectric constant, Internal field in solids, Clausius Mosotti equation, Electronic, Ionic and Orientational polarization (Qualitative), Ferroelectricity; Magnetisation, Permeability, Susceptibility, Classification and properties of dia, para and ferro magnetic materials on the basis of magnetic moment, Hysteresis curve.

MODULE –V: LASERS AND FIBER OPTICS (09)

Characteristics of lasers, Spontaneous and stimulated emission of radiation, Metastable state, Population inversion, Lasing action, Ruby laser, He-Ne laser and Applications of lasers. Principle and construction of an optical fiber, Acceptance angle, Numerical aperture, Types of optical fibers (Single mode, multimode, step index, graded index), Optical fiber communication system with block diagram and Applications of optical fibers.

IV. TEXT BOOKS:

1. Dr. K Vijay Kumar and Dr. S Chandralingam, “Modern Engineering Physics” Volume-1&2, S Chand.Co, 2018.
2. Dr. M. N. Avadhanulu, Dr. P. G. Kshirsagar, “A Text Book of Engineering Physics”, S. Chand.
3. B. K Pandey and S. Chaturvedi, “Engineering Physics”, Cengage learning.

V. REFERENCE BOOKS:

1. J. Singh, “Semiconductor Optoelectronics: Physics and Technology”, McGraw-Hill Inc. (1995).
2. P. Bhattacharya, “Semiconductor Optoelectronic Devices”, Prentice Hall of India (1997).
3. Monica Katiyar and Deepak Gupta, "Optoelectronic Materials and Devices", NPTEL Online course.

VI. WEB REFERENCES:

1. http://link.springer.com/book
2. http://www.thphys.physics.ox.ac.uk
3. http://www.sciencedirect.com/science
4. http://www.e-booksdirectory.com

VII. E-TEXT BOOKS:

1. http://www.peaceone.net/basic/Feynman/
2. http://physicsdatabase.com/free-physics-books/
3. http://www.damtp.cam.ac.uk/user/tong/statphys/sp.pdf
4. http://www.freebookcentre.net/Physics/Solid-State-Physics-Books.html

PROGRAMMING FOR PROBLEM SOLVING USING C

II Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC04	Foundation	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There are no prerequisites to take this course.

I. COURSE OVERVIEW:

The main emphasis of the course will be on problem solving aspects in through C programming. The students will understand programming language, programming, concepts of loops, reading a set of data, step wise refinements, functions, control structures, arrays, dynamic memory allocations, enumerated data types, structures, unions, and file handling. This course provides adequate knowledge to solve problems in their respective domains.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Problem-solving through programming
II. Programming language, programming, reading a set of Data, stepwise refinement, concepts of Loops, Functions, Control structure, Arrays, Structure, Pointer and File concept.
III. To build efficient programs in 'C' language essential for future programming and software engineering courses.

III. COURSE SYLLABUS:

MODULE-I: INTRODUCTION (10)

Introduction to components of a computer: Memory, processor, I/O Devices, storage, operating system; Concept of assembler, compiler, interpreter, loader and linker.

Idea of Algorithms: Algorithms, Flowcharts, Pseudo code with examples, From algorithm to Programs and Source Code

Introduction to C Programming Language: History of C, Basic structure of a C program, Process of compiling and running a C program; C Tokens: Keywords, Identifiers, Constants, Strings, Special symbols, Variables, Data types; Operators, Precedence of Operators, Expression evaluation, Formatted Input/Output functions, Type Conversion and type casting.

MODULE-II: CONTROL STRUCTURES (08)

Decision Making Statements: Simple if, if-else, else if ladder, Nested if, switch case statement;
Loop control statements: for, while and do while loops, nested loops;
Unconditional Control Structures: break, continue and goto statements.

MODULE-III: ARRAYS AND FUNCTIONS (10)

Arrays: Introduction, Single dimensional array and multi-dimensional array: declaration, initialization, accessing elements of an array; Operations on arrays: traversal, reverse, insertion, deletion, merge, search; **Strings:** Arrays of characters, Reading and writing strings, String handling functions, Operations on strings; array of strings.

Functions: Concept of user defined functions, Function declaration, return statement, Function prototype, Types of functions, Inter function communication, Function calls, Parameter passing mechanisms; Recursion; Passing arrays to functions, passing strings to functions; Storage classes.

MODULE-IV: POINTER AND STRUCTURES (10)

Pointers: Basics of pointers, Pointer arithmetic, pointer to pointers, array of pointers, Generic pointers, Null pointers, Pointers as functions arguments, Functions returning pointers; Dynamic memory allocation.
Structures: Structure definition, initialization, structure members, nested structures, arrays of structures, structures and functions, structures and pointers, self-referential structures; Unions: Union definition, initialization, accessing union members; bit fields, typedef, enumerations, Preprocessor directives.

MODULE-V: FILE HANDLING AND APPLICATIONS IN C (07)

File Handling: Concept of a file, text files and binary files, streams, standard I/O, formatted I/O, file I/O operations, error handling, Line I/O, miscellaneous functions; Applications in C.

IV.TEXT BOOKS:

1. Byron Gottfried, "Programming with C", Schaum's Outlines Series, McGraw Hill Education, 3rd Edition, 2017.
2. Reema Thareja, "Programming in C", Oxford university press, 2nd Edition, 2016.

V. REFERENCE BOOKS:

I. W. Kernighan Brian, Dennis M. Ritchie, "The C Programming Language", PHI Learning, 2nd Edition, 1988.
II. Yashavant Kanetkar, "Exploring C", BPB Publishers, 2nd Edition, 2003.
III. Schildt Herbert, "C: The Complete Reference", Tata McGraw Hill Education, 4th Edition, 2014.
IV. R. S. Bichkar, "Programming with C", Universities Press, 2nd Edition, 2012.
V. Dey Pradeep, Manas Ghosh, "Computer Fundamentals and Programming in C", Oxford University Press, 2nd Edition, 2006.
VI. Stephen G. Kochan, "Programming in C", Addison-Wesley Professional, 4th Edition, 2014.

VI.WEB REFERENCES:

1. https://www.calvin.edu/~pribeiro/courses/engr315/EMFT_Book.pdf
2. https://www.web.mit.edu/viz/EM/visualizations/coursenotes/modules/guide02.pdf
3. https://www.nptel.ac.in/courses/108106073/
4. https://www.iare.ac.in

VII.E-TEXT BOOKS:

1. http://www.freebookcentre.net/Language/Free-C-Programming-Books-Download.htm
2. http://www.imada.sdu.dk/~svalle/courses/dm14-2005/mirror/c/
3. http://www.enggnotebook.weebly.com/uploads/2/2/7/1/22718186/ge6151-notes.pdf

ENGLISH LANGUAGE AND COMMUNICATION SKILLS LABORATORY

I SEMESTER: **AE / ECE / EEE / ME / CE**

II SEMESTER: **CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC04	Foundation	-	-	2	1	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: 45				Total Classes: 45		

Prerequisite: **There are no prerequisites to take this course.**

I. COURSE OVERVIEW

The sole aim of the course is to enhance the communication skills of upcoming engineering graduates to meet the requirements and challenges in a competitive global world. This course includes General Introduction to Listening Skills, Speaking Skills, Vocabulary and Grammar, Reading Skills, and Writing Skills.

II. COURSE OBJECTIVES:

The students will try to:

I. Improve their ability to listen and comprehend a given text.
II. Upgrade the fluency and acquire a functional knowledge of English Language.
III. Enrich thought process by viewing a problem through multiple angles.

III. COURSE SYLLABUS:

Week-l: LISTENING SKILL

a. Listening to conversations and interviews of famous personalities in various fields; Listening practice related to the TV talk shows and news.
b. Listening for specific information; Listening for summarizing information – Testing..

Week-2: LISTENING SKILL

a. Listening to films of short duration and monologues for taking notes; Listening to answer multiple choice questions.
b. Listening to telephonic conversations; Listening to native Indian: Abdul Kalam, British: Helen Keller and American: Barrack Obama speakers to analyze intercultural differences – Testing.

Week-3: SPEAKING SKILL

a. Functions of English Language; Introduction to pronunciation; Vowels and Consonants
b. Tips on how to develop fluency, body language and communication; Introducing oneself: Talking about yourself, others, leave taking.

Week-4: SPEAKING SKILL

a. Sounds - Speaking exercises involving the use of Vowels and Consonant sounds in different contexts;Exercises on Homophones and Homographs
b. Just a minute (JAM) session.

Week-5: SPEAKING SKILL

a. Stress patterns.
b. Situational Conversations: common everyday situations; Acting as a compere and newsreader; Greetings for differentoccasionswithfeedbackpreferablythroughvideorecording.

Week-6: READING SKILL

a. Intonation.
b. Reading newspaper and magazine articles; Reading selective autobiographies for critical commentary.

Week-7: READING SKILL

a. Improving pronunciation through tongue twisters.
b. Reading advertisements, pamphlets; Reading comprehension exercises with critical and analytical questions based on context.

Week-8: WRITING SKILL

a. Listening to inspirational short stories and Writing messages
b. Writing leaflets, Notice; Writing tasks; Flashcards – Exercises

Week-9: WRITING SKILL

a. Write the review on a video clipping of short duration (5 to 10minutes).
b. Write a slogan related to the image; Write a short story of 6-10 lines based on the hints given.

Week-10: WRITING SKILL

a. Minimising Mother Tongue interference to improve fluency through watching educational videos.
b. Writing practices – précis writing; Essay writing

Week-11: THINKING SKILL

a. Correcting common errors in day to day conversations.

Practice in preparing thinking blocks to decode diagrammatical representations into English words,expressions, idioms, proverbs.

IV. TEXT BOOK:

1. "English Language and Communication Skills" Lab Manual - Prepared by the faculty of English, IARE.

V. REFERENCE BOOKS:

1. Meenakshi Raman, Sangeetha Sharma, "Technical Communication Principles and Practices", Oxford University Press, New Delhi, 3rd Edition, 2015.
2. Rhirdion, Daniel, "Technical Communication", Cengage Learning, New Delhi, 1st Edition, 2009.

PHYSICS LABORATORY

I Semester: AE / ME / CE / ECE / EEE

II Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC05	Foundation	-	-	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

Pre-Requisites: Basic principles of Physics

I. COURSE OVERVIEW:

This course is designed to lay a strong foundation in Engineering Physics that forms a basis to various branches of Engineering. It helps the students to perform experiments, to correlate theory with experimental data, analyse using graphical representations and present them as part of a clear, well-organized lab report. At the end of the course, students will be able to demonstrate a working knowledge of fundamentals of Physics and communicate their ideas effectively, both orally and in writing.

II. COURSE OBJECTIVES:

The students will try to learn:

1. Experimental skills in using optical instruments to determine physical constants.
2. The real time applications of electromagnetic theory.
3. The working principles of various electronic devices.

III. COURSE SYLLABUS:

Week-1: HALL EFFECT (LORENTZ FORCE)
Determination of charge carrier density.

Week-2: MELDE'E EXPERIMENT
Determination of frequency of a given tuning fork.

Week-3: STEWART GEE'S APPARATUS
Magnetic field along the axis of current carrying coil-Stewart and Gee's method.

Week-4: B-H CURVE WITH CRO
To determine the energy loss per unit volume of a given magnetic material per cycle by tracing the Hysteresis loop (B-H curve).

Week-5: ENERGY GAP OF A SEMICONDUCTOR DIODE
Determination of energy gap of a semiconductor diode.

Week-6: PHOTO DIODE
Studying V-I characteristics of photo diode.

Week-7: OPTICAL FIBER
Evaluation of numerical aperture of a given optical fiber.

Week-8: WAVE LENGTH OF LASER LIGHT
Determination of wavelength of a given laser light using diffraction grating.

Week-9: PLANCK'S CONSTANT
Determination of Planck's constant using LED.

Week-10: LIGHT EMITTING DIODE
Studying V-I characteristics of LED

Week-11: NEWTONS RINGS
Determination of radius of curvature of a given plano-convex lens.

Week-12: SINGLE SLIT DIFFRACTION
Determination of width of a given single slit.

IV. MANUALS:

1. C. L. Arora, "Practical Physics", S. Chand & Co., New Delhi, 3rd Edition, 2012.
2. VijayKumar, Dr.T.Radhakrishna, "Practical Physics for Engineering Students", SM Enterprises, 2nd Edition, 2014.

V. WEB REFERENCE:

http://www.iare.ac.in

PROGRAMMING FOR PROBLEM SOLVING USING C LABORATORY

II Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE /EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC05	Foundation	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes:36		

Prerequisite: There are no prerequisites to take this course.

I. COURSE OVERVIEW

This course introduces students to writing computer programs. This course presents the principles of structured programming using the Python language, one of the most increasingly preferred languages for programming today. Because of its ease of use, it is ideal as a first programming language and runs on both the PC and Macintosh platforms. However, the knowledge gained in the course can be applied later to other languages such as C and Java. The course uses iPython Notebook to afford a more interactive experience.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Acquire logical thinking and identify efficient ways of solving problems using C programming language.
II. Develop programs by using decision making, branching and looping constructs.
II. Implement real time applications using the concept of array, pointers, functions and structures.

III. COURSE SYLLABUS:

Week – 1: OPERATORS AND EVALUATION OF EXPRESSIONS

e. Design and develop a flowchart and algorithm to read a number and implement using a C program to check whether the given number is even or odd using ternary operator.
f. Design and develop a flowchart and algorithm to read two integers and implement using a C program to perform the addition of two numbers without using + operator.
g. Develop a C program to evaluate the following arithmetic expressions by reading appropriate input from the standard input device. Understand the priority of operators while evaluating expressions.
   i. 6*2/(2+1 * 2/3 +6) +8 * (8/4)
   ii. 17 – 8 / 4 * 2 + 3 - ++2
   iii. ! (x > 10) && (y = = 2)
h. Develop a C program to display the size of various built-in data types in C language.

Week – 2: CONTROL STRUCTURES

a. Design and develop a flowchart and algorithm to read a year as an input and find whether it is leap year or not. Implement a C program for the same and execute for all possible inputs with appropriate messages. Also consider end of the centuries.
b. Design and develop a flowchart and algorithm to find the square root of a given number N. Implement a C program for the same and execute for all possible inputs with appropriate messages. (Note: Don't use library function sqrt(n), Hint: Use Newton-Raphson method to find the square root).
c. Design and develop a flowchart and algorithm to generate a Fibonacci sequence up to a given number N. A Fibonacci sequence is defined as follows: The first and second terms in the sequence are 0 and 1. Subsequent terms are found by adding the preceding two terms in the sequence. Implement a C program for the developed flowchart/algorithm and execute the same to generate the first N terms of the sequence.
d. Design and develop a flowchart and algorithm that takes three coefficients (a, b, and c) of a Quadratic equation ($ax^2+bx+c=0$) as input and compute all possible roots. Implement a C program for the developed flowchart/algorithm and execute the same to output the possible roots for a given set of coefficients with appropriate messages.

Week – 3: CONTROL STRUCTURES

a. Design and develop an algorithm to find the reverse of an integer number N and check whether it is PALINDROME or NOT. Implement a C program for the developed algorithm that takes an integer number as input and output the reverse of the same with suitable messages. Ex: N: 2020, Reverse: 0202, Not a Palindrome.

b. Draw the flowchart and write C Program to compute sin(x) using Taylor series approximation given by

$$\sin(x) = x - (x^3/3!) + (x^5/5!) - (x^7/7!) + \ldots\ldots$$

Compare the result with the built- in Library function and print both the results with appropriate messages.

c. Design and develop an algorithm and flowchart to read a three digit number and check whether the given number is Armstrong number or not. Write a C program to implement the same and also display the Armstrong numbers between the ranges 1 to 1000.

d. Design and develop an algorithm for evaluating the polynomial $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$, for a given value of x and its coefficients using Horner's method. Implement a C program for the same and execute the program for different sets of values of coefficients and x.

Week – 4: ARRAYS

e. Develop, implement and execute a C program to read a list of integers and store it in a single dimensional array. Write a C program to print the second largest integer in a list of integers.

f. Develop, implement and execute a C program to read a list of integers and store it in a single dimensional array. Write a C program to count and display positive, negative, odd and even numbers in an array.

g. Develop, implement and execute a C program to read a list of integers and store it in a single dimensional array. Write a C program to find the frequency of a particular number in a list of integers.

h. Develop, implement and execute a C program that reads two matrices A (m x n) and B (p x q) and Compute the product A and B. Read matrix A and matrix B in row major order respectively. Print both the input matrices and resultant matrix with suitable headings and output should be in matrix format only. Program must check the compatibility of orders of the matrices for multiplication. Report appropriate message in case of incompatibility.

Week – 5: STRINGS

a. Develop a user-defined function **STRCOPY (str1, str2)** to simulate the built-in library function **strcpy (str1, str2)** that copies a string str2 to another string str1. Write a C program that invokes this function to perform string copying. Also perform the same operation using built-in function.

b. Develop a user-defined function **STRCONCT (str1, str2)** to simulate the built-in library function **strcat (str1, str2)** that takes two arguments str1 and str2, concatenates str2 and str1 and stores the result in str1. Write a C program that invokes this function to perform string concatenation. Also perform the same operation using built-in function.

c. Develop a C program that returns a pointer to the first occurrence of the string in a given string using built-in library function **strstr()**. Example: **strstr()** function is used to locate first occurrence of the string "test" in the string "This is a test string for testing". Pointer is returned at first occurrence of the string "test".

d. Develop a C program using the library function **strcmp (str1, str2)** that compares the string pointed to by str1 to the string pointed to by str2 and returns an integer. Display appropriate messages based on the return values of this function as follows −

if return value < 0 then it indicates str1 is less than str2.
if return value > 0 then it indicates str2 is less than str1.
if return value = 0 then it indicates str1 is equal to str2.

Week – 6: FUNCTIONS

a. Design and develop a recursive and non-recursive function **FACT(num)** to find the factorial of a number, n!, defined by FACT(n) = 1, if n = 0. Otherwise FACT(n) = n * FACT(n-1). Using this function, write a C program to compute the binomial coefficient. Tabulate the results for different values of n and r with suitable messages

b. Design and develop a recursive function **GCD (num1, num2**) that accepts two integer arguments. Write a C program that invokes this function to find the greatest common divisor of two given integers.

c. Design and develop a recursive function **FIBO (num)** that accepts an integer argument. Write a C program that invokes this function to generate the Fibonacci sequence up to num.

d. Design and develop a C function **ISPRIME** (num) that accepts an integer argument and returns 1 if the argument is prime, a 0 otherwise. Write a C program that invokes this function to generate prime numbers between the given ranges.

e. Design and develop a function **REVERSE (str**) that accepts a string arguments. Write a C program that invokes this function to find the reverse of a given string.

Week – 7: POINTERS

a. Develop a C program using pointers to compute the sum, mean and standard deviation of all elements stored in an array of n real numbers.

b. Develop a C program to read a list of integers and store it in an array. Then read the array elements using a pointer and print the value along with the memory addresses.

c. Design and develop non-recursive functions **input_matrix(matrix, rows, col**s) and **print_matrix(matrix, rows, cols)** that stores integers into a two-dimensional array and displays the integers in matrix form. Write a C program to input and print elements of a two dimensional array using pointers and functions.

d. Develop a C program to a store a list of integers in a single dimensional array using dynamic memory allocation (limit will be at run time) using malloc() function. Write a C program to read the elements and print the sum of all elements along with the entered elements. Also use free() function to release the memory.

Week – 8: **STRUCTURES AND UNIONS**

e. Write a C program that uses functions to perform the following operations:
   i. Reading a complex number
   ii. Writing a complex number
   iii. Addition and subtraction of two complex numbers
   Note: represent complex number using a structure.

f. Write a C program to compute the monthly pay of 100 employees using each employee_s name, basic pay. The DA is computed as 52% of the basic pay. Gross-salary (basic pay + DA). Print the employees name and gross salary.

g. Create a Book structure containing book_id, title, author name and price. Write a C program to pass a structure as a function argument and print the book details.

h. Create a union containing 6 strings: name, home_address, hostel_address, city, state and zip. Write a C program to display your present address.

Week – 9: ADDITIONAL PROGRAMS

a. Write a C program to read in two numbers, x and n, and then compute the sum of this geometric progression: $1+x+x^2+x^3+\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots+x^n$. For example: if n is 3 and x is 5, then the program computes 1+5+25+125. Print x, n, the sum. Perform error checking. For example, the formula does not make sense for negative exponents – if n is less than 0. Have your program print an error message if n<0, then go back and read in the next pair of numbers of without computing the sum. Are any values of x also illegal? If so, test for them too.

b. Develop a C program to find the 2's complement of a given binary number. 2's complement is obtained by scanning it from right to left and complementing all the bits after the first appearance of a 1. Thus 2's complement of 11100 is 00100. Write a C program to find the 2's complement of a binary number.

c. Develop a C program to convert a Roman numeral to its decimal equivalent. E.g. check for the inputs - Roman number IX is equivalent to 9 and Roman number XI is equivalent to 11.

Week – 10: PREPROCESSOR DIRECTIVES

a. Define a macro with one parameter to compute the volume of a sphere. Write a C program using this macro to compute the volume for spheres of radius 5, 10 and 15meters.

b. Define a macro that receives an array and the number of elements in the array as arguments. Write a C program for using this macro to print the elements of the array.

c. Write symbolic constants for the binary arithmetic operators +, -, *, and /. Write a C program to illustrate the use of these symbolic constants.

Week – 11: FILES

a. Create an employee file **employee.txt** and write 5 records having employee name, designation, salary, branch and city. Develop a C program to display the contents of **employee.txt** file.

b. Create a **studentolddata.txt** file containing student name, roll no, branch, section, address. Develop a C program to copy the contents of **studentolddata.txt** file to another file **studentnewdata.txt**.

c. Develop a C program to create a text file **info.txt** to store the information given below. Implement using a C program to count the number of words and characters in the file **info.txt**.

   Test Data:
   Input the file name to be opened : info.txt
   Expected Output:
   The content of the file info.txt are :
   Welcome to IARE
   Welcome to Computer Programming

The number of words in the file info.txt are : 7
The number of characters in the file info.txt are : 46

d. Given two university information files "**studentname.txt**" and "**roll_number.txt**" that contains students Name and Roll numbers respectively. Write a C program to create a new file called "**output.txt**" and copy the content of files "**studentname.txt**" and "**roll_number.txt**" into output file. Display the contents of output file "**output.txt**" on to the screen.

studname.txt	**roll_number.txt**
Asha	20951A1201
Bharath	20951A0502
Uma	20951A0456
Shilpa	20951A0305

Week – 12: **COMMAND LINE ARGUMENTS**

a. Develop a C program to read a set of arguments and display all arguments given through command line.
b. Develop a C program to read a file at command line argument and display the contents of the file.
c. Develop a C program to read N integers and find the sum of N integer numbers using command line arguments.
d. Develop a C program to read three integers and find the largest integer among three using command line argument.

IV. REFERENCE BOOKS:

1. Yashavant Kanetkar, "Let Us C", BPB Publications, New Delhi, 13th Edition, 2012.
2. Oualline Steve, "Practical C Programming", O'Reilly Media, 3rd Edition, 1997.
3. King KN, "C Programming: A Modern Approach", Atlantic Publishers, 2nd Edition, 2015.
4. Kochan Stephen G, "Programming in C: A Complete Introduction to the C Programming Language", Sam's Publishers, 3rd Edition, 2004.
5. Linden Peter V, "Expert C Programming: Deep C Secrets'', Pearson India, 1st Edition, 1994.

V. WEB REFERENCES:

1. http://www.sanfoundry.com/c-programming-examples
2. http://www.geeksforgeeks.org/c
3. http://www.cprogramming.com/tutorial/c
4. http://www.cs.princeton.edu

DATA STRUCTURES

III Semester: **Common for all branches**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC08	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: **Python Programming**

I. COURSE OVERVIEW:

The course covers some of the general-purpose data structures and algorithms, and software development. Topics covered include managing complexity, analysis, static data structures, dynamic data structures and hashing mechanisms. The main objective of the course is to teach the students how to select and design data structures and algorithms that are appropriate for problems that they might encounter in real life. This course reaches to student by power point presentations, lecture notes, and lab which involve the problem solving in mathematical and engineering areas.

II. COURSE OBJECTIVES:

The students will try to learn:

I. To provide students with skills needed to understand and analyze performance trade-offs of different algorithms / implementations and asymptotic analysis of their running time and memory usage.
II. To provide knowledge of basic abstract data types (ADT) and associated algorithms: stacks, queues, lists, tree, graphs, hashing and sorting, selection and searching.
III. The fundamentals of how to store, retrieve, and process data efficiently.
IV. To provide practice by specifying and implementing these data structures and algorithms in Python.
V. Understand essential for future programming and software engineering courses.

III. SYLLABUS:

MODULE – I: INTRODUCTION TO DATA STRUCTURES, SEARCHING AND SORTING (09)

Basic concepts: Introduction to data structures, classification of data structures, operations on data structures; Algorithm Specification, Recursive algorithms, Data Abstraction, Performance analysis- time complexity and space complexity, Asymptotic Notation-Big O, Omega, and Theta notations. Introduction to Linear and Non Linear data structures, Searching techniques: Linear and Binary search; Sorting techniques: Bubble, Selection, Insertion, Quick and Merge Sort and comparison of sorting algorithms.

MODULE – II: LINEAR DATA STRUCTURES (09)

Stacks: Stack ADT, definition and operations, Implementations of stacks using array, applications of stacks, Arithmetic expression conversion and evaluation; Queues: Primitive operations; Implementation of queues using Arrays, applications of linear queue, circular queue and double ended queue (deque).

MODULE – III: LINKED LISTS (09)

Linked lists: Introduction, singly linked list, representation of a linked list in memory, operations on a single linked list; Applications of linked lists: Polynomial representation and sparse matrix manipulation.

Types of linked lists: Circular linked lists, doubly linked lists; Linked list representation and operations of Stack, linked list representation and operations of queue.

MODULE - IV NON LINEAR DATA STRUCTURES (09)

Trees: Basic concept, binary tree, binary tree representation, array and linked representations, binary tree traversal, binary tree variants, threaded binary trees, application of trees, Graphs: Basic concept, graph terminology, Graph Representations - Adjacency matrix, Adjacency lists, graph implementation, Graph traversals – BFS, DFS, Application of graphs, Minimum spanning trees – Prims and Kruskal algorithms.

MODULE - V BINARY TREES AND HASHING (09)

Binary search trees: Binary search trees, properties and operations; Balanced search trees: AVL trees; Introduction to M-Way search trees, B trees; Hashing and collision: Introduction, hash tables, hash functions, collisions, applications of hashing.

IV. TEXT BOOKS:

1. Rance D. Necaise, "Data Structures and Algorithms using Python", Wiley Student Edition.
2. Benjamin Baka, David Julian, "Python Data Structures and Algorithms", Packt Publishers, 2017.

V. REFERENCE BOOKS:

1. S. Lipschutz, "Data Structures", Tata McGraw Hill Education, 1st Edition, 2008.
2. D. Samanta, "Classic Data Structures", PHI Learning, 2nd Edition, 2004.

VI. WEB REFERENCES:

1. https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
2. https://www.codechef.com/certification/data-structures-and-algorithms/prepare
3. https://www.cs.auckland.ac.nz/software/AlgAnim/dsToC.html
4. https://online-learning.harvard.edu/course/data-structures-and-algorithms

MATHEMATICAL FOUNDATION FOR CYBER SECURITY

III Semester: **CSE(CS)**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC01	Core	3	1	0	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisites: **There are no prerequisites to take this course.**

. COURSE OVERVIEW:

This course presents mathematical concepts in cyber security domain. The cryptography is widely used to secure data against eavesdropping. The cryptographic algorithms are based on mathematical algorithms where these algorithms use the secret key for a secure transformation of data. Number theory, coding theory and cryptography are well-known areas of information security as both are necessary for today's technology oriented; online-based world. Essentially, coding theory is associated with error correcting codes. Cyber security studies will require a strong math background. Strong analytics and statistical analysis skills needed. Encryption and programming will go hand in hand. Cyber security is a technical field and one that at its core, requires strong quantitative skills.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The basics of mathematical models used in information security.
II. The general understanding of cyber security relationship with numbers
III. The security model and analyze them before being used in many commercial, industrial as well as web application.
IV. The role of mathematics in a complex system such as the Internet.

III.SYLLABUS:

MODULE – I: INTRODUCTION TO NUMBER THEORY (09)

Definition - Divisibility - Greatest common divisor - Prime numbers - Fundamental theorem of arithmetic - Mersenne primes - Fermat numbers - Euclidean algorithm - Fermat's theorem - Euler totient function - Euler's theorem. Congruences: Definition - Basic properties of congruences - Residue classes - Chinese remainder theorem.

MODULE – II: ALGEBRAIC STRUCTURE (09)

Algebraic Structures: Groups – Cyclic groups, Cosets, Modulo groups - Primitive roots - Discrete logarithms. Rings – Sub rings, ideals and quotient rings, Integral domains. Fields – Finite fields – GF(pn), GF(2n) - Classification - Structure of finite fields. Lattice, Lattice as Algebraic system, sub lattices, some special lattices.

MODULE – III: PROBABILITY THEORY (09)

Introduction – Concepts of Probability - Conditional Probability - Baye's Theorem - Random Variables – discrete and continuous- central Limit Theorem-Stochastic ProcessMarkov Chain

Bayesian methods of estimation. Random Processes: general concepts, power spectrum, discrete-time processes, random walks and other applications, Markov chains, transition probabilities.

MODULE - IV: CODING THEORY (09)

Coding Theory: Introduction - Basic concepts: codes, minimum distance, equivalence of codes, Linear codes - Linear codes - Generator matrices and parity-check matrices - Syndrome decoding – Hamming codes - Hadamard Code - Goppa codes.

MODULE - V: PSEUDORANDOM NUMBER GENERATION (09)

Introduction and examples - Indistinguishability of Probability Distributions - Next Bit Predictors - The Blum-Blum-Shub Generator – Security of the BBS Generator.

IV. TEXT BOOKS:

1. D. S. Malik, J. Mordeson, M. K. Sen, "Fundamentals of Abstract Algebra, Tata McGraw Hill.
2. P. K. Saikia, "Linear Algebra", Pearson Education, 2009.
3. Niven, H.S. Zuckerman and H. L. Montgomery, "An Introduction to the Theory of Numbers", John Wiley and Sons, 2004.

4. D P Bersekas and J N Tsitsiklis, “Introduction to Probability”, Athena Scientific, 2008.
5. C.L. Liu, ‘Elements of Discrete mathematics’, McGraw Hill, 2008.

V. REFERENCE BOOKS:

1. Douglas Stinson, ‘Cryptography – Theory and Practice’, CRC Press, 2006.
2. Sheldon M Ross, “Introduction to Probability Models”, Academic Press, 2003.
3. Leigh Metcalf, William Casey, “Cybersecurity and Applied Mathematics”, Syngress Publisher(s): Released in June 2016.
4. Chuck Easttom, “Modern Cryptography: Applied Mathematics for Encryption and Information Security”, 1st Edition 2006.

VI. WEB REFERENCES:

1. https://www.amrita.edu/course/mathematical-foundations-cyber-security-systems
2. https://www.oreilly.com/library/view/cybersecurity-and-applied/9780128044995/?code=msdeal
3. https://cybersecurityguide.org/resources/math-in-cybersecurity/

ANALOG AND DIGITAL ELECTRONICS

III Semester: CSE / IT / CSIT / CSE (AI&ML) / CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AECC08	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: No Prerequisites

I. COURSE OVERVIEW:

This course provides the basic knowledge over the construction and functionality of the basic electronic devices such as diodes and transistors. It also provides the information about the uncontrollable and controllable electronic switches and the flow of current through these switches in different biasing conditions and also will make them to learn the basic theory of switching circuits and their applications in specified relationship between signals at the input and output terminals. They will be able to design combinational and sequential circuitsdetail. Starting from a problem statement they will learn to design circuits of logic gates that have a.They will learn to design counters, adders, sequence detectors.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The Fundamental knowledge of the operational principles and characteristics of semiconductor devices and their applications.
II. The basic concept of number systems, boolean algebra and optimized implementation of combinational and sequential circuits.
III. The perceive subsequent studies in the area of microprocessors, microcontrollers, VLSI design and embedded systems effectively use of fundamentals of digital electronics.

III. SYLLABUS:

MODULE – I: DIODE AND APPLICATIONS

Diode - Static and Dynamic resistances, Equivalent circuit, Load line analysis, Diffusion and Transition Capacitances, Diode Applications: Switch-Switching times. Rectifier - Half Wave Rectifier, Full Wave Rectifier, Bridge Rectifier, Rectifiers with Capacitive Filter.

MODULE – II: BIPOLAR JUNCTION TRANSISTOR (BJT)

Principle of Operation and characteristics - Common Emitter, Common Base, Common Collector Configurations, Operating point, DC & AC load lines, Transistor Hybrid parameter model, Determination of h-parameters from transistor characteristics, Conversion of h-parameters.

MODULE – III: NUMBER SYSTEMS

Number systems, Complements of Numbers, Codes- Weighted and Non-weighted codes and its Properties, Parity check code and Hamming code.

Boolean Algebra: Basic Theorems and Properties, Switching Functions- Canonical and Standard Form, Algebraic Simplification, Digital Logic Gates, EX-OR gates, Universal Gates, Multilevel NAND/NOR realizations.

MODULE - IV: MINIMIZATION OF BOOLEAN FUNCTIONS

Karnaugh Map Method - Up to five Variables, Don't Care Map Entries, Tabular Method,
Combinational Logic Circuits: Adders, Subtractors, comparators, Multiplexers, Demultiplexers, Encoders, Decoders and Code converters, Hazards and Hazard Free Relations.

MODULE – V: SEQUENTIAL CIRCUITS FUNDAMENTALS

Basic Architectural Distinctions between Combinational and Sequential circuits, SR Latch, Flip Flops: SR, JK, JK Master Slave, D and T Type Flip Flops, Excitation Table of all Flip Flops, Timing and Triggering Consideration, Conversion from one type of Flip-Flop to another.
Registers and Counters: Shift Registers – Left, Right and Bidirectional Shift Registers, Applications of Shift Registers - Design and Operation of Ring and Twisted Ring Counter, Operation of Asynchronous and
Synchronous Counters.

IV. TEXT BOOKS:

1. Jacob Millman , “Electronic Devices and Circuits”, McGraw Hill Education, 2017
2. Robert L. Boylestead, Louis Nashelsky, “Electronic Devices and Circuits Theory”, Pearson, 11th Edition, 2009.
3. ZviKohavi&Niraj K. Jha, “Switching and Finite Automata Theory”, Cambridge, 3rd Edition, 2010.
4. R. P. Jain, “Modern Digital Electronics” Tata McGraw-Hill, 3rd Edition, 2007.

V. REFERENCE BOOKS:

1. J. Millman, H. Taub and Mothiki S. Prakash Rao, “Pulse, Digital and Switching Waveforms”, McGraw Hill 2nd Edition, 2008.
2. S. Salivahanan, N.Suresh Kumar, AVallvaraj , “Electronic Devices and Circuits”, TMH. 2nd Edition, 2008.
3. Morris Mano, “Digital Design, PHI, 4th Edition, 2006.
4. Fredriac J. Hill, Gerald R. Peterson, “Introduction to Switching Theory and Logic Design”, John Wiley &SonsInc. 3rd Edition, 2006.

VI. WEB REFERENCES:

1. http://www-mdp.eng.cam.ac.uk/web/library/enginfo/electrical/hong1.pdf
2. https://archive.org/details/ElectronicDevicesCircuits
3. http://nptel.ac.in/courses/Webcourse-contents/IIT-ROORKEE/BASICELECTRONICS/home_page.htm
4. mcsbzu.blogspot.com
5. http://books.askvenkat.com
6. http://worldclassprogramme.com

VII. WEB REFERENCES:

1. http://services.eng.uts.edu.au/pmcl/ec/Downloads/LectureNotes.pdf
2. http://nptel.ac.in/courses/122106025/
3. http://www.freebookcentre.net/electronics-ebooks-download/Electronic-Devices-and-Circuits-(PDF-313p).html
4. https://books.google.co.in/books/about/Switching_Theory_and_Logic_Design
5. https://www.smartzworld.com/notes/switching-theory-and-logic-design-stld
6. https://www.researchgate.net/.../295616521_Switching_Theory_and_Logic_Design

COMPUTER ORGANIZATION AND ARCHITECTURE

III Semester: **CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC07	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: **Programming For Problem Solving**

I. COURSE OVERVIEW:

This course introduces the principles of basic computer organization, CPU organization, and the basic architecture concepts. The course emphasizes performance and cost analysis, instruction set design, register transfer languages, arithmetic, logic and shift micro operations, pipelining, memory technology, memory hierarchy, virtual memory management, and I/O organization of computer, parallel processing and inter process communication and synchronization.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Understand the organization and architecture of computer systems and electronic computers.
II. Study the assembly language program execution, instruction format and instruction cycle.
III. Design a simple computer using hardwired and micro-programmed control methods.
IV. Study the basic components of computer systems besides the computer arithmetic.
V. Understand input-output organization, memory organization and management, and pipelining.

III. SYLLABUS:

MODULE – I: INTRODUCTION TO COMPUTER ORGANIZATION

Basic computer organization, CPU organization, memory subsystem organization and interfacing, input or output subsystem organization and interfacing, a simple computer levels of programming languages, assembly language instructions, instruction set architecture design, a simple instruction set architecture.

MODULE – II: ORGANIZATION OF A COMPUTER

Register transfer: Register transfer language, register transfer, bus and memory transfers, arithmetic micro operations, logic micro operations, shift micro operations; Control unit: Control memory, address sequencing, micro program example, and design of control unit.

MODULE – III: CPU AND COMPUTER ARITHMETIC

CPU design: Instruction cycle, data representation, memory reference instructions, input-output, and interrupt, addressing modes, data transfer and manipulation, program control.
Computer arithmetic: Addition and subtraction, floating point arithmetic operations, decimal arithmetic unit.

MODULE - IV: INPUT-OUTPUT ORGANIZATION AND MEMORY ORGANIZATION

Memory organization: Memory hierarchy, main memory, auxiliary memory, associative memory, cache memory, virtual memory; Input or output organization: Input or output Interface, asynchronous data transfer, modes of transfer, priority interrupt, direct memory access.

MODULE – V: MULTIPROCESSORS

Pipeline: Parallel processing, pipelining-arithmetic pipeline, instruction pipeline; Multiprocessors: Characteristics of multiprocessors, inter connection structures, inter processor arbitration, inter processor communication and synchronization.

IV. TEXT BOOKS:

1. M. Morris Mano, "Computer Systems Architecture", Pearson, 3rd Edition, 2015.
2. John D. Carpinelli, "Computer Systems Organization and Architecture", Pearson, 1st Edition, 2001.
3. Patterson, Hennessy, "Computer Organization and Design: The Hardware/Software Interface", Morgan Kaufmann, 5th Edition, 2013.

V. REFERENCE BOOKS:

1. John. P. Hayes, "Computer System Architecture", McGraw-Hill, 3rd Edition, 1998.
2. Carl Hamacher, Zvonko G Vranesic, Safwat G Zaky, "Computer Organization", McGraw-Hill, 5th Edition, 2002.
3. William Stallings, "Computer Organization and Architecture", Pearson Edition, 8th Edition, 2010.

VI. WEB REFERENCES:

1. https://www.tutorialspoint.com/computer_logical_organization/
2. https://www.courseera.org/learn/comparch
3. https://www.cssimplified.com/.../computer-organization-and-assembly-language-programming

OPERATING SYSTEMS

III Semester: CSE(AI & ML) / CSE (CS) / CSE(DS)

IV Semester: CSE / IT/ CSIT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC12	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites:

I. COURSE OVERVIEW:

Operating system is system software that manages computer hardware, software resources, and provides common services for computer programs. This course provides a comprehensive introduction to operating systems design concepts, data structures and algorithms. It is designed to provide in-depth critique on the problems of resource management, scheduling, concurrency, synchronization, memory management, file management, peripheral management, protection and security. It deals with the transfer of programs in and out of memory; organizes processing time between programs and users. Various applications of operating systems include security, job accounting, error detection aids, coordination between other software's and users.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Understand the functionalities of main components in operating systems.
II. Analyze the algorithms used in memory and process management.
III. Understand the clock synchronization protocols
IV. Interpret the concepts of input and output storage for file management.

III. SYLLABUS:

MODULE – I: INTRODUCTION

Operating systems objectives and functions: Computer system architecture, operating systems structure, operating systems operations; Evolution of operating systems: Simple batch, multi programmed, time shared, personal computer, parallel distributed systems, real time systems, special purpose systems, operating system services, user operating systems interface; Systems calls: Types of systems calls, system programs, protection and security, operating system design and implementation, operating systems structure, virtual machines.

MODULE – II: PROCESS AND CPU SCHEDULING, PROCESS COORDINATION

Process concepts: The process, process state, process control block, threads; Process scheduling: Scheduling queues, schedulers, context switch, preemptive scheduling, dispatcher, scheduling criteria, scheduling algorithms, multiple processor scheduling; Real time scheduling; Thread scheduling; Case studies Linux windows; Process synchronization, the critical section problem; Peterson's solution, synchronization hardware, semaphores and classic problems of synchronization, monitors

MODULE – III: MEMORY MANAGEMENT AND VIRTUAL MEMORY

Logical and physical address space: Swapping, contiguous memory allocation, paging, structure of page table.

Segmentation: Segmentation with paging, virtual memory, demand paging; Performance of demand paging: Page replacement, page replacement algorithms, allocation of frames, thrashing.

MODULE – IV: FILE SYSTEM INTERFACE, MASS-STORAGE STRUCTURE

The concept of a file, access methods, directory structure, file system mounting, file sharing, protection, file system structure, file system implementation, allocation methods, free space management, directory implementation, efficiency and performance; Overview of mass storage structure: Disk structure, disk attachment, disk scheduling, disk management, swap space management; Dynamic memory allocation: Basic concepts; Library functions.

MODULE – V: DEADLOCKS, PROTECTION

System model: Deadlock characterization, methods of handling deadlocks, deadlock prevention, dead lock avoidance, dead lock detection and recovery form deadlock system protection, goals of protection, principles of protection, domain of protection, access matrix, implementation of access matrix, access control, revocation of access rights, capability based systems, language based protection.

IV. TEXT BOOKS:

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, "Operating System Principles", Wiley Student Edition, 8th Edition, 2010.
2. William Stallings, "Operating System- Internals and Design Principles", Pearson Education, 6th Edition, 2002.

V. REFERENCE BOOKS:

1. Andrew S Tanenbaum, "Modern Operating Systems", PHI, 3rd Edition, 2007.
2. D. M. Dhamdhere, "Operating Systems a Concept Based Approach", Tata McGraw-Hill, 2nd Edition, 2006.

VI. WEB REFERENCES:

1. www.smartzworld.com/notes/operatingsystems
2. www.scoopworld.in
3. www.sxecw.edu.in
4. www.technofest2u.blogspot.com

EXPERIENTIAL ENGINEERING EDUCATION (EXEED) – PROTOTYPE / DESIGN BUILDING

III Semester: **Common for all branches**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC09	Foundation	2	0	0	1	30	70	100
Contact Classes: 28	Tutorial Classes: Nil	Practical Classes: 28				Total Classes: 28		

Prerequisite: There are no prerequisites to take this course

I. COURSE OVERVIEW:

This course provides an overall exposure to the various methods and tools of prototyping. This course discusses Low- Fidelity, paper, wireframing and tool based prototyping techniques along with design principles and patterns.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The basic principles and design aspect of prototyping.
II. The various techniques, design guidelines and patterns.
III. The applications of prototyping using various tools and platforms.

WEEK NO	TOPIC
WEEK – I	An introduction to Prototyping
WEEK – II	Low - Fidelity Prototyping and Paper Prototyping
WEEK – III	Wireframing and Tool based Prototyping
WEEK – IV	Physical Low- Fidelity Prototyping
WEEK – V	Tool based prototyping
WEEK – VI	Design Principles and Patterns- Graphic Design
WEEK – VII	Design Principles and Patterns- Interaction Design
WEEK –VIII	Commercial design guidelines and standards.
WEEK - IX	Universal design: Sensory and cognitive impairments
WEEK - X	Universal design: Tools, Limitations and standards
WEEK - XI	Introduction platforms and context : Mobile UI design, Wearable
WEEK - XII	Introduction platforms and context : Automotive user interface
WEEK - XIII	Introduction platforms and context : IoT and Physical Computing
WEEK - XIV	Assessment

DATA STRUCTURES LABORATORY

III Semester: Common for all branches								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
ACSC10	**Core**	0	0	3	1.5	30	70	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Programming for Problem Solving using C and Python Programming								

I. COURSE OVERVIEW:

The course covers some of the general-purpose data structures and algorithms, and software development. Topics covered include managing complexity, analysis, static data structures, dynamic data structures and hashing mechanisms. The main objective of the course is to teach the students how to select and design data structures and algorithms that are appropriate for problems that they might encounter in real life. This course reaches to student by power point presentations, lecture notes, and lab which involve the problem solving in mathematical and engineering areas.

II. COURSES OBJECTIVES:

The students will try to learn

I. The skills needed to understand and analyze performance trade-offs of different algorithms / implementations and asymptotic analysis of their running time and memory usage.

II. The basic abstract data types (ADT) and associated algorithms: stacks, queues, lists, tree, graphs, hashing and sorting, selection and searching.

III. The fundamentals of how to store, retrieve, and process data efficiently.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 Interpret the complexity of algorithm using the asymptotic notations.

CO 2 Select appropriate searching and sorting technique for a given problem.

CO 3 Construct programs on performing operations on linear and nonlinear data structures for organization of a data

CO 4 Make use of linear data structures and nonlinear data structures solving real time applications.

CO 5 Describe hashing techniques and collision resolution methods for efficiently accessing data with respect to performance.

CO 6 Compare various types of data structures; in terms of implementation, operations and performance.

IV. COURSE CONTENT:

EXERCISES FOR DATA STRUCTURES LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Implicit Recursion

A specific type of recursion called **implicit recursion** occurs when a function calls itself without making an explicit recursive call. This can occur when a function calls another function, which then calls the original code once again and starts a recursive execution of the original function.

Using implicit recursion find the second-largest elements from the array.

In this case, the **find_second_largest** method calls the **find_largest()** function via implicit recursion to locate the second-largest number in a provided list of numbers. Implicit recursion can be used in this way to get the second-largest integer without having to write any more code

Input: nums = [1, 2, 3, 4, 5]

Output: 4

```
def find_largest(numbers):
     # Write code here
     …

def find_second_largest(numbers):
    # Write code here
     …

# Driver code
numbers = [1, 2, 3, 4, 5]

# Function call
second_largest = find_second_largest(numbers)
print(second_largest)
```

1.2 Towers of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, and C) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod (here considered C), obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Input: 2

Output: Disk 1 moved from A to B

Disk 2 moved from A to C
Disk 1 moved from B to C

Input: 3

Output: Disk 1 moved from A to C

Disk 2 moved from A to B
Disk 1 moved from C to B

Disk 3 moved from A to C

Disk 1 moved from B to A

Disk 2 moved from B to C

Disk 1 moved from A to C

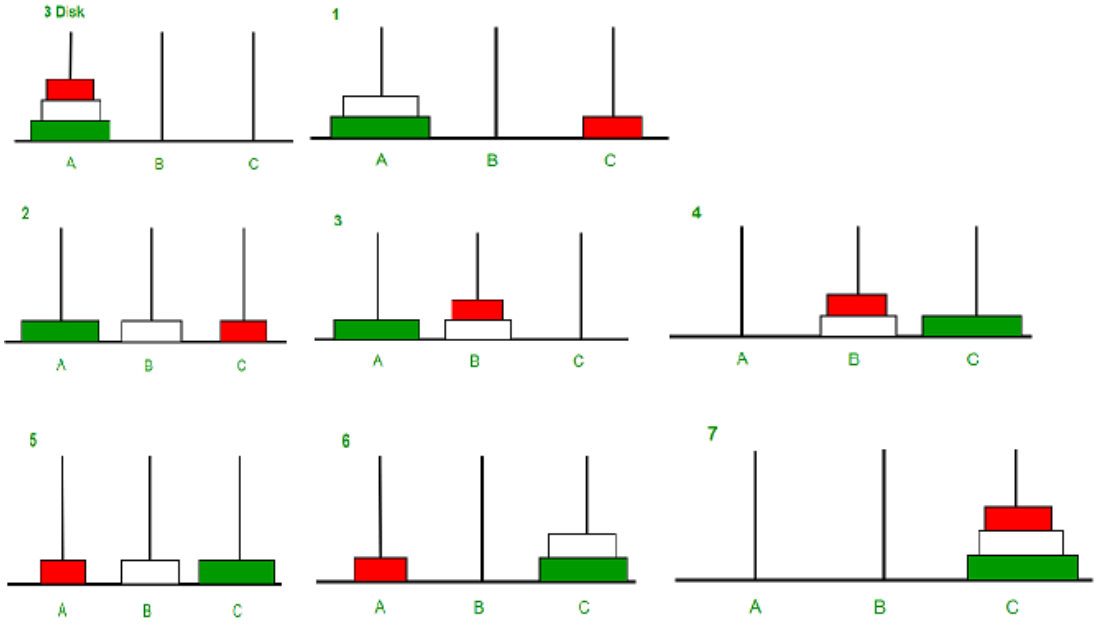
Tower of Hanoi using Recursion:

The idea is to use the helper node to reach the destination using recursion. Below is the pattern for this problem:

- Shift 'N-1' disks from 'A' to 'B', using C.
- Shift last disk from 'A' to 'C'.
- Shift 'N-1' disks from 'B' to 'C', using A.

Follow the steps below to solve the problem:

- Create a function towerOfHanoi where pass the N (current number of disk), from_rod, to_rod, aux_rod.
- Make a function call for N – 1 th disk.
- Then print the current the disk along with from_rod and to_rod
- Again make a function call for N – 1 th disk.




```
# Recursive Python function to solve Tower of Hanoi
def TowerOfHanoi(n, from_rod, to_rod, aux_rod):
    if n == 0:
        return
    # Write code here
    …
```

```
# Driver code
N = 3

# A, C, B are the name of rods
TowerOfHanoi(N, 'A', 'C', 'B')
```

1.3 Recursively Remove all Adjacent Duplicates

Given a string, recursively remove adjacent duplicate characters from the string. The output string should not have any adjacent duplicates.

Input: s = "azxxzy"

Output: "ay"

Explanation:

- First "azxxzy" is reduced to "azzy".
- The string "azzy" contains duplicates
- So it is further reduced to "ay"

Input: "caaabbbaacdddd"

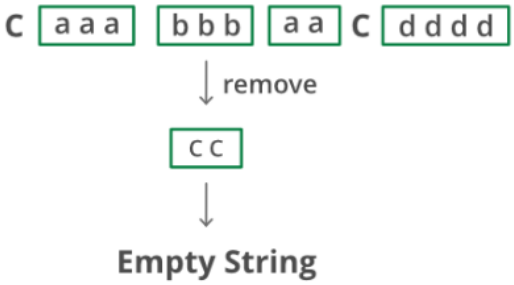
Output: Empty String

Input: "acaaabbbacdddd"

Output: "acac"

Procedure to remove duplicates:

- Start from the leftmost character and remove duplicates at left corner if there are any.
- The first character must be different from its adjacent now. Recur for string of length n-1 (string without first character).
- Let the string obtained after reducing right substring of length n-1 be rem_str. There are three possible cases
  - If first character of rem_str matches with the first character of original string, remove the first character from rem_str.
  - If remaining string becomes empty and last removed character is same as first character of original string. Return empty string.
  - Else, append the first character of the original string at the beginning of rem_str.
- Return rem_str.



```
# Program to remove all adjacent duplicates from a string

# Recursively removes adjacent duplicates from str and returns
# new string. last_removed is a pointer to last_removed character

def removeUtil(string, last_removed):

      # Write code here
        …
def remove(string):
    # Write code here
    …
# Utility functions
def toList(string):
    x = []
    for i in string:
        x.append(i)
    return x

def toString(x):
    return ''.join(x)

# Driver program
string1 = "azxxxzy"
print remove(string1)

string2 = "caaabbbaac"
print remove(string2)

string3 = "gghhg"
print remove(string3)

string4 = "aaaacddddcappp"
print remove(string4)

string5 = "aaaaaaaaaa"
print remove(string5)
```

1.4 Product of Two Numbers using Recursion

Given two numbers x and y find the product using recursion.

Input: x = 5, y = 2

Output: 10

Input: x = 100, y = 5

Output: 500

Procedure

1. If x is less than y, swap the two variables value
2. Recursively find y times the sum of x
3. If any of them become zero, return 0

```
# Find Product of two Numbers using Recursion

# recursive function to calculate multiplication of two numbers
def product( x , y ):
```

```
    # Write code here
    …
# Driver code
x = 5
y = 2
print( product(x, y))
```

1.5 Binary to Gray Code using Recursion

Given the Binary code of a number as a decimal number, we need to convert this into its equivalent Gray Code. Assume that the binary number is in the range of integers. For the larger value, we can take a binary number as string.

In gray code, only one bit is changed in 2 consecutive numbers.

Input: 1001

Output: 1101

Explanation: 1001 -> 1101 -> 1101 -> 1101

Input: 11

Output: 10

Explanation: 11 -> 10

Procedure:

The idea is to check whether the last bit and second last bit are same or not, if it is same then move ahead otherwise add 1.

Follow the steps to solve the given problem:

binary_to_grey(n)
if n == 0
    grey = 0;
else if last two bits are opposite to each other
    grey = 1 + 10 * binary_to_gray(n/10))
else if last two bits are same
    grey = 10 * binary_to_gray(n/10))

```
# Convert Binary to Gray code using recursion
# Function to change Binary to Gray using recursion
def binary_to_gray(n):
    # write code here
    …
# Driver Code
binary_number = 1011101
print(binary_to_gray(binary_number), end='')
```

1.6 Count Set-bits of a number using Recursion

Given a number N. The task is to find the number of set bits in its binary representation using recursion.

Input: 21

Output: 3

Explanation: 21 represented as 10101 in binary representation

Input: 16

Output: 1

Explanation: 16 represented as 10000 in binary representation

Procedure:

1. First, check the LSB of the number.
2. If the LSB is 1, then we add 1 to our answer and divide the number by 2.
3. If the LSB is 0, we add 0 to our answer and divide the number by 2.
4. Then we recursively follow step 1 until the number is greater than 0.

```
# Find number of set bits in a number
# Recursive function to find number of set bits in a number
def CountSetBits(n):
    # write code here
    …
# Driver code
n = 21;
# Function call
print(CountSetBits(n));
```

1.7 Fibonacci Series in Reverse Order using Recursion

Given an integer N, the task is to print the first N terms of the Fibonacci series in reverse order using Recursion.

Input: N = 5

Output: 3 2 1 1 0

Explanation: First five terms are – 0 1 1 2 3

Input: N = 10

Output: 34 21 13 8 5 3 2 1 1 0

The idea is to use recursion in a way that keeps calling the same function again till N is greater than 0 and keeps on adding the terms and after that starts printing the terms.

Follow the steps below to solve the problem:

1. Define a function fibo (int N, int a, int b) where
   i. N is the number of terms and
   ii. a and b are the initial terms with values 0 and 1.
2. If N is greater than 0, then call the function again with values N-1, b, a+b.
3. After the function call, print a as the answer.

```
# Function to print the Fibonacci series in reverse order.
def fibo(n, a, b):
    # write code here
    …
# Driver Code
N = 10
fibo(N, 0, 1)
```

1.8 Length of Longest Palindromic Sub-string using Recursion

Given a string S, the task is to find the length longest sub-string which is a palindrome.

Input: S = "aaaabbaa"

Output: 6

Explanation: Sub-string "aabbaa" is the longest palindromic sub-string.

Input: S = "banana"

Output: 5

Explanation: Sub-string "anana" is the longest palindromic sub-string.

The idea is to use recursion to break the problem into smaller sub-problems. In order to break the problem into two smaller sub-problems, compare the start and end characters of the string and recursively call the function for the middle substring.

```
# Find the length of longest palindromic sub-string using Recursion
# Function to find maximum of the two variables
def maxi(x, y):
    if x > y:
        return x
    else:
        return y
 # Function to find the longest palindromic substring: Recursion
def longestPalindromic(strn, i, j, count):
    # write code here
    …
# Function to find the longest palindromic sub-string
def longest_palindromic_substr(strn):
    # write code here
    …
strn = "aaaabbaa"
# Function Call
print(longest_palindromic_substr(strn))
```

1.9 Find the Value of a Number Raised to its Reverse

Given a number N and its reverse R. The task is to find the number obtained when the number is raised to the power of its own reverse

Input : N = 2, R = 2

Output: 4

Explanation: Number 2 raised to the power of its reverse 2 gives 4 which gives 4 as a result after performing modulo 10^9+7

Input: N = 57, R = 75

Output: 262042770

Explanation: 57^{75} modulo 10^9+7 gives us the result as 262042770

```
# Function to return ans with modulo
def PowerOfNum(N, R):
    # write code here
     …
# Driver code
N = 57
R = 75
# Function call
print(int(PowerOfNum(N, R)))
```

1.10 Mean of Array using Recursion

Find the mean of the elements of the array.

Mean = (Sum of elements of the Array) / (Total no of elements in Array)

Input: 1 2 3 4 5

Output: 3

Input: 1 2 3

Output: 2

To find the mean using recursion assume that the problem is already solved for N-1 i.e. you have to find for n

Sum of first N-1 elements = (Mean of N-1 elements) * (N-1)

Mean of N elements = (Sum of first N-1 elements + N-th elements) / (N)

```
# Program to find mean of array
# Function definition of findMean function
def findMean(A, N):
    # write code here
    …
# Driver Code
```

```
Mean = 0
A = [1, 2, 3, 4, 5]
N = len(A)
print(findMean(A, N))
```

Try:

1. Given two numbers **N** and **r**, find the value of ${}^{N}C_{r}$ using recursion.

$$C(n,r) = C(n-1,r-1) + C(n-1,r)$$

Input: N = 5, r = 2

Output: 10

Explanation: The value of 5C2 is 10

2. Predict the output of the following program. What does the following fun() do in general?

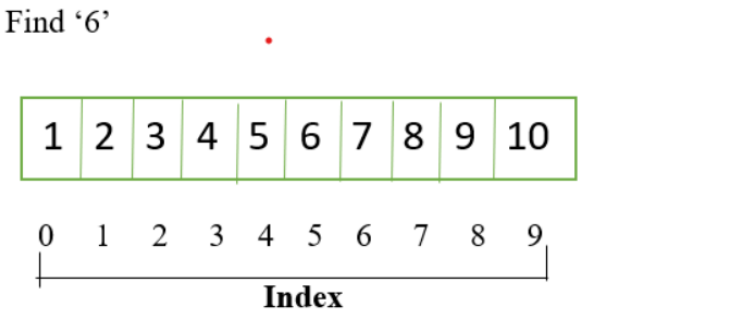
```
fp = 15
def fun(n):
    global fp
    if (n <= 2):
        fp = 1
        return 1

    t = fun(n - 1)
    f = t + fp
    fp = t
    return f

# Driver code
print(fun(5))
```

3. **Tail recursion:** Calculate factorial of a number using a Tail-Recursive function.

2. Searching

2.1 Linear / Sequential Search

Linear search is defined as the searching algorithm where the list or data set is traversed from one end to find the desired value. Given an array arr[] of n elements, write a recursive function to search a given element x in arr[].



Note : We find '6' at index '5' through linear search

Linear search procedure:

1. Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
2. If x matches with an element, return the index.

3. If x doesn't match with any of the elements, return -1.

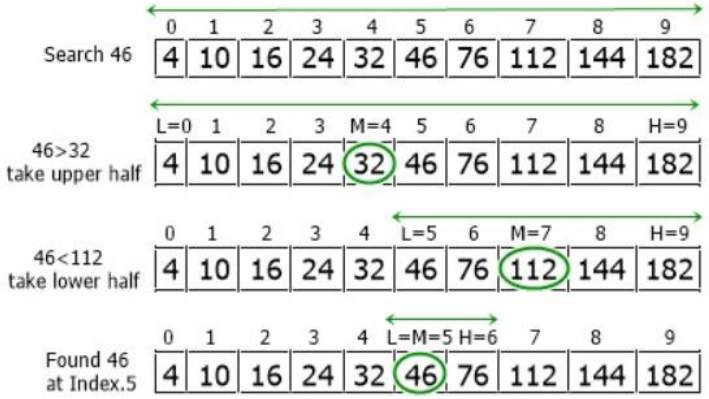
Input: arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 110;
Output: 6
Element x is present at index 6

Input: arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 175;
Output: -1
Element x is not present in arr[].

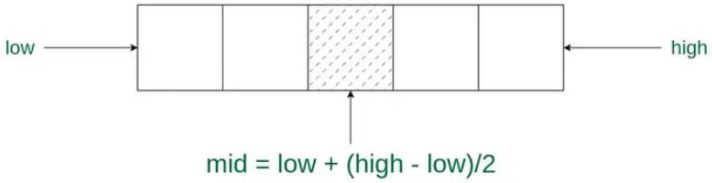
```
# Recursive linear search
def linear_search(arr, curr_index, key):
    # write code here
    …

# Driver code
arr = [10, 20, 80, 30, 60, 50, 110, 100, 130, 170]
x = 110
linear_search(arr, 0, x)
```

2.2 Binary Search

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).



Conditions for Binary Search algorithm:
1. The data structure must be sorted.
2. Access to any element of the data structure takes constant time.



Binary Search Procedure:

1. Divide the search space into two halves by finding the middle index "mid".
2. Compare the middle element of the search space with the key.
3. If the key is found at middle element, the process is terminated.
4. If the key is not found at middle element, choose which half will be used as the next search space.
    a. If the key is smaller than the middle element, then the left side is used for next search.
    b. If the key is larger than the middle element, then the right side is used for next search.
5. This process is continued until the key is found or the total search space is exhausted.

Input: arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
Output: target = 23
Element 23 is present at index 5

```
# Program for recursive binary search.

# Returns index of x in arr if present, else -1
def binarySearch(arr, l, r, x):
    # write code here
    …

# Driver Code
arr = [2, 3, 4, 10, 40]
x = 10
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print("Element is present at index", result)
else:
    print("Element is not present in array")
```

2.3 Uniform Binary Search

Uniform Binary Search is an optimization of Binary Search algorithm when many searches are made on same array or many arrays of same size. In normal binary search, we do arithmetic operations to find the mid points. Here we precompute mid points and fills them in lookup table. The array look-up generally works faster than arithmetic done (addition and shift) to find the mid-point.

Input: array = {1, 3, 5, 6, 7, 8, 9}, v=3

Output: Position of 3 in array = 2

Input: array = {1, 3, 5, 6, 7, 8, 9}, v=7

Output: Position of 7 in array = 5

The algorithm is very similar to Binary Search algorithm, the only difference is a lookup table is created for an array and the lookup table is used to modify the index of the pointer in the array which makes the search faster. Instead of maintaining lower and upper bound the algorithm maintains an index and the index is modified using the lookup table.

```
# Implementation of above approach

MAX_SIZE = 1000

# lookup table
```

```
lookup_table = [0] * MAX_SIZE

# create the lookup table for an array of length n
def create_table(n):
    # write code here
    …
# binary search

def binary(arr, v):
    # write code here
    …

# Driver code
arr = [1, 3, 5, 6, 7, 8, 9]
n = len(arr)

# create the lookup table
create_table(n)

# print the position of the array
print("Position of 3 in array = ", binary(arr, 3))
```

2.4 Interpolation Search

Interpolation search works better than Binary Search for a Sorted and Uniformly Distributed array. Binary search goes to the middle element to check irrespective of search-key. On the other hand, Interpolation search may go to different locations according to search-key. If the value of the search-key is close to the last element, Interpolation Search is likely to start search toward the end side. Interpolation search is more efficient than binary search when the elements in the list are uniformly distributed, while binary search is more efficient when the elements in the list are not uniformly distributed.

Interpolation search can take longer to implement than binary search, as it requires the use of additional calculations to estimate the position of the target element.

Input: arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Output: target = 5

```
# Interpolation search
def interpolation_search(arr, target):
    # write code here
    …


# Driver code
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target = 5

index = interpolation_search(arr, target)
if index == -1:
    print(f"{target} not found in the list")
else:
    print(f"{target} found at index {index}")
```

2.5 Fibonacci Search

Given a sorted array arr[] of size n and an element x to be searched in it. Return index of x if it is present in array else return -1.

Input: arr[] = {2, 3, 4, 10, 40}, x = 10
Output: 3
Element x is present at index 3.

Input: arr[] = {2, 3, 4, 10, 40}, x = 11
Output: -1
Element x is not present.

Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1. First few Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Fibonacci Search Procedure:

Let the searched element be x. The idea is to first find the smallest Fibonacci number that is greater than or equal to the length of the given array. Let the found Fibonacci number be fib (m'th Fibonacci number). We use (m-2)'th Fibonacci number as the index (If it is a valid index). Let (m-2)'th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for subarray after i, else we recur for subarray before i.

Let arr[0..n-1] be the input array and the element to be searched be x.

1. Find the smallest Fibonacci number greater than or equal to n. Let this number be fibM [m'th Fibonacci number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)'th Fibonacci Number] and fibMm2 [(m-2)'th Fibonacci Number].

2. While the array has elements to be inspected:

   i. Compare x with the last element of the range covered by fibMm2

   ii. If x matches, return index

   iii. Else If x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.

   iv. Else x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate the elimination of approximately front one-third of the remaining array.

3. Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index.

```
# Fibonacci search
from bisect import bisect_left

# Returns index of x if present, else returns -1

def fibMonaccianSearch(arr, x, n):
    # write code here
```

```
    …

# Driver Code
arr = [10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100,235]
n = len(arr)
x = 235
ind = fibMonaccianSearch(arr, x, n)
if ind>=0:
  print("Found at index:",ind)
else:
  print(x,"isn't present in the array");
```

3. Sorting

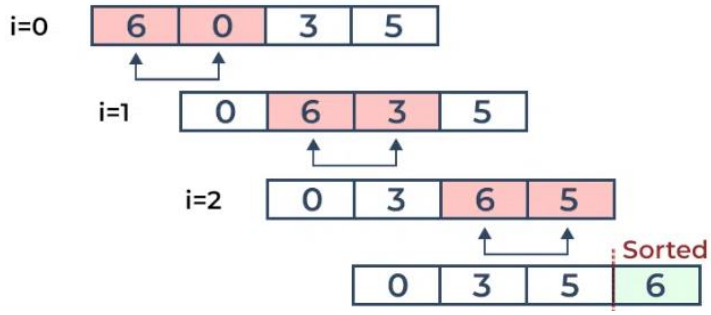
3.1 Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

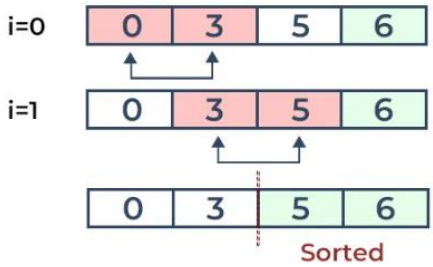
Bubble Sort Procedure:

1. Traverse from left and compare adjacent elements and the higher one is placed at right side.
2. In this way, the largest element is moved to the rightmost end at first.
3. This process is then continued to find the second largest and place it and so on until the data is sorted.

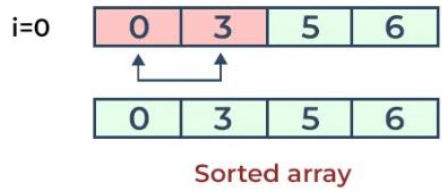
Input: arr = [6, 3, 0, 5]
Output:
First Pass:



Second Pass:



Third Pass:




```
# Implementation of Bubble Sort

def bubbleSort(arr):
      # write code here
      …

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print("Sorted array:")
for i in range(len(arr)):
    print("%d" % arr[i], end=" ")
```

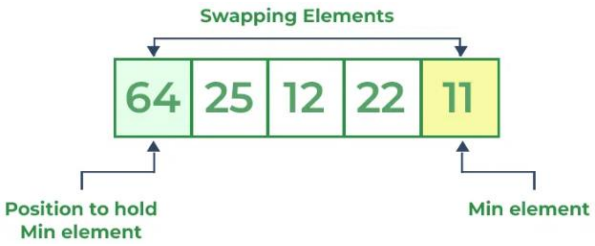
3.2 Selection Sort

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted.

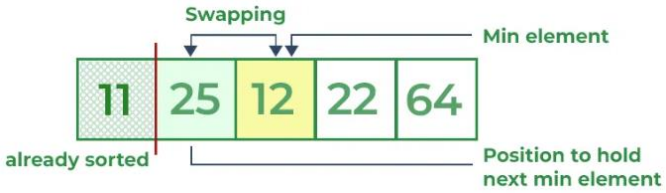
Input: arr = [64, 25, 12, 22, 11]

Output: arr = [11, 12, 22, 25, 64]

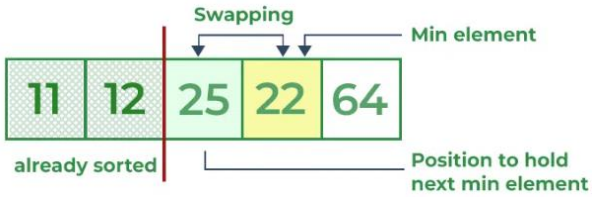
First Pass: For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value. Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.



Second Pass: For the second position, where 25 is present, again traverse the rest of the array in a sequential manner. After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.



Third Pass: Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array. While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.



Fourth Pass: Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array. As 25 is the 4th lowest value hence, it will place at the fourth position.

Fifth Pass: At last the largest value present in the array automatically get placed at the last position in the array. The resulted array is the sorted array.

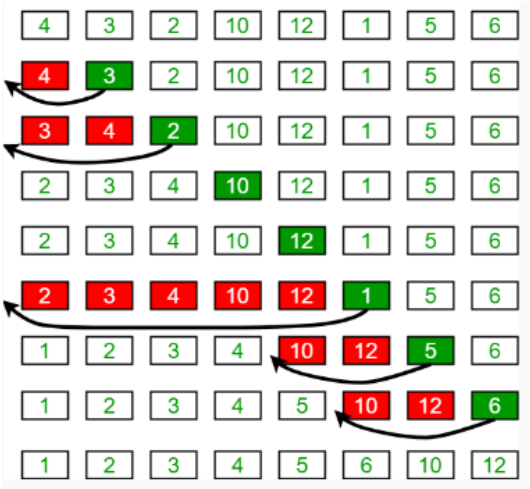

```
# Implementation of selection sort
import sys
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):
      # write code here
       …
# Driver code
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i],end=" , ")
```

3.3 Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Insertion Sort Procedure:

1. To sort an array of size N in ascending order iterate over the array and compare the current element (key) to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before.
2. Move the greater elements one position up to make space for the swapped element.



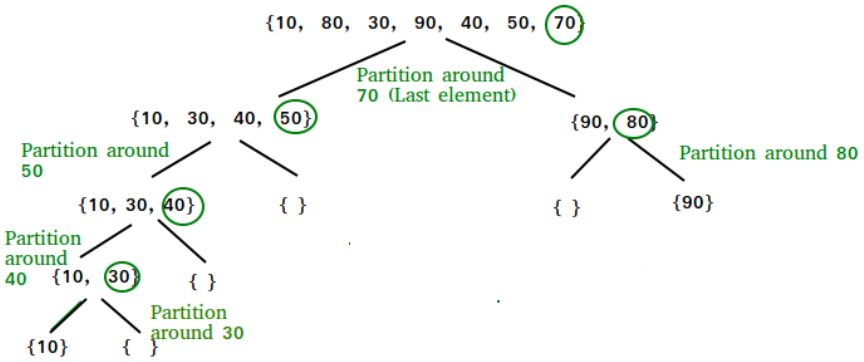
Input: arr = [4, 3, 2, 10, 12, 1, 5, 6]
Output: arr = [1, 2, 3, 4, 5, 6, 10, 12]

```
# Implementation of Insertion Sort

# Function to do insertion sort
def insertionSort(arr):
    # write code here
    …

# Driver code
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
    print ("% d" % arr[i])
```

4. Divide and Conquer

4.1 Quick Sort

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array. The key process in quickSort is a partition(). The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot. Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.



The quick sort method can be summarized in three steps:
1. **Pick:** Select a pivot element.
2. **Divide:** Split the problem set, move smaller parts to the left of the pivot and larger items to the right.
3. **Repeat and combine:** Repeat the steps and combine the arrays that have previously been sorted.

Algorithm for Quick Sort Function:

```
//start –> Starting index,  end --> Ending index
Quicksort(array, start, end)
{
        if (start < end)
        {
                pIndex = Partition(A, start, end)
                Quicksort(A,start,pIndex-1)
                Quicksort(A,pIndex+1, end)
        }
}
```

Algorithm for Partition Function:

```
partition (array, start, end)
{
        // Setting rightmost Index as pivot
        pivot = arr[end];

        i = (start - 1)  // Index of smaller element and indicates the
            // right position of pivot found so far
        for (j = start; j <= end- 1; j++)
        {
                // If current element is smaller than the pivot
                if (arr[j] < pivot)
                {
                        i++;    // increment index of smaller element
                        swap arr[i] and arr[j]
                }
        }
        swap arr[i + 1] and arr[end])
        return (i + 1)
}
```

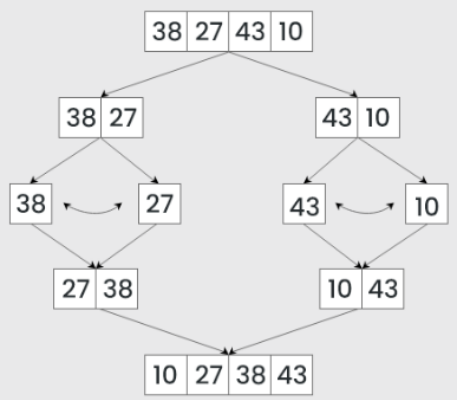
Input: arr = [10, 80, 30, 90, 40, 50, 70]

Output: arr = [10, 30, 40, 50, 70, 80, 90]

```
# Implementation of QuickSort

# Function to find the partition position
def partition(array, low, high):
    # write code here
    …

# Function to perform quicksort
def quicksort(array, low, high):
    # write code here
    …

# Driver code
array = [10, 7, 8, 9, 1, 5]
N = len(array)

# Function call
quicksort(array, 0, N - 1)
print('Sorted array:')
for x in array:
    print(x, end=" ")
```

4.2 Merge Sort

Merge sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array. In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.



Input: arr = [12, 11, 13, 5, 6, 7]
Output: arr = [5, 6, 7, 11, 12, 13]

```
# Implementation of MergeSort

def mergeSort(arr):
    # write code here
    …

# print the list
```

```
def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()


# Driver Code
arr = [12, 11, 13, 5, 6, 7]
print("Given array is")
printList(arr)
mergeSort(arr)
print("\nSorted array is ")
printList(arr)
```

4.3 Heap Sort

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

Heap Sort Procedure:

First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.

- Build a heap from the given input array.
- Repeat the following steps until the heap contains only one element:
    - Swap the root element of the heap (which is the largest element) with the last element of the heap.
    - Remove the last element of the heap (which is now in the correct position).
    - Heapify the remaining elements of the heap.
    - The sorted array is obtained by reversing the order of the elements in the input array.

Input: arr = [12, 11, 13, 5, 6, 7]
Output: Sorted array is 5 6 7 11 12 13

```
# Implementation of heap Sort
# To heapify subtree rooted at index i.
# n is size of heap

def heapify(arr, N, i):
    # write code here
    …

# The main function to sort an array of given size
def heapSort(arr):
    # write code here
    …

# Driver code
arr = [12, 11, 13, 5, 6, 7]
```

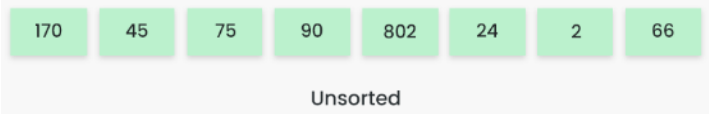
```
# Function call
heapSort(arr)
N = len(arr)
print("Sorted array is")
for i in range(N):
    print("%d" % arr[i], end=" ")
```

4.4 Radix Sort

Radix Sort is a linear sorting algorithm that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys. Rather than comparing elements directly, Radix Sort distributes the elements into buckets based on each digit's value. By repeatedly sorting the elements by their significant digits, from the least significant to the most significant, Radix Sort achieves the final sorted order.

Radix Sort Procedure:
The key idea behind Radix Sort is to exploit the concept of place value.
1. It assumes that sorting numbers digit by digit will eventually result in a fully sorted list.
2. Radix Sort can be performed using different variations, such as Least Significant Digit (LSD) Radix Sort or Most Significant Digit (MSD) Radix Sort.

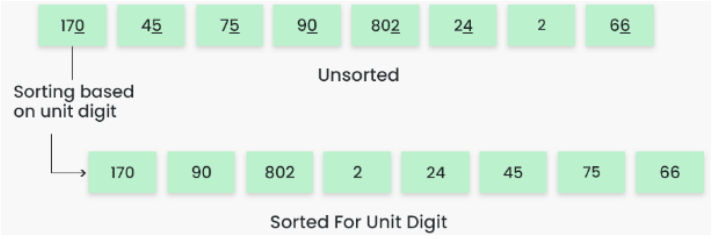
To perform radix sort on the array [170, 45, 75, 90, 802, 24, 2, 66], we follow these steps:



Step 1: Find the largest element in the array, which is 802. It has three digits, so we will iterate three times, once for each significant place.

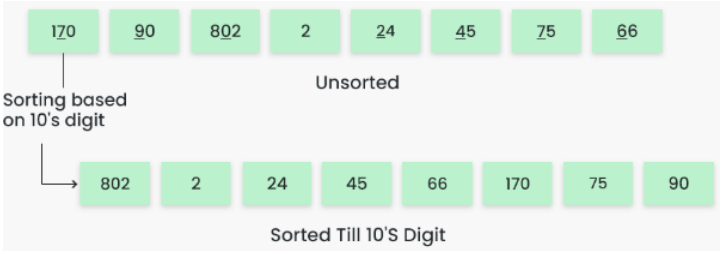
Step 2: Sort the elements based on the unit place digits (X=0). We use a stable sorting technique, such as counting sort, to sort the digits at each significant place.

Sorting based on the unit place:
Perform counting sort on the array based on the unit place digits.
The sorted array based on the unit place is [170, 90, 802, 2, 24, 45, 75, 66]



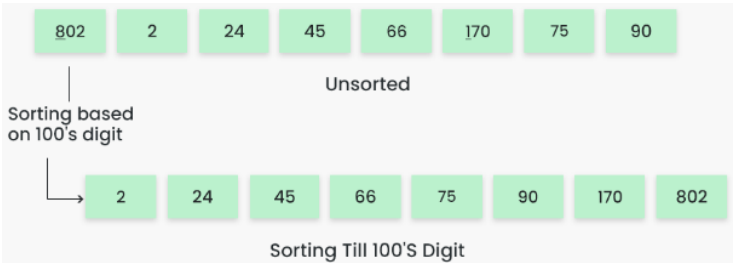
Step 3: Sort the elements based on the tens place digits.

Sorting based on the tens place:
Perform counting sort on the array based on the tens place digits.
The sorted array based on the tens place is [802, 2, 24, 45, 66, 170, 75, 90]



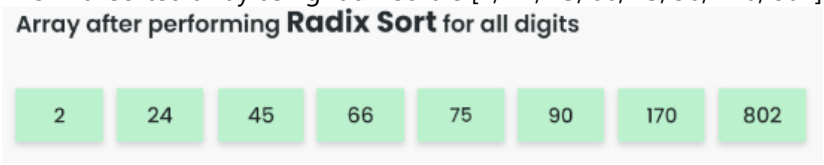
Step 4: Sort the elements based on the hundreds place digits.

Sorting based on the hundreds place:
Perform counting sort on the array based on the hundreds place digits.
The sorted array based on the hundreds place is [2, 24, 45, 66, 75, 90, 170, 802]



Step 5: The array is now sorted in ascending order.

The final sorted array using radix sort is [2, 24, 45, 66, 75, 90, 170, 802]




```
# Implementation of Radix Sort
# A function to do counting sort of arr[] according to the digit represented by exp.

def countingSort(arr, exp1):
    # write code here
    …


# Method to do Radix Sort
def radixSort(arr):
    # write code here
    …

# Driver code
arr = [170, 45, 75, 90, 802, 24, 2, 66]

# Function Call
radixSort(arr)

for i in range(len(arr)):
    print(arr[i],end=" ")
```

4.5 Shell Sort

Shell sort is mainly a variation of Insertion Sort. In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved. The idea of ShellSort is to allow the exchange of far items. In Shell sort, we make the array h-sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element are sorted.

Shell Sort Procedure:
1. Initialize the value of gap size h
2. Divide the list into smaller sub-part. Each must have equal intervals to h
3. Sort these sub-lists using insertion sort
4. Repeat this step 1 until the list is sorted.
5. Print a sorted list.

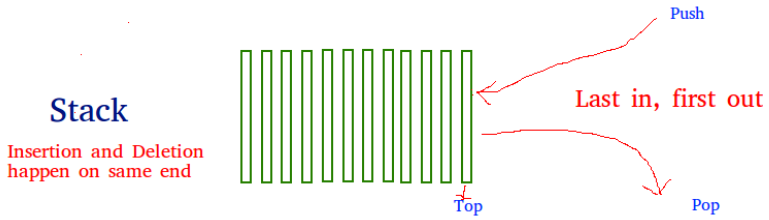
```
Procedure Shell_Sort(Array, N)
  While Gap < Length(Array) /3 :
            Gap = ( Interval * 3 ) + 1
  End While Loop
  While Gap > 0 :
    For (Outer = Gap; Outer < Length(Array); Outer++):
        Insertion_Value = Array[Outer]
            Inner = Outer;
        While Inner > Gap-1 And Array[Inner – Gap] >= Insertion_Value:
            Array[Inner] = Array[Inner – Gap]
            Inner = Inner – Gap
         End While Loop
           Array[Inner] = Insertion_Value
    End For Loop
    Gap = (Gap -1) /3;
  End While Loop
End Shell_Sort
```

```
# Implementation of Shell Sort

def shellSort(arr, n):
    # write code here
    …

# Driver code
arr = [12, 34, 54, 2, 3]
print("input array:",arr)

shellSort(arr,len(arr))
print("sorted array",arr)
```

5. Stack

5.1 Stack implementation using List

A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- **empty()** – Returns whether the stack is empty
- **size()** – Returns the size of the stack
- **top() / peek()** – Returns a reference to the topmost element of the stack
- **push(a)** – Inserts the element 'a' at the top of the stack
- **pop()** – Deletes the topmost element of the stack

```
# Stack implementation using list
top=0
mymax=5
def createStack():
    stack=[]
    return stack
def isEmpty(stack):
    # write code here
    …
def Push(stack,item):
    # write code here
    …
def Pop(stack):
    # write code here
    …
# create a stack object
stack = createStack()
while True:
    print("1.Push")
    print("2.Pop")
    print("3.Display")
    print("4.Quit")
    # write code here
    …
```

5.2 Balanced Parenthesis Checking

Given an expression string, write a python program to find whether a given string has balanced parentheses or not.

Input: {[]{()}}

Output: Balanced

Input: [{}{}(]

Output: Unbalanced

Using stack One approach to check balanced parentheses is to use stack. Each time, when an open parentheses is encountered push it in the stack, and when closed parenthesis is encountered, match it with the top of stack and pop it. If stack is empty at the end, return Balanced otherwise, Unbalanced.

```
# Check for balanced parentheses in an expression
open_list = ["[","{","("]
close_list = ["]","}",")"]

# Function to check parentheses
def check(myStr):
    # write code here
    …
```

5.3 Evaluation of Postfix Expression

Given a postfix expression, the task is to evaluate the postfix expression. Postfix expression: The expression of the form "a b operator" (ab+) i.e., when a pair of operands is followed by an operator.

Input: str = "2 3 1 * + 9 -"

Output: -4

Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.

Input: str = "100 200 + 2 / 5 * 7 +"

Output: 757

Procedure for evaluation postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

```
# Evaluate value of a postfix expression

# Class to convert the expression
class Evaluate:

    # Constructor to initialize the class variables
    def __init__(self, capacity):
        self.top = -1
        self.capacity = capacity

        # This array is used a stack
        self.array = []

    # Check if the stack is empty
    def isEmpty(self):
        # write code here
```

```
        …
    def peek(self):
        # write code here
        …
    def pop(self):
        # write code here
        …
    def push(self, op):
        # write code here
        …
    def evaluatePostfix(self, exp):
        # write code here
        …
# Driver code
exp = "231*+9-"
obj = Evaluate(len(exp))

# Function call
print("postfix evaluation: %d" % (obj.evaluatePostfix(exp)))
```

5.4 Infix to Postfix Expression Conversion

For a given Infix expression, convert it into Postfix form.
Infix expression: The expression of the form "a operator b" (a + b) i.e., when an operator is in-between every pair of operands.
Postfix expression: The expression of the form "a b operator" (ab+) i.e., When every pair of operands is followed by an operator.

Infix to postfix expression conversion procedure:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, put it in the postfix expression.
3. Otherwise, do the following
    - If the precedence and associativity of the scanned operator are greater than the precedence and associativity of the operator in the stack [or the stack is empty or the stack contains a '('], then push it in the stack. ['^' operator is right associative and other operators like '+','–','*' and '/' are left-associative].
    - Check especially for a condition when the operator at the top of the stack and the scanned operator both are '^'. In this condition, the precedence of the scanned operator is higher due to its right associativity. So it will be pushed into the operator stack.
    - In all the other cases when the top of the operator stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.
    - Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator.

- After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is a '(', push it to the stack.
5. If the scanned character is a ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-5 until the infix expression is scanned.
7. Once the scanning is over, Pop the stack and add the operators in the postfix expression until it is not empty.
8. Finally, print the postfix expression.

Input: A + B * C + D
Output: A B C * + D +

Input: ((A + B) – C * (D / E)) + F
Output: A B + C D E / * - F +

```
# Convert infix expression to postfix
# Class to convert the expression

class Conversion:

    # Constructor to initialize the class variables
    def __init__(self, capacity):
        self.top = -1
        self.capacity = capacity

        # This array is used a stack
        self.array = []


        # Precedence setting
        self.output = []
        self.precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}

    # Check if the stack is empty
    def isEmpty(self):
        # write code here
        …

    # Return the value of the top of the stack
    def peek(self):
        # write code here
        …

    # Pop the element from the stack
    def pop(self):
        # write code here
        …
```

```
    # Push the element to the stack
    def push(self, op):
        # write code here
        …

    # A utility function to check is the given character is operand
    def isOperand(self, ch):
        # write code here
        …

    # Check if the precedence of operator is strictly less than top of stack or not
    def notGreater(self, i):
        # write code here
        …

    # The main function that converts given infix expression
    # to postfix expression
    def infixToPostfix(self, exp):
       # write code here
        …

# Driver code
exp = "a+b*(c^d-e)^(f+g*h)-i"
obj = Conversion(len(exp))

# Function call
obj.infixToPostfix(exp)
```

5.5 Reverse a Stack

The stack is a linear data structure which works on the LIFO concept. LIFO stands for last in first out. In the stack, the insertion and deletion are possible at one end the end is called the top of the stack. Define two recursive functions BottomInsertion() and Reverse() to reverse a stack using Python. Define some basic function of the stack like push(), pop(), show(), empty(), for basic operation like respectively append an item in stack, remove an item in stack, display the stack, check the given stack is empty or not.

BottomInsertion(): this method append element at the bottom of the stack and BottomInsertion accept two values as an argument first is stack and the second is elements, this is a recursive method.

Reverse(): the method is reverse elements of the stack, this method accept stack as an argument Reverse() is also a Recursive() function. Reverse() is invoked BottomInsertion() method for completing the reverse operation on the stack.

Input: Elements = [1, 2, 3, 4, 5]
Output: Original Stack
5
4
3
2
1

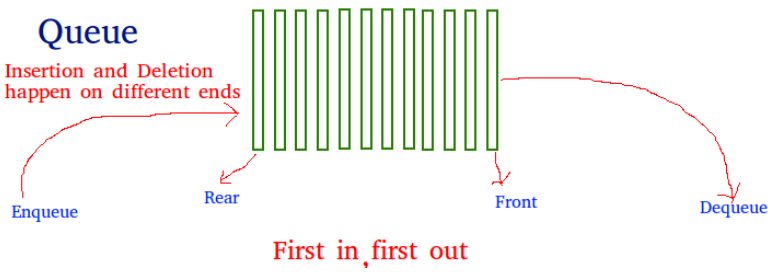
Stack after Reversing
1
2
3
4
5

```
# create class for stack
class Stack:

    # create empty list
    def __init__(self):
        self.Elements = []

    # push() for insert an element
    def push(self, value):
        self.Elements.append(value)

    # pop() for remove an element
    def pop(self):
        return self.Elements.pop()

    # empty() check the stack is empty of not
    def empty(self):
        return self.Elements == []

    # show() display stack
    def show(self):
        for value in reversed(self.Elements):
            print(value)
 # Insert_Bottom() insert value at bottom
def BottomInsert(s, value):
   # write code here
   …
# Reverse() reverse the stack
def Reverse(s):
   # write code here
   …
# create object of stack class
stk = Stack()

stk.push(1)
stk.push(2)
stk.push(3)
stk.push(4)
stk.push(5)

print("Original Stack")
stk.show()

print("\nStack after Reversing")
Reverse(stk)
stk.show()
```

6. Queue

6.1 Linear Queue

Linear queue is a linear data structure that stores items in First in First out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.




```
# Static implementation of linear queue
front=0
rear=0
mymax=5
def createQueue():
    queue=[]   #empty list
    return queue
def isEmpty(queue):
    # write code here
    …
def enqueue(queue,item):  # insert an element into the queue
    # write code here
    …
def dequeue(queue):  #remove an element from the queue
    # write code here
    …
# Driver code
queue = createQueue()
while True:
    print("1.Enqueue")
    print("2.Dequeue")
    print("3.Display")
    print("4.Quit")
    # write code here
    …
```

6.2 Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Input:

["MyStack", "push", "push", "top", "pop", "empty"]

[[], [1], [2], [], [], []]

Output:

[null, null, null, 2, 2, false]

```
class MyStack:

    def __init__(self):
       # write code here
       …
    def push(self, x: int) -> None:
       # write code here
       …
    def pop(self) -> int:
       # write code here
       …
    def top(self) -> int:
       # write code here
       …
    def empty(self) -> bool:
       # write code here
       …
# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

6.3 Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.

- boolean empty() Returns true if the queue is empty, false otherwise.

Input:

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output:

[null, null, null, 1, 1, false]

```
class MyQueue:

    def __init__(self):
        # write code here
        …
    def push(self, x: int) -> None:
        # write code here
        …
    def pop(self) -> int:
        # write code here
        …
    def peek(self) -> int:
        # write code here
        …
    def empty(self) -> bool:
        # write code here
        …
# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```

6.4 Circular Queue

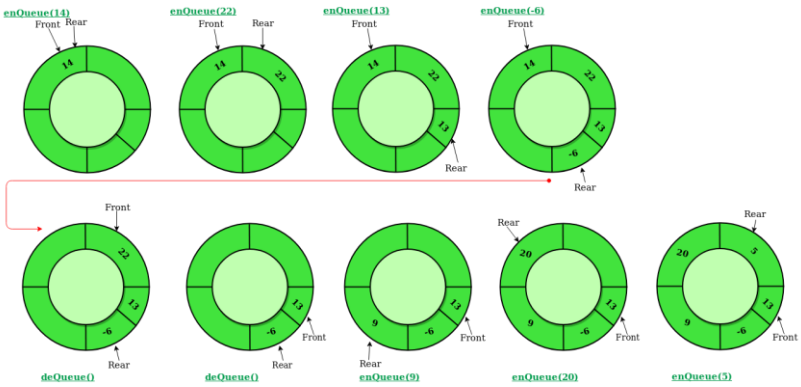
A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'.

Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.
    - Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].

- If it is full then display Queue is full.
    - If the queue is not full then, insert an element at the end of the queue.

- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
    - Check whether the queue is Empty.
    - If it is empty then display Queue is empty.
        - If the queue is not empty, then get the last element and remove it from the queue.



Implement Circular Queue using Array:

1. Initialize an array queue of size **n**, where n is the maximum number of elements that the queue can hold.
2. Initialize two variables front and rear to -1.
3. **Enqueue:** To enqueue an element **x** into the queue, do the following:
    - Increment rear by 1.
    - If **rear** is equal to n, set **rear** to 0.
    - If **front** is -1, set **front** to 0.
    - Set queue[rear] to x.
4. **Dequeue:** To dequeue an element from the queue, do the following:
    - Check if the queue is empty by checking if **front** is -1.
    - If it is, return an error message indicating that the queue is empty.
    - Set **x** to queue [front].
    - If **front** is equal to **rear**, set **front** and **rear** to -1.
    - Otherwise, increment **front** by 1 and if **front** is equal to n, set **front** to 0.
    - Return x.

```
class CircularQueue():

    # constructor
    def __init__(self, size): # initializing the class
        self.size = size

        # initializing queue with none
        self.queue = [None for i in range(size)]
        self.front = self.rear = -1

    def enqueue(self, data):
         # Write code here
         …

    def dequeue(self):
         # Write code here
         …

    def display(self):
        # Write code here
        …
# Driver Code
ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
```

6.5 Deque (Doubly Ended Queue)

In a Deque (Doubly Ended Queue), one can perform insert (append) and delete (pop) operations from both the ends of the container. There are two types of Deque:

1. **Input Restricted Deque:** Input is limited at one end while deletion is permitted at both ends.
2. **Output Restricted Deque:** Output is limited at one end but insertion is permitted at both ends.

Operations on Deque:

1. **append():** This function is used to insert the value in its argument to the right end of the deque.
2. **appendleft():** This function is used to insert the value in its argument to the left end of the deque.
3. **pop():** This function is used to delete an argument from the right end of the deque.
4. **popleft():** This function is used to delete an argument from the left end of the deque.
5. **index(ele, beg, end):** This function returns the first index of the value mentioned in arguments, starting searching from beg till end index.
6. **insert(i, a):** This function inserts the value mentioned in arguments(a) at index(i) specified in arguments.
7. **remove():** This function removes the first occurrence of the value mentioned in arguments.

8. **count():** This function counts the number of occurrences of value mentioned in arguments.

9. **len(dequeue):** Return the current size of the dequeue.
10. **Deque[0]:** We can access the front element of the deque using indexing with de[0].
11. **Deque[-1]:** We can access the back element of the deque using indexing with de[-1].
12. **extend(iterable):** This function is used to add multiple values at the right end of the deque. The argument passed is iterable.
13. **extendleft(iterable):** This function is used to add multiple values at the left end of the deque. The argument passed is iterable. Order is reversed as a result of left appends.
14. **reverse():** This function is used to reverse the order of deque elements.
15. **rotate():** This function rotates the deque by the number specified in arguments. If the number specified is negative, rotation occurs to the left. Else rotation is to right.

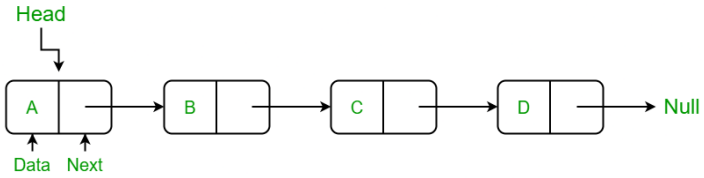
```
# importing "collections" for deque operations
import collections

# initializing deque
de = collections.deque([1, 2, 3])
print("deque: ", de)

# using append() to insert 4 at the end of deque
# Write code here

# Printing modified deque
# Write code here

# using appendleft() to insert 6 at the beginning of deque
# Write code here


# Printing modified deque
# Write code here

# using pop() to delete 4 from the right end of deque
# Write code here

# Printing modified deque
# Write code here

# using popleft() to delete 6 from the left end of deque
# Write code here

# Printing modified deque
# Write code here

# using insert() to insert the value 3 at 5th position
# Write code here

# printing modified deque
# Write code here

# using count() to count the occurrences of 3
```

```
# Write code here

# using remove() to remove the first occurrence of 3
# Write code here

# Printing modified deque
# Write code here

# Printing current size of deque
# Write code here

# using pop() to delete 6 from the right end of deque
# Write code here

# Printing modified deque
# Write code here

# Printing current size of deque
# Write code here

# Accessing the front element of the deque
# Write code here

# Accessing the back element of the deque
# Write code here

# using extend() to add 4,5,6 to right end
# Write code here

# Printing modified deque
# Write code here

# using extendleft() to add 7,8,9 to left end
# Write code here

# Printing modified deque
# Write code here

# using rotate() to rotate the deque rotates by 3 to left
# Write code here

# Printing modified deque
# Write code here

# using reverse() to reverse the deque
# Write code here

# Printing modified deque
# Write code here
```

7. Linked List

7.1 Singly Linked List

A singly linked list is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.



Creating a linked list involves the following operations:

1. Creating a Node class:
2. Insertion at beginning:
3. Insertion at end
4. Insertion at middle
5. Update the node
6. Deletion at beginning
7. Deletion at end
8. Deletion at middle
9. Remove last node
10. Linked list traversal
11. Get length

```
# Create a Node class to create a node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Create a LinkedList class
class LinkedList:
    def __init__(self):
        self.head = None

    # Method to add a node at begin of LL
    def insertAtBegin(self, data):
        # Write code here
        …
    # Method to add a node at any index, Indexing starts from 0.
    def insertAtIndex(self, data, index):
        # Write code here
        …
```

```
     # Method to add a node at the end of LL
    def insertAtEnd(self, data):
        # Write code here
        …
     # Update node of a linked list at given position
    def updateNode(self, val, index):
        # Write code here
        …
    # Method to remove first node of linked list
    def remove_first_node(self):
        # Write code here
        …
    # Method to remove last node of linked list
    def remove_last_node(self):
        # Write code here
        …
    # Method to remove at given index
    def remove_at_index(self, index):
        # Write code here
        …
     # Method to remove a node from linked list
    def remove_node(self, data):
        # Write code here
        …
     # Print the size of linked list
    def sizeOfLL(self):
        # Write code here
        …
    # print method for the linked list
    def printLL(self):
        # Write code here
        …
# create a new linked list
llist = LinkedList()

# add nodes to the linked list
llist.insertAtEnd('a')
llist.insertAtEnd('b')
llist.insertAtBegin('c')
llist.insertAtEnd('d')
llist.insertAtIndex('g', 2)
```

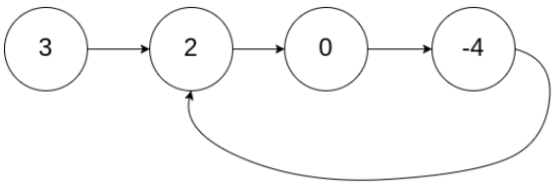
```
# print the linked list
print("Node Data")
llist.printLL()
# remove a nodes from the linked list
print("\nRemove First Node")
llist.remove_first_node()
print("Remove Last Node")
llist.remove_last_node()
print("Remove Node at Index 1")
llist.remove_at_index(1)
# print the linked list again
print("\nLinked list after removing a node:")
llist.printLL()
print("\nUpdate node Value")
llist.updateNode('z', 0)
llist.printLL()
print("\nSize of linked list :", end=" ")
print(llist.sizeOfLL())
```

7.2 Linked List Cycle

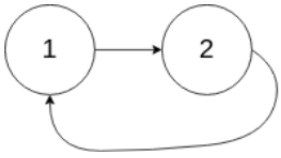
Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to.

Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.



Input: head = [3, 2, 0, -4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).



Input: head = [1, 2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Input: head = [1], pos = -1

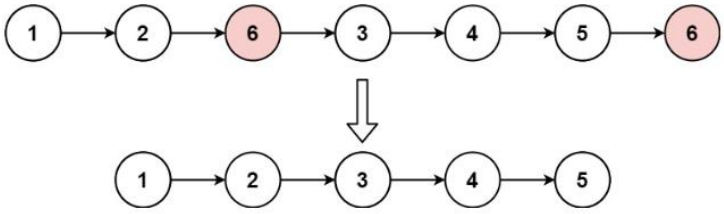
Output: false

Explanation: There is no cycle in the linked list.

```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class Solution:
    def hasCycle(self, head):
         # Write code here
         …
```

7.3 Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.



Input: head = [1, 2, 6, 3, 4, 5, 6], val = 6

Output: [1, 2, 3, 4, 5]

Input: head = [ ], val = 1

Output: [ ]

Input: head = [7, 7, 7, 7], val = 7

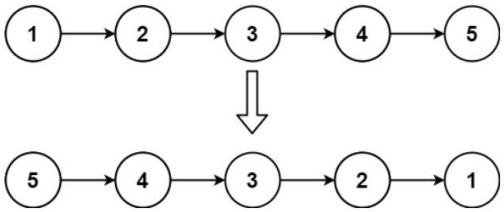
Output: [ ]

```
# Definition for singly-linked list.
class ListNode:
   def __init__(self, val=0, next=None):
       self.val = val
       self.next = next

class Solution:
    def removeElements(self, head, val):
         # Write code here
         …
```

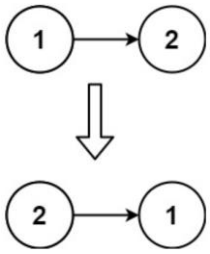
7.4 Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Input: head = [1, 2, 3, 4, 5]
Output: [5, 4, 3, 2, 1]



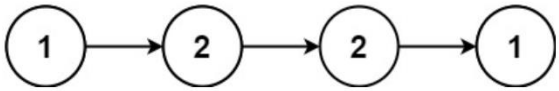
Input: head = [1, 2]
Output: [2, 1]



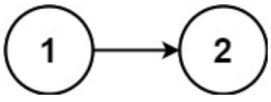

```
# Definition for singly-linked list.
class ListNode:
   def __init__(self, val=0, next=None):
       self.val = val
       self.next = next

class Solution:
    def reverseList(self, head):
        # Write code here
        …
```

7.5 Palindrome Linked List

Given the head of a singly linked list, return true if it is a palindrome or false otherwise.



Input: head = [1, 2, 2, 1]
Output: true



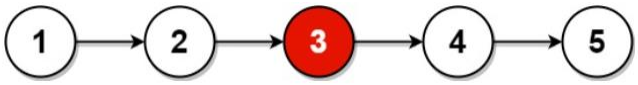
Input: head = [1, 2]
Output: false

```
# Definition for singly-linked list.
class ListNode:
   def __init__(self, val=0, next=None):
       self.val = val
```

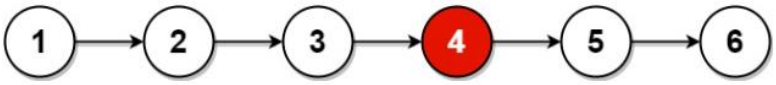
```
        self.next = next

class Solution:
    def isPalindrome(self, head):
        # Write code here
        …
```

7.6 Middle of the Linked List

Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node.



Input: head = [1, 2, 3, 4, 5]
Output: [3, 4, 5]
Explanation: The middle node of the list is node 3.



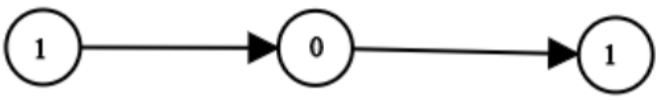
Input: head = [1, 2, 3, 4, 5, 6]
Output: [4, 5, 6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

```
# Definition for singly-linked list.
class ListNode:
   def __init__(self, val=0, next=None):
       self.val = val
       self.next = next

class Solution:
    def middleNode(self, head):
        # Write code here
        …
```

7.7 Convert Binary Number in a Linked List to Integer

Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the decimal value of the number in the linked list. The most significant bit is at the head of the linked list.



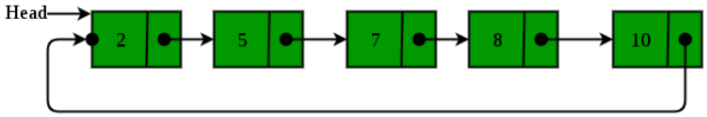
Input: head = [1, 0, 1]
Output: 5
Explanation: (101) in base 2 = (5) in base 10
Input: head = [0]
Output: 0

```
# Definition for singly-linked list.
class ListNode:
   def __init__(self, val=0, next=None):
       self.val = val
       self.next = next

class Solution:
    def getDecimalValue(self, head):
        # Write code here
        …
```

8. Circular Single Linked List and Doubly Linked List

8.1 Circular Linked List

The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.



Operations on the circular linked list:

1. Insertion at the beginning
2. Insertion at the end
3. Insertion in between the nodes
4. Deletion at the beginning
5. Deletion at the end
6. Deletion in between the nodes
7. Traversal

```
# Circular linked list operations

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class CircularLinkedList:
    def __init__(self):
        self.last = None
    def addToEmpty(self, data):
        # Write code here
        …

    # add node to the front
    def addFront(self, data):
        # Write code here
        …

    # add node to the end
    def addEnd(self, data):
        # Write code here
        …
```

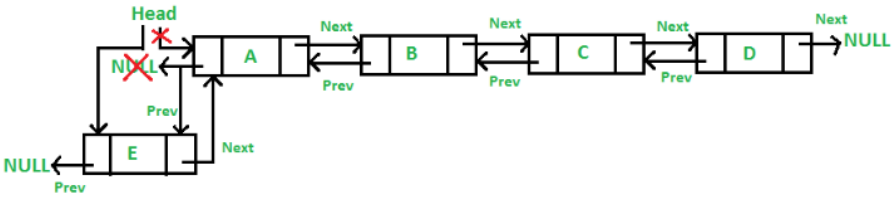
```
    # insert node after a specific node
    def addAfter(self, data, item):
        # Write code here
        …

    # delete a node
    def deleteNode(self, last, key):
        # Write code here
        …

    def traverse(self):
        # Write code here
        …
# Driver Code
cll = CircularLinkedList()

last = cll.addToEmpty(6)
last = cll.addEnd(8)
last = cll.addFront(2)
last = cll.addAfter(10, 2)

cll.traverse()
last = cll.deleteNode(last, 8)
print()
cll.traverse()
```

8.2 Doubly Linked List

The A doubly linked list is a type of linked list in which each node consists of 3 components:

1. *prev - address of the previous node
2. data - data item
3. *next - address of next node.

Double Linked List Node



Operations on the Double Linked List:

1. Insertion at the beginning
2. Insertion at the end
3. Insertion in between the nodes
4. Deletion at the beginning
5. Deletion at the end
6. Deletion in between the nodes
7. Traversal

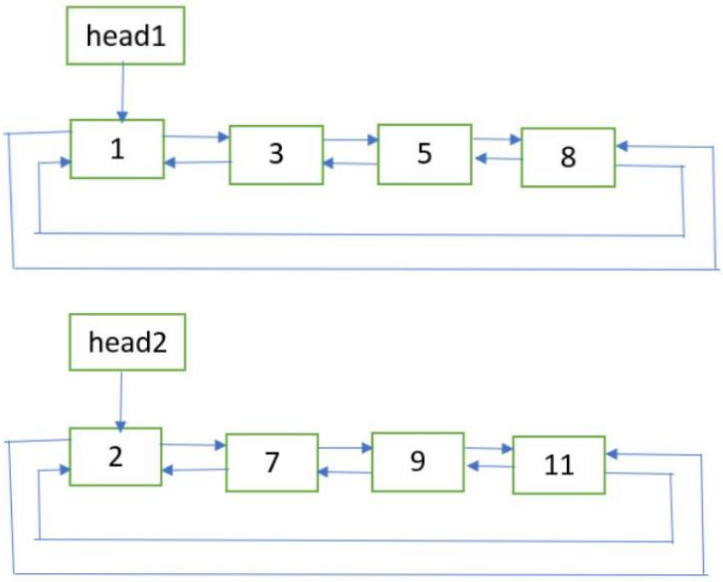
```
# Implementation of doubly linked list
class Node:
    def __init__(self,data):
        self.data=data
        self.next=self.prev=None

class DLinkedList:
    def __init__(self):
        self.head=None
        self.ctr=0
    def insert_beg(self,data):
        # Write code here
        …
    def insert_end(self,data):
        # Write code here
        …
    def delete_beg(self):
        # Write code here
        …
    def delete_end(self):
        # Write code here
        …
    def insert_pos(self,pos,data):
        # Write code here
        …
    def delete_pos(self,pos):
        # Write code here
        …
    def traverse_f(self):
        # Write code here
        …
    def traverse_r(self):
        # Write code here
        …

def menu():
    print("1.Insert at beginning")
    print("2.Insert at position")
    print("3.Insert at end")
    print("4.Delete at beginning")
    print("5.Delete at position")
    print("6.Delete at end")
    print("7.Count no of nodes")
    print("8.Traverse forward")
    print("9.Traverse reverse")
    print("10.Quit")
    ch=eval(input("Enter choice:"))
    return ch

print("******************Double linked list**************")
d=DLinkedList()
while True :
    ch=menu()
    if ch==1:
        data=eval(input("Enter data:"))
        d.insert_beg(data)

    elif ch==2:
```

```
        data=eval(input("Enter data:"))
        pos=int(input("Enter position:"))
        d.insert_pos(pos,data)

    elif ch==3:
        data=eval(input("Enter data:"))
        d.insert_end(data)

    elif ch==4:
        d.delete_beg()

    elif ch==5:
        pos=int(input("Enter position:"))
        d.delete_pos(pos)

    elif ch==6:
        d.delete_end()

    elif ch==7:
        print("Number of nodes",d.ctr)

    elif ch==8:
        d.traverse_f()
    elif ch==9:
        d.traverse_r()

    else:
        print("Exit")
        break
```

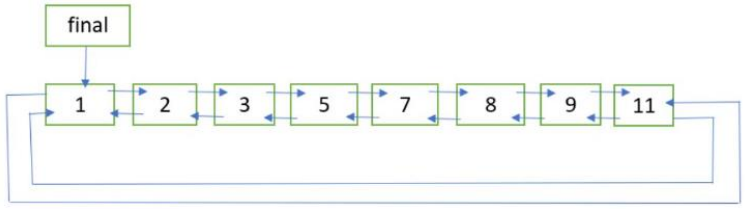
8.3 Sorted Merge of Two Sorted Doubly Circular Linked Lists

Given two sorted Doubly circular Linked List containing n1 and n2 nodes respectively. The problem is to merge the two lists such that resultant list is also in sorted order.

Input: List 1 and List 2



Output: Merged List



Procedure for Merging Doubly Linked List:

1. If head1 == NULL, return head2.
2. If head2 == NULL, return head1.
3. Let **last1** and **last2** be the last nodes of the two lists respectively. They can be obtained with the help of the previous links of the first nodes.
4. Get pointer to the node which will be the last node of the final list. If last1.data < last2.data, then **last_node** = last2, Else **last_node** = last1.
5. Update last1.next = last2.next = NULL.
6. Now merge the two lists as two sorted doubly linked list are being merged. Refer **merge** procedure of this post. Let the first node of the final list be **finalHead**.
7. Update finalHead.prev = last_node and last_node.next = finalHead.
8. Return **finalHead**.

```
# Implementation for Sorted merge of two sorted doubly circular linked list

import math

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

# A utility function to insert a new node at the beginning
# of doubly circular linked list
def insert(head_ref, data):
     # Write code here
     …
# function for Sorted merge of two sorted doubly linked list
def merge(first, second):
    # Write code here
    …
# function for Sorted merge of two sorted doubly circular linked list
def mergeUtil(head1, head2):
    # Write code here
    …
# function to print the list
def printList(head):
    # Write code here
```

```
    …
# Driver Code
head1 = None
head2 = None
# list 1:
head1 = insert(head1, 8)
head1 = insert(head1, 5)
head1 = insert(head1, 3)
head1 = insert(head1, 1)
# list 2:
head2 = insert(head2, 11)
head2 = insert(head2, 9)
head2 = insert(head2, 7)
head2 = insert(head2, 2)

newHead = mergeUtil(head1, head2)

print("Final Sorted List: ", end = "")
printList(newHead)
```

8.4 Delete all occurrences of a given key in a Doubly Linked List

Given a doubly linked list and a key x. The problem is to delete all occurrences of the given key x from the doubly linked list.

Input: 2 <-> 2 <-> 10 <-> 8 <-> 4 <-> 2 <-> 5 <-> 2
x = 2
Output: 10 <-> 8 <-> 4 <-> 5

Algorithm:
delAllOccurOfGivenKey (head_ref, x)
if head_ref == NULL
return
Initialize **current** = head_ref
Declare **next**
while current != NULL
if current->data == x
next = current->next
deleteNode(head_ref, current)
current = next
else
current = current->next

```
# Implementation to delete all occurrences of a given key in a doubly linked list
import math

# a node of the doubly linked list
class Node:
    def __init__(self,data):
        self.data = data
        self.next = None
        self.prev = None

# Function to delete a node in a Doubly Linked List.
# head_ref --> pointer to head node pointer.
# del --> pointer to node to be deleted.
def deleteNode(head, delete):
    # Write code here
```

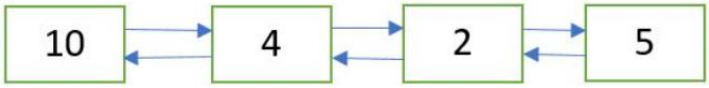
```
    …

# function to delete all occurrences of the given key 'x'
def deleteAllOccurOfX(head, x):
    # Write code here
    …

# Function to insert a node at the beginning of the Doubly Linked List
def push(head,new_data):
    # Write code here
    …

# Function to print nodes in a given doubly linked list
def printList(head):
    # Write code here
    …

# Driver Code
# Start with the empty list
head = None
# Create the doubly linked list:
head = push(head, 2)
head = push(head, 5)
head = push(head, 2)
head = push(head, 4)
head = push(head, 8)
head = push(head, 10)
head = push(head, 2)
head = push(head, 2)
print("Original Doubly linked list:")
printList(head)
x = 2
# delete all occurrences of 'x'
head = deleteAllOccurOfX(head, x)
print("\nDoubly linked list after deletion of ",x,":")
printList(head)
```

8.5 Delete a Doubly Linked List Node at a Given Position

Given a doubly linked list and a position n. The task is to delete the node at the given position n from the beginning.

Input: Initial doubly linked list

Output: Doubly Linked List after deletion of node at position n = 2



Procedure:

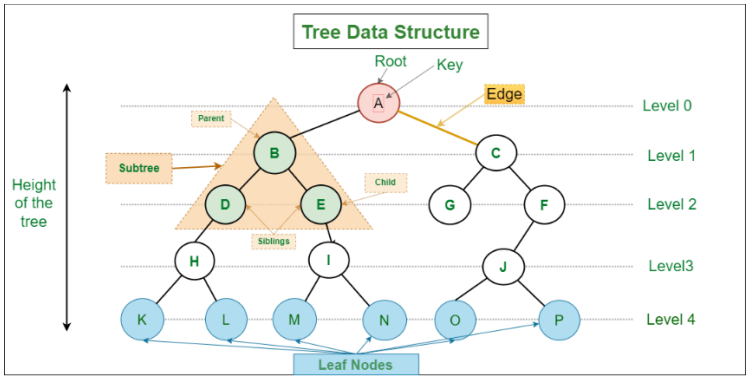
1. Get the pointer to the node at position n by traversing the doubly linked list up to the nth node from the beginning.
2. Delete the node using the pointer obtained in Step 1.

```
# Python implementation to delete a doubly Linked List node
# at the given position

# A node of the doubly linked list
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

# Function to delete a node in a Doubly Linked List.
# head_ref -. pointer to head node pointer.
# del -. pointer to node to be deleted.
def deleteNode(head_ref, del_):
    # Write code here
    …

# Function to delete the node at the given position
# in the doubly linked list
def deleteNodeAtGivenPos(head_ref, n):
    # Write code here
    …

# Function to insert a node at the beginning of the Doubly Linked List
def push(head_ref, new_data):
    # Write code here
    …

# Function to print nodes in a given doubly linked list
def printList(head):
    # Write code here
    …
# Driver Code
# Start with the empty list
head = None
head = push(head, 5)
head = push(head, 2)
head = push(head, 4)
head = push(head, 8)
head = push(head, 10)
print("Doubly linked list before deletion:")
printList(head)

n = 2

# delete node at the given position 'n'
head = deleteNodeAtGivenPos(head, n)
print("\nDoubly linked list after deletion:")
printList(head)
```

9. Trees

9.1 Tree Creation and Basic Tree Terminologies

A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.



Basic Terminologies in Tree:

1. **Parent Node:** The node which is a predecessor of a node is called the parent node of that node. {B} is the parent node of {D, E}.
2. **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: {D, E} are the child nodes of {B}.
3. **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {A} is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.
4. **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. {K, L, M, N, O, P} are the leaf nodes of the tree.
5. **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node. {A, B} are the ancestor nodes of the node {E}
6. **Descendant:** Any successor node on the path from the leaf node to that node. {E, I} are the descendants of the node {B}.
7. **Sibling:** Children of the same parent node are called siblings. {D, E} are called siblings.
8. **Level of a node:** The count of edges on the path from the root node to that node. The root node has level 0.
9. **Internal node:** A node with at least one child is called Internal Node.
10. **Neighbour of a Node:** Parent or child nodes of that node are called neighbors of that node.
11. **Subtree:** Any node of the tree along with its descendant.

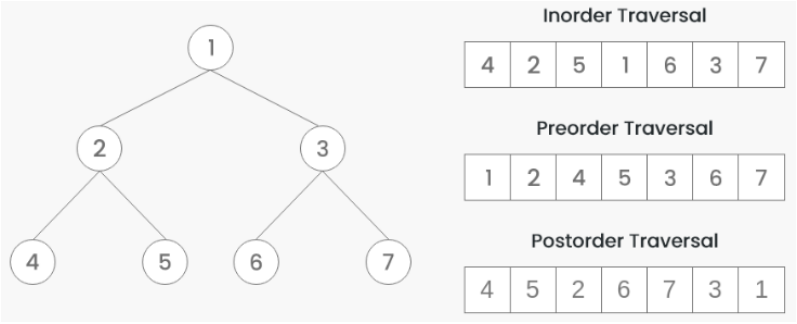
```
# Demonstration of Tree Basic Terminologies
# Function to add an edge between vertices x and y

# Function to print the parent of each node
def printParents(node, adj, parent):
    # Write code here
```

```
    …

# Function to print the children of each node
def printChildren(Root, adj):
    # Write code here
    …

# Function to print the leaf nodes
def printLeafNodes(Root, adj):
    # Write code here
    …

# Function to print the degrees of each node
def printDegrees(Root, adj):
    # Write code here
    …

# Driver code
# Number of nodes
N = 7
Root = 1

# Adjacency list to store the tree
adj = []
for i in range(0, N+1):
    adj.append([])

# Creating the tree
adj[1].append(2)
adj[2].append(1)

adj[1].append(3)
adj[3].append(1)

adj[1].append(4)
adj[4].append(1)

adj[2].append(5)
adj[5].append(2)

adj[2].append(6)
adj[6].append(2)

adj[4].append(7)
adj[7].append(4)

# Printing the parents of each node
print("The parents of each node are:")
printParents(Root, adj, 0)
# Printing the children of each node
print("The children of each node are:")
printChildren(Root, adj)

# Printing the leaf nodes in the tree
print("The leaf nodes of the tree are:")
printLeafNodes(Root, adj)

# Printing the degrees of each node
print("The degrees of each node are:")
printDegrees(Root, adj)
```

9.2 Binary Tree Traversal Techniques

A binary tree data structure can be traversed in following ways:

1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Level Order Traversal



Algorithm Inorder (tree)

1. Traverse the left subtree, i.e., call Inorder(left->subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right->subtree)

Algorithm Preorder (tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left->subtree)
3. Traverse the right subtree, i.e., call Preorder(right->subtree)

Algorithm Postorder (tree)

1. Traverse the left subtree, i.e., call Postorder(left->subtree)
2. Traverse the right subtree, i.e., call Postorder(right->subtree)
3. Visit the root.

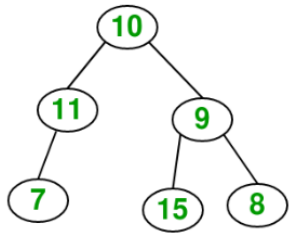
```
# Program to create a binary tree and print traversal orders
class Node:
    def __init__(self,data):
        self.data=data
        self.l=None
        self.r=None

class BT:
    def __init__(self):
        self.root=None

    def insert(self,n):
        # Write code here
        …


      def postorder(self,root):
        # Write code here
        …
```

```
    def preorder(self,root):
        # Write code here
        …

    def inorder(self,root):
        # Write code here
        …

# Driver code
b=BT()
print("******************TREE USING DOUBLE LINKED LIST************************")
while True:
    print("1.Insert data to tree")
    print("2.Post Order Traversal")
    print("3.Pre Order Traversal")
    print("4.In Order Traversal")
    print("5.Exit")
    ch=int(input("Enter choice:"))
    if ch==1:
        n=int(input("Enter number of nodes:"))
        b.insert(n)
    elif ch==2:
        b.postorder(b.root)
    elif ch==3:
        b.preorder(b.root)
    elif ch==4:
        b.inorder(b.root)
    else:
        print("Exit")
        break
```

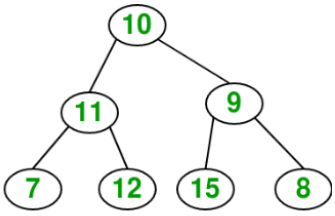
9.3 Insertion in a Binary Tree in Level Order

Given a binary tree and a key, insert the key into the binary tree at the first position available in level order.

Input: Consider the tree given below



Output:



After inserting 12

The idea is to do an iterative level order traversal of the given tree using queue. If we find a node whose left child is empty, we make a new key as the left child of the node. Else if we find a node whose right child is empty, we make the new key as the right child. We keep traversing the tree until we find a node whose either left or right child is empty.

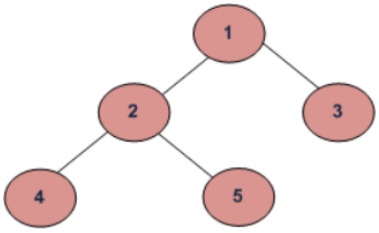
```
# Insert element in binary tree
class newNode():
    def __init__(self, data):
        self.key = data
        self.left = None
        self.right = None

# Inorder traversal of a binary tree
def inorder(temp):
    # Write code here
    …

# function to insert element in binary tree
def insert(temp,key):
   # Write code here
   …

# Driver code
root = newNode(10)
root.left = newNode(11)
root.left.left = newNode(7)
root.right = newNode(9)
root.right.left = newNode(15)
root.right.right = newNode(8)
print("Inorder traversal before insertion:", end = " ")
inorder(root)

key = 12
insert(root, key)

print()
print("Inorder traversal after insertion:", end = " ")
inorder(root)
```

9.4 Finding the Maximum Height or Depth of a Binary Tree

Given a binary tree, the task is to find the height of the tree. The height of the tree is the number of edges in the tree from the root to the deepest node.

Note: The height of an empty tree is 0.

Input: Consider the tree below



Recursively calculate the height of the left and the right subtrees of a node and assign height to the node as max of the heights of two children plus 1.

maxDepth('1') = max(maxDepth('2'), maxDepth('3')) + 1 = 2 + 1
because recursively
maxDepth('2') = max (maxDepth('4'), maxDepth('5')) + 1 = 1 + 1 and (as height of both '4' and '5' are 1)
maxDepth('3') = 1

Procedure:

- Recursively do a Depth-first search.
- If the tree is empty then return 0
- Otherwise, do the following
  - Get the max depth of the left subtree recursively i.e. call maxDepth(tree->left-subtree)
  - Get the max depth of the right subtree recursively i.e. call maxDepth(tree->right-subtree)
  - Get the max of max depths of left and right subtrees and add 1 to it for the current node.

$$max_depth = max(maxdepthofleftsubtree, maxdepthofrightsubtree) + 1$$

  - Return max_depth.

```
# Find the maximum depth of tree
# A binary tree node
class Node:
    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Compute the "maxDepth" of a tree -- the number of nodes
# along the longest path from the root node down to the farthest leaf node

def maxDepth(node):
    # Write code here
    …

# Driver program to test above function
root = Node(1)
root.left = Node(2)
```

```
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print("Height of tree is %d" % (maxDepth(root)))
```

9.5 Deletion in a Binary Tree

Given a binary tree, delete a node from it by making sure that the tree shrinks from the bottom (i.e. the deleted node is replaced by the bottom-most and rightmost node).

Input: Delete 10 in below tree

```
     10
   /   \
 20     30
```

Output:

```
     30
   /
 20
```

Input: Delete 20 in below tree

```
     10
   /   \
 20     30
          \
           40
```

Output:

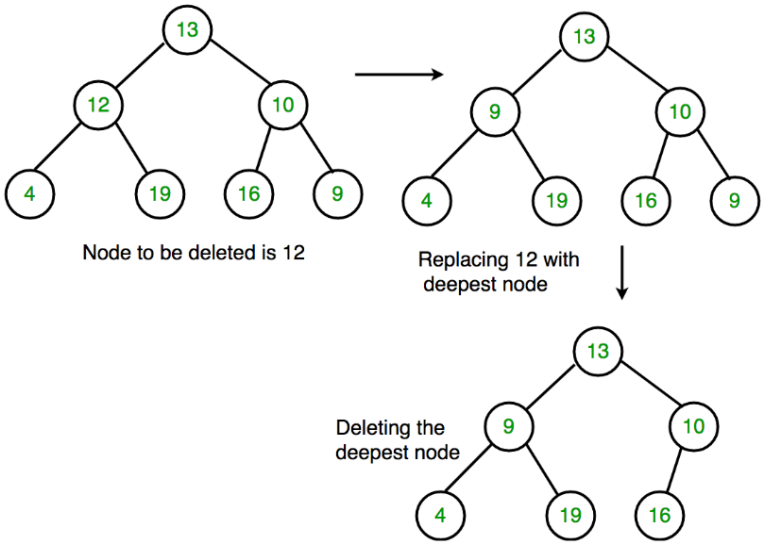
```
     10
  /     \
40       30
```

Algorithm:

1. Starting at the root, find the deepest and rightmost node in the binary tree and the node which we want to delete.

2. Replace the deepest rightmost node's data with the node to be deleted.

3. Then delete the deepest rightmost node.




```
# Deletion in a Binary Tree

# Create a node with data, left child and right child.

class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Inorder traversal of a binary tree
def inorder(temp):
    # Write code here
    …

# function to delete the given deepest node (d_node) in binary tree
def deleteDeepest(root, d_node):
    # Write code here
    …

# function to delete element in binary tree
def deletion(root, key):
    # Write code here
    …

# Driver code
root = Node(10)
root.left = Node(11)
root.left.left = Node(7)
root.left.right = Node(12)
root.right = Node(9)
root.right.left = Node(15)
root.right.right = Node(8)
print("The tree before the deletion: ", end = "")
inorder(root)
key = 11
root = deletion(root, key)
print();
print("The tree after the deletion: ", end = "")
inorder(root)
```

10. Binary Search Tree (BST)

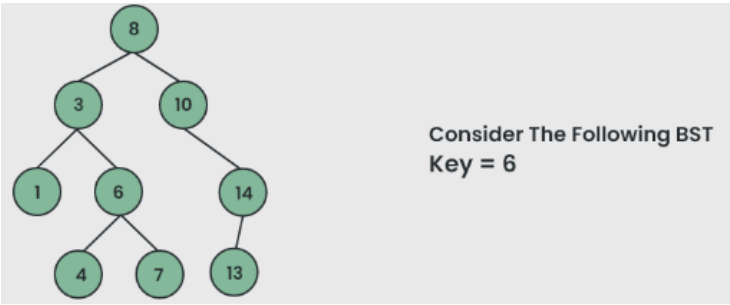
10.1 Searching in Binary Search Tree

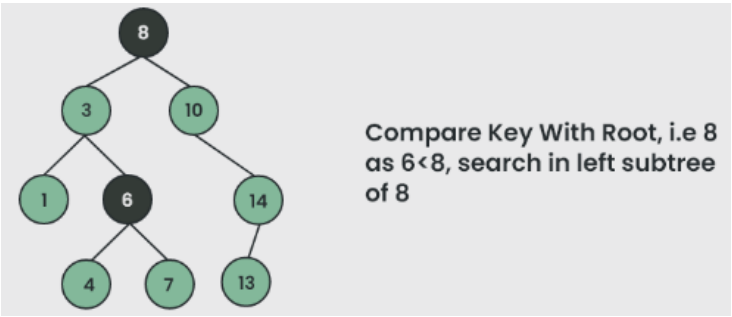
Given a BST, the task is to delete a node in this BST. For searching a value in BST, consider it as a sorted array. Perform search operation in BST using Binary Search Algorithm.

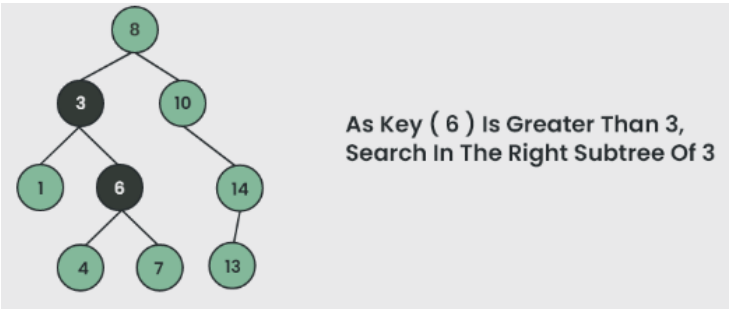
Algorithm to search for a key in a given Binary Search Tree:

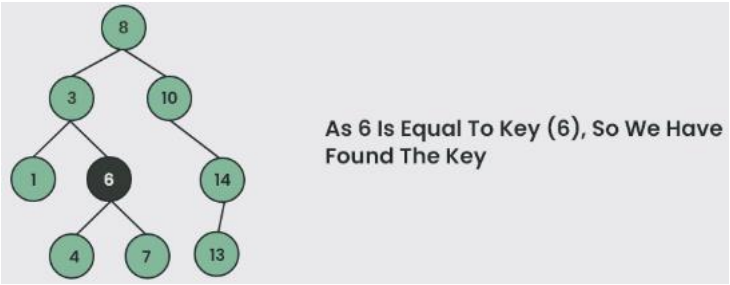
Let's say we want to search for the number **X,** We start at the root. Then:

- We compare the value to be searched with the value of the root.
- If it's equal we are done with the search if it's smaller we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger.
- Repeat the above step till no more traversal is possible
- If at any iteration, key is found, return True. Else False.







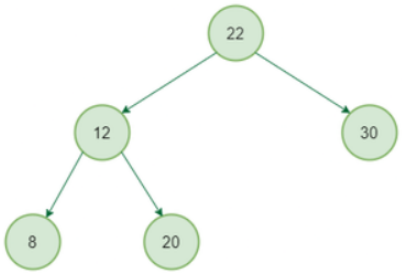



```
# Search a given key in a given BST

class Node:
    # Constructor to create a new node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

# A utility function to insert
# a new node with the given key in BST
def insert(node, key):
    # Write code here
    …

# Utility function to search a key in a BST
def search(root, key):
    # Write code here
    …

# Driver Code
root = None
root = insert(root, 50)
insert(root, 30)
insert(root, 20)
insert(root, 40)
insert(root, 70)
insert(root, 60)
insert(root, 80)
# Key to be found
key = 6

# Searching in a BST
if search(root, key) is None:
    print(key, "not found")

else:
    print(key, "found")

key = 60

# Searching in a BST
if search(root, key) is None:
    print(key, "not found")
else:
    print(key, "found")
```

10.2 Find the node with Minimum Value in a BST

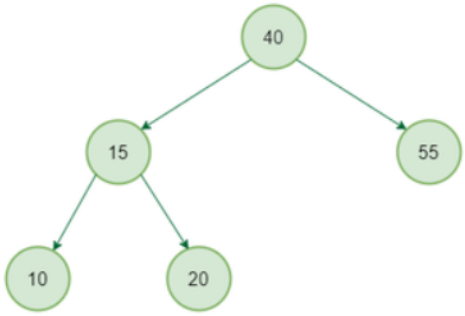
Write a function to find the node with minimum value in a Binary Search Tree.

Input: Consider the tree given below



Output: 8

Input: Consider the tree given below



Output: 10

```
from typing import List
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Give a binary search tree and a number, inserts a new node with the given number
# in the correct place in the tree. Returns the new root pointer
def insert(node: Node, data: int) -> Node:
    # Write code here
    …
# Given a non-empty binary search tree, inorder traversal for
# the tree is stored in the list sorted_inorder. Inorder is LEFT, ROOT, RIGHT.

def inorder(node: Node, sorted_inorder: List[int]) -> None:
    # Write code here
    …
# Driver Code
root = None
root = insert(root, 4)
insert(root, 2)
insert(root, 1)
insert(root, 3)
insert(root, 6)
insert(root, 4)
insert(root, 5)
sorted_inorder = []
```

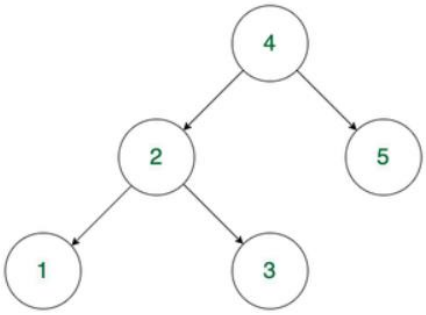
```
inorder(root, sorted_inorder)  # calling the recursive function

# Values of all nodes will appear in sorted order in the list sorted_inorder
print(f"Minimum value in BST is {sorted_inorder[0]}")
```

10.3 Check if a Binary Tree is BST or not

A binary search tree (BST) is a node-based binary tree data structure that has the following properties.

1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. Both the left and right subtrees must also be binary search trees.
4. Each node (item in the tree) has a distinct key.

Input: Consider the tree given below



Output: Check if max value in left subtree is smaller than the node and min value in right subtree greater than the node, then print it "Is BST" otherwise "Not a BST"

Procedure:

1. If the current node is null then return true
2. If the value of the left child of the node is greater than or equal to the current node then return false
3. If the value of the right child of the node is less than or equal to the current node then return false
4. If the left subtree or the right subtree is not a BST then return false
5. Else return true

```
# Program to check if a binary tree is BST or not
# A binary tree node has data, pointer to left child and a pointer to right child

class Node:
     def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def maxValue(node):
    # Write code here
    …

def minValue(node):
    # Write code here
    …

# Returns true if a binary tree is a binary search tree
def isBST(node):
    # Write code here
```

```
    …

# Driver code
root = Node(4)
root.left = Node(2)
root.right = Node(5)
# root.right.left = Node(7)
root.left.left = Node(1)
root.left.right = Node(3)

# Function call
if isBST(root) is True:
    print("Is BST")
else:
    print("Not a BST")
```

10.4 Second Largest Element in BST

Given a Binary search tree (BST), find the second largest element.

Input: Root of below BST

```
          10
         /
        5
```

Output: 5

Input: Root of below BST

```
          10

         /  \

       5    20

             \

              30
```

Output: 20

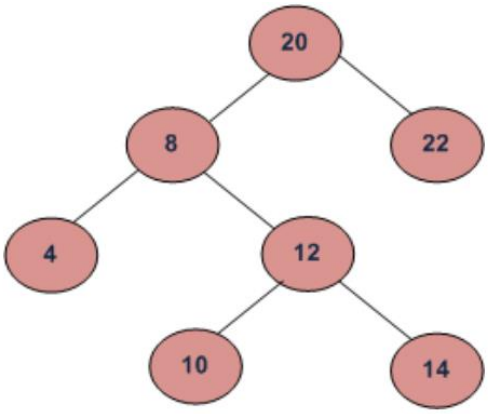
Procedure: The second largest element is second last element in inorder traversal and second element in reverse inorder traversal. We traverse given Binary Search Tree in reverse inorder and keep track of counts of nodes visited. Once the count becomes 2, we print the node.

```
# Find the second largest element in
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.key = data
        self.left = None
        self.right = None

# A function to find 2nd largest element in a given tree.
def secondLargestUtil(root, c):
```

```
    # Write code here
    …
# Function to find 2nd largest element
def secondLargest(root):
    # Write code here
    …

# A utility function to insert a new node with given key in BST
def insert(node, key):

# Driver Code
# Let us create following BST
#       50
#     /    \
#    30     70
#   / \     / \
#  20  40  60  80

root = None
root = insert(root, 50)
insert(root, 30)
insert(root, 20)
insert(root, 40)
insert(root, 70)
insert(root, 60)
insert(root, 80)
secondLargest(root)
```

Try:

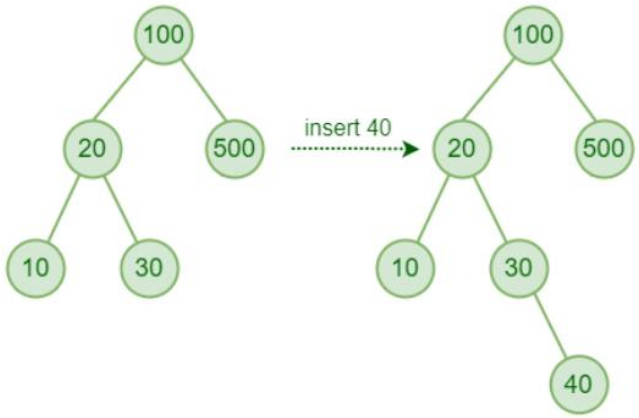
1. **Kth largest element in BST when modification to BST is not allowed:** Given a Binary Search Tree (BST) and a positive integer k, find the k'th largest element in the Binary Search Tree. For a given BST, if k = 3, then output should be 14, and if k = 5, then output should be 10.



10.5 Insertion in Binary Search Tree (BST)

Given a Binary search tree (BST), the task is to insert a new node in this BST.

Input: Consider a BST and insert the element 40 into it.



Procedure for inserting a value in a BST:

A new key is always inserted at the leaf by maintaining the property of the binary search tree. We start searching for a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node. The below steps are followed while we try to insert a node into a binary search tree:

- Check the value to be inserted (say X) with the value of the current node (say val) we are in:
  - If X is less than val move to the left subtree.
  - Otherwise, move to the right subtree.
  - Once the leaf node is reached, insert X to its right or left based on the relation between X and the leaf node's value.

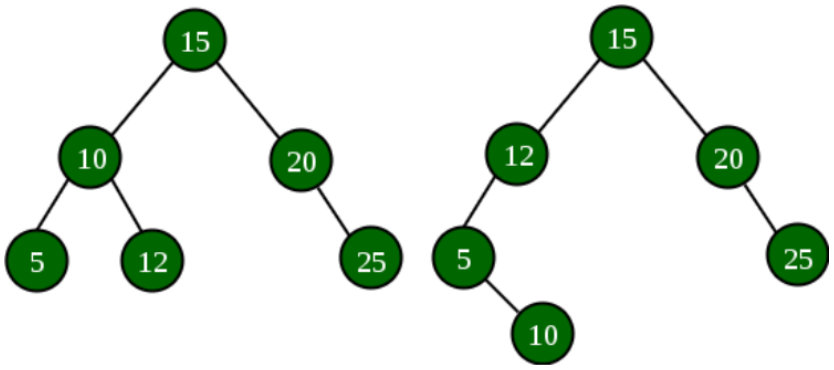
```
# insert operation in binary search tree
# A utility class that represents an individual node in a BST
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

 # A utility function to insert a new node with the given key
def insert(root, key):
    # Write code here
    …

 # A utility function to do inorder tree traversal
def inorder(root):
    # Write code here
    …

# Driver code
# Let us create the following BST
#       50
```

```
#     /     \
#    30     70
#   / \     / \
#  20 40   60 80
r = Node(50)
r = insert(r, 30)
r = insert(r, 20)
r = insert(r, 40)
r = insert(r, 70)
r = insert(r, 60)
r = insert(r, 80)

# Print inorder traversal of the BST
inorder(r)
```

Try:

1. **Check if two BSTs contain same set of elements:** Given two Binary Search Trees consisting of unique positive elements, we have to check whether the two BSTs contain the same set of elements or not.

Input: Consider two BSTs which contains same set of elements {5, 10, 12, 15, 20, 25}, but the structure of the two given BSTs can be different.



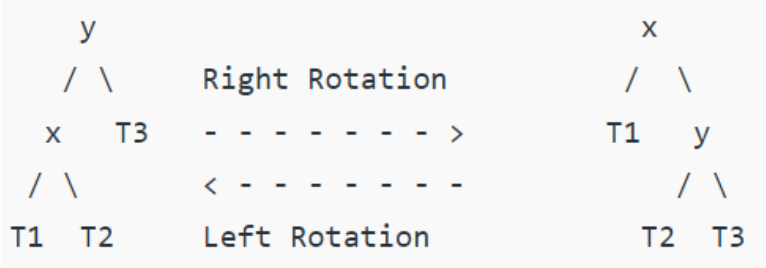
11. AVL Tree

11.1 Insertion in an AVL Tree

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes. To make sure that the given tree remains AVL after every insertion, we must augment the standard BST insert operation to perform some re-balancing. Following are two basic operations that can be performed to balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

- Left Rotation
- Right Rotation

T1, T2 and T3 are subtrees of the tree, rooted with y (on the left side) or x (on the right side)



Keys in both of the above trees follow the following order

keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)

So BST property is not violated anywhere.

Procedure for inserting a node into an AVL tree
Let the newly inserted node be w

- Perform standard BST insert for w.
- Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.
- Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that need to be handled as x, y and z can be arranged in 4 ways.
- Following are the possible 4 arrangements:
  - y is the left child of z and x is the left child of y (Left Left Case)
  - y is the left child of z and x is the right child of y (Left Right Case)
  - y is the right child of z and x is the right child of y (Right Right Case)
  - y is the right child of z and x is the left child of y (Right Left Case)

```
# Insert a node in AVL tree

# Generic tree node class
class TreeNode(object):
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
        self.height = 1

# AVL tree class which supports the insert operation
class AVL_Tree(object):

    # Recursive function to insert key in subtree rooted with node and returns
    # new root of subtree.
    def insert(self, root, key):
        # Write code here
        …

    def leftRotate(self, z):
        # Write code here
        …

    def rightRotate(self, z):
        # Write code here
        …

    def getHeight(self, root):
        # Write code here
        …

    def getBalance(self, root):
        # Write code here
        …

    def preOrder(self, root):
        # Write code here
        …

# Driver code
myTree = AVL_Tree()
root = None

root = myTree.insert(root, 10)
root = myTree.insert(root, 20)
root = myTree.insert(root, 30)
root = myTree.insert(root, 40)
root = myTree.insert(root, 50)
root = myTree.insert(root, 25)

"""The constructed AVL Tree would be
            30
           /  \
         20   40
        /  \     \
       10  25    50"""

# Preorder Traversal
```

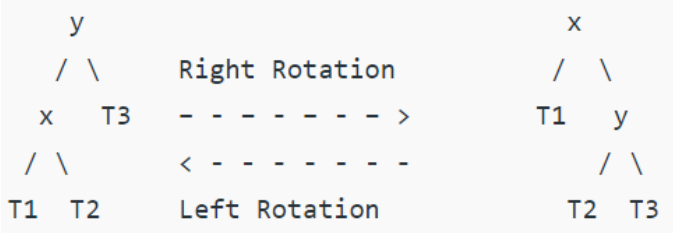
```
print("Preorder traversal of the",
      "constructed AVL tree is")
myTree.preOrder(root)
print()
```

11.2 Deletion in an AVL Tree

Given an AVL tree, make sure that the given tree remains AVL after every deletion, we must augment the standard BST delete operation to perform some re-balancing. Following are two basic operations that can be performed to re-balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

1. Left Rotation
2. Right Rotation

T1, T2 and T3 are subtrees of the tree rooted with y (on left side)
or x (on right side)



Keys in both of the above trees follow the following order
keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)
So BST property is not violated anywhere.

Procedure to delete a node from AVL tree:
Let w be the node to be deleted

1. Perform standard BST delete for w.

2. Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z, and x be the larger height child of y. Note that the definitions of x and y are different from insertion here.

3. Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:

   i. y is left child of z and x is left child of y (Left Left Case)

   ii. y is left child of z and x is right child of y (Left Right Case)

   iii. y is right child of z and x is right child of y (Right Right Case)

   iv. y is right child of z and x is left child of y (Right Left Case)

```
# delete a node in AVL tree

class TreeNode(object):
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
        self.height = 1

# AVL tree class which supports insertion, deletion operations
class AVL_Tree(object):

    def insert(self, root, key):
        # Write code here
        …

    # Recursive function to delete a node with given key from subtree
    # with given root. It returns root of the modified subtree.
    def delete(self, root, key):
        # Write code here
        …

    def leftRotate(self, z):
        # Write code here
        …

    def rightRotate(self, z):
        # Write code here
        …

    def getHeight(self, root):
        # Write code here
        …

    def getBalance(self, root):
        # Write code here
        …

    def getMinValueNode(self, root):
        # Write code here
        …

    def preOrder(self, root):
        # Write code here
        …

myTree = AVL_Tree()
root = None
nums = [9, 5, 10, 0, 6, 11, -1, 1, 2]

for num in nums:
    root = myTree.insert(root, num)


# Preorder Traversal
print("Preorder Traversal after insertion -")
myTree.preOrder(root)
print()
```

```
# Delete
key = 10
root = myTree.delete(root, key)

# Preorder Traversal
print("Preorder Traversal after deletion -")
myTree.preOrder(root)
print()
```

11.3 Count Greater Nodes in AVL Tree

Given an AVL tree, calculate number of elements which are greater than given value in AVL tree.

Input: x = 5

Root of below AVL tree

```
      9
     / \
    1   10
   / \   \
  0   5   11
 /   / \
-1   2   6
```

Output: 4

Explanation: There are 4 values which are greater than 5 in AVL tree which are 6, 9, 10 and 11.

```
# Count greater nodes in an AVL tree

class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1
        self.desc = 0

 def height(N):
    if N is None:
        return 0
    return N.height

# A utility function to get maximum of two integers

def max(a, b):
    if a > b:
        return a
    return b

def newNode(key):
    # Write code here
    …
```

```
# A utility function to right rotate subtree rooted with y

def rightRotate(y):
    # Write code here
    …

def leftRotate(x):
    # Write code here
    …
def getBalance(N):
    # Write code here
    …

def insert(root, key):
    # Write code here
    …

def minValueNode(node):
    # Write code here
    …

# Recursive function to delete a node with given key # from subtree with given root. It
returns root of the modified subtree.

def deleteNode(root, key):
    # Write code here
    …

def preOrder(root):
    # Write code here
    …

def CountGreater(root, x):
    # Write code here
    …

# Driver program to test above function
root = None
root = insert(root, 9)
root = insert(root, 5)
root = insert(root, 10)
root = insert(root, 0)
root = insert(root, 6)
root = insert(root, 11)
root = insert(root, -1)
root = insert(root, 1)
root = insert(root, 2)

print("Preorder traversal of the constructed AVL tree is")
preOrder(root)

print("Number of elements greater than 9 are")
print(CountGreater(root, 9))

root = deleteNode(root, 10)

print("Preorder traversal after deletion of 10")
preOrder(root)
print('Number of elements greater than 9 are')
print(CountGreater(root, 9))
```

11.4 Minimum Number of Nodes in an AVL Tree with given Height

Given the height of an AVL tree 'h', the task is to find the minimum number of nodes the tree can have.

Input: H = 0

Output: N = 1

Only '1' node is possible if the height
of the tree is '0' which is the root node.

Input: H = 3

Output: N = 7

Recursive approach:
In an AVL tree, we have to maintain the height balance property, i.e. difference in the height of the left and the right subtrees cannot be other than -1, 0 or 1 for each node.

We will try to create a recurrence relation to find minimum number of nodes for a given height, n(h).

- For height = 0, we can only have a single node in an AVL tree, i.e. n(0) = 1
- For height = 1, we can have a minimum of two nodes in an AVL tree, i.e. n(1) = 2
- Now for any height 'h', root will have two subtrees (left and right). Out of which one has to be of height h-1 and other of h-2. [root node excluded]
- So, n(h) = 1 + n(h-1) + n(h-2) is the required recurrence relation for h>=2 [1 is added for the root node]

```
# Function to find minimum number of nodes

def AVLnodes(height):
    # Write code here
    …
# Driver Code
H = 3
print(AVLnodes(H))
```

12. Graph Traversal

12.1 Breadth First Search

The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
For a given graph G, print BFS traversal from a given source vertex.

```
# BFS traversal from a given source vertex.

from collections import defaultdict
 # This class represents a directed graph using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):
```

```
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):
      # Write code here
      …

# Create a graph given in the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is Breadth First Traversal" " (starting from vertex 2)")
g.BFS(2)
```

Output: Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

12.2 Depth First Search

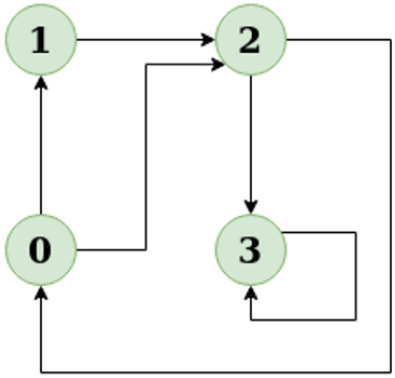
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

For a given graph G, print DFS traversal from a given source vertex.

Input: n = 4, e = 6
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

Output: DFS from vertex 1: 1 2 0 3

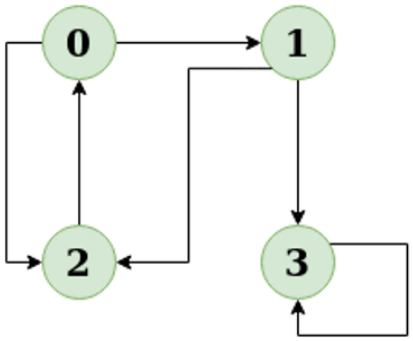
Explanation:
DFS Diagram:



Input: n = 4, e = 6
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: DFS from vertex 2: 2 0 1 3

Explanation:
DFS Diagram:




```
# DFS traversal from a given graph
from collections import defaultdict

# This class represents a directed graph using adjacency list representation
class Graph:
    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)
    # A function used by DFS
    def DFSUtil(self, v, visited):
        # Write code here
        …

    # The function to do DFS traversal. It uses recursive DFSUtil()

    def DFS(self, v):
      # Write code here
       …
```

```
# Driver's code
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
print("Following is Depth First Traversal (starting from vertex 2)")
# Function call
g.DFS(2)
```

12.3 Best First Search (Informed Search)

The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

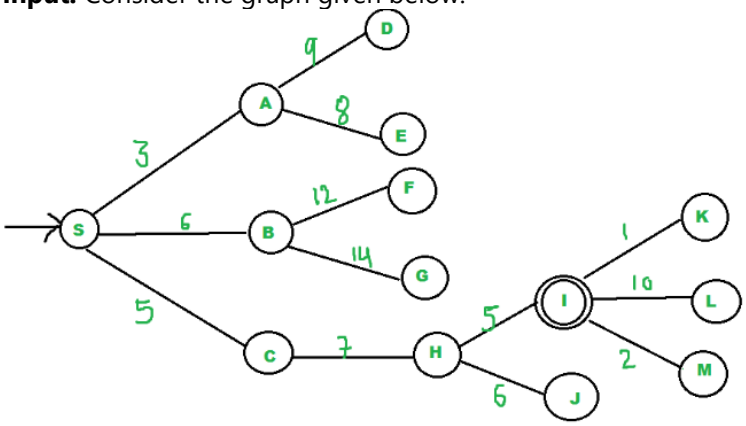
Implementation of Best First Search:
We use a priority queue or heap to store the costs of nodes that have the lowest evaluation function value. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

Algorithm:
Best-First-Search(Graph g, Node start)

1) Create an empty PriorityQueue
   PriorityQueue pq;
2) Insert "start" in pq.
   pq.insert(start)
3) Until PriorityQueue is empty
   u = PriorityQueue.DeleteMin
   If u is the goal
   Exit
   Else
   Foreach neighbor v of u
   If v "Unvisited"
   Mark v "Visited"
   pq.insert(v)
   Mark u "Examined"

End procedure

Input: Consider the graph given below.



- We start from source “S” and search for goal “I” using given costs and Best First search.
- pq initially contains S
    - We remove S from pq and process unvisited neighbors of S to pq.
    - pq now contains {A, C, B} (C is put before B because C has lesser cost)
- We remove A from pq and process unvisited neighbors of A to pq.
    - pq now contains {C, B, E, D}
- We remove C from pq and process unvisited neighbors of C to pq.
    - pq now contains {B, H, E, D}
- We remove B from pq and process unvisited neighbors of B to pq.
    - pq now contains {H, E, D, F, G}
- We remove H from pq.
- Since our goal “I” is a neighbor of H, we return.

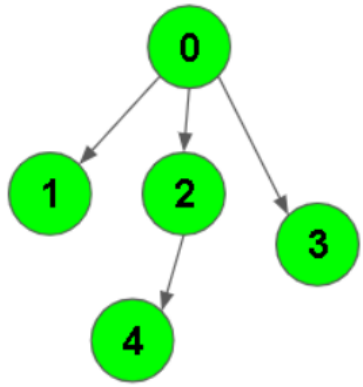
```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

# Function For Implementing Best First Search
# Gives output path having lowest cost

def best_first_search(actual_Src, target, n):
    # Write code here
    …
# Function for adding edges to graph
def addedge(x, y, cost):
    # Write code here
    …
# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
```

```
best_first_search(source, target, v)
```

12.4 Breadth First Traversal of a Graph

Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0.
One can move from node u to node v only if there's an edge from u to v. Find the BFS traversal of the graph starting from the 0th vertex, from left to right according to the input graph. Also, you should only take nodes directly or indirectly connected from Node 0 in consideration.

Input: Consider the graph given below where V = 5, E = 4, edges = {(0,1), (0,2), (0,3), (2,4)}



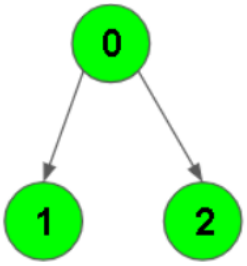
Output: 0 1 2 3 4

Explanation:

0 is connected to 1, 2, and 3.

2 is connected to 4.
So starting from 0, it will go to 1 then 2 then 3. After this 2 to 4, thus BFS will be 0 1 2 3 4.

Input: Consider the graph given below where V = 3, E = 2, edges = {(0, 1), (0, 2)}



Output: 0 1 2
Explanation:
0 is connected to 1, 2. So starting from 0, it will go to 1 then 2, thus BFS will be 0 1 2.
Your task is to complete the function **bfsOfGraph()** which takes the integer V denoting the number of vertices and adjacency list as input parameters and returns a list containing the BFS traversal of the graph starting from the 0th vertex from left to right.

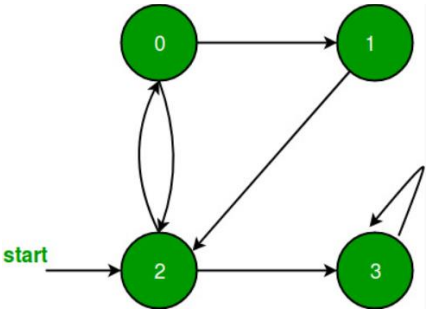
```
from typing import List
from queue import Queue
class Solution:

    # Function to return Breadth First Traversal of given graph.
    def bfsOfGraph(self, V: int, adj: List[List[int]]) -> List[int]:
        # Write code here
        …
```

```
# Driver Code
T=int(input())
for i in range(T):
	V, E = map(int, input().split())
	adj = [[] for i in range(V)]
	for _ in range(E):
		u, v = map(int, input().split())
		adj[u].append(v)
	ob = Solution()
	ans = ob.bfsOfGraph(V, adj)
	for i in range(len(ans)):
			print(ans[i], end = " ")
	print()
```

12.5 Depth First Search (DFS) for Disconnected Graph

Given a Disconnected Graph, the task is to implement DFS or Depth First Search Algorithm for this Disconnected Graph.

Input: Consider the graph given below.



Output: 0 1 2 3

Procedure for DFS on Disconnected Graph:
Iterate over all the vertices of the graph and for any unvisited vertex, run a DFS from that vertex.

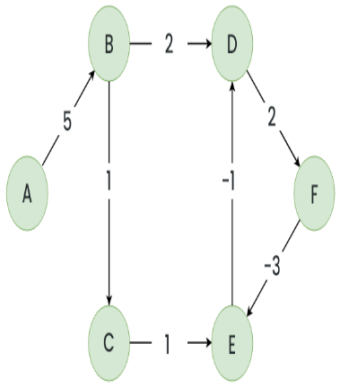
```
# DFS traversal for complete graph
from collections import defaultdict

# This class represents a directed graph using adjacency list representation
class Graph:
    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        # Write code here
        …

    # A function used by DFS
    def DFSUtil(self, v, visited):
        # Write code here
        …

    # The function to do DFS traversal.
    # It uses recursive DFSUtil
    def DFS(self):
```

```
        # Write code here
        …
# Driver's code
print("Following is Depth First Traversal")
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

# Function call
g.DFS()
```

Try:

1. **Detect a negative cycle in a Graph (Bellman Ford):** A Bellman-Ford algorithm is also guaranteed to find the shortest path in a graph, similar to Dijkstra's algorithm. Although Bellman-Ford is slower than Dijkstra's algorithm, it is capable of handling graphs with negative edge weights, which makes it more versatile. The shortest path cannot be found if there exists a negative cycle in the graph. If we continue to go around the negative cycle an infinite number of times, then the cost of the path will continue to decrease (even though the length of the path is increasing).

Consider a graph G and detect a negative cycle in the graph using Bellman Ford algorithm.



13. Minimum Spanning Tree (MST)

13.1 Kruskal's Algorithm

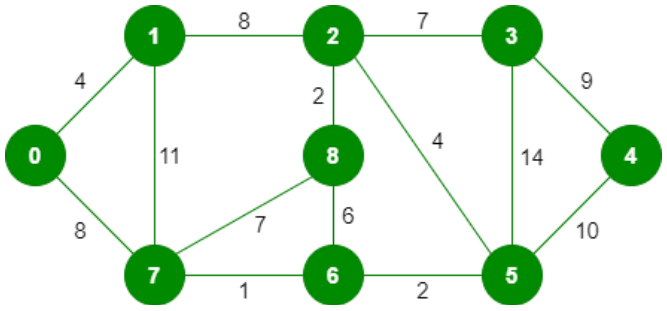
In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last.

MST using Kruskal's algorithm:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Input: For the given graph G find the minimum cost spanning tree.



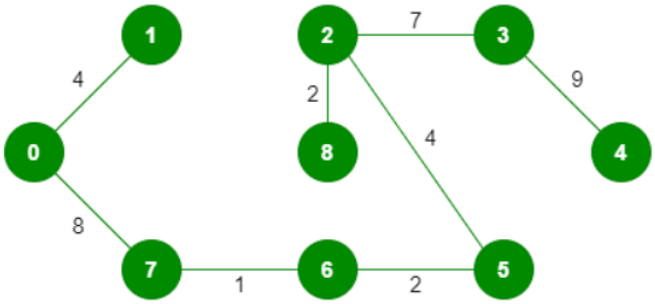
The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges.

Output:




```
# Kruskal's algorithm to find minimum Spanning Tree of a given connected,
# undirected and weighted graph

# Class to represent a graph
```

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    # Function to add an edge to graph
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        …

    def union(self, parent, rank, x, y):
        …

    def KruskalMST(self):
      # write your code here
      …

# Driver code
g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

# Function call
g.KruskalMST()
```

Output: Following are the edges in the constructed MST
2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning Tree: 19

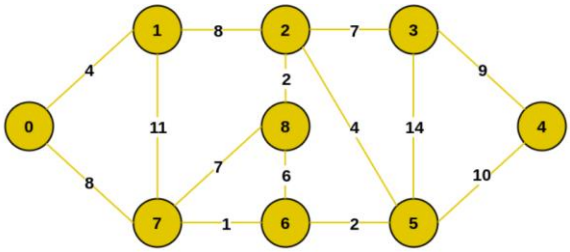
13.2 Prim's Algorithm

The Prim's algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

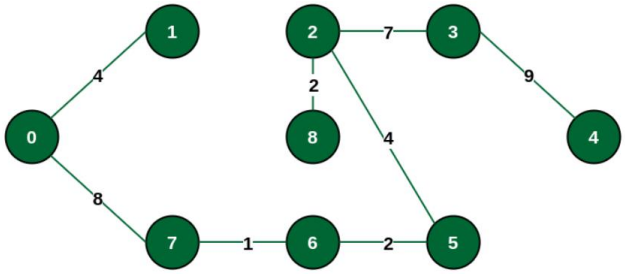
Prim's Algorithm:
The working of Prim's algorithm can be described by using the following steps:

1. Determine an arbitrary vertex as the starting vertex of the MST.
2. Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
3. Find edges connecting any tree vertex with the fringe vertices.
4. Find the minimum among these edges.
5. Add the chosen edge to the MST if it does not form any cycle.
6. Return the MST and exit

Input: For the given graph G find the minimum cost spanning tree.



Output: The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.




```
# Prim's Minimum Spanning Tree (MST) algorithm.
# The program is for adjacency matrix representation of the graph

# Library for INT_MAX
import sys

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    # A utility function to print
    # the constructed MST stored in parent[]
    def printMST(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minKey(self, key, mstSet):
        # write your code here
        …

    def primMST(self):
       # write your code here
        …


# Driver's code
g = Graph(5)
g.graph = [[0, 2, 0, 6, 0],
           [2, 0, 3, 8, 5],
           [0, 3, 0, 0, 7],
```

```
            [6, 8, 0, 0, 9],
            [0, 5, 7, 9, 0]]

g.primMST()
```

Output:

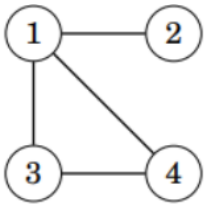
Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

13.3 Total Number of Spanning Trees in a Graph

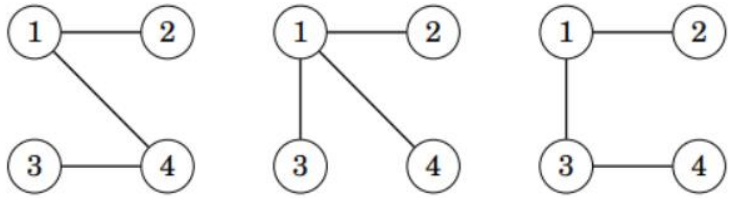
If a graph is a complete graph with n vertices, then total number of spanning trees is $n^{(n-2)}$ where n is the number of nodes in the graph. In complete graph, the task is equal to counting different labeled trees with n nodes for which have Cayley's formula.

Laplacian matrix:
A Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.

Kirchhoff's theorem provides a way to calculate the number of spanning trees for a given graph as a determinant of a special matrix. Consider the following graph,



All possible spanning trees are as follows:



In order to calculate the number of spanning trees, construct a Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.
for the above graph, The Laplacian matrix will look like this

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

The number of spanning trees equals the determinant of a matrix.

The Determinant of a matrix that can be obtained when we remove any row and any column from L.
For example, if we remove the first row and column, the result will be,

$$\det\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}\right) = 3.$$

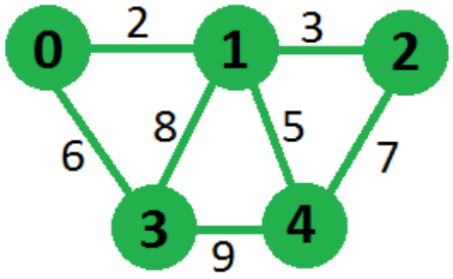
The determinant is always the same, regardless of which row and column we remove from L.

```
# Finds the number of spanning trees in a graph using Matrix Chain Multiplication.
MAX = 100
MOD = 1000000007

# Matrix Multiplication
def multiply(A, B, C):
    # write your code here
    …

# Function to find Nth power of A
def power(A, N, result):
    # write your code here
    …
# Function to find number of Spanning Trees in a Graph
# using Matrix Chain Multiplication.
def numOfSpanningTree(graph, V):
    # write your code here
    …

# Driver program
V = 4 # Number of vertices in graph
E = 5 # Number of edges in graph
graph = [[0, 1, 1, 1],
         [1, 0, 1, 1],
         [1, 1, 0, 1],
         [1, 1, 1, 0]]
print(numOfSpanningTree(graph, V))
```

13.4 Minimum Product Spanning Tree

A minimum product spanning tree for a weighted, connected, and undirected graph is a spanning tree with a weight product less than or equal to the weight product of every other spanning tree. The weight product of a spanning tree is the product of weights corresponding to each edge of the spanning tree. All weights of the given graph will be positive for simplicity.

Input:



Output: Minimum Product that we can obtain is 180 for above graph by choosing edges 0-1, 1-2, 0-3 and 1-4

This problem can be solved using standard minimum spanning tree algorithms like Kruskal and prim's algorithm, but we need to modify our graph to use these algorithms. Minimum spanning tree algorithms tries to minimize the total sum of weights, here we need to minimize the total product of weights. We can use the property of logarithms to overcome this problem.

log(w1* w2 * w3 * .... * wN) = log(w1) + log(w2) + log(w3) ..... + log(wN)

We can replace each weight of the graph by its log value, then we apply any minimum spanning tree algorithm which will try to minimize the sum of log(wi) which in turn minimizes the weight product.

```
# Minimum product spanning tree
import math

# Number of vertices in the graph
V = 5

# A utility function to find the vertex with minimum key value, from the set
# of vertices not yet included in MST
def minKey(key, mstSet):
    # write your code here
    …

# A utility function to print the constructed MST stored in parent[] and
# print Minimum Obtainable product
def printMST(parent, n, graph):
    # write your code here
    …

# Function to construct and print MST for a graph represented using adjacency
# matrix representation inputGraph is sent for printing actual edges and
# logGraph is sent for actual MST operations
def primMST(inputGraph, logGraph):
    # write your code here
    …

# Method to get minimum product spanning tree
def minimumProductMST(graph):
    # write your code here
    …

# Driver code
graph = [ [ 0, 2, 0, 6, 0 ],
          [ 2, 0, 3, 8, 5 ],
          [ 0, 3, 0, 0, 7 ],
          [ 6, 8, 0, 0, 9 ],
```

```
          [ 0, 5, 7, 9, 0 ], ]

# Print the solution
minimumProductMST(graph)
```

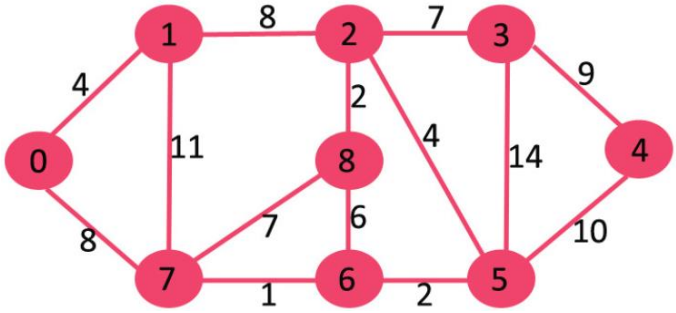
13.5 Reverse Delete Algorithm for Minimum Spanning Tree

In Reverse Delete algorithm, we sort all edges in decreasing order of their weights. After sorting, we one by one pick edges in decreasing order. We include current picked edge if excluding current edge causes disconnection in current graph. The main idea is delete edge if its deletion does not lead to disconnection of graph.

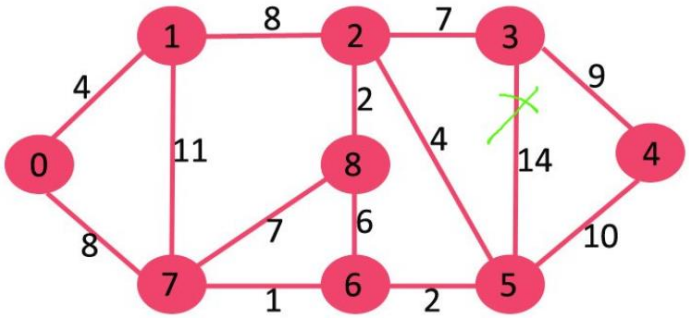
Algorithm:

1. Sort all edges of graph in non-increasing order of edge weights.
2. Initialize MST as original graph and remove extra edges using step 3.
3. Pick highest weight edge from remaining edges and check if deleting the edge disconnects the graph or not.
   If disconnects, then we don't delete the edge.
   Else we delete the edge and continue.

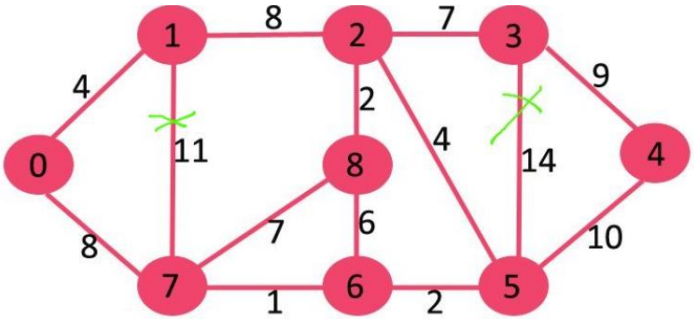
Input: Consider the graph below



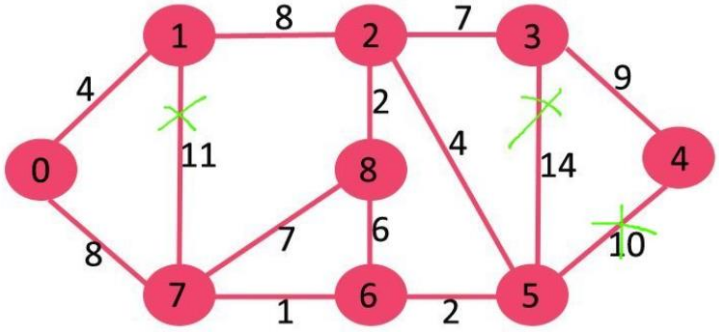
If we delete highest weight edge of weight 14, graph doesn't become disconnected, so we remove it.



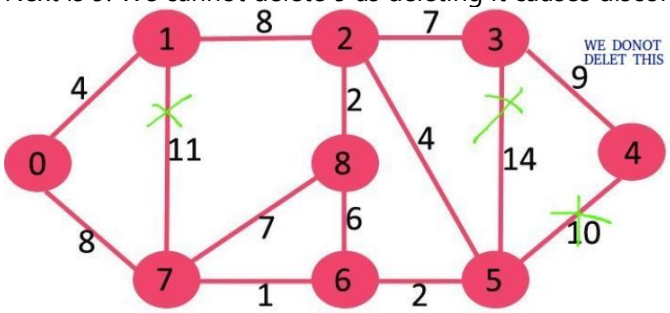
Next we delete 11 as deleting it doesn't disconnect the graph.



Next we delete 10 as deleting it doesn't disconnect the graph.



Next is 9. We cannot delete 9 as deleting it causes disconnection.



We continue this way and following edges remain in final MST.

Edges in MST

(3, 4)
(0, 7)
(2, 3)
(2, 5)
(0, 1)
(5, 6)
(2, 8)
(6, 7)

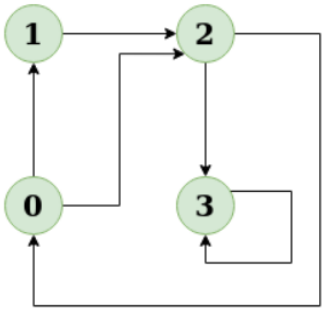
```
# Find Minimum Spanning Tree of a graph using Reverse Delete Algorithm

# Graph class represents a directed graph using adjacency list representation
class Graph:
    def __init__(self, v):

        # No. of vertices
        self.v = v
```

```
        self.adj = [0] * v
        self.edges = []
        for i in range(v):
            self.adj[i] = []

    # function to add an edge to graph
    def addEdge(self, u: int, v: int, w: int):
        # write code here
        …

    def dfs(self, v: int, visited: list):
       # write code here
        …

    # Returns true if graph is connected
    # Returns true if given graph is connected, else false
    def connected(self):
       # write code here
       …

    # This function assumes that edge (u, v) exists in graph or not
    def reverseDeleteMST(self):
        # write code here
        …

# Driver Code
# create the graph given in above figure
V = 9
g = Graph(V)

# making above shown graph
g.addEdge(0, 1, 4)
g.addEdge(0, 7, 8)
g.addEdge(1, 2, 8)
g.addEdge(1, 7, 11)
g.addEdge(2, 3, 7)
g.addEdge(2, 8, 2)
g.addEdge(2, 5, 4)
g.addEdge(3, 4, 9)
g.addEdge(3, 5, 14)
g.addEdge(4, 5, 10)
g.addEdge(5, 6, 2)
g.addEdge(6, 7, 1)
g.addEdge(6, 8, 6)
g.addEdge(7, 8, 7)

g.reverseDeleteMST()
```

Try:

1. **Detect Cycle in a Directed Graph:** Given the root of a Directed graph, The task is to check whether the graph contains a cycle or not.

Input: N = 4, E = 6



Output: Yes
Explanation: The diagram clearly shows a cycle 0 -> 2 -> 0

14. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

Name of the contest	Minimum number of problems to solve	Required score
• CodeChef	20	200
• Leetcode	20	200
• GeeksforGeeks	20	200
• SPOJ	5	50
• InterviewBit	10	1000
• Hackerrank	25	250
• Codeforces	10	100
• BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certifications:

1. HackerRank - Problem Solving Skills Certification (Basic and Intermediate)
2. GeeksforGeeks – Data Structures and Algorithms Certification
3. CodeChef - Learn Data Structures and Algorithms Certification
4. Interviewbit – DSA pro / Python pro
5. Edx – Data Structures and Algorithms
5. NPTEL – Programming, Data Structures and Algorithms
6. NPTEL – Introduction to Data Structures and Algorithms
7. NPTEL – Data Structures and Algorithms
8. NPTEL – Programming and Data Structure

V. TEXT BOOKS:

1. Rance D. Necaise, "Data Structures and Algorithms using Python", Wiley Student Edition.
2. Benjamin Baka, David Julian, "Python Data Structures and Algorithms", Packt Publishers, 2017.

VI. REFERENCE BOOKS:

1. S. Lipschutz, "Data Structures", Tata McGraw Hill Education, 1st edition, 2008.
2. D. Samanta, "Classic Data Structures", PHI Learning, 2nd edition, 2004.

VII. ELECTRONICS RESOURCES:

1. https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
2. https://www.codechef.com/certification/data-structures-and-algorithms/prepare
3. https://www.cs.auckland.ac.nz/software/AlgAnim/dsToC.html
4. https://online-learning.harvard.edu/course/data-structures-and-algorithms

VIII. MATERIALS ONLINE

1. Course Content
2. Lab manual

PROGRAMMING WITH OBJECTS LABORATORY

III Semester: Common for CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AITC03	**Core**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	3	1.5	30	70	100
Contact Classes: NIL	**Tutorial Classes: NIL**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Programming for Problem Solving using C								

I. COURSE OVERVIEW:

This course provides students with hands-on experience in developing programs and applications using object oriented programming. It covers classes, objects, inheritance, polymorphism, exception handling, files, multi- threading, database connectivity and AWT. It helps the students to develop real-world applications and enhances their programming skills.

II. COURSES OBJECTIVES:

The students will try to learn

I. The basic concepts of object oriented programming.
II. The application of object oriented features for developing flexible and extensible applications.
III. The Graphical User Interface (GUI) with database connectivity to develop web applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Demonstrate object oriented programming concepts that helps to organize complex problems solving.
CO 2	Make use of the programming constructs like control Structures, arrays, parameter passing techniques and constructors to solve the real time problems.
CO 3	Utilize the abstraction, encapsulation and polymorphism Techniques to solve different complex problems.
CO 4	Experiment all threading and thread synchronization problems in soft real time systems.
CO 5	Make use of inheritance, interfaces, packages and files to implement reusability in soft real time systems.
CO 6	Construct GUI based applications along with Exception handling using AWT, Swings and JDBC connectivity.

PROGRAMMING WITH OBJECTS LABORATORY

1. Getting Started Exercises

1.1 HelloWorld

1. Install JDK on your machine. Follow the instructions in "How to Install JDK".
2. Write a Hello-world program using JDK and a source-code editor, such as:
   - For All Platforms: Sublime Text, Atom
   - For Windows: TextPad, NotePad++
   - For macOS: jEdit, gedit
   - For Ubuntu: gedit

1.2 CheckOddEven

Write a program called **CheckOddEven** which prints "`Odd Number`" if the `int` variable "`number`" is odd, or "`Even Number`" otherwise. The program shall always print "`bye!`" before exiting.

Hints

n is an even number if (*n* `% 2`) is `0`; otherwise, it is an odd number. Use == for comparison, e.g., `(n % 2) == 0`.

```
/**
 * Trying if-else statement and modulus (%) operator.
 */
public class CheckOddEven {   // Save as "CheckOddEven.java"
   public static void main(String[] args) {  // Program entry point
      int number = 49;       // Set the value of "number" here!
      System.out.println("The number is " + number);
      if ( ...... ) {
         System.out.println( ...... );   // even number
      } else {
         System.out.println( ...... );   // odd number
      }
      System.out.println( ...... );
   }
}
```

Try

`number = 0`, `1`, `88`, `99`, `-1`, `-2` and verify your results.

Again, take note of the source-code indentation! Make it a good habit to ident your code properly, for ease of reading your program.

1.3 PrintDayInWord

Write a program called **PrintDayInWord** which prints "Sunday", "Monday", ... "Saturday" if the `int` variable "`dayNumber`" is `0`, `1`, ..., `6`, respectively. Otherwise, it shall print "Not a valid day". Use (a) a "nested-`if`" statement; (b) a "`switch-case-default`" statement.

Try

`dayNumber = 0`, 1, 2, 3, 4, 5, 6, 7 and verify your results.

1.4 Fibonacci (Decision & Loop)

Write a program called **Fibonacci** to print the first `20` Fibonacci numbers `F(n)`, where `F(n)=F(n-1)+F(n-2)` and `F(1)=F(2)=1`. Also compute their average. The output shall look like:

```
The first 20 Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
The average is 885.5
```

Hints

```
/**
 * Print first 20 Fibonacci numbers and their average
 */
public class Fibonacci {
   public static void main (String[] args) {
      int n = 3;          // The index n for F(n), starting from n=3, as n=1 and n=2 are
pre-defined
      int fn;             // F(n) to be computed
      int fnMinus1 = 1;   // F(n-1), init to F(2)
      int fnMinus2 = 1;   // F(n-2), init to F(1)
      int nMax = 20;      // maximum n, inclusive
      int sum = fnMinus1 + fnMinus2;  // Need sum to compute average
      double average;

      System.out.println("The first " + nMax + " Fibonacci numbers are:");
      ......

      while (n <= nMax) {  // n starts from 3
            // n = 3, 4, 5, ..., nMax
         // Compute F(n), print it and add to sum
         ......
         // Increment the index n and shift the numbers for the next iteration
         ++n;
         fnMinus2 = fnMinus1;
         fnMinus1 = fn;
      }

      // Compute and display the average (=sum/nMax).
      // Beware that int/int gives int.
      ......
   }
}
```

Try

1. *Tribonacci numbers* are a sequence of numbers `T(n)` similar to *Fibonacci numbers*, except that a number is formed by adding the three previous numbers, i.e., `T(n)=T(n-1)+T(n-2)+T(n-3)`, `T(1)=T(2)=1`, and `T(3)=2`. Write a program called **`Tribonacci`** to produce the first twenty Tribonacci numbers.

1.5 `ExtractDigits` (Decision & Loop)

Write a program called **`ExtractDigits`** to extract each digit from an `int`, in the reverse order. For example, if the `int` is 15423, the output shall be "3 2 4 5 1", with a space separating the digits.

Hints

The *coding pattern* for extracting individual digits from an integer *n* is:

1. Use `(n % 10)` to extract the last (least-significant) digit.
2. Use `n = n / 10` to drop the last (least-significant) digit.
3. Repeat if `(n > 0)`, i.e., more digits to extract.

Take note that *n* is destroyed in the process. You may need to clone a copy.

```
int n = ...;
while (n > 0) {
   int digit = n % 10;  // Extract the least-significant digit
   // Print this digit
   ......
   n = n / 10;  // Drop the least-significant digit and repeat the loop
}
```

Try

Write a program that prompts user for a positive integer. The program shall read the input as `int`; compute and print the sum of all its digits

1.6 `InputValidation` (Loop with `boolean` flag)

Your program often needs to validate the user's inputs, e.g., marks shall be between 0 and 100.
Write a program that prompts user for an integer between `0-10` or `90-100`. The program shall read the input as `int`; and repeat until the user enters a valid input. For examples,

```
Enter a number between 0-10 or 90-100: -1
Invalid input, try again...
Enter a number between 0-10 or 90-100: 50
Invalid input, try again...
Enter a number between 0-10 or 90-100: 101
Invalid input, try again...
Enter a number between 0-10 or 90-100: 95
You have entered: 95
```

Hints

Use the following *coding pattern* which uses a do-while loop controlled by a `boolean` flag to do input validation. We use a do-while instead of while-do loop as we need to execute the body to prompt and process the input at least once.

```
// Declare variables
int numberIn;      // to be input
boolean isValid;   // boolean flag to control the loop
......

// Use a do-while loop controlled by a boolean flag
// to repeatably read the input until a valid input is entered
isValid = false;   // default assuming input is not valid
do {
   // Prompt and read input
   ......

   // Validate input by setting the boolean flag accordingly
   if (numberIn ......) {
      isValid = true;   // exit the loop
   } else {
      System.out.println(......);  // Print error message and repeat
   }
} while (!isValid);
......
```

Try

Write a program that prompts user for the mark (between `0-100` in `int`) of 3 students; computes the average (in `double`); and prints the result rounded to 2 decimal places. Your program needs to perform input validation.

Hints

```
// Declare constant
final int NUM_STUDENTS = 3;

// Declare variables
int numberIn;
boolean isValid;   // boolean flag to control the input validation loop
int sum = 0;
double average;
......

for (int studentNo = 1; studentNo <= NUM_STUDENTS; ++studentNo) {
    // Prompt user for mark with input validation
    ......
    isValid = false;   // reset assuming input is not valid
    do {
       ......
    } while (!isValid);

    sum += ......;
}
......
```

1.7 IncomeTaxCalculator (Decision)

The progressive income tax rate is mandated as follows:

Taxable Income	Rate (%)
First $20,000	0
Next $20,000	10
Next $20,000	20
The remaining	30

For example, suppose that the taxable income is `$85000`, the income tax payable is `$20000*0% + $20000*10% + $20000*20% + $25000*30%`.

Write a program called **`IncomeTaxCalculator`** that reads the taxable income (in `int`). The program shall calculate the income tax payable (in `double`); and print the result rounded to 2 decimal places. For examples,

```
Enter the taxable income: $41234
The income tax payable is: $2246.80

Enter the taxable income: $67891
The income tax payable is: $8367.30

Enter the taxable income: $85432
The income tax payable is: $13629.60

Enter the taxable income: $12345
The income tax payable is: $0.00
```

Hints

```
// Declare constants first (variables may use these constants)
// The keyword "final" marked these as constant (i.e., cannot be changed).
// Use uppercase words joined with underscore to name constants
final double TAX_RATE_ABOVE_20K = 0.1;
final double TAX_RATE_ABOVE_40K = 0.2;
final double TAX_RATE_ABOVE_60K = 0.3;

// Declare variables
int taxableIncome;
double taxPayable;
......

// Compute tax payable in "double" using a nested-if to handle 4 cases
if (taxableIncome <= 20000) {         // [0, 20000]
   taxPayable = ......;
} else if (taxableIncome <= 40000) {  // [20001, 40000]
   taxPayable = ......;
} else if (taxableIncome <= 60000) {  // [40001, 60000]
   taxPayable = ......;
} else {                              // [60001, ]
   taxPayable = ......;
}
// Alternatively, you could use the following nested-if conditions
// but the above follows the table data
```

```
      //if (taxableIncome > 60000) {          // [60001, ]
      //   ......
      //} else if (taxableIncome > 40000) {   // [40001, 60000]
      //   ......
      //} else if (taxableIncome > 20000) {   // [20001, 40000]
      //   ......
      //} else {                              // [0, 20000]
      //   ......
      //}

      // Print results rounded to 2 decimal places
      System.out.printf("The income tax payable is: $%.2f%n", ...);
```

Try

Suppose that a `10%` tax rebate is announced for the income tax payable, capped at `$1,000`, modify your program to handle the tax rebate. For example, suppose that the tax payable is `$12,000`, the rebate is `$1,000`, as `10%` of `$12,000` exceeds the cap.

1.8 IncomeTaxCalculatorWithSentinel (Decision & Loop)

Based on the previous exercise, write a program called `IncomeTaxCalculatorWithSentinel` which shall repeat the calculation until user enter `-1`. For example,

```
Enter the taxable income (or -1 to end): $41000
The income tax payable is: $2200.00

Enter the taxable income (or -1 to end): $62000
The income tax payable is: $6600.00

Enter the taxable income (or -1 to end): $73123
The income tax payable is: $9936.90

Enter the taxable income (or -1 to end): $84328
The income tax payable is: $13298.40

Enter the taxable income: $-1
bye!
```

The `-1` is known as the *sentinel value*. (Wiki: In programming, a *sentinel value*, also referred to as a flag value, trip value, rogue value, signal value, or dummy data, is a special value which uses its presence as a condition of termination.)

Hints

The *coding pattern* for handling input with sentinel value is as follows:

```
      // Declare constants first
      final int SENTINEL = -1;    // Terminating value for input
      ......

      // Declare variables
      int taxableIncome;
      double taxPayable;
      ......

      // Read the first input to "seed" the while loop
      System.out.print("Enter the taxable income (or -1 to end): $");
      taxableIncome = in.nextInt();
```

```
      while (taxableIncome != SENTINEL) {
         // Compute tax payable
         ......
         // Print result
         ......

         // Read the next input
         System.out.print("Enter the taxable income (or -1 to end): $");
         taxableIncome = in.nextInt();
            // Repeat the loop body, only if the input is not the SENTINEL value.
            // Take note that you need to repeat these two statements inside/outside the
loop!
      }
      System.out.println("bye!");
```

Take note that we repeat the input statements inside and outside the loop. Repeating statements is NOT a good programming practice. This is because it is easy to repeat (Cntl-C/Cntl-V), but hard to maintain and synchronize the repeated statements. In this case, we have no better choices!

2. Exercises on Patterns and Arrays

2.1 SquarePattern (nested-loop)

Write a program called **SquarePattern** that prompts user for the size (a non-negative integer in `int`); and prints the following square pattern using two nested for-loops.

```
Enter the size: 5
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

Hints

The *code pattern* for printing 2D patterns using nested loops is:

```
      // Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            System.out.print( ...... );    // Use print() without newline inside the
inner loop
            ......
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

Notes

1. You should name the loop indexes `row` and `col`, NOT `i` and `j`, or `x` and `y`, or `a` and `b`, which are meaningless.
2. The `row` and `col` could start at 1 (and upto `size`), or start at `0` (and upto `size-1`). As computer counts from `0`, it is probably more efficient to start from `0`. However, since humans counts from `1`, it is easier to read if you start from `1`.

Try

Rewrite the above program using nested while-do loops.

2.2 CheckerPattern (nested-loop)

Write a program called **CheckerPattern** that prompts user for the size (a non-negative integer in `int`); and prints the following checkerboard pattern.

```
Enter the size: 7
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
```

Hints

```
      // Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            if ((row % 2) == 0) {   // row 2, 4, 6, ...
               ......
            }
            System.out.print( ...... );    // Use print() without newline inside the
inner loop
            ......
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

2.3 MultiplicationTable (nested-loop)

Write a program called **MultiplicationTable** that prompts user for the size (a positive integer in `int`); and prints the multiplication table as shown:

```
Enter the size: 10
 * |   1   2   3   4   5   6   7   8   9  10
-----------------------------------------------
 1 |   1   2   3   4   5   6   7   8   9  10
 2 |   2   4   6   8  10  12  14  16  18  20
 3 |   3   6   9  12  15  18  21  24  27  30
 4 |   4   8  12  16  20  24  28  32  36  40
 5 |   5  10  15  20  25  30  35  40  45  50
 6 |   6  12  18  24  30  36  42  48  54  60
 7 |   7  14  21  28  35  42  49  56  63  70
 8 |   8  16  24  32  40  48  56  64  72  80
 9 |   9  18  27  36  45  54  63  72  81  90
10 |  10  20  30  40  50  60  70  80  90 100
```

Hints

Use `printf()` to format the output, e.g., each cell is %4d.

2.4 TriangularPattern (nested-loop)

Write 4 programs called **`TriangularPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints each of the patterns as shown:

```
Enter the size: 8

#                   # # # # # # # #       # # # # # # # #                   #
# #                 # # # # # # #           # # # # # # #                 # #
# # #               # # # # # #               # # # # # #               # # #
# # # #             # # # # #                   # # # # #             # # # #
# # # # #           # # # #                       # # # #           # # # # #
# # # # # #         # # #                           # # #         # # # # # #
# # # # # # #       # #                               # #       # # # # # # #
# # # # # # # #     #                                   #     # # # # # # # #
     (a)                 (b)                 (c)                 (d)
```

Hints

1. On the main diagonal, `row = col`. On the opposite diagonal, `row + col = size + 1`, where `row` and `col` begin from 1.
2. You need to print the *leading* blanks, in order to push the # to the right. The *trailing* blanks are optional, which does not affect the pattern.
3. For pattern (a), `if (row >= col) print #`. Trailing blanks are optional.
4. For pattern (b), `if (row + col <= size + 1) print #`. Trailing blanks are optional.
5. For pattern (c), `if (row >= col) print #; else print blank`. Need to print the *leading* blanks.
6. For pattern (d), `if (row + col >= size + 1) print #; else print blank`. Need to print the *leading* blanks.
7. The *coding pattern* is:

```
// Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            if (......) {
                System.out.print("# ");
            } else {
                System.out.print("  ");  // Need to print the "leading" blanks
            }
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

2.5 BoxPattern (nested-loop)

Write 4 programs called **`BoxPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints the pattern as shown:

```
Enter the size: 8

# # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #
#           #      #                       #        #       #      # #       # #
#           #        #                   #            #   #        #   #   #   #
#           #          #               #                #         #     #     #
#           #            #           #                  #   #        #   #   #   #
#           #              #       #                    #       #      # #       # #
# # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #    # # # # # # #
     (a)              (b)             (c)              (d)             (e)
```

Hints

1. On the main diagonal, `row = col`. On the opposite diagonal, `row + col = size + 1`, where `row` and `col` begin from 1.
2. For pattern (a), `if (row == 1 || row == size || col == 1 || col == size) print #; else print blank`. Need to print the intermediate blanks.
3. For pattern (b), `if (row == 1 || row == size || row == col) print #; else print blank`.

2.6 `HillPattern` (nested-loop)

Write 3 programs called **HillPatternX** (X = A, B, C, D) that prompts user for the size (a non-negative integer in int); and prints the pattern as shown:

```
Enter the rows: 6

         #              # # # # # # # # # # #              #             # # # # # # # # # # #
       # # #              # # # # # # # # #              # # #           # # # # #   # # # # #
     # # # # #              # # # # # # #              # # # # #         # # # #       # # # #
   # # # # # # #              # # # # #              # # # # # # #       # # #           # # #
 # # # # # # # # #              # # #              # # # # # # # # #     # #               # #
# # # # # # # # # # #              #              # # # # # # # # # # #   #                   #
        (a)                    (b)                  # # # # # # # # # #     # #               # #
                                                      # # # # # # # #       # # #           # # #
                                                        # # # # # #         # # # #       # # # #
                                                          # # # #           # # # # #   # # # # #
                                                            #               # # # # # # # # # # #
                                                           (c)                     (d)
```

Hints

For pattern (a):

```
for (int row = 1; ......) {
         // numCol = 2*numRows - 1
         for (int col = 1; ......) {
            if ((row + col >= numRows + 1) && (row >= col - numRows + 1)) {
               ......;
            } else {
               ......;
            }
         }
         ......;
      }
```

or, use 2 sequential inner loops to print the columns:

```
      for (int row = 1; row <= rows; row++) {
         for (int col = 1; col <= rows; col++) {
            if ((row + col >= rows + 1)) {
```

```
            ......
         } else {
            ......
         }
      }
      for (int col = 2; col <= rows; col++) { // skip col = 1
         if (row >= col) {
            ......
         } else {
            ......
         }
      }
      ......
   }
```

2.7 NumberPattern (nested-loop)

Write 4 programs called **NumberPatternX** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints the pattern as shown:

```
Enter the size: 8

1                   1 2 3 4 5 6 7 8                   1       8 7 6 5 4 3 2 1
1 2                   1 2 3 4 5 6 7                   2 1     7 6 5 4 3 2 1
1 2 3                   1 2 3 4 5 6                   3 2 1   6 5 4 3 2 1
1 2 3 4                   1 2 3 4 5                   4 3 2 1 5 4 3 2 1
1 2 3 4 5                   1 2 3 4                   5 4 3 2 1 4 3 2 1
1 2 3 4 5 6                   1 2 3                   6 5 4 3 2 1 3 2 1
1 2 3 4 5 6 7                   1 2                   7 6 5 4 3 2 1 2 1
1 2 3 4 5 6 7 8                   1                   8 7 6 5 4 3 2 1 1
     (a)                 (b)                 (c)             (d)
```

2.8 PrintArray(Array)

Write a program called `PrintArray` which prompts user for the number of items in an array (a non-negative integer), and saves it in an `int` variable called `NUM_ITEMS`. It then prompts user for the values of all the items and saves them in an `int` array called `items`. The program shall then print the contents of the array in the form of `[x1, x2, ..., xn]`. For example,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 3 2 5 6 9
The values are: [3, 2, 5, 6, 9]
```

Hints

```
      // Declare variables
      tinal int NUM_ITEMS;
      int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
      ......

      // Prompt for for the number of items and read the input as "int"
      ......
      NUM_ITEMS = ......

      // Allocate the array
```

```
      items = new int[NUM_ITEMS];

      // Prompt and read the items into the "int" array, if array length > 0
      if (items.length > 0) {
         ......
         for (int i = 0; i < items.length; ++i) {  // Read all items
            ......
         }
      }

      //  Print  array  contents,  need  to  handle  first  item  and  subsequent  items
differently
      ......
      for (int i = 0; i < items.length; ++i) {
         if (i == 0) {
            // Print the first item without a leading commas
            ......
         } else {
            // Print the subsequent items with a leading commas
            ......
         }
         // or, using a one liner
         //System.out.print((i == 0) ? ...... : ......);
      }
```

2.9 PrintArrayInStars (Array)

Write a program called **printArrayInStars** which prompts user for the number of items in an array (a non-negative integer), and saves it in an `int` variable called `NUM_ITEMS`. It then prompts user for the values of all the items (non-negative integers) and saves them in an `int` array called `items`. The program shall then print the contents of the array in a graphical form, with the array index and values represented by number of stars. For examples,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 7 4 3 0 7
0: *******(7)
1: ****(4)
2: ***(3)
3: (0)
4: *******(7)
```

Hints

```
      // Declare variables
      final int NUM_ITEMS;
      int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
      ......
      ......
      // Print array in "index: number of stars" using a nested-loop
      // Take note that rows are the array indexes and columns are the value in that
index
      for (int idx = 0; idx < items.length; ++idx) {  // row
         System.out.print(idx + ": ");
         // Print value as the number of stars
         for (int starNo = 1; starNo <= items[idx]; ++starNo) {  // column
            System.out.print("*");
         }
```

```
      ......
   }
   ......
```

2.10 GradesStatistics (Array)

Write a program which prompts user for the number of students in a class (a non-negative integer), and saves it in an `int` variable called `numStudents`. It then prompts user for the grade of each of the students (integer between 0 to 100) and saves them in an `int` array called `grades`. The program shall then compute and print the average (in `double` rounded to 2 decimal places) and minimum/maximum (in `int`).

```
Enter the number of students: 5
Enter the grade for student 1: 98
Enter the grade for student 2: 78
Enter the grade for student 3: 78
Enter the grade for student 4: 87
Enter the grade for student 5: 76
The average is: 83.40
The minimum is: 76
The maximum is: 98
```

2.11 Hex2Bin (Array for Table Lookup)

Write a program called **`Hex2Bin`** that prompts user for a hexadecimal string and print its equivalent binary string. The output shall look like:

```
Enter a Hexadecimal string: 1abc
The equivalent binary for hexadecimal "1abc" is: 0001 1010 1011 1100
```

Hints

1. Use an array of 16 `String`s containing binary strings corresponding to hexadecimal number `0-9A-F` (or `a-f`), as follows

```
final String[] HEX_BITS = {"0000", "0001", "0010", "0011",
                   "0100", "0101", "0110", "0111",
                   "1000", "1001", "1010", "1011",
                   "1100", "1101", "1110", "1111"};
```

2.12 Dec2Hex (Array for Table Lookup)

Write a program called **`Dec2Hex`** that prompts user for a positive decimal number, read as `int`, and print its equivalent hexadecimal string. The output shall look like:

```
Enter a decimal number: 1234
The equivalent hexadecimal number is 4D2
```

3. Magic(Special) Numbers

3.1 Amicable umbers

Two different numbers are said to be so Amicable numbers if each sum of divisors is equal to the other number. Write a Java program to check whether the given numbers are amicable or not. For example,

```
Enter 1st number: 228
Enter 2nd number: 220
The numbers are Amicable Numbers.
```

Hints

220 and 284 are Amicable Numbers.

Divisors of 220 = 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110

1+2+4+5+10+11+20+22+44+55+110 = 284

Divisors of 284 = 1, 2, 4, 71, 142

1+2+4+71+142 = 220

Try

1. Print 5 pairs of amicable numbers.

Hints

Amicable Numbers are: (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368).

3.2 Armstrong Number

Armstrong number is a positive number if it is equal to the sum of cubes of its digits is called Armstrong number and if its sum is not equal to the number then it's not a Armstrong number. For example,

```
Enter number:145
145 is not an Armstrong Number

Enter number: 153
153 is an Armstrong Number
```

Hints

Examples: 153 is Armstrong

(1*1*1)+(5*5*5)+(3*3*3) = 153

Try

Print all Armstrong numbers below 10000

3.3 Capricorn Number

A number is called Capricorn (or Kaprekar) number whose square is divided into two parts in any conditions and parts are added, the additions of parts is equal to the number, is called Capricorn or Kaprekar number. For example,

```
Enter a number: 45
45 is a Capricorn number
Enter a number: 297
297 is a Capricorn number
Enter a number: 44
```

```
44 is not a Capricorn number
```

Hints

```
Number = 45
(45)^2   = 2025

All parts for 2025:
202 + 5 = 207 (not 45)
20 + 25 = 45
2+ 025  = 27 (not 45)
```

From the above we can see one combination is equal to number so that 45 is Capricorn or Kaprekar number.

Try

Write a Java program to generate and show all Kaprekar numbers less than 1000.

3.4 Circular Prime

A circular prime is a prime number with the property that the number generated at each intermediate step when cyclically permuting its digits will be prime. For example, 1193 is a circular prime, since 1931, 9311 and 3119 all are also prime. For example,

```
Enter a number: 137
137 is a Circular Prime
Enter a number: 44
44 is not a Circular Prime
```

Try

Write Java code to display all circular primes from 1 to 1000.

3.5 Happy Number

A happy number is a natural number in a given number base that eventually reaches 1 when iterated over the perfect digital invariant function for. Those numbers that do not end in 1 are -unhappy numbers. For example,

```
Enter a number: 31
31 is a Happy number

Enter a number: 32
32 is not a Happy number
```

Try

Print all happy numbers from 1 to 1000.

3.6. Automorphic Number

An Automorphic number is a number whose square "ends" in the same digits as the number itself. For example,

```
Enter a number: 5
5 is a Automorphic Number

Enter a number: 25
```

25 is a Automorphic Number

Enter a number: 2
2 is not a Automorphic Number

Hints

5*5 = 25, 6*6 = 36, 25*25 = 625

5,6,25 are automorphic numbers

Try

Print all automorphic numbers from 1 to 10000

3.7 Disarium Number

A number is called Disarium number if the sum of its power of the positions from left to right is equal to the number. For example,

Enter a number: 135
135 is a Disarium Number

Enter a number: 32
32 is not a Disarium Number

Hints

$1^1 + 3^2 + 5^3 = 1 + 9 + 125 = 135$

Try

Print all Disarium numbers from 1 to 10000

3.8 Magic Number

Magic number is the sum of its digits recursively are calculated till a single digit If the single digit is 1 then the number is a magic number. Magic number is very similar with Happy Number. For example,

Enter a number: 226
226 is a Magic Number

Enter a number: 32
32 is not a Magic Number

Hints

226 is said to be a magic number

2+2+6=10 sum of digits is 10 then again 1+0=1 now we get a single digit number is 1. if we single digit number will now 1 them it would not a magic number.

Try

1. A neon number is a number where the sum of digits of square of the number is equal to the number. For example if the input number is 9, its square is 9*9 = 81 and sum of the digits is 9. i.e. 9 is a neon number. For example,

Enter a number: 9
9 is a Neon Number

```
Enter a number: 8
8 is not a Neon Number
```

2. A palindromic number is a number that remains the same when its digits are reversed. For example,

```
Enter a number: 16461
16461 is a Palindromic Number

Enter a number: 1234
1234 is not a Palindromic Number
```

3. A perfect number is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. For instance, 6 has divisors 1, 2 and 3, and 1 + 2 + 3 = 6, so 6 is a perfect number. For example,

```
Enter a number: 6
6 is a Perfect Number

Enter a number: 3
3 is not a Perfect Number
```

4. A number is said to be special number when the sum of factorial of its digits is equal to the number itself. Example- 145 is a Special Number as 1!+4!+5!=145. For example,

```
Enter a number: 145
145 is a Special Number

Enter a number: 23
23 is not a Special Number
```

5. A spy number is a number where the sum of its digits equals the product of its digits. For example, 1124 is a spy number, the sum of its digits is 1+1+2+4=8 and the product of its digits is 1*1*2*4=8. For example,

```
Enter a number: 1124
1124 is a Spy Number

Enter a number: 12
12 is not a Spy Number
```

6. A number is said to be an Ugly number if positive numbers whose prime factors only include 2, 3, 5. For example, 6(2×3), 8(2x2x2), 15(3×5) are ugly numbers while 14(2×7) is not ugly since it includes another prime factor 7. Note that 1 is typically treated as an ugly number. For example,

```
Enter a number: 6
6 is an Ugly Number

Enter a number: 14
14 is not an Ugly Number
```

3.9 swap() (Array & Method)

Write a method called **`swap()`**, which takes two arrays of `int` and swap their contents if they have the same length. It shall return `true` if the contents are successfully swapped. The method's signature is as follows:

```
public static boolean swap(int[] array1, int[] array2)
```

Also write a test driver to test this method.

Hints

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

3.10 reverse() (Array & Method)

Write a method called **reverse()**, which takes an array of `int` and reverse its contents. For example, the reverse of `[1,2,3,4]` is `[4,3,2,1]`. The method's signature is as follows:

```
public static void reverse(int[] array)
```

Take note that the array passed into the method can be modified by the method (this is called "*pass by reference*"). On the other hand, primitives passed into a method cannot be modified. This is because a clone is created and passed into the method instead of the original copy (this is called "*pass by value*").

Also write a test driver to test this method.

Hints

You might use two indexes in the loop, one moving forward and one moving backward to point to the two elements to be swapped.

```
for (int fIdx = 0, bIdx = array.length - 1; fIdx < bIdx; ++fIdx, --bIdx) {
   // Swap array[fIdx] and array[bIdx]
   // Only need to transverse half of the array elements
}
```

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

3.11 GradesStatistics (Array & Method)

Write a program called **GradesStatistics**, which reads in *n* grades (of `int` between `0` and `100`, inclusive) and displays the *average*, *minimum*, *maximum*, *median* and *standard deviation*. Display the floating-point values upto 2 decimal places. Your output shall look like:

```
Enter the number of students: 4
Enter the grade for student 1: 50
Enter the grade for student 2: 51
Enter the grade for student 3: 56
Enter the grade for student 4: 53
The grades are: [50, 51, 56, 53]
The average is: 52.50
The median is: 52.00
The minimum is: 50
The maximum is: 56
The standard deviation is: 2.29
```

The formula for calculating standard deviation is:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i{}^2 - \mu^2}, \text{ where } \mu \text{ is the mean}$$

Hints:

```
public class GradesStatistics {
   public static int[] grades;  // Declare an int[], to be allocated later.
                                // This array is accessible by all the methods.

   public static void main(String[] args) {
      readGrades();  // Read and save the inputs in global int[] grades
      System.out.println("The grades are: ");
      print(grades);
      System.out.println("The average is " + average(grades));
      System.out.println("The median is " + median(grades));
      System.out.println("The minimum is " + min(grades));
      System.out.println("The maximum is " + max(grades));
      System.out.println("The standard deviation is " + stdDev(grades));
   }

   // Prompt user for the number of students and allocate the global "grades" array.
   // Then, prompt user for grade, check for valid grade, and store in "grades".
   public static void readGrades() { ..... }

   // Print the given int array in the form of [x1, x2, x3,..., xn].
   public static void print(int[] array) { ..... }

   // Return the average value of the given int[]
   public static double average(int[] array) { ..... }

   // Return the median value of the given int[]
   // Median is the center element for odd-number array,
   // or average of the two center elements for even-number array.
   // Use Arrays.sort(anArray) to sort anArray in place.
   public static double median(int[] array) { ..... }

   // Return the maximum value of the given int[]
   public static int max(int[] array) {
      int max = array[0];   // Assume that max is the first element
      // From second element, if the element is more than max, set the max to this
element.
      ......
   }

   // Return the minimum value of the given int[]
   public static int min(int[] array) { ..... }

   // Return the standard deviation of the given int[]
   public static double stdDev(int[] array) { ..... }
}
```

Take note that besides `readGrade()` that relies on global variable `grades`, all the methods are *self-contained general utilities* that operate on any given array.

3.12 GradesHistogram (Array & Method)

Write a program called **GradesHistogram**, which reads in *n* grades (as in the previous exercise), and displays the horizontal and vertical histograms. For example:

```
 0 -  9: ***
10 - 19: ***
20 - 29:
30 - 39:
40 - 49: *
50 - 59: *****
60 - 69:
70 - 79:
80 - 89: *
90 -100: **

                              *
                              *
 *     *                      *
 *     *                      *                      *
 *     *                  *   *                  *   *
0-9  10-19 20-29 30-39 40-49 50-59 60-69 70-79 80-89 90-100
```

4. Exercises on String and char Operations

4.1 ReverseString (String & char)

Write a program called **ReverseString**, which prompts user for a `String`, and prints the *reverse* of the `String` by extracting and processing each character. The output shall look like:

```
Enter a String: abcdef
The reverse of the String "abcdef" is "fedcba".
```

Hints

For a `String` called `inStr`, you can use `inStr.length()` to get the *length* of the `String`; and `inStr.charAt(idx)` to retrieve the `char` at the `idx` position, where `idx` begins at 0, up to `instr.length() - 1`.

```
// Define variables
String inStr;        // input String
int inStrLen;        // length of the input String
......

// Prompt and read input as "String"
System.out.print("Enter a String: ");
inStr = in.next();   // use next() to read a String
inStrLen = inStr.length();
```

```
      // Use inStr.charAt(index) in a loop to extract each character
      // The String's index begins at 0 from the left.
      // Process the String from the right
      for (int charIdx = inStrLen - 1; charIdx >= 0; --charIdx) {
            // charIdx = inStrLen-1, inStrLen-2, ... ,0
         ......
      }
```

4.2 CountVowelsDigits (String & char)

Write a program called **CountVowelsDigits**, which prompts the user for a `String`, counts the number of vowels (`a`, `e`, `i`, `o`, `u`, `A`, `E`, `I`, `O`, `U`) and digits (`0`-`9`) contained in the string, and prints the counts and the percentages (rounded to 2 decimal places). For example,

```
Enter a String: testing12345
Number of vowels: 2 (16.67%)
Number of digits: 5 (41.67%)
```

Hints

1. To check if a `char c` is a digit, you can use `boolean` expression `(c >= '0' && c <= '9')`; or use built-in `boolean` function `Character.isDigit(c)`.
2. You could use `in.next().toLowerCase()` to convert the input `String` to lowercase to reduce the number of cases.
3. To print a `%` using `printf()`, you need to use `%%`. This is because `%` is a prefix for format specifier in `printf()`, e.g., `%d` and `%f`.

4.3 PhoneKeyPad (String & char)

On you phone keypad, the alphabets are mapped to digits as follows:

`ABC(2)`, `DEF(3)`, `GHI(4)`, `JKL(5)`, `MNO(6)`, `PQRS(7)`, `TUV(8)`, `WXYZ(9)`.

Write a program called **PhoneKeyPad**, which prompts user for a `String` (case insensitive), and converts to a sequence of keypad digits. Use (a) a nested-`if`, (b) a `switch-case-default`.

Hints

1. You can use `in.next().toLowerCase()` to read a `String` and convert it to lowercase to reduce your cases.
2. In `switch-case`, you can handle multiple cases by omitting the `break` statement, e.g.,

```
switch (inChar) {
   case 'a': case 'b': case 'c':  // No break for 'a' and 'b', fall thru 'c'
      System.out.print(2); break;
   case 'd': case 'e': case 'f':
      ......
   default:
      ......
}
```

4.4 Caesar's Code (String & char)

Caesar's Code is one of the simplest encryption techniques. Each letter in the plaintext is replaced by a letter some fixed number of position (n) down the alphabet cyclically. In this exercise, we shall pick `n=3`. That is, `'A'` is replaced by `'D'`, `'B'` by `'E'`, `'C'` by `'F'`, ..., `'X'` by `'A'`, ..., `'Z'` by `'C'`.

Write a program called **CaesarCode** to cipher the Caesar's code. The program shall prompt user for a plaintext string consisting of mix-case letters only; compute the ciphertext; and print the ciphertext in uppercase. For example,

```
Enter a plaintext string: Testing
The ciphertext string is: WHVWLQJ
```

Hints

1. Use `in.next().toUpperCase()` to read an input string and convert it into uppercase to reduce the number of cases.
2. You can use a big nested-if with 26 cases (`'A'-'Z'`). But it is much better to consider `'A'` to `'W'` as one case; `'X'`, `'Y'` and `'Z'` as 3 separate cases.
3. Take note that `char 'A'` is represented as Unicode number 65 and `char 'D'` as 68. However, `'A' + 3 gives 68`. This is because `char + int` is implicitly casted to `int + int` which returns an `int` value. To obtain a `char` value, you need to perform explicit type casting using `(char)('A' + 3)`. Try printing `('A' + 3)` with and without type casting.

4.5 Decipher Caesar's Code (String & char)

Write a program called **DecipherCaesarCode** to decipher the Caesar's code described in the previous exercise. The program shall prompts user for a ciphertext string consisting of mix-case letters only; compute the plaintext; and print the plaintext in uppercase. For example,

```
Enter a ciphertext string: wHVwLQJ
The plaintext string is: TESTING
```

4.6 Exchange Cipher (String & char)

This simple cipher exchanges `'A'` and `'Z'`, `'B'` and `'Y'`, `'C'` and `'X'`, and so on.
Write a program called **ExchangeCipher** that prompts user for a plaintext string consisting of mix-case letters only. You program shall compute the ciphertext; and print the ciphertext in uppercase. For examples,

```
Enter a plaintext string: abcXYZ
The ciphertext string is: ZYXCBA
```

Hints

1. Use `in.next().toUpperCase()` to read an input string and convert it into uppercase to reduce the number of cases.
2. You can use a big nested-if with 26 cases (`'A'-'Z'`), or use the following relationship:

```
'A' + 'Z' == 'B' + 'Y' == 'C' + 'X' == ... == plainTextChar + cipherTextChar
Hence, cipherTextChar = 'A' + 'Z' - plainTextChar
```

4.7 TestPalindromicWord and TestPalindromicPhrase (String & char)

A word that reads the same backward as forward is called a *palindrome*, e.g., "mom", "dad", "racecar", "madam", and "Radar" (case-insensitive).

Write a program called **TestPalindromicWord**, that prompts user for a word and prints ""xxx" is|is not a palindrome".

A phrase that reads the same backward as forward is also called a palindrome, e.g., "Madam, I'm Adam", "A man, a plan, a canal - Panama!" (ignoring punctuation and capitalization).

Modify your program (called **TestPalindromicPhrase**) to check for palindromic phrase. Use in.nextLine() to read a line of input.

Hints

1. Maintain two indexes, `forwardIndex (fIdx)` and `backwardIndex (bIdx)`, to scan the phrase forward and backward.

```
int fIdx = 0, bIdx = strLen - 1;
while (fIdx < bIdx) {
   ......
   ++fIdx;
   --bIdx;
}
// or
for (int fIdx = 0, bIdx = strLen - 1; fIdx < bIdx; ++fIdx, --bIdx) {
   ......
}
```

2. You can check if a `char c` is a letter either using built-in `boolean` function `Character.isLetter(c)`; or `boolean` expression `(c >= 'a' && c <= 'z')`. Skip the index if it does not contain a letter.

4.8 CheckBinStr (String & char)

The binary number system uses 2 symbols, 0 and 1. Write a program called **`CheckBinStr`** to verify a binary string. The program shall prompt user for a binary string; and decide if the input string is a valid binary string. For example,

```
Enter a binary string: 10101100
"10101100" is a binary string

Enter a binary string: 10120000
"10120000" is NOT a binary string
```

Hints

Use the following coding pattern which involves a `boolean` flag to check the input string.

```
// Declare variables
      String inStr;     // The input string
      int inStrLen;     // The length of the input string
      char inChar;      // Each char of the input string
      boolean isValid;  // "is" or "is not" a valid binary string?
      ......

      isValid = true;  // Assume that the input is valid, unless our check fails
      for (......) {
         inChar = ......;
```

```
         if (!(inChar == '0' || inChar == '1')) {
            isValid = false;
            break;   // break the loop upon first error, no need to continue for
more errors
                     // If this is not encountered, isValid remains true after the loop.
         }
      }
      if (isValid) {
         System.out.println(......);
      } else {
         System.out.println(......);
      }
      // or using one liner
      //System.out.println(isValid ? ... : ...);
```

4.9 CheckHexStr (String & char)

The hexadecimal (hex) number system uses 16 symbols, `0-9` and `A-F` (or `a-f`). Write a program to verify a hex string. The program shall prompt user for a hex string; and decide if the input string is a valid hex string. For examples,

```
Enter a hex string: 123aBc
"123aBc" is a hex string

Enter a hex string: 123aBcx
"123aBcx" is NOT a hex string
```

Hints

```
if (!((inChar >= '0' && inChar <= '9')
      || (inChar >= 'A' && inChar <= 'F')
      || (inChar >= 'a' && inChar <= 'f'))) { // Use positive logic and then reverse
   ......
}
```

4.10 Bin2Dec (String & char)

Write a program called `Bin2Dec` to convert an input binary string into its equivalent decimal number. Your output shall look like:

```
Enter a Binary string: 1011
The equivalent decimal number for binary "1011" is: 11

Enter a Binary string: 1234
error: invalid binary string "1234"
```

4.11 Hex2Dec (String & char)

Write a program called **`Hex2Dec`** to convert an input hexadecimal string into its equivalent decimal number. Your output shall look like:

```
Enter a Hexadecimal string: 1a
The equivalent decimal number for hexadecimal "1a" is: 26
```

```
Enter a Hexadecimal string: 1y3
error: invalid hexadecimal string "1y3"
```

4.12 Oct2Dec (String & char)

Write a program called **Oct2Dec** to convert an input Octal string into its equivalent decimal number. For example,

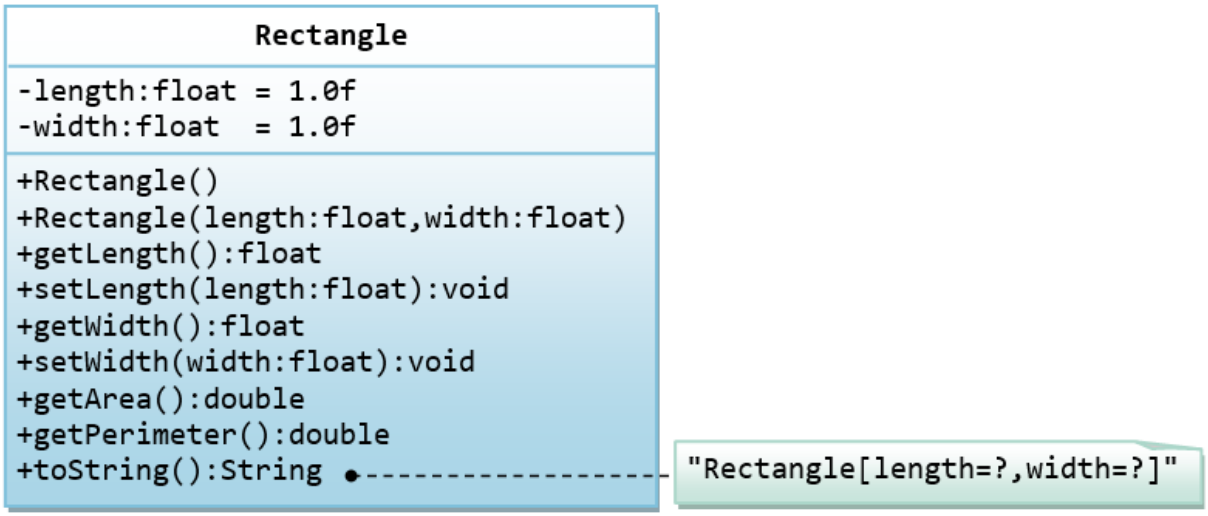
```
Ente an Octal string: 147
The equivalent decimal number "147" is: 103
```

5. Exercises on Classes and Objects

5.1 The Rectangle Class

A class called `Rectangle`, which models a rectangle with a length and a width (in `float`), is designed as shown in the following class diagram. Write the `Rectangle` class.

Hints:



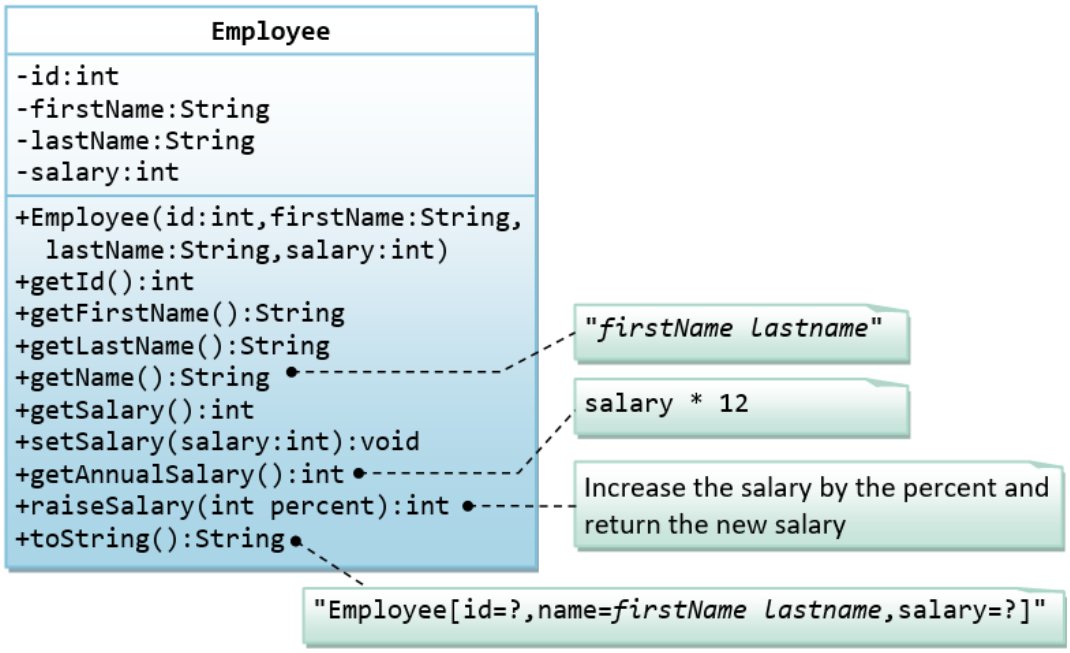
The expected output is:

```
Rectangle[length=1.2,width=3.4]
Rectangle[length=1.0,width=1.0]
Rectangle[length=5.6,width=7.8]
length is: 5.6
width is: 7.8
area is: 43.68
perimeter is: 26.80
```

5.2 The `Employee` Class

A class called `Employee`, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method `raiseSalary(percent)` increases the salary by the given percentage. Write the `Employee` class.

Hints:



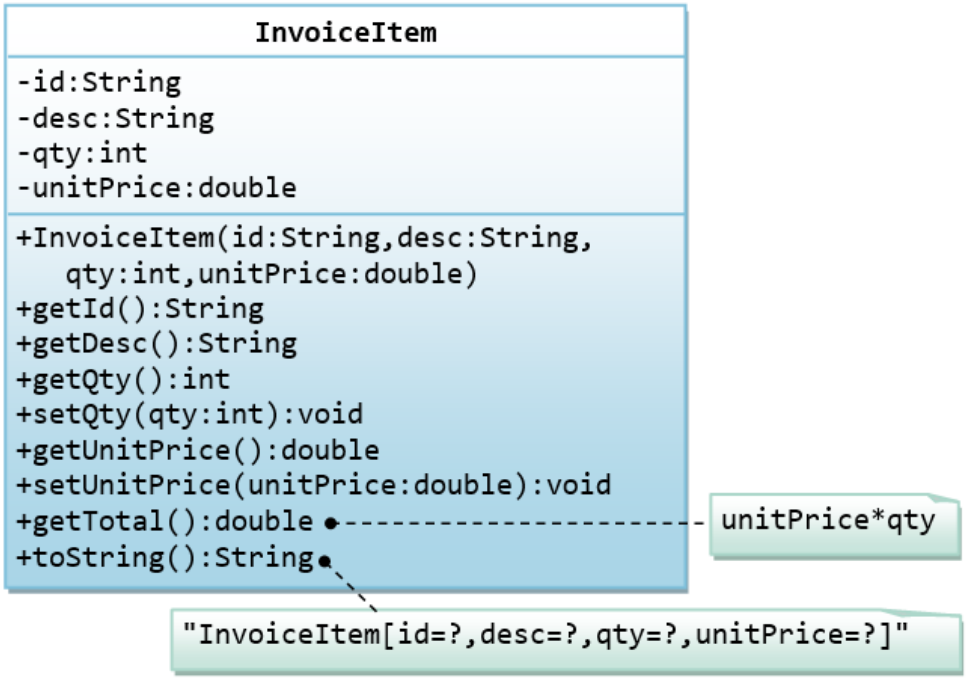
The expected out is:

```
Employee[id=8,name=Peter Tan,salary=2500]
Employee[id=8,name=Peter Tan,salary=999]
id is: 8
firstname is: Peter
lastname is: Tan
salary is: 999
name is: Peter Tan
annual salary is: 11988
1098
Employee[id=8,name=Peter Tan,salary=1098]
```

5.3 The `InvoiceItem` Class

A class called `InvoiceItem`, which models an item of an invoice, with ID, description, quantity and unit price, is designed as shown in the following class diagram. Write the `InvoiceItem` class.

Hints:



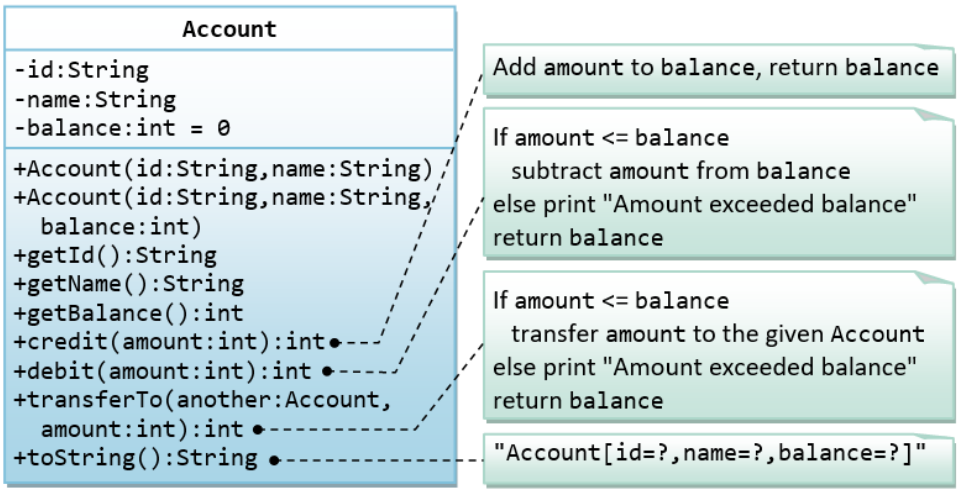
The expected output is:

```
InvoiceItem[id=A101,desc=Pen Red,qty=888,unitPrice=0.08]
InvoiceItem[id=A101,desc=Pen Red,qty=999,unitPrice=0.99]
id is: A101
desc is: Pen Red
qty is: 999
unitPrice is: 0.99
The total is: 989.01
```

5.4 The Account Class

A class called `Account`, which models a bank account of a customer, is designed as shown in the following class diagram. The methods `credit(amount)` and `debit(amount)` add or subtract the given `amount` to the `balance`. The method `transferTo(anotherAccount, amount)` transfers the given `amount` from this `Account` to the given `anotherAccount`. Write the `Account` class.

Hints:



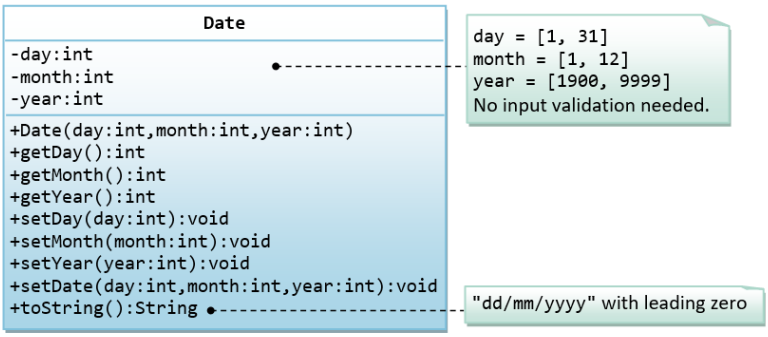
The expected output is:

```
Account[id=A101,name=Tan Ah Teck, balance=88]
Account[id=A102,name=Kumar, balance=0]
ID: A101
Name: Tan Ah Teck
Balance: 88
Account[id=A101,name=Tan Ah Teck, balance=188]
Account[id=A101,name=Tan Ah Teck, balance=138]
Amount exceeded balance
Account[id=A101,name=Tan Ah Teck, balance=138]
Account[id=A101,name=Tan Ah Teck, balance=38]
Account[id=A102,name=Kumar, balance=100]
```

5.5 The `Date` Class

A class called `Date`, which models a calendar date, is designed as shown in the following class diagram. Write the `Date` class.

Hints:



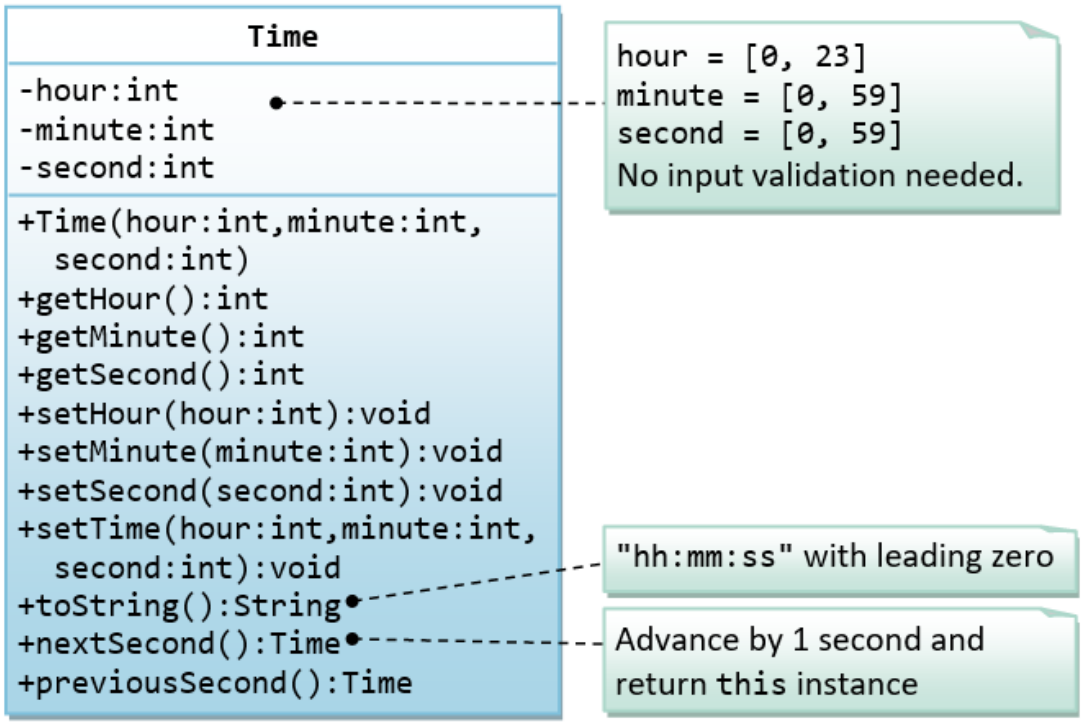
The expected output is:

```
01/02/2014
09/12/2099
Month: 12
Day: 9
Year: 2099
03/04/2016
```

5.6 Ex: The `Time` Class

A class called `Time`, which models a time instance, is designed as shown in the following class diagram. The methods `nextSecond()` and `previousSecond()` shall advance or rewind this instance by one second, and return this instance, so as to support chaining operation such as `t1.nextSecond().nextSecond()`. Write the `Time` class.

Hints:



The expected output is:

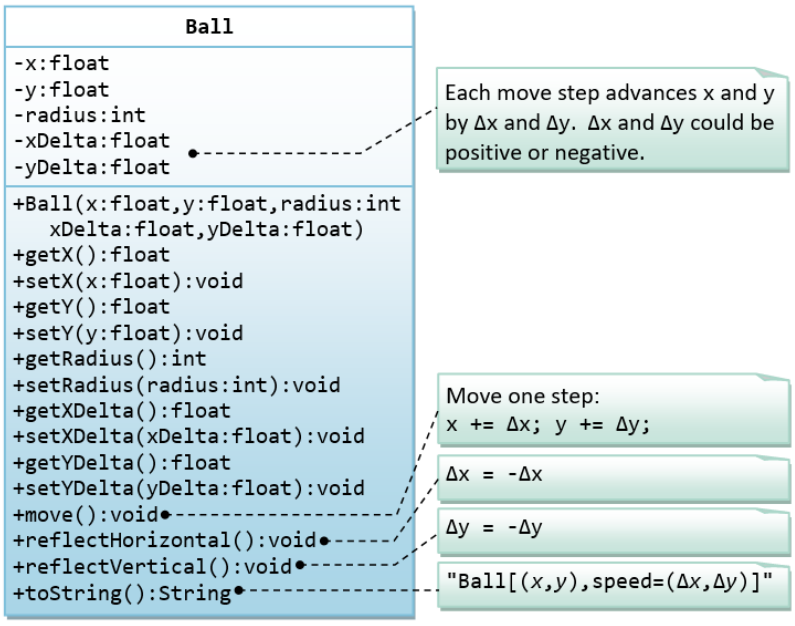
```
01:02:03
04:05:06
Hour: 4
Minute: 5
Second: 6
23:59:58
23:59:59
00:00:01
00:00:00
23:59:58
```

5.7 The `Ball` Class

A class called `Ball`, which models a bouncing ball, is designed as shown in the following class diagram. It contains its radius, x and y position. Each move-step advances the x and y by delta-x and delta-y, respectively. delta-x and delta-y could be positive or negative.

The `reflectHorizontal()` and `reflectVertical()` methods could be used to bounce the ball off the walls. Write the `Ball` class. Study the test driver on how the ball bounces.

Hints:



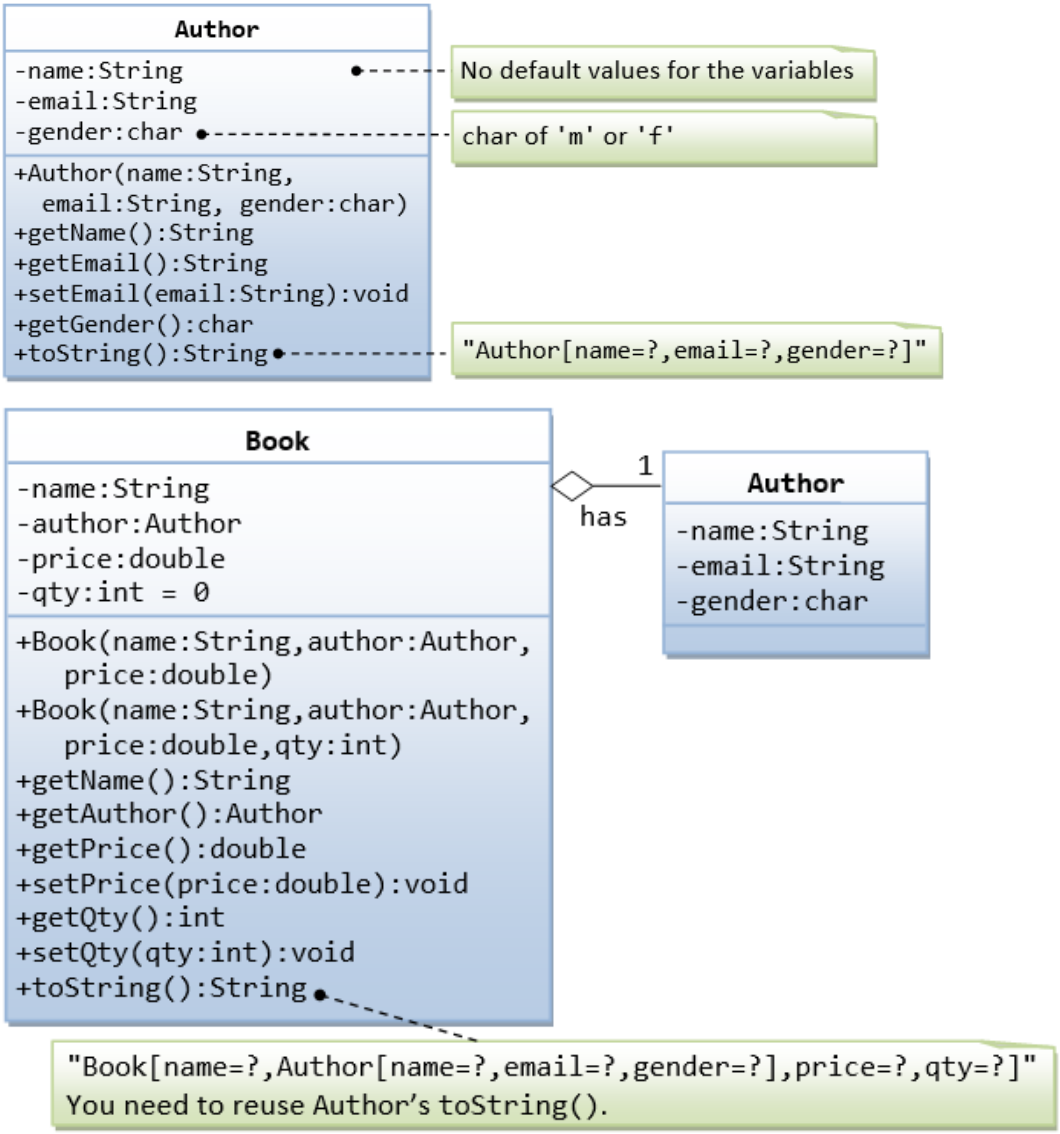
The expected output is:

```
Ball[(1.1,2.2),speed=(3.3,4.4)]
Ball[(80.0,35.0),speed=(4.0,6.0)]
x is: 80.0
y is: 35.0
radius is: 5
xDelta is: 4.0
yDelta is: 6.0
Ball[(84.0,41.0),speed=(4.0,6.0)]
Ball[(88.0,47.0),speed=(4.0,6.0)]
Ball[(92.0,41.0),speed=(4.0,-6.0)]
Ball[(96.0,35.0),speed=(4.0,-6.0)]
Ball[(92.0,29.0),speed=(-4.0,-6.0)]
Ball[(88.0,23.0),speed=(-4.0,-6.0)]
Ball[(84.0,17.0),speed=(-4.0,-6.0)]
Ball[(80.0,11.0),speed=(-4.0,-6.0)]
Ball[(76.0,5.0),speed=(-4.0,-6.0)]
Ball[(72.0,-1.0),speed=(-4.0,-6.0)]
Ball[(68.0,5.0),speed=(-4.0,6.0)]
Ball[(64.0,11.0),speed=(-4.0,6.0)]
Ball[(60.0,17.0),speed=(-4.0,6.0)]
Ball[(56.0,23.0),speed=(-4.0,6.0)]
Ball[(52.0,29.0),speed=(-4.0,6.0)]
```

6. Exercises on Composition

6.1 The Author and Book Classes

This first exercise shall lead you through all the concepts involved in OOP Composition.

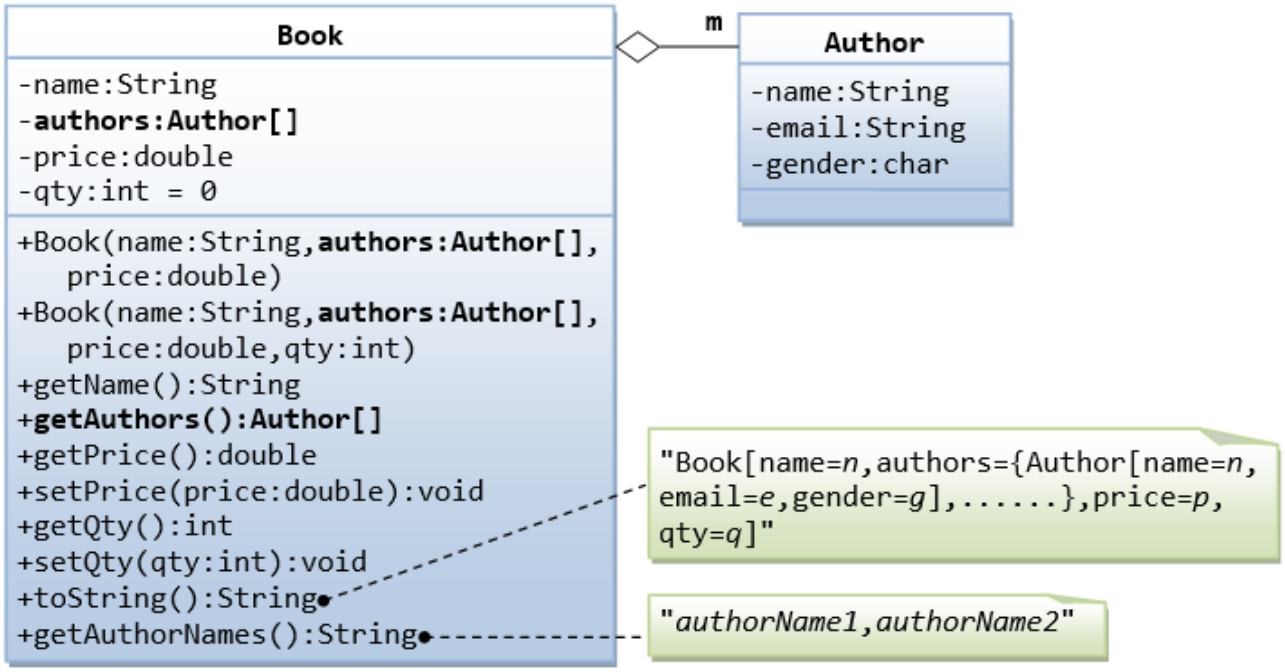


TRY

1. Printing the `name` and `email` of the author from a `Book` instance.
   (Hint: `aBook.getAuthor().getName()`, `aBook.getAuthor().getEmail()`).
2. Introduce new methods called `getAuthorName()`, `getAuthorEmail()`, `getAuthorGender()` in the `Book` class to return the `name`, `email` and `gender` of the author of the book. For example,

```
public String getAuthorName() {
   return author.getName();   // cannot use author.name as name is private in Author class
}
```

6.2 The Author and Book Classes - An Array of Objects as an Instance Variable



In the earlier exercise, a book is written by one and only one author. In reality, a book can be written by one or more author. Modify the Book class to support one or more authors by changing the instance variable `authors` to an `Author` array.

Notes:

- The constructors take an array of `Author` (i.e., `Author[]`), instead of an `Author` instance. In this design, once a Book instance is constructor, you cannot add or remove author.
- The `toString()` method shall return **`"Book[name=?,authors={Author[name=?,email=?,gender=?],.....},price=?,qty=?]"`.**

You are required to:

1. Write the code for the Book class. You shall re-use the `Author` class written earlier.
2. Write a test driver (called `TestBook`) to test the Book class.

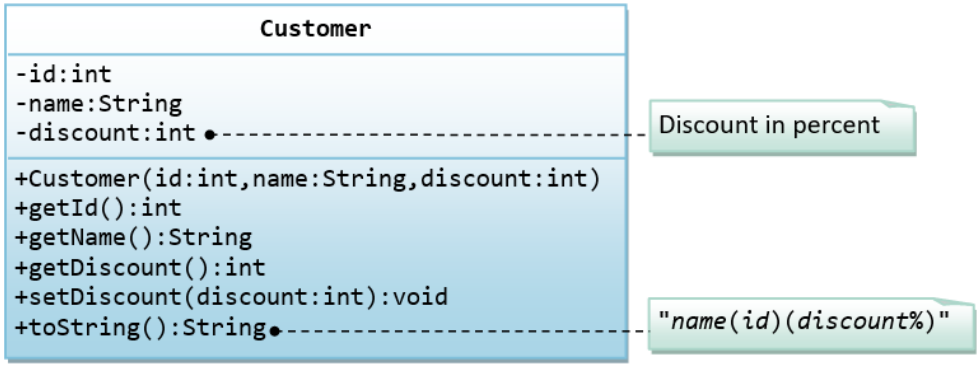
Hints

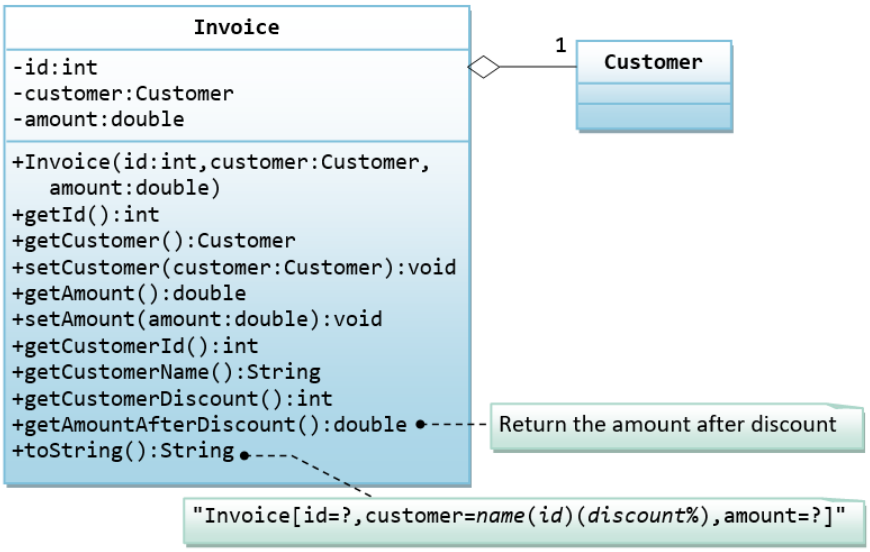
```
// Declare and allocate an array of Authors
Author[] authors = new Author[2];
authors[0] = new Author("Tan Ah Teck", "AhTeck@somewhere.com", 'm');
authors[1] = new Author("Paul Tan", "Paul@nowhere.com", 'm');

// Declare and allocate a Book instance
Book javaDummy = new Book("Java for Dummy", authors, 19.99, 99);
System.out.println(javaDummy);  // toString()
```

6.3 The `Customer` and `Invoice` classes

A class called `Customer`, which models a customer in a transaction, is designed as shown in the class diagram. A class called `Invoice`, which models an invoice for a particular customer and composes an instance of `Customer` as its instance variable, is also shown. Write the `Customer` and `Invoice` classes.

Hints:

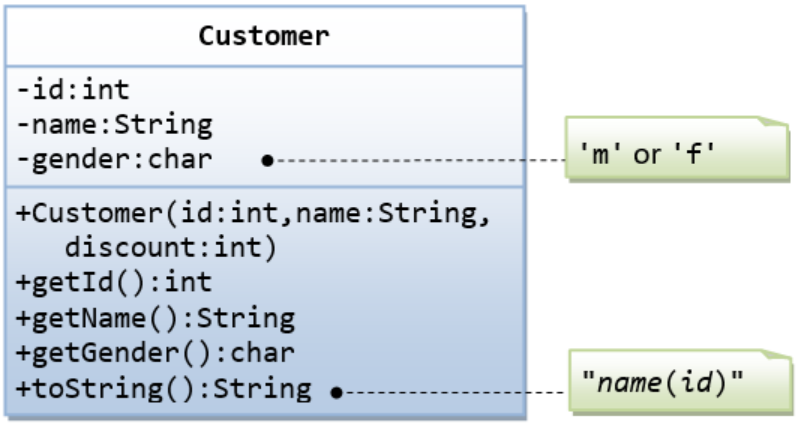




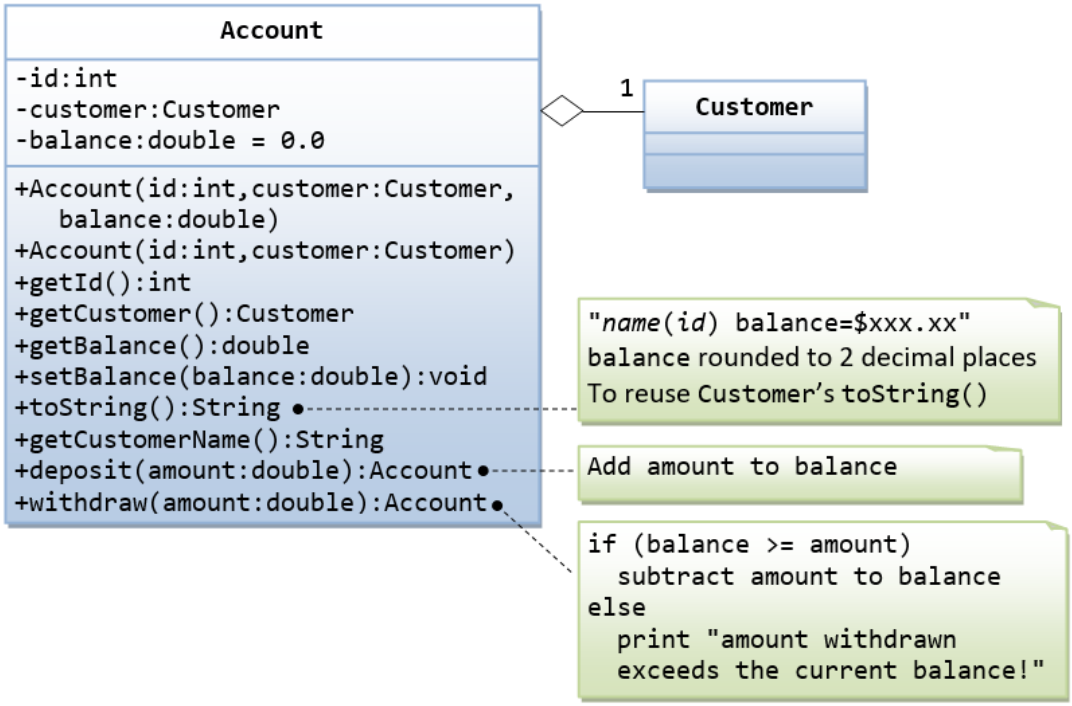
The expected output is:

```
Tan Ah Teck(88)(10%)
Tan Ah Teck(88)(8%)
id is: 88
name is: Tan Ah Teck
discount is: 8
Invoice[id=101,customer=Tan Ah Teck(88)(8%),amount=888.8]
Invoice[id=101,customer=Tan Ah Teck(88)(8%),amount=999.9]
id is: 101
customer is: Tan Ah Teck(88)(8%)
amount is: 999.9
customer's id is: 88
customer's name is: Tan Ah Teck
customer's discount is: 8
amount after discount is: 919.91
```

6.4 Ex: The `Customer` and `Account` classes



The `Customer` class models a customer is design as shown in the class diagram. Write the codes for the `Customer` class and a test driver to test all the `public` methods.



The `Account` class models a bank account, design as shown in the class diagram, composes a `Customer` instance (written earlier) as its member. Write the codes for the `Account` class and a test driver to test all the `public` methods.

It contains:

- A method called `distance(int x, int y)` that returns the distance from *this* point to another point at the given `(x, y)` coordinates, e.g.,

```
MyPoint p1 = new MyPoint(3, 4);
```

```
System.out.println(p1.distance(5, 6));
```

- An overloaded `distance(MyPoint another)` that returns the distance from *this* point to the given `MyPoint` instance (called `another`), e.g.,

```
MyPoint p1 = new MyPoint(3, 4);
MyPoint p2 = new MyPoint(5, 6);
System.out.println(p1.distance(p2));
```

- Another overloaded `distance()` method that returns the distance from `this` point to the origin `(0,0)`, e.g.

```
MyPoint p1 = new MyPoint(3, 4);
System.out.println(p1.distance());
```

You are required to:

1. Write the code for the class `MyPoint`. Also write a test program (called `TestMyPoint`) to test all the methods defined in the class,

Hints:

```
// Overloading method distance()
// This version takes two ints as arguments
public double distance(int x, int y) {
   int xDiff = this.x - x;
   int yDiff = ......
   return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
}

// This version takes a MyPoint instance as argument
public double distance(MyPoint another) {
   int xDiff = this.x - another.x;
   .......
}
```

Try

Write a program that allocates `10` points in an array of `MyPoint`, and initializes to `(1, 1)`, `(2, 2)`, ... `(10, 10)`.

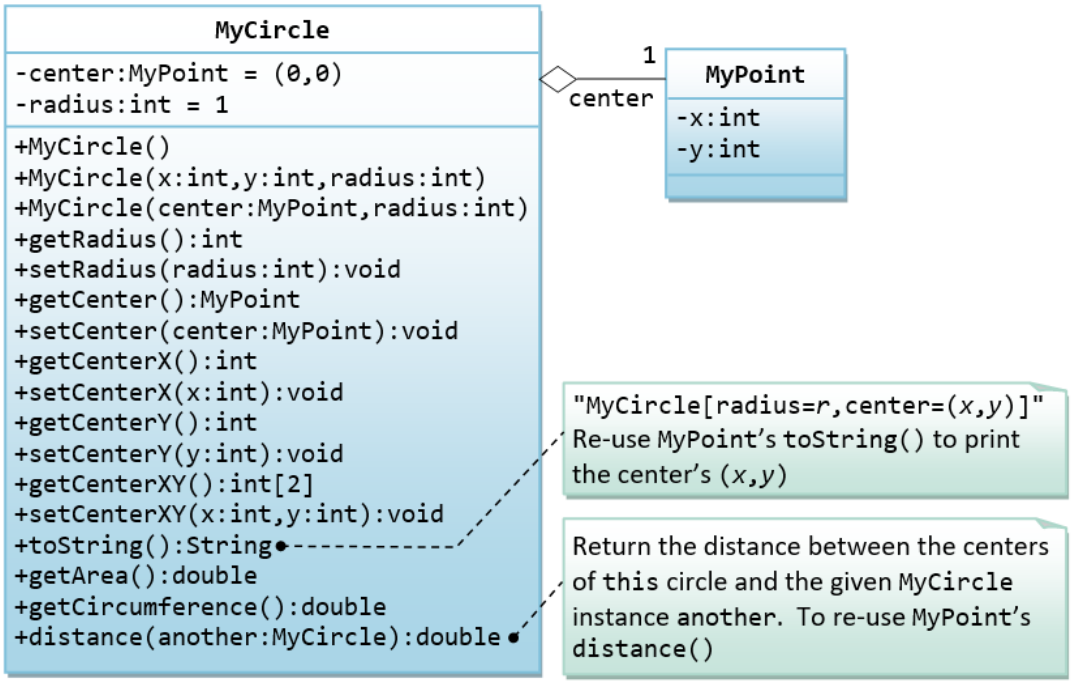
Hints

You need to allocate the array, as well as each of the 10 `MyPoint` instances. In other words, you need to issue 11 `new`, 1 for the array and 10 for the `MyPoint` instances.

```
MyPoint[] points = new MyPoint[10];  // Declare and allocate an array of MyPoint
for (int i = 0; i < points.length; i++) {
   points[i] = new MyPoint(...);    // Allocate each of MyPoint instances
}
// use a loop to print all the points
```

6.5 Ex: The `MyCircle` and `MyPoint` Classes

A class called `MyCircle`, which models a circle with a center and a radius, is designed as shown in the class diagram. The `MyCircle` class uses a `MyPoint` instance (written in the earlier exercise) as its center.

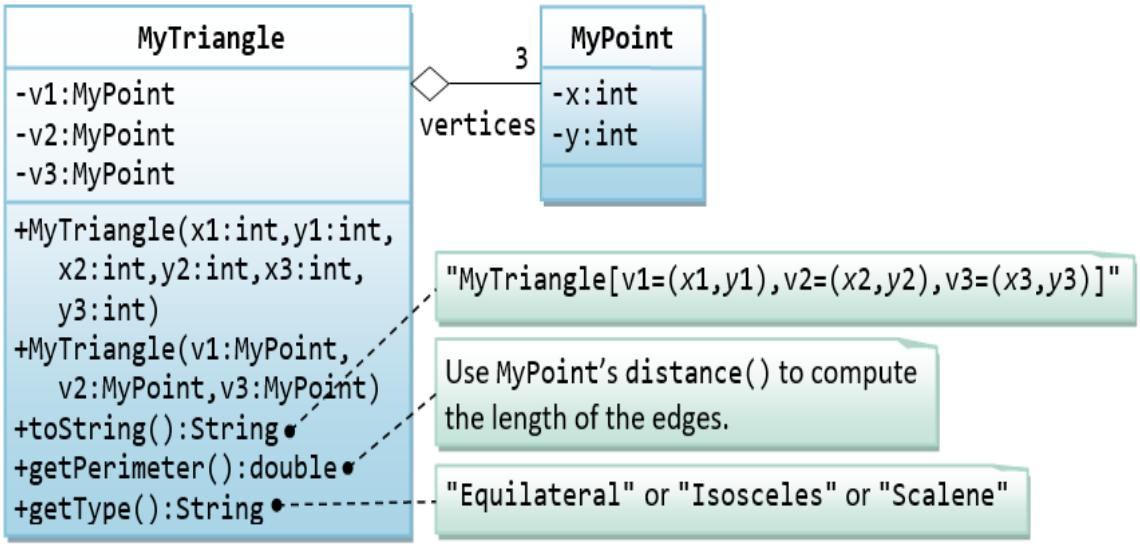


Hints:

```
// Constructors
public MyCircle(int x, int y, int radius) {
   // Need to construct an instance of MyPoint for the variable center
   center = new MyPoint(x, y);
   this.radius = radius;
}
public MyCircle(MyPoint center, int radius) {
   // An instance of MyPoint already constructed by caller; simply assign.
   this.center = center;
   ......
}
public MyCircle() {
   center = new MyPoint(.....);  // construct MyPoint instance
   this.radius = ......
}

// Returns the x-coordinate of the center of this MyCircle
public int getCenterX() {
   return center.getX();   // cannot use center.x and x is private in MyPoint
}

// Returns the distance of the center for this MyCircle and another MyCircle
public double distance(MyCircle another) {
   return center.distance(another.center); // use distance() of MyPoint
}
```

6.6 Ex: The `MyTriangle` and `MyPoint` Classes

A class called `MyTriangle`, which models a triangle with 3 vertices, is designed as shown in the class diagram. The `MyTriangle` class uses three `MyPoint` instances (created in the earlier exercise) as the three vertices.



It contains:

- Three `private` instance variables `v1`, `v2`, `v3` (instances of `MyPoint`), for the three vertices.
- A constructor that constructs a `MyTriangle` with three set of coordinates, `v1=(x1, y1)`, `v2=(x2, y2)`, `v3=(x3, y3)`.
- An overloaded constructor that constructs a `MyTriangle` given three instances of `MyPoint`.
- A `toString()` method that returns a string description of the instance in the format "`MyTriangle[v1=(x1,y1),v2=(x2,y2),v3=(x3,y3)]`".
- A `getPerimeter()` method that returns the length of the perimeter in double. You should use the `distance()` method of `MyPoint` to compute the perimeter.
- A method `printType()`, which prints "`equilateral`" if all the three sides are equal, "`isosceles`" if any two of the three sides are equal, or "`scalene`" if the three sides are different.

Write the `MyTriangle` class. Also write a test driver (called `TestMyTriangle`) to test all the `public` methods defined in the class.

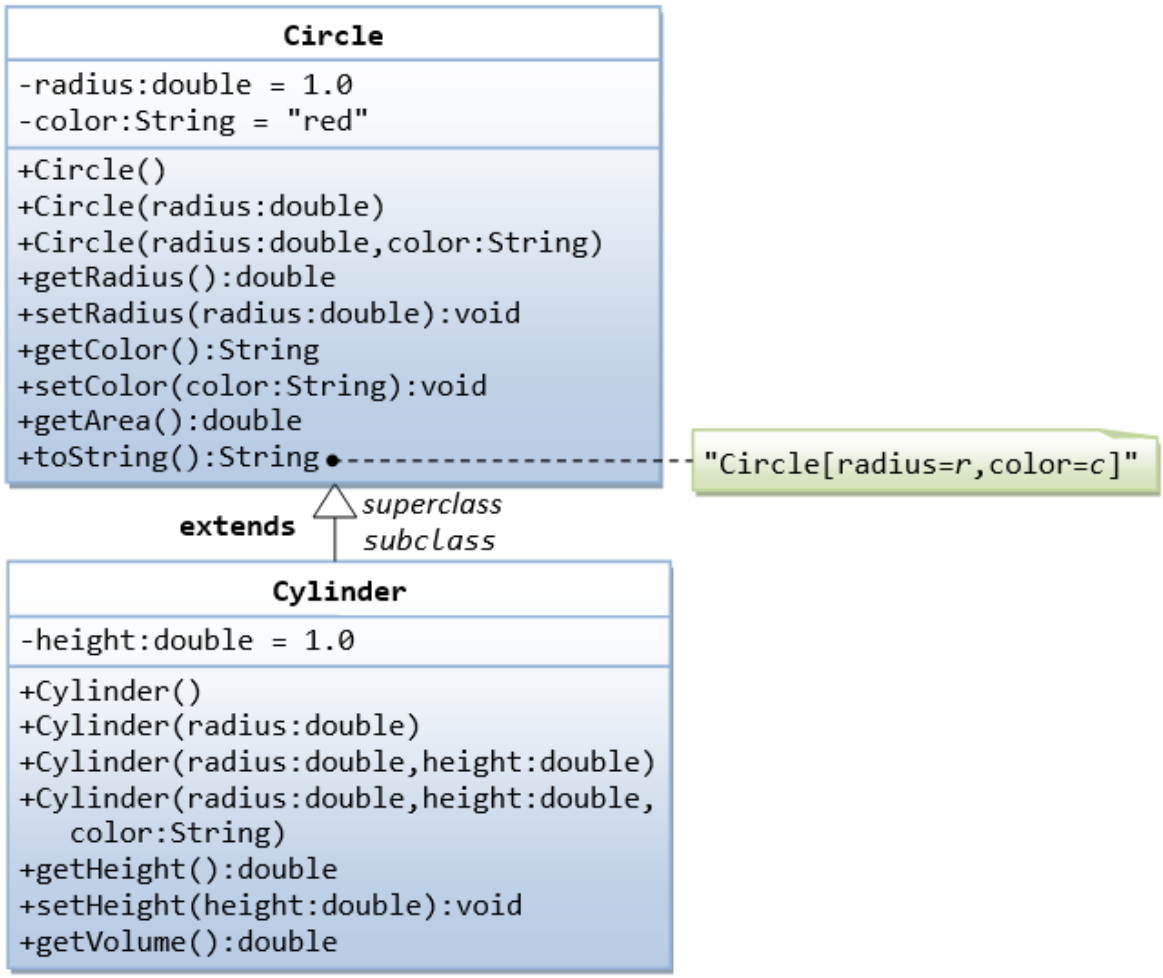
Try

Design a `MyRectangle` class which is composed of two `MyPoint` instances as its *top-left* and *bottom-right* corners. Draw the class diagrams, write the codes, and write the test drivers.

7. Exercises on Inheritance

7.1 An Introduction to OOP Inheritance: The `Circle` and `Cylinder` Classes

This exercise shall guide you through the important concepts in inheritance.



In this exercise, a subclass called `Cylinder` is derived from the superclass `Circle` as shown in the class diagram (where an an arrow pointing up from the subclass to its superclass). Study how the subclass `Cylinder` invokes the superclass' constructors (via `super()` and `super(radius)`) and inherits the variables and methods from the superclass `Circle`.

You can reuse the `Circle` class that you have created in the previous exercise. Make sure that you keep "`Circle.class`" in the same directory.

```
public class Cylinder extends Circle {  // Save as "Cylinder.java"
   private double height;  // private variable

   // Constructor with default color, radius and height
   public Cylinder() {
      super();        // call superclass no-arg constructor Circle()
      height = 1.0;
   }
   // Constructor with default radius, color but given height
   public Cylinder(double height) {
      super();        // call superclass no-arg constructor Circle()
```

```
      this.height = height;
   }
   // Constructor with default color, but given radius, height
   public Cylinder(double radius, double height) {
      super(radius);  // call superclass constructor Circle(r)
      this.height = height;
   }

   // A public method for retrieving the height
   public double getHeight() {
      return height;
   }

   // A public method for computing the volume of cylinder
   //  use superclass method getArea() to get the base area
   public double getVolume() {
      return getArea()*height;
   }
}
```

Method Overriding and "Super": The subclass `Cylinder` inherits `getArea()` method from its superclass Circle. Try *overriding* the `getArea()` method in the subclass `Cylinder` to compute the surface area (=2π×radius×height + 2×base-area) of the cylinder instead of base area. That is, if `getArea()` is called by a `Circle` instance, it returns the area. If `getArea()` is called by a `Cylinder` instance, it returns the surface area of the cylinder.

If you override the `getArea()` in the subclass `Cylinder`, the `getVolume()` no longer works. This is because the `getVolume()` uses the *overridden* `getArea()` method found in the same class. (Java runtime will search the superclass only if it cannot locate the method in this class). Fix the `getVolume()`.

Hints: After overridding the `getArea()` in subclass `Cylinder`, you can choose to invoke the `getArea()` of the superclass `Circle` by calling `super.getArea()`.

Try

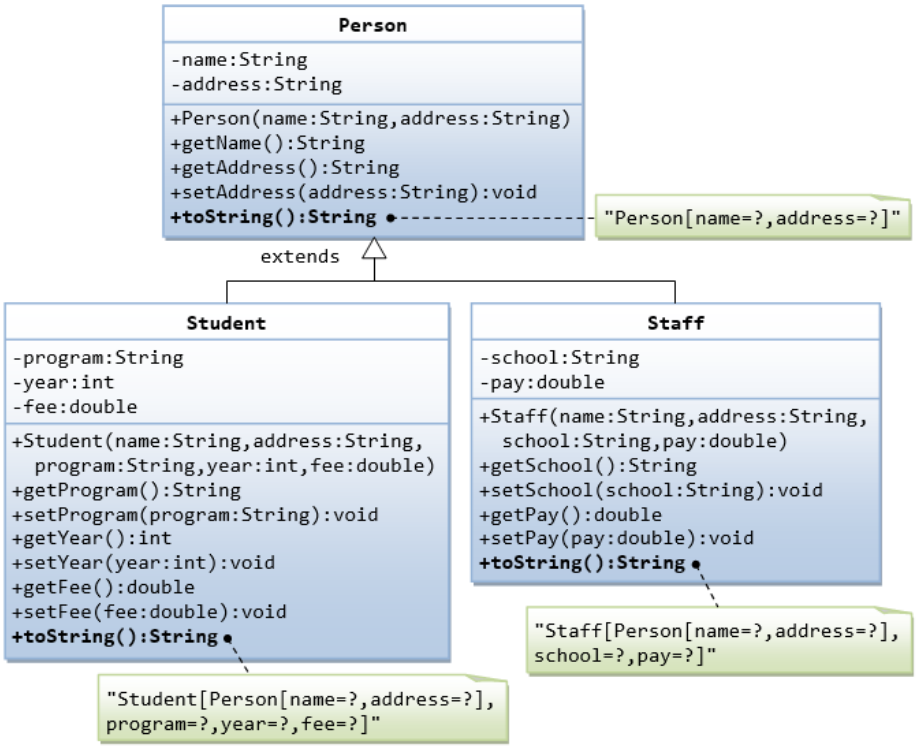
Provide a `toString()` method to the `Cylinder` class, which overrides the `toString()` inherited from the superclass `Circle`, e.g.,

```
@Override
public String toString() {      // in Cylinder class
   return "Cylinder: subclass of " + super.toString()  // use Circle's toString()
          + " height=" + height;
}
```

Try out the `toString()` method in `TestCylinder`.

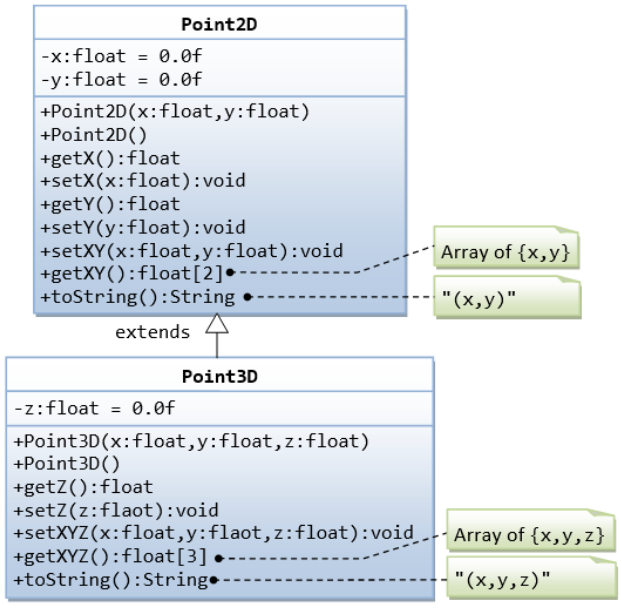
7.2 Superclass `Person` and its subclasses

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation `@Override`.



7.3 `Point2D` and `Point3D`

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation `@Override`.



Hints:

1. You cannot assign floating-point literal say `1.1` (which is a `double`) to a `float` variable, you need to add a suffix f, e.g. `0.0f`, `1.1f`.
2. The instance variables x and y are `private` in `Point2D` and cannot be accessed directly in the subclass `Point3D`. You need to access via the `public` getters and setters. For example,

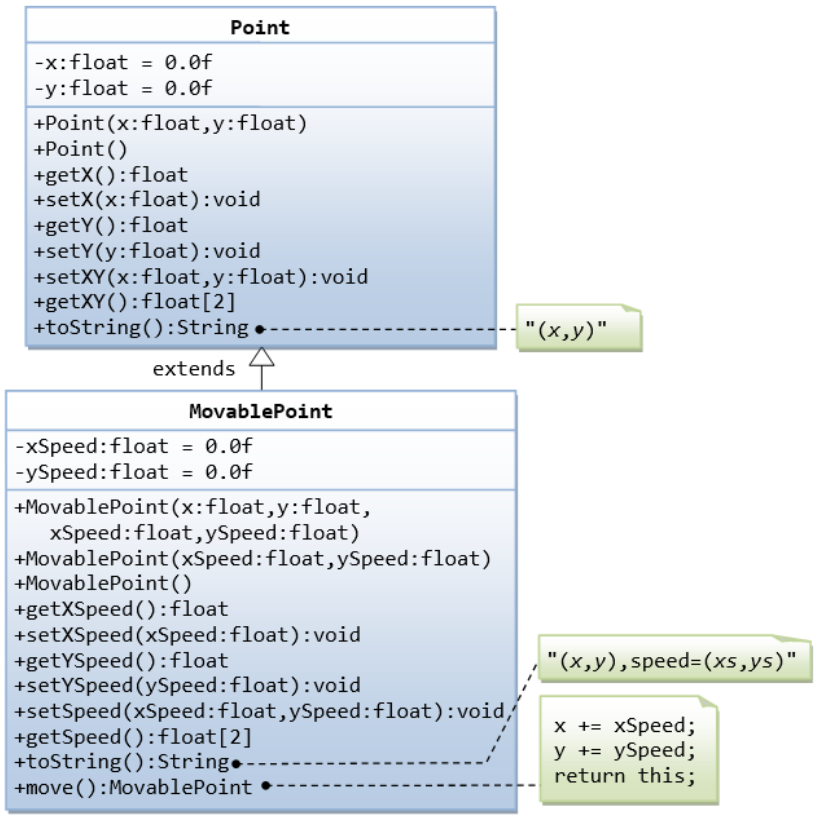
```
public void setXYZ(float x, float y, float z) {
       setX(x);     // or super.setX(x), use setter in superclass
       setY(y);
       this.z = z;
    }
```

3. The method `getXY()` shall return a `float` array:

```
public float[] getXY() {
       float[] result = new float[2];  // construct an array of 2 elements
       result[0] = ...
       result[1] = ...
       return result;  // return the array
    }
```

7.4 `Point` and `MovablePoint`

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation `@Override`.



Hints

1. You cannot assign floating-point literal say `1.1` (which is a `double`) to a `float` variable, you need to add a suffix f, e.g. `0.0f`, `1.1f`.
2. The instance variables x and y are `private` in `Point` and cannot be accessed directly in the subclass `MovablePoint`. You need to access via the `public` getters and setters. For example, you cannot write `x += xSpeed`, you need to write `setX(getX() + xSpeed)`.

7.5 Superclass `Shape` and its subclasses `Circle`, `Rectangle` and `Square`

- Write a superclass called Shape (as shown in the class diagram)
- Write a test program to test all the methods defined in Shape.
- Write two subclasses of Shape called `Circle` and `Rectangle`, as shown in the class diagram.
- Write a class called Square, as a subclass of `Rectangle`. Convince yourself that Square can be modeled as a subclass of `Rectangle`. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

Provide the appropriate constructors (as shown in the class diagram).

Hints:

```
public Square(double side) {
   super(side, side);  // Call superclass Rectangle(double, double)
}
```

- Override the `toString()` method to return "`A Square with side=xxx, which is a subclass of yyy`", where yyy is the output of the `toString()` method from the superclass.
- Do you need to override the `getArea()` and `getPerimeter()`? Try them out.
- Override the `setLength()` and `setWidth()` to change both the `width` and `length`, so as to maintain the square geometry.

Exercises on Composition vs Inheritance

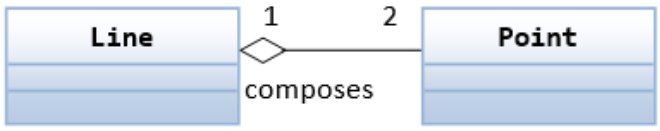
7.6 The `Point` and `Line` Classes

They are two ways to reuse a class in your applications: *composition* and *inheritance*.

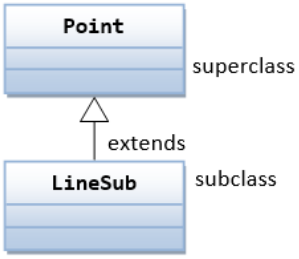
Let us begin with *composition* with the statement "a line composes of two points".

Complete the definition of the following two classes: `Point` and `Line`. The class `Line` composes 2 instances of class `Point`, representing the beginning and ending points of the line. Also write test classes for `Point` and `Line` (says `TestPoint` and `TestLine`).

The class diagram for *composition* is as follows (where a diamond-hollow-head arrow pointing to its constituents):



Instead of *composition*, we can design a `Line` class using `inheritance`. Instead of "a line composes of two points", we can say that "a line is a point extended by another point", as shown in the following class diagram:



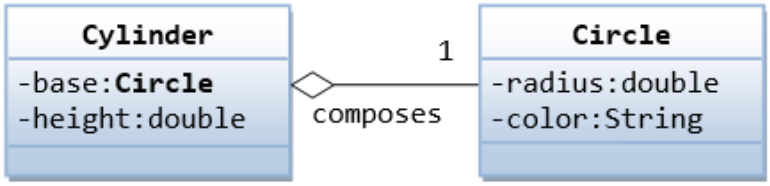
Let's re-design the `Line` class (called `LineSub`) as a subclass of class `Point`. `LineSub` inherits the starting point

from its superclass `Point`, and adds an ending point. Complete the class definition. Write a testing class called `TestLineSub` to test `LineSub`.

Try

There are two approaches that you can design a line, `composition` or `inheritance`. "A line composes two points" or "A line is a point extended with another point"".
Compare the `Line` and `LineSub` designs: `Line` uses *composition* and `LineSub` uses *inheritance*. Which design is better?

7.7 The `Circle` and `Cylinder` Classes Using Composition



Try

rewriting the `Circle-Cylinder` of the previous exercise using *composition* (as shown in the class diagram) instead of *inheritance*. That is, "a cylinder is composed of a base circle and a height".

```
public class Cylinder {
   private Circle base;   // Base circle, an instance of Circle class
   private double height;

   // Constructor with default color, radius and height
   public Cylinder() {
      base = new Circle(); // Call the constructor to construct the Circle
      height = 1.0;
   }
   ......
}
```

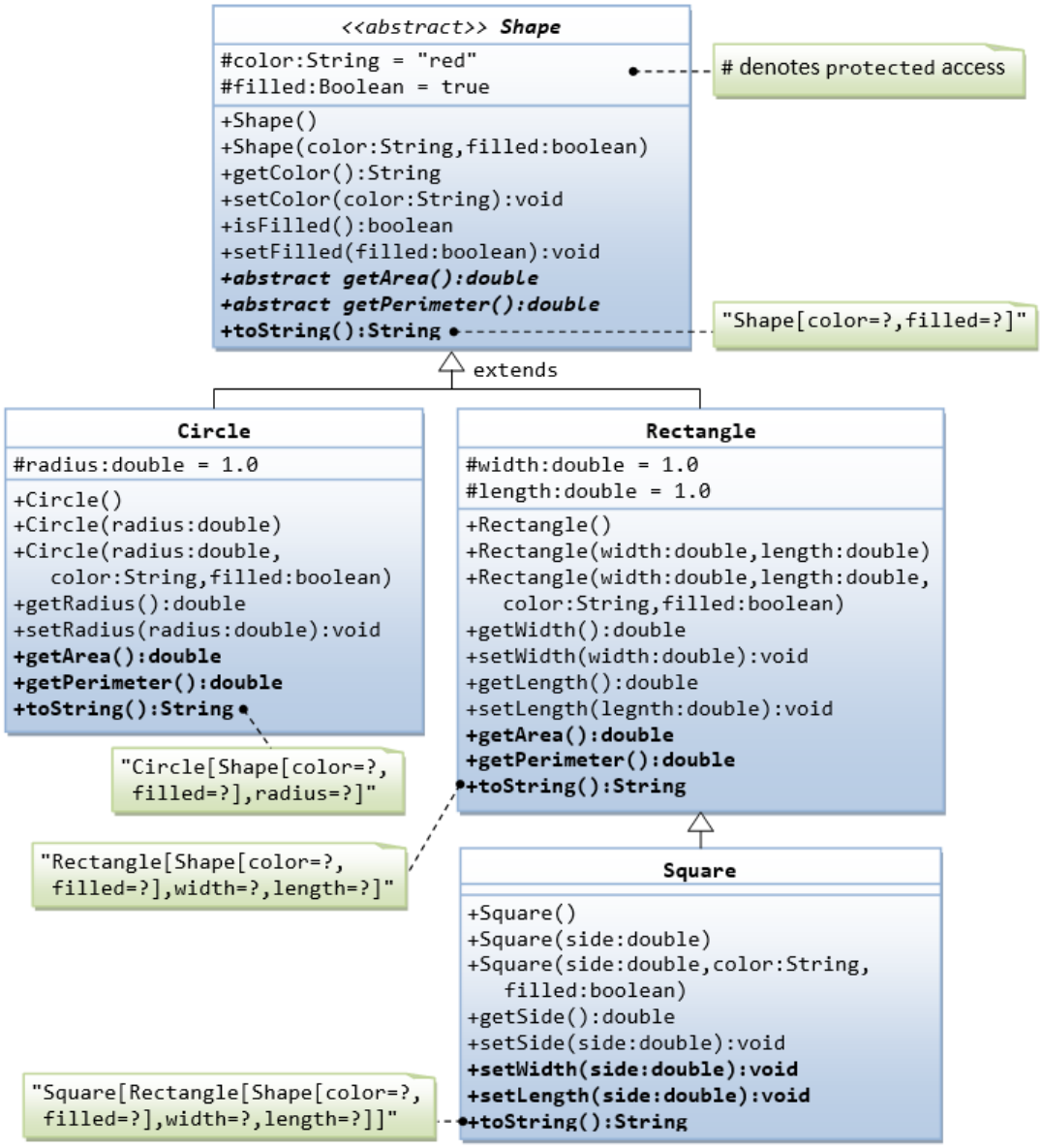
Which design (inheritance or composition) is better?

8. Exercises on Polymorphism, Abstract Classes and Interfaces

8.1 Ex: Abstract Superclass Shape and Its Concrete Subclasses

Rewrite the superclass Shape and its subclasses `Circle`, `Rectangle` and `Square`, as shown in the class diagram.

Shape is an `abstract` class containing 2 `abstract` methods: `getArea()` and `getPerimeter()`, where its concrete subclasses must provide its implementation. All instance variables shall have `protected` access, i.e., accessible by its subclasses and classes in the same package. Mark all the overridden methods with annotation `@Override`.



In this exercise, Shape shall be defined as an `abstract` class, which contains:

- Two protected instance variables color(String) and filled(boolean). The `protected` variables can be accessed by its subclasses and classes in the same package. They are denoted with a `'#'` sign in the class diagram.
- Getter and setter for all the instance variables, and `toString()`.
- Two `abstract` methods `getArea()` and `getPerimeter()` (shown in italics in the class diagram).

The subclasses `Circle` and `Rectangle` shall *override* the `abstract` methods `getArea()` and `getPerimeter()` and provide the proper implementation. They also *override* the `toString()`.

Write a test class to test these statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors, if any.

```
Shape s1 = new Circle(5.5, "red", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
```

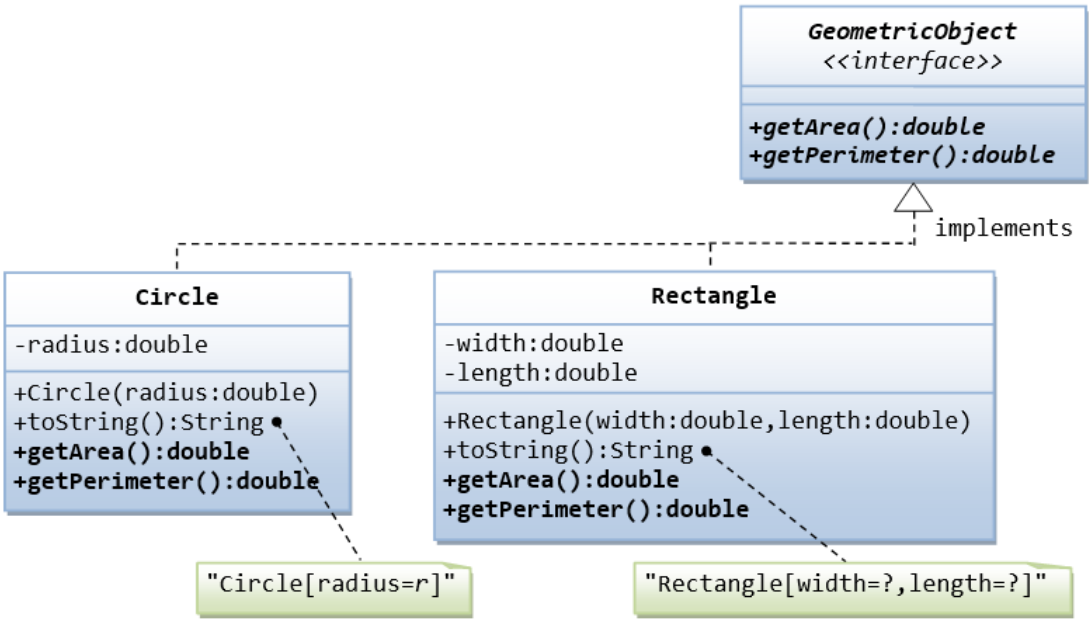
```
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                   // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "red", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Try

Explain the usage of the `abstract` method and `abstract` class?

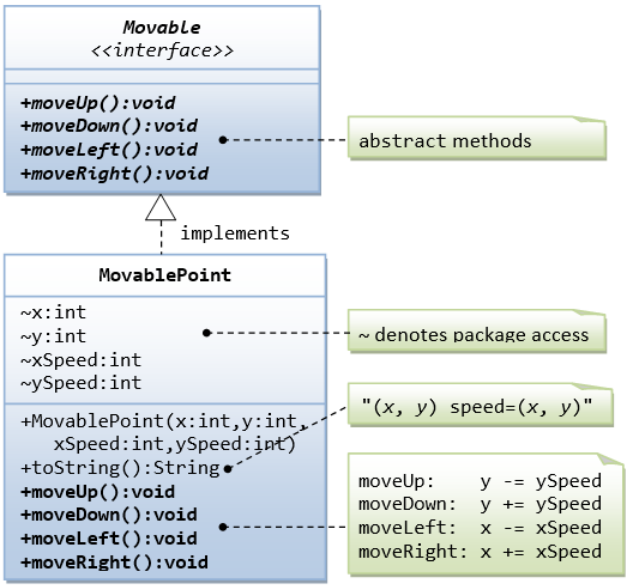
8.2 `GeometricObject` Interface and its Implementation Classes `Circle` and `Rectangle`

Write an interface called `GeometricObject`, which contains 2 abstract methods: getArea() and `getPerimeter()`, as shown in the class diagram. Also write an implementation class called `Circle`. Mark all the overridden methods with annotation `@Override`.



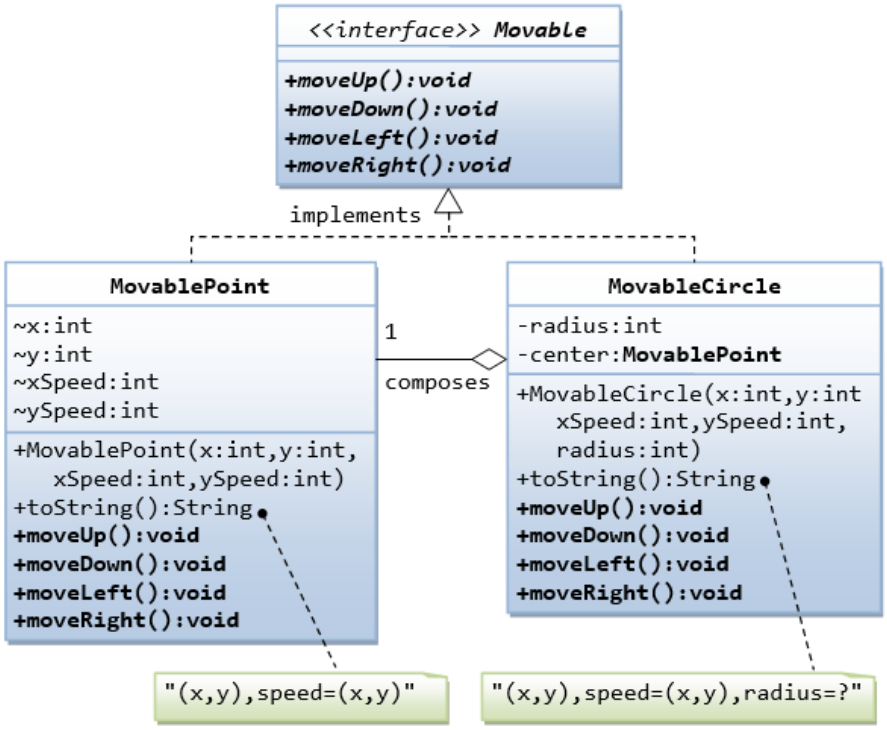
8.3 Ex: `Movable` Interface and its Implementation `MovablePoint` Class

Write an interface called `Movaable`, which contains 4 abstract methods moveUp(), moveDown(), moveLeft() and `moveRight()`, as shown in the class diagram. Also write an implementation class called `MovablePoint`. Mark all the overridden methods with annotation `@Override`.

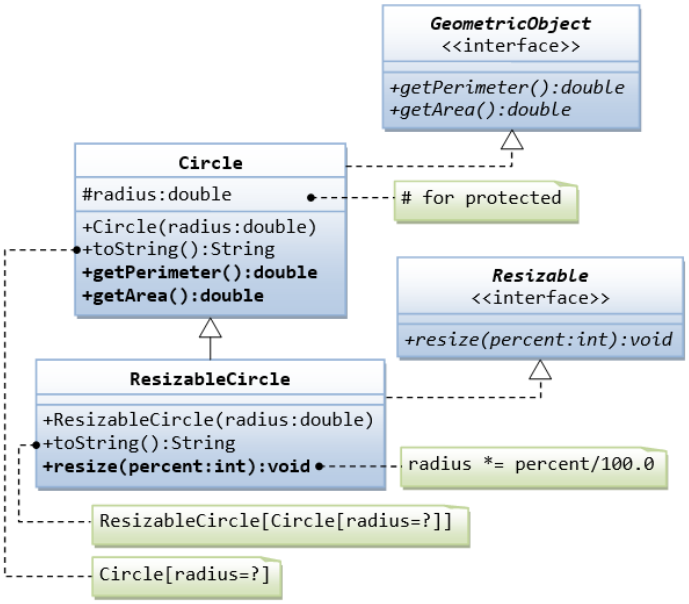


8.4 Movable Interface and Classes MovablePoint and MovableCircle

Write an interface called `Movable`, which contains 4 abstract methods `moveUp()`, `moveDown()`, `moveLeft()` and `moveRight()`, as shown in the class diagram. Also write the implementation classes called `MovablePoint` and `MovableCircle`. Mark all the overridden methods with annotation `@Override`.



8.5 Interfaces Resizable and GeometricObject



Write the `interface` called `GeometricObject`, which declares two abstract methods: getParameter() and `getArea()`, as specified in the class diagram.

Hints:

```
public interface GeometricObject {
   public double getPerimeter();
   ......
}
```

Write the implementation class `Circle`, with a protected variable `radius`, which implements the interface `GeometricObject`.

Hints:

```
public class Circle implements GeometricObject {
   // Private variable
   ......

   // Constructor
   ......

   // Implement methods defined in the interface GeometricObject
   @Override
   public double getPerimeter() { ...... }

   ......
}
```

Write a test program called `TestCircle` to test the methods defined in `Circle`.

The class `ResizableCircle` is defined as a subclass of the class `Circle`, which also implements an interface called `Resizable`, as shown in class diagram. The interface `Resizable` declares an `abstract` method `resize()`, which modifies the dimension (such as `radius`) by the given percentage. Write the interface `Resizable` and the class `ResizableCircle`.

Hints:

```
public interface Resizable {
   public double resize(...);
}
```

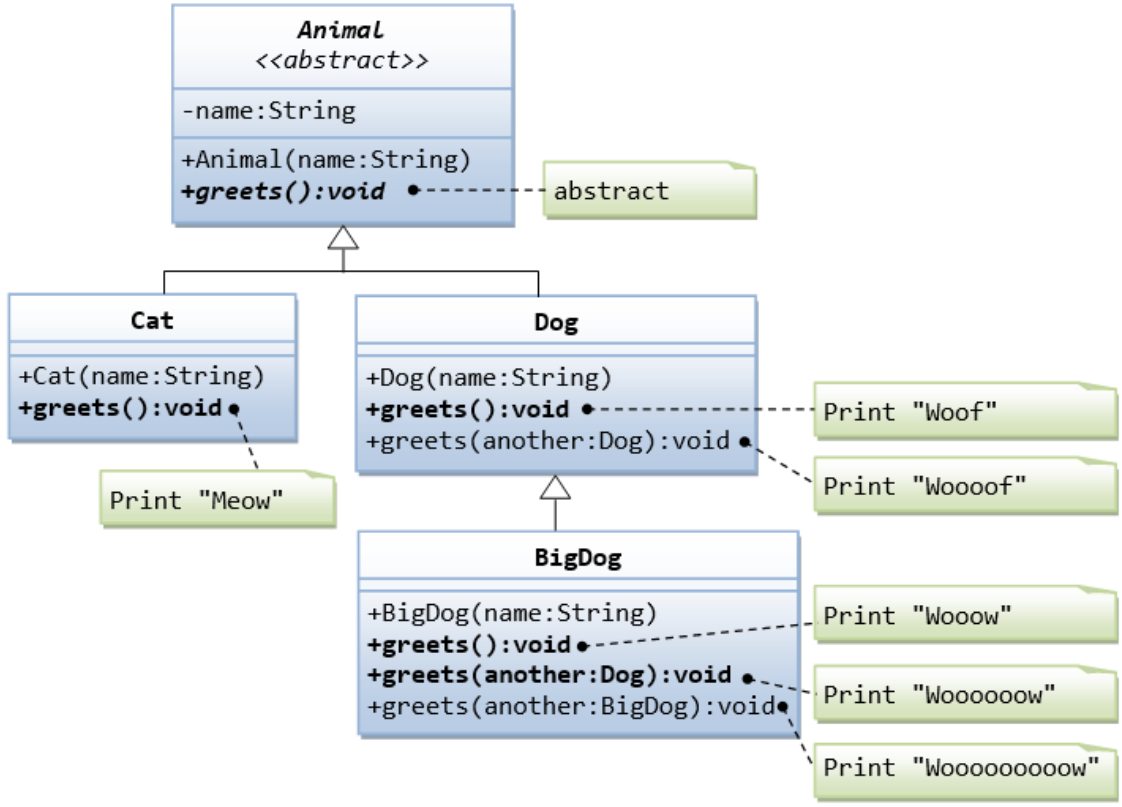
```
public class ResizableCircle extends Circle implements Resizeable {

   // Constructor
   public ResizableCircle(double radius) {
      super(...);
   }

   // Implement methods defined in the interface Resizable
   @Override
   public double resize(int percent) { ...... }
}
```

Try

Write a test program called `TestResizableCircle` to test the methods defined in `ResizableCircle`.

8.6 Abstract Superclass `Animal` and its Implementation Subclasses

Write the codes for all the classes shown in the class diagram. Mark all the overridden methods with annotation @Override.



8.7 Another View of Abstract Superclass `Animal` and its Implementation Subclasses

Examine the following codes and draw the class diagram.

```
abstract public class Animal {
   abstract public void greeting();
}
```

```
public class Cat extends Animal {
   @Override
   public void greeting() {
      System.out.println("Meow!");
   }
}
```

```
public class Dog extends Animal {
   @Override
   public void greeting() {
      System.out.println("Woof!");
   }

   public void greeting(Dog another) {
      System.out.println("Wooooooooooof!");
   }
}
```

```
public class BigDog extends Dog {
   @Override
   public void greeting() {
      System.out.println("Woow!");
   }

   @Override
   public void greeting(Dog another) {
      System.out.println("Wooooooowwwww!");
   }
}
```

Try

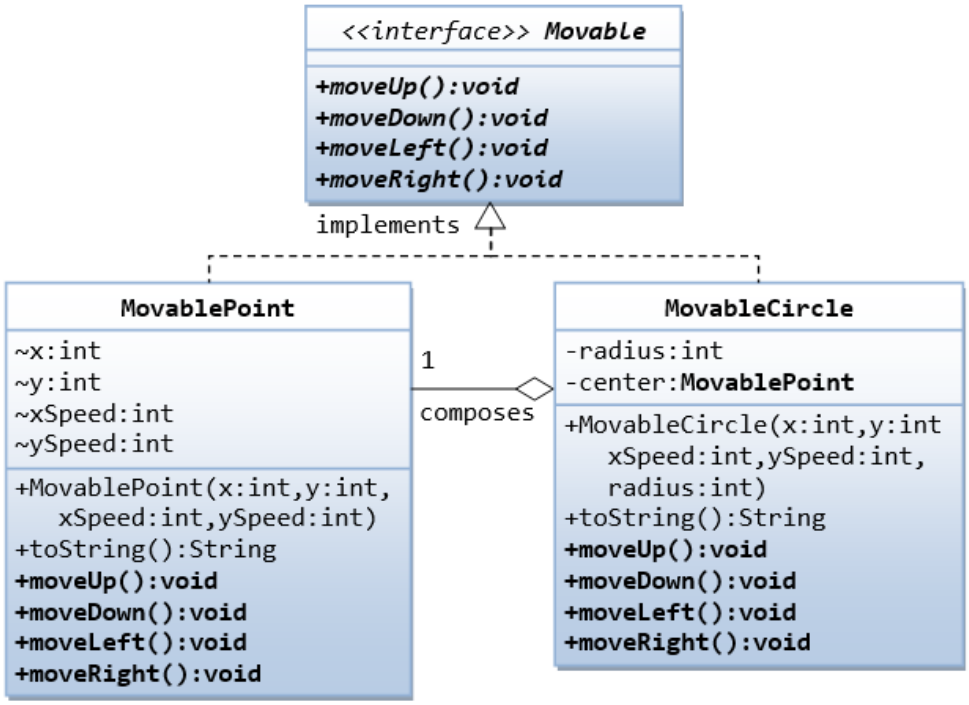
Explain the outputs (or error) for the following test program.

```
public class TestAnimal {
   public static void main(String[] args) {
      // Using the subclasses
      Cat cat1 = new Cat();
      cat1.greeting();
      Dog dog1 = new Dog();
      dog1.greeting();
      BigDog bigDog1 = new BigDog();
      bigDog1.greeting();

      // Using Polymorphism
      Animal animal1 = new Cat();
      animal1.greeting();
      Animal animal2 = new Dog();
      animal2.greeting();
      Animal animal3 = new BigDog();
      animal3.greeting();
      Animal animal4 = new Animal();

      // Downcast
      Dog dog2 = (Dog)animal2;
      BigDog bigDog2 = (BigDog)animal3;
      Dog dog3 = (Dog)animal3;
      Cat cat2 = (Cat)animal2;
      dog2.greeting(dog3);
      dog3.greeting(dog2);
      dog2.greeting(bigDog2);
      bigDog2.greeting(dog2);
      bigDog2.greeting(bigDog1);
   }
}
```

8.8 Interface `Movable` and its subclasses MovablePoint & MovableCircle

Suppose that we have a set of objects with some common behaviors: they could move up, down, left or right. The exact behaviors (such as how to move and how far to move) depend on the objects themselves. One common way to model these common behaviors is to define an *interface* called `Movable`, with `abstract` methods `moveUp()`, `moveDown()`, `moveLeft()` and `moveRight()`. The classes that implement the `Movable` interface will provide actual implementation to these `abstract` methods.

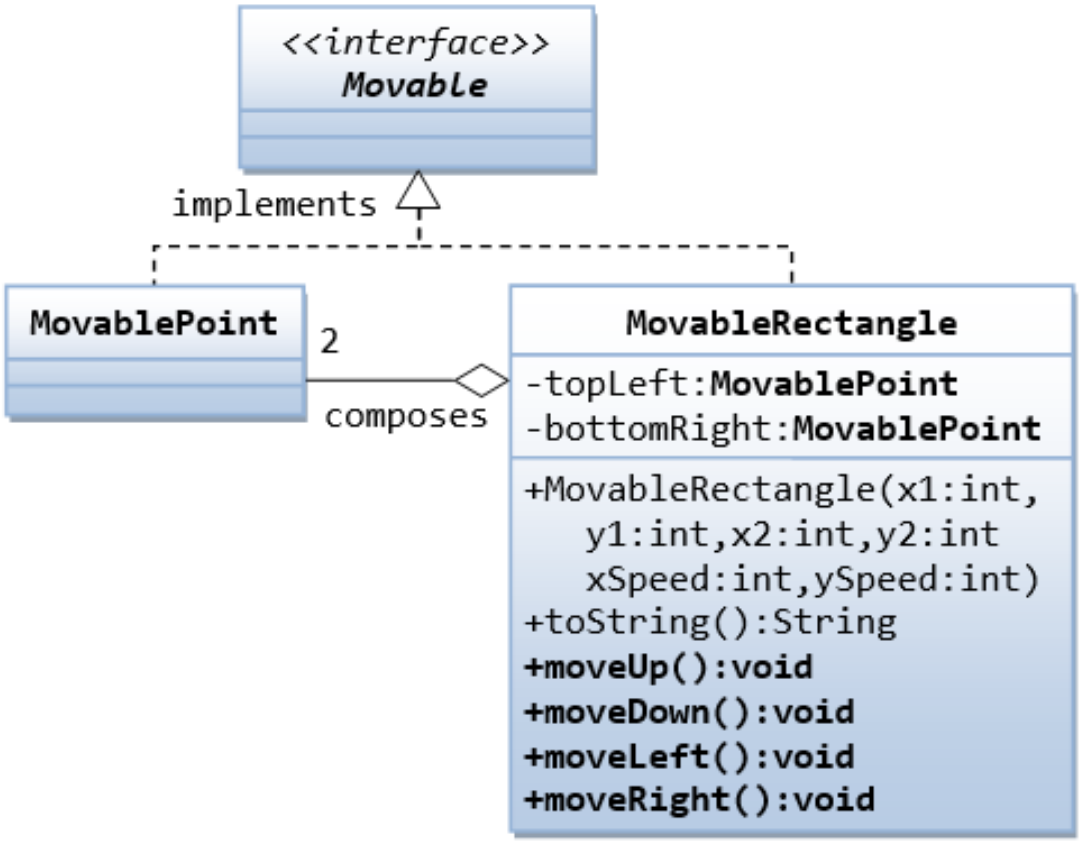
Let's write two concrete classes - `MovablePoint` and `MovableCircle` - that implement the Movable interface.



Write a test program and try out these statements:

```
Movable m1 = new MovablePoint(5, 6, 10, 15);     // upcast
System.out.println(m1);
m1.moveLeft();
System.out.println(m1);

Movable m2 = new MovableCircle(1, 2, 3, 4, 20);  // upcast
System.out.println(m2);
m2.moveRight();
System.out.println(m2);
```

Write a new class called `MovableRectangle`, which composes two `MovablePoints` (representing the top-left and bottom-right corners) and implementing the `Movable` Interface. Make sure that the two points has the same speed.



Try
Develop classes that shows the difference between an interface and an abstract class

9. Exercises on Exception Handling

9.1 Usage of the try and catch and finally block.

In this example, we are implementing try and catch block to handle the exception. The error code written is in try block and catch block handles the raised exception. The finally block will be executed on every condition.

Hints:

```
class ExceptionTest{
        public static void main(String[] args){
                int a = 40, b = 4, c = 4;
                int result;
                try{
                        result = a / (b-c);
                }
                catch (...){
                        ...;
                }
                finally{
                        ...;
                }
                ...;
                System.out.println("Result: "+result);
        }
```

```
}
```

Try

Experiment

- A scenario where NumberFormatException occurs
- A scenario where NullPointerException occurs
- A scenario where ArrayIndexOutOfBoundsException occurs

9.2 Multiple catch block using command line argument

The catch block is used to handle the exception which is raised in try block. A single try block may contain more than one catch block. Below example shows how to use to multiple catch block.

Hints:

```
class Check_Exception{
        public static void main(String[] args){
                try{
                        int a = ...;
                        int b = ...;
                        int c = a / b;
                        System.out.println("Result: "+c);
                }
                catch (...){

                }
                catch(...){

                }
                catch(NumberFormatException ne){

                }
                finally{
                        ...
                }
        }
}
```

9.3 Java throw Keyword

Create a validate method that takes integer value (age) as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

Hints

The syntax of the Java throw keyword is given below.

```
throw new exception_class("error message");
```

Example:

```
throw new IOException("sorry device error");
```

Try
Implement both unchecked and checked exceptions using throw keyword

9.4 Java throws keyword

The throws keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs, if matched with the declared ones, is thrown automatically then. Write Java code that demonstrates the working of throws keyword in exception handling.

Hints

Syntax:

```
return_type method_name() throws exception_class_name{
        //method code
}
```

Advantages:
1. Checked Exception can be propagated (forwarded in call stack).
2. It provides information to the caller of the method about the exception.

9.5 Chained Exceptions

Java allows relating one exception with another exception. i.e. one exception describes the cause of another exception. Write a program to explain the chained exception in Java.

9.6 Custom Exceptions

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Following are few of the reasons to use custom exceptions:

- To catch and provide specific treatment to a subset of existing Java exceptions.
- Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create custom exception, we need to extend Exception class that belongs to java.lang package.
Develop a java program to implement custom exeptions

```
// class representing custom exception
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
        super(str);
    }
}

// class that uses custom exception InvalidAgeException
```

```
...
// main method
.....
```

10. Exercises on File Handling

10.1 Reading text file

Consider a text file abc.txt that contains data. Now your task is to extract the content of the file and display it

Hints:

1. Your program should accept the path to the text file as a command-line argument.
2. Implement proper error handling to account for file-related exceptions.
3. Design and implement a function/method for reading and displaying the content of the text file.

10.2 Reading file content line by line

The file file.txt is a text file that contains a list of names. Each line in the file contains a single name. The names in the file are separated by newline characters.
Write a Java program to read the content of the file file.txt line by line. For each line, print the line to the console.

Hints:

1. Create a BufferedReader object to read the file file.txt.
2. Use a while loop to read each line from the file.
3. For each line, print the line to the console.

10.3 Appending data to an existing file.

Develop a program that allows users to add new content to an already existing file. This problem aims to assess your understanding of file handling, proper exception management, and effective modification of file content.

Hints

1. Create a Java program that accepts the following inputs from the user:
   - The path to the existing text file.
   - The new content that the user wants to append to the file.
2. Open the existing file and append the provided content to it.
3. Implement appropriate error handling to manage exceptions during file operations.
4. After appending the data, display a confirmation message to the user.

10.4 Reading first 4 lines from a text file

Your task is to develop java program that reads and displays the first four lines of a text file.

Hints

1. Take the path to the text file as a command-line argument.
2. Implement error handling to address potential file-related exceptions.
3. Design a function/method that reads and displays the first four lines of the text file.

10.5 Copy the content of one file to another file

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream. Develop java code to copy the content of one file to another.

TRY

Implement the same using Character Streams like FileReader and FileWriter.

10.6 List files and Directories

Write a Java program to get a list of all file/directory names in the given directory.

TRY

1. Check if a file or directory specified by pathname exists or not.
2. Get specific files with extensions from a specified folder.
3. Check if a file or directory has read and write permissions.

11. Exercises on Multithreading

11.1 Concurrent Thread Increment

Create and Start Multiple Threads - Write a Java program to create and start multiple threads that increment a shared counter variable concurrently.

Hints

1. First, define a "Counter" class that represents a shared counter variable. It has a synchronized increment() method that increments the counter variable by one.
2. Next define an "IncrementThread" class that extends Thread. Each IncrementThread instance increments the shared counter by a specified number of increments.
3. In the Main class, we create a 'Counter' object, specify the number of threads and increments per thread, and create an array of 'IncrementThread' objects. We then iterate over the array, creating and starting each thread.
4. After starting all the threads, we use the join() method to wait for each thread to finish before proceeding. After all threads have finished, we print the shared counter's final count.

11.2 wait() and notify() for Thread Synchronization

Write a Java program to create a producer-consumer scenario using the wait() and notify() methods for thread synchronization.

Hints

1. The "Producer" class implements the Runnable interface and represents the producer thread. It continuously produces items by adding values to the shared buffer. When the buffer is full, the producer waits until the consumer consumes an item and notifies it.
2. The "Consumer" class also implements the Runnable interface and represents the consumer thread. It continuously consumes items by removing values from the shared buffer. As soon as the buffer is empty, the consumer uses the wait() method to wait until a new item is produced by the producer.
3. In the main() method, we create instances of the Producer and Consumer classes as separate threads and start them concurrently.

11.3 Synchronizing Threads with Reentrant for Shared Resource

A reentrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities.

A ReentrantLock is owned by the thread last successfully locking, but not yet unlocking it. A thread invoking lock will return, successfully acquiring the lock, when the lock is not owned by another thread. The method will return immediately if the current thread already owns the lock. This can be checked using methods isHeldByCurrentThread(), and getHoldCount().

Write a Java program that uses the ReentrantLock class to synchronize access to a shared resource among multiple threads.

11.4 Thread Synchronization with Semaphores

In computer science, a semaphore is a variable or abstract data type used to control access to a common resource by multiple threads and avoid critical section problems in a concurrent system such as a multitasking operating system. Semaphores are a type of synchronization primitive.

Write a Java program to demonstrate Semaphore usage for thread synchronization.

11.5 Concurrent Read-Write Access with ReadWriteLock

A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive.

All ReadWriteLock implementations must guarantee that the memory synchronization effects of writeLock operations (as specified in the Lock interface) also hold with respect to the associated readLock. That is, a thread successfully acquiring the read lock will see all updates made upon previous release of the write lock.

A read-write lock allows for a greater level of concurrency in accessing shared data than that permitted by a mutual exclusion lock. It exploits the fact that while only a single thread at a time (a writer thread) can modify the shared data, in many cases any number of threads can concurrently read the data (hence reader threads). In theory, the increase in concurrency permitted by the use of a read-write lock will lead to performance improvements over the use of a mutual exclusion lock. In practice this increase in concurrency will only be fully realized on a multi-processor, and then only if the access patterns for the shared data are suitable.

Write a Java program to illustrate the usage of the ReadWriteLock interface for concurrent read-write access to a shared resource.

12. JDBC Programming

12.1 Database Connection and Query

Create a Java program that establishes a connection to a database using JDBC. You are required to perform a basic SQL query to retrieve information from a table and display the results.

Hints

1. Set up a local database (e.g., MySQL) with a sample table.
2. Develop a Java program that establishes a JDBC connection to the database.
3. Write an SQL query to retrieve specific information from the table.

4. Execute the query and display the results in a readable format.
5. Handle any potential exceptions that may occur during database operations.

12.2 Prepared Statements and Parameterized Queries

Design a Java program that utilizes prepared statements for executing parameterized queries. Your program should demonstrate the use of placeholders to execute safe and efficient database operations.

Hints

1. Modify the previous program to use prepared statements for SQL queries.
2. Implement parameterized queries by using placeholders for dynamic values.
3. Allow the user to input values for the query parameters.
4. Execute the prepared statement and display the results.

12.3 Transaction Management

Develop a Java program that demonstrates the concept of transaction management in JDBC. Your program should include operations that simulate a transaction, with both successful and failed cases.

Hints

1. Create a sample database table suitable for transaction simulation.
2. Implement a transaction that involves multiple SQL statements.
3. Perform operations that simulate a successful transaction.
4. Perform operations that simulate a failed transaction (e.g., due to an error or violation).
5. Handle transaction rollbacks and commits accordingly.

12.4 Inserting Multiple Records

Design a Java program that showcases batch processing using JDBC. Your program should allow the insertion of multiple records into a database table in a single operation.

Hints

1. Create a suitable database table for record insertion.
2. Develop a Java program that inserts multiple records into the table.
4. Allow the user to input data for multiple records.
5. Use JDBC techniques to efficiently insert records in a single operation.

12.5 ResultSet and Data Retrieval

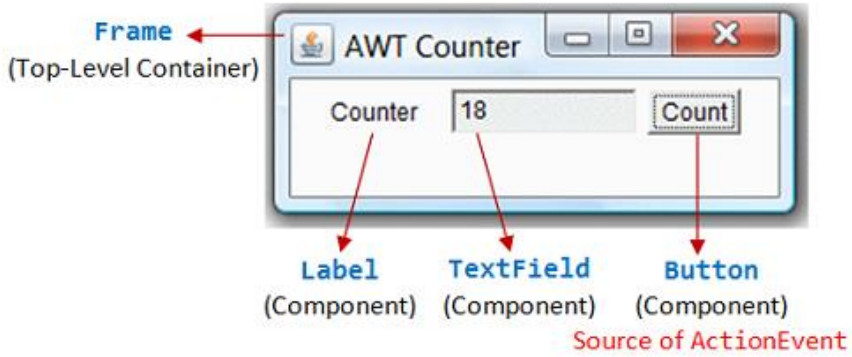
Create a Java program that focuses on retrieving and processing data from a database using JDBC's ResultSet. Your program should demonstrate the fetching and manipulation of data retrieved from a database table.

Hints

1. Set up a database table with relevant sample data.
2. Develop a Java program that connects to the database and retrieves data using a SQL query.
3. Process and manipulate the retrieved data using the ResultSet interface.
4. Display the processed data in a clear and organized manner.

13. AWT GUI Applications

13.1 AWTCounter



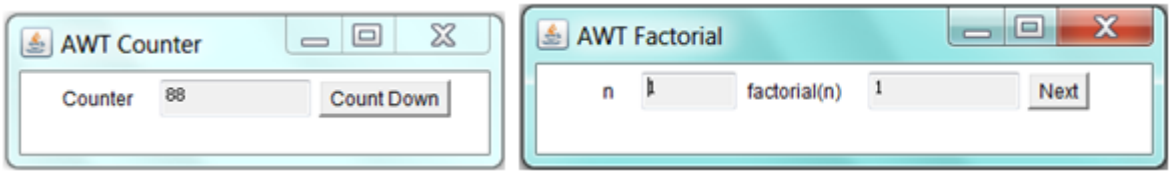
Write an AWT GUI application (called `AWTCounter`) as shown in the Figure. Each time the "Count" button is clicked, the counter value shall increase by 1.

The program has three components:

1. a `java.awt.Label` "`Counter`";
2. a non-editable `java.awt.TextField` to display the counter value; and
3. a `java.awt.Button` "`Count`".

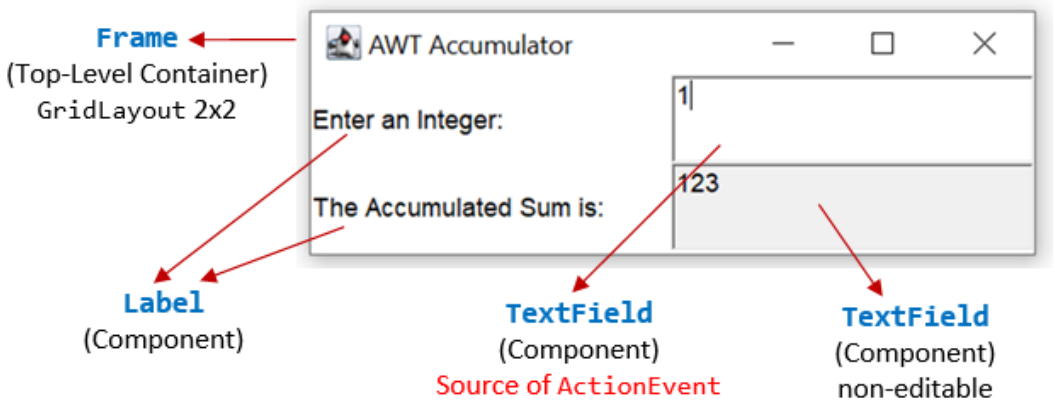
The components are placed inside the top-level AWT container `java.awt.Frame`, arranged in `FlowLayout`. You have to use control-c, or "close" the CMD shell, or hit the "terminate" button on Eclipse's Console to terminate the program. This is because the program does not process the `WindowEvent` fired by the "window-close" button.

Try



1. Modify the program (called `AWTCounterDown`) to count down, with an initial value of 88, as shown.
2. Modify the program (called `AWTFactorial`) to display n and factorial of n, as shown. Clicking the "Next" button shall increase n by 1. n shall begin at 1.

13.2 AWTAccumulator



Write an AWT GUI application called `AWTAccumulator`, which has four components:

1. a `java.awt.Label` "Enter an integer and press enter";
2. an input `java.awt.TextField`;
3. a `java.awt.Label` "The accumulated sum is", and
4. a protected (read-only) `java.awt.TextField` for displaying the accumulated sum.

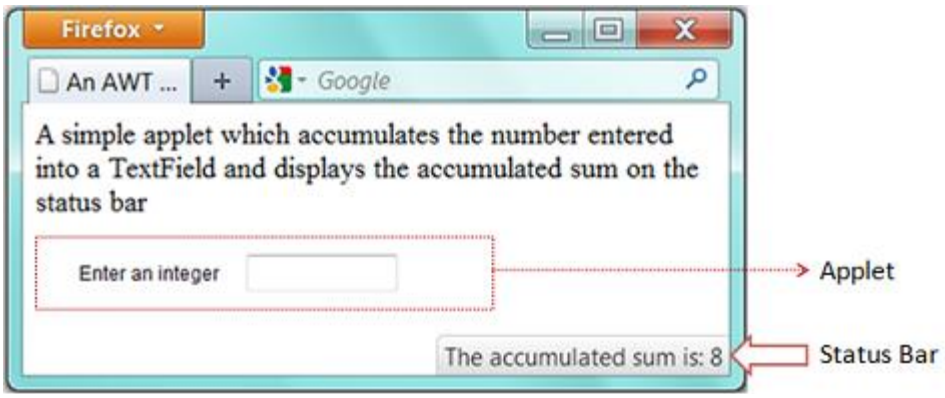
The four GUI components are placed inside a container `java.awt.Frame`, arranged in `GridLayout` of 2 rows 2 columns. The program shall accumulate the numbers entered into the input `TextField`, and display the accumulated sum on the display `TextField`.

Try

1. Modify the program (called `AWTAccumulatorLabel`) to display the sum using a `Label` instead of a protected `TextField`, as shown (using `FlowLayout`).

2. Modify the program (called `AWTFactorialTextField`) to display the factorial of the input number, as shown (using `FlowLayout`).

13.3 AWTAccumulatorApplet (Obsolete)



NOTE: Most browsers today do not support Java applets anymore. Keep this section for nostalgia.

An Java *applet* is a graphics program run inside a browser.
Write a Java applet (called `AWTAccumulatorApplet`) which contains:

1. a `label` "Enter an integer:",
2. a `TextField` for user to enter a number.
3. The applet shall accumulate all the integers entered and show it on the status bar of the browser's window.

Note:

- An applet extends from `java.applet.Applet`, whereas a standalone GUI application extends from `java.awt.Frame`. You cannot `setTitle()` and `setSize()` on Applet.
- Applet uses `init()` to create the GUI, while standalone GUI application uses the constructor (invoked in `main()`).

HTML codes: AWTAccumulatorApplet.html

Applet runs inside a web browser. A separate HTML script (says `AWTAccumulatorApplet.html`) is required, which uses an <applet> tag to embed the applet

Try

1. Modify the applet to run the "Counter" application (as in `AWTCounter`).
2. Modify the applet to run the "Factorial" application (as in `AWTFactorial`).

13.4 WindowEvent and WindowListener

Modify the `AWTCounter` program (called `AWTCounterWithClose`) to process the "Window-Close" button.

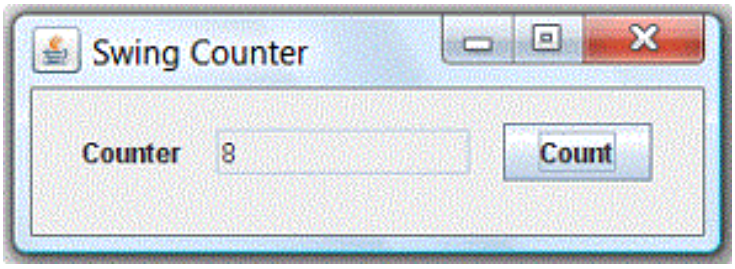
14.1 Converting from AWT to Swing

Convert all the previous AWT exercises (`AWTCounter`, `AWTAccumulator`, `AWTFactorial`, etc.) to Swing applications (called `SwingCounter`, `SwingAccumulator`, `SwingFactorial`, etc.).

Notes:

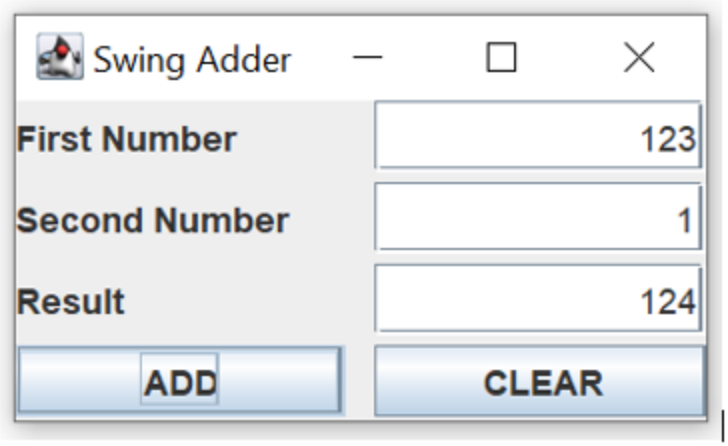
- Swing Components are kept in package `javax.swing`. They begin with a prefix "`J`", e.g., `JButton`, `JLabel`, `JFrame`.
- Swing Components are to be added onto the `ContentPane` of the top-level container `JFrame`. You can retrieve the `ContentPane` via method `getContentPane()` from a `JFrame`.

```
Container cp = getContentPane();  // of JFrame
cp.setLayout(......);
cp.add(......);
```



14.2 SwingAdder

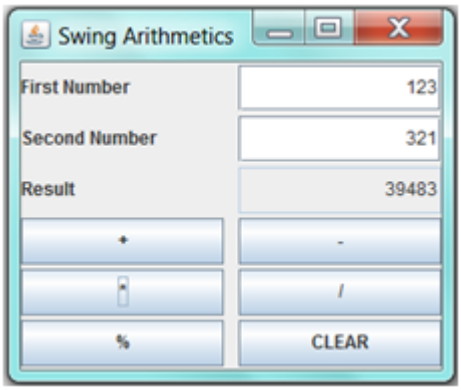
Write a Swing application called `SwingAdder` as shown. The "ADD" button adds the two integers and display the result. The "CLEAR" button shall clear all the text fields.



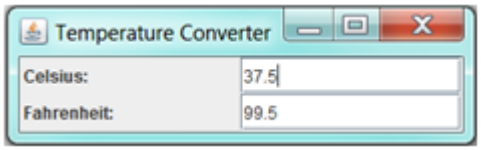
Hints: Set the content-pane to 4x2 GridLayout. The components are added from left-to-right, top-to-bottom.

Try

Modify the above exercise (called `SwingArithmetics`) to include buttons "`+`", "`-`", "`*`", "`/`", "`%`" (remainder) and "`CLEAR`" as shown.



14.3 SwingTemperatureConverter

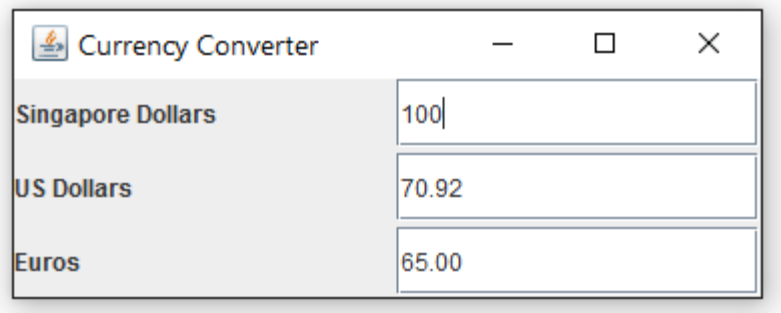


Write a GUI program called `SwingTemperatureConverter` to convert temperature values between Celsius and Fahrenheit. User can enter either the Celsius or the Fahrenheit value, in floating-point number.

Hints

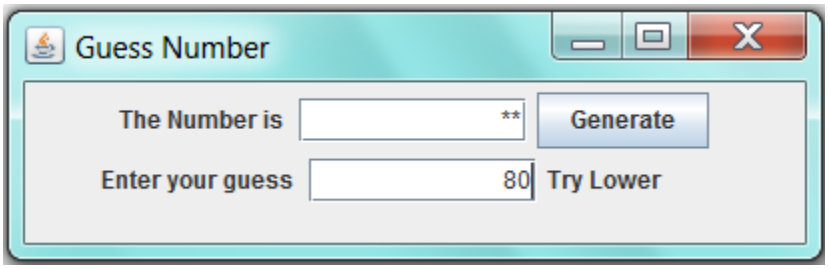
To display a floating-point number in a specific format (e.g., 1 decimal place), use the `static` method `String.format()`, which has the same form as `printf()`. For example, `String.format("%.1f", 1.234)` returns String "`1.2`".

14.4 SwingCurrencyConverter



Write a simple currency converter, as shown in the figure. User can enter the amount of "Singapore Dollars", "US Dollars", or "Euros", in floating-point number. The converted values shall be displayed to 2 decimal places. Assume that 1 USD = 1.41 SGD, 1 USD = 0.92 Euro, 1 SGD = 0.65 Euro.

14.5 SwingNumberGuess



Write a number guessing game in Swing (as shown in the Figure). The program shall generate a random number between 1 to 100. It shall mask out the random number generated and output "Yot Got it", "Try Higher" or "Try Lower" depending on the user's input.

Hints

- You can use `Math.random()` to generate a random number in `double` in the range of `[0.0, 1.0)`.

14.6 SwingPhoneApp

Write a Software Phone App using Java Swing as illustrated in the figure. The user enters the phone number and pushes the "CALL" button to start a phone call. Once the call is started, the label of the "CALL" button changes to "HANG UP". When the user hangs up, the display is cleared.

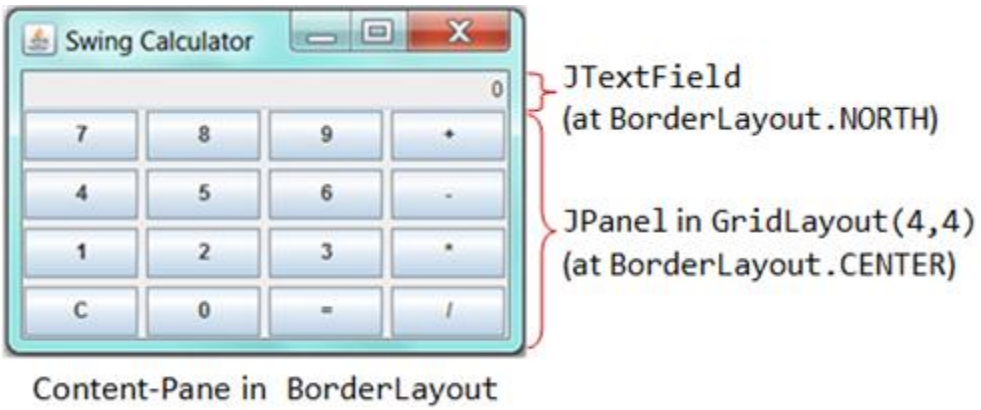
Assume that the following 2 methods are available for handling phone call:

```
public void call(String phoneNumber);  // to make a phone call with the phoneNumber
public void hangup();  // to terminate the existing call
```

Hints

- Use a 10-element `JButton` array to hold the 10 numeric buttons. Construct a common instance of a named inner class as the `ActionListener` for the 10 numeric buttons.
- Use a `boolean` flag (says `isCalling`) to keep track of the status.

14.7 SwingCalculator

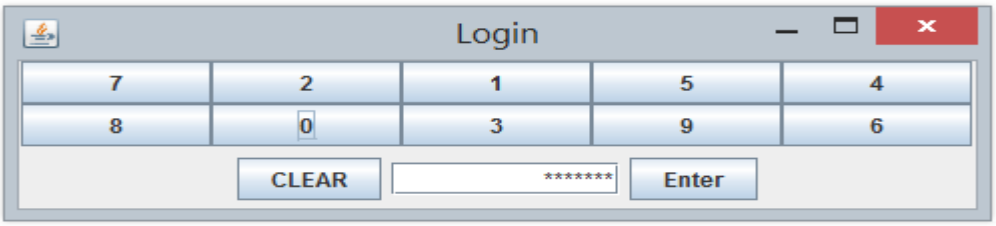


Implement a simple calculator (called SwingCalculator) as shown.

Hints

- Set the ContentPane to BorderLayout. Add a JTextField (tfDisplay) to the NORHT. Add a JPanel (panelButtons) to the CENTER. Set the JPanel to GridLayout of 4x4, and add the 16 buttons.
- All the number buttons can share the same listener as they can be processed with the same codes. Use event.getActionCommand() to get the label of the button that fires the event.
- The operator buttons "+", "-", "*", "/", "%" and "=" can share a common listener.
- Use an anonymous inner class for "C" button.
- You need to keep track of the *previous* operator. For example in "1 + 2 =", the current operator is "=", while the *previous* operator is "+". Perform the operation specified by the previous operator.

14.8 SwingLoginPanel

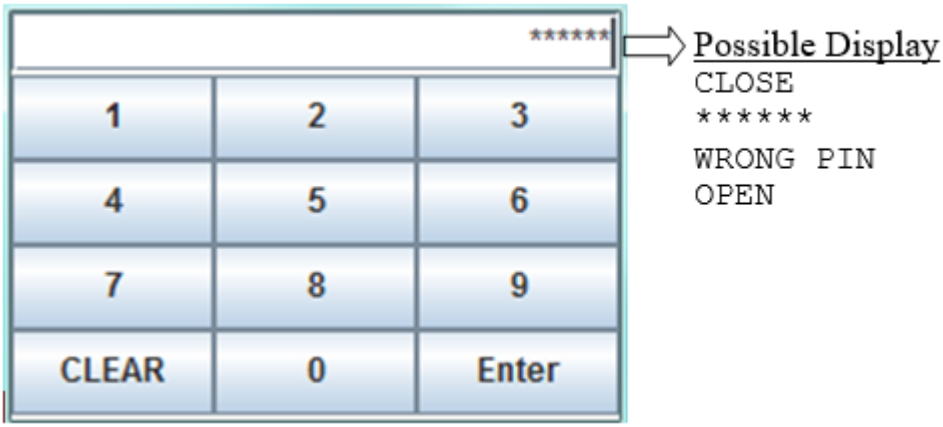


A Java Swing application has a login page as shown in the Figure. Users are required to enter the correct passcode to start the application. The system uses a scramble keypad with a randomly allocated set of numbers from 0 to 9. The display shall show "Enter passcode" initially, and show an asterisk (*) for each number entered. Upon pushing the "Enter" button, the system verifies the passcode. If the passcode is correct, the system invokes a method called `startApp()` to start the application. Otherwise, it displays "Wrong passcode". The "Clear" button shall clear the display.

Assume that the following methods are available:

```
public String getPasscode(); // return the passcode
public void startApp();      // Start the application
public void shuffleArray(int[] array)
   // Shuffle (Randomize) the given int array, e.g.,
   // int[] numbers = {1, 2, 3, 4, 5};
   // shuffleArray(numbers); // randomize the elements
```

14.9 SwingLock



Write a Java Swing application for an electronic lock as shown in the figure. The display shall show the state of either "CLOSE" or "OPEN". In the "CLOSE" state, the user types his PIN followed by the "Enter" key to unlock the system.

The display shall show an asterisk (*) for each number entered. The display shall show "WRONG PIN" if the PIN is incorrect. The "Clear" button clears the number entered (if any), locks the system and sets the display to "CLOSE".

Assume that the following methods are available:

```
public boolean checkPIN(String PIN);  // return true for correct PIN
public void unlock();   // Unlock the system
public void lock();     // Lock the system
```

Hints

- Use a 10-element `JButton` array to hold the 10 numberic buttons. Construct a common instance of a named inner class as their `ActionListener`.
- Use a `boolean` flag (says `isLocked`) to keep track of the status.

15. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging.

There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)

- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

Name of the contest	Minimum number of problems to solve	Required score
CodeChef	20	200
Leetcode	20	200
GeeksforGeeks	20	200
SPOJ	5	50
InterviewBit	10	1000
Hackerrank	25	250
Codeforces	10	100
BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certifications:

- HackerRank – Java Basic Skills Certification
- Oracle Certified Associate Java Programmer OCAJP
- CodeChef - Learn Java Certification
- NPTEL – Programming in Java
- NPTEL – Data Structures and Algorithms in Java

V. TEXT BOOKS:

1. Schildt, Herbert. *Java: The Complete Reference* 11th Edition, McGraw-Hill Education, 2018.
2. Deitel, Paul and Deitel, Harvey. *Java: How to Program*, Pearson, 11th Edition, 2018.

VI. REFERENCE BOOKS:

1. Evans, Benjamin J. and Flanagan, David. *Java in a Nutshell*, O'Reilly Media, 7th Edition, 2018.
2. Bloch, Joshua. *Effective Java*, Addison-Wesley Professional, 3rd Edition, 2017.
3. Sierra, Kathy and Bates, Bert. *Head First Java*, O'Reilly Media, 2nd Edition, 2005
4. Farrell, Joyce.*Java Programming*, Cengage Learning B S Publishers, 8th Edition, 2020

VII. ELECTRONICS RESOURCES:

1. https://docs.oracle.com/en/java/
2. https://www.geeksforgeeks.org/java
3. https://www.tutorialspoint.com/java/index.htm
4. https://www.coursera.org/courses?query=java

VIII. MATERIALS ONLINE

1. Course Content
2. Lab manual

LINUX PROGRAMMING LABORATORY

III Semester: CSE (CS) IV Semester: CSE / CSE (DS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
ACSC16	**CORE**	0	1	2	2	30	70	100
Contact Classes 15	**Tutorial Classes**	**Practical Classes: 30**				**Total Classes: 45**		
Prerequisite: Programming for Problem Solving using C Laboratory								

I. COURSE OVERVIEW:

This course aims to provide a well-rounded introduction to Linux, operating systems, and software development in a Linux environment. Linux is the blend of innovative concepts required by its unique environment involving kernel concepts, basic commands, shell scripting, file processing, socket programming, processes and Inter process communication (IPC). This course equip individuals with the knowledge and skills needed to work effectively in a Linux-based environment, software development, system administration and security awareness.

II. COURSES OBJECTIVES:

The students will try to learn

VI. The fundamental concepts of operating system including bourne again shell (bash) with the linux command line environment.

VII. The shell programming using arithmetic operations, control structures and functions in shell scripts in vi editor.

VIII. The process management and inter-process communication used for exchanging data between multiple threads in one or more processes.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 Demonstrate text processing utilities, file handling utilities, security by file permissions, process utilities, disk utilities and networking commands with different options available for solving problems.

CO 2 Make use of bourne shell constructs, decision structures and loops in designing programs for

CO 3 Interpret to write, compile, debug and run C language program in linux shell environment for implementing kernel level concepts.

CO 4 Identify basic methods and techniques used in solving simple programming tasks in the area of execution environment, processes signals and threads.

CO 5 Experiment with IPC mechanisms such as pipes, named pipes, shared memory, message queues, semaphores and sockets for inter-process communication.

CO 6 Choose the appropriate protocol such as TCP or UDP for effective communication in client-server applications.

1. Getting Started Exercises

1.1 Help and man pages

You can issue help command:

```
$ help                          // Display the help menu for the bash shell
$ help <command-name>  // Display the help menu for the command
```

Most of the commands also support an help option, but may exist in various style:

```
$ <command-name> -h        // Unix-style: dash followed by a single character
$ <command-name> -?
$ <command-name> -help      // X-style: dash followed by a keyword
$ <command-name> --help     // GNU-style: double-dash followed by a keyword
```

Unix systems provide so called man pages (or manual pages) for all the commands and utilities. To display the man page for a particular command, use man command:

```
$ man <command-name>          // Display manual page for the command
                        // You can use Up/Down/PgUp/PgDown keys to scroll the texts
$ info <command-name>
$ man <command-name> | less   // Display in page-mode
```

To search for commands:

```
$ man -k <keyword>     // Search for commands relevant to keyword
$ apropos <keyword>    // Same as above
```

1.2 Useful Commands/Utilities

These are the commands/utilities that a good Unix programmer is expected to know. Check the man pages ("man *command-name*" or google) to get the detailed description.

- File related:
  - **pwd**: Print current working directory. In bash shell, the current working directory is also shown in the command prompt.
  - **cd *pathname***: Change current working directory. The *pathname* could be either absolute or relative (to the current working directory). Special notations "." and ".." refer to the current and parent directories, respectively.
  - **ls**: List files (in short-format). "ls -l" lists file in long-format; "ls -a" lists also the hidden files.
  - **cat**: Concatenate files and print its content.
  - **less**, **more**: View file in pages.
  - **touch *filename***: Create the file if it does not exist; otherwise, update the last-modified timestamp.
  - **export *name*=*value***: Set a variable and export to global environment.
  - **top**: Print resource usage and top processes.
  - **hostname**: Print hostname.
  - **uptime**: Print how long the system has been running.
  - **date**: Print date/time.
- Utilities:
  - **which *program-name***: Print the location of the program-name.
  - **whereis *program-name***: List all files related to the program-name.
  - **whatis *program-name***: Print one-line description of program-name.

- **locate *filename***: Search for files in local system.
- **man *command-name***: Display manual pages for the command.

- Editors:
  - **vi/vim**, **nano**, **emacs**: Console-based (text-based) editors.
  - **gedit**: graphical text editor.

- Programming:
  - **make**: Install programs.
  - **gcc**, **g++**: GNU C/C++ compiler.
- More: diff, gzip, tar, ping, ssh, history, su, sudo, adduser, addgroup, etc.

More on cd (change directory) command

Read "Change Directory (cd) command" for basic usage.
You can use "cd *path*" to change the current working directory. The new path could be an absolute path, beginning with root "\" or home "~"; or relative to the current working directory (PWD).
In cd command, you can use "/" to denote the root directory, "~" to denote home directory of the current login user; ".." (double-dot) to refer to the parent directory; "." (single-dot) to refer to the current directory; and "-" (dash) to refer to the previous working directory (OLDPWD).
By default, in "cd *relative-path*", the new path is relative to the current working directory. Nonetheless, you can set the environment variable CDPATH to change the base. If CDPATH is not set, it is defaulted to current working directory. CDPATH could contain multiple directories separated by ":" (colon). For example,

```
$ cd                // home directory
$ pwd
/home/peter
$ mkdir local       // create a directory local (/home/peter/local)

$ export CDPATH=/usr    // set base for relative cd to /usr

$ cd local          // relative to CDPATH
/usr/local

$ export CDPATH=.:/usr  // set base for relative cd to current directory and /usr
$ cd                // home directory
$ pwd
/home/peter
$ cd local          // found local relative to current directory
/home/peter/local
```

1.3 Pipe and Input/Output Redirection

By default, the output of a command goes to the screen (called STDOUT), and the input of a command comes from the keyboard (called STDIN). You can use a *redirection operator* to redirect input and output from/to a file or another command:

- > (output redirection): Writes the output to a file (or a device such as printer), instead of the screen (STDOUT).
- >> (output append redirection): Appends the output to a file, instead of the screen.
- < (input redirection): Reads the input from a file or a device, instead of the keyboard (STDIN).
- | (pipe): Pipes the output of one command as the input of another command.
- tee: sends output to standard output and to file(s). Named after T-pipe, which splits water into two directions.

An output redirector '>' involves a program and a sink (destination). An input redirector '<' involves a program and a source. A pipe '|' involves two programs.

For examples,

```
// Redirect the output of the ls command to a file, instead of screen
$ ls -l > listing.txt

// Pipe the output of ls command as the input of program less for view page-by-page
$ ls -l | less
// Pipe the output of ls command as the input of program wc to count the lines
$ ls -l | wc -l
// Pipe the output of ls command as the input of program grep to filter lines containing "xxxx"
$ ls -l | grep xxxx

// Pipe the output of ls command as the input of tee, which sends the output to standout and file "listing.txt"
$ ls -l | tee listing.txt
// Pipe the output of ls command as the input of grep.
// grep then sends its output to standout and files "listing1.txt" and "listing2.txt"
$ ls -l | grep xxxx | tee listing1.txt listing2.txt
```

1.4 File Handling Utilities:

cp: ***The cp command copies a file or a group of files. It creates an exact image of the file on disk with a different name.***

Ex: $cp t1 t2
This will copy the contents of t1 into t2. If t2 does not exist, it will be created. However, if t2 already exists, then its contents are overwritten.
Ex: $cp -i t1 t2
cp: overwrite t2 (Yes/No)?
This will copy the contents of t1 into t2. If t2 does not exist, it will be created. However, if t2 already exists, then it warns the user before overwritten contents to a file. If "y" at this prompt overwrites the file, any other response leaves it uncopied.

mv:

The mv command renames files. It has two functions:

1. It renames a file (or directory).
2. It moves a group of files to a different directory.

Syn: $mv <sourcefilename> <renamefile>/<directoryname>. For example,

$ Ex: $mv s1 t1

Then s1 is renamed as t1.

Ex: $mv file1 file2 newdir

On execution of this command 'file1' and 'file2' are no longer present at their original location, but are moved to the directory 'newdir'.

rm: (Removing files): The rm command removes the given file or files supplied to it.
Syn: $rm options <filename(s)>

Ex: $rm –i file1
Where –i is a switch, removes file1 interactively; i.e. you are asked for confirmation before deleting the file

mkdir: The mkdir command is used to create a new directory

Syn: $mkdir options <directoryname>

Ex: $ mkdir book
The above command creates a directory named 'book'

Among the options available with mkdir is -p, which allows us to create multiple generations of directories specified in the given path too.

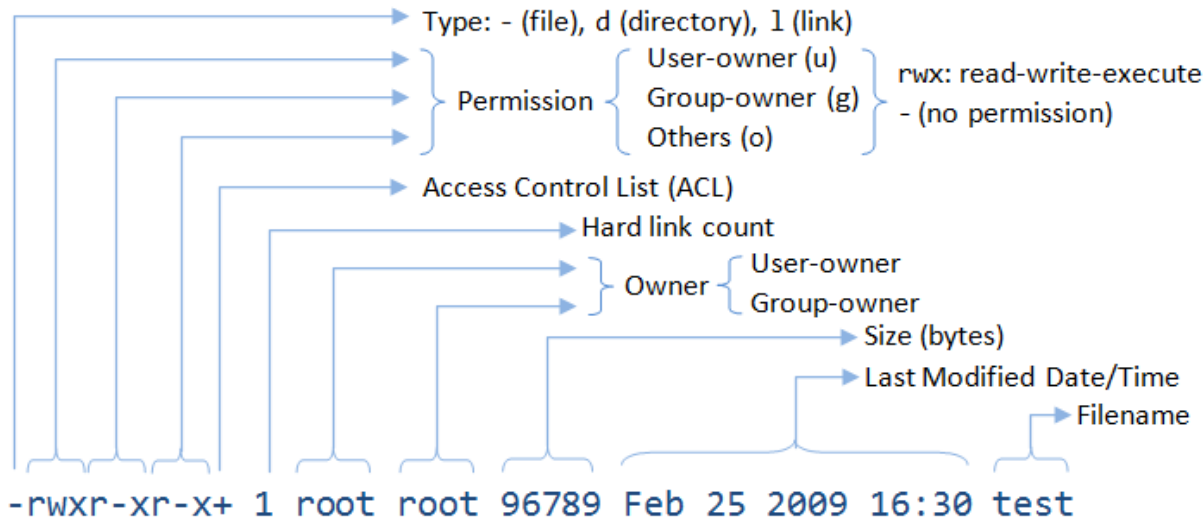
rmdir: To remove empty directory
Syn: $rmdir options <directoryname>
The rmdir is used to remove directories.

With '-p' option, it removes not only the specified directory but also its parent directories. However rmdir removes only the empty directories.

Ex: $ rmdir -p works/dir1/unix/book
It removes the directory book.

1.5 File/Directory Ownerships and Permissions

You can issue command "ls -l" to list files/directory in long format, which shows all the file attributes, e.g.,



For each entry:

- The first character indicates the type, with - for file and d for directory. The other type codes are l for link (symlink or hard link), b for block device, c for character device, p for named pipe, and s for socket.
- A file/directory has a user-owner and a group-owner, as indicated. The user-owner may or may not belong to the group.
- A file/directory has 3 permission settings, read/list (r), write (w) and execute/access (x), for user-owner (u), group-owner (g) and others (or the world) (o), respectively. The permissions are represented with 9 characters, in 3 groups of "rwx" for user-owner, group-owner and others, as shown in the above listing. "-" indicates absence of permission. For examples,

- -rwxr-x--- peter devel test.php
- // File,
- // filename is test.php,
- // user-owner is peter, group-owner is devel,
- // user-owner has read, write and execute permissions,
- // group-owner has read and execute permissions,
- // others (the world) have no permission
- -rwx------ root root myconfig
- // File,
- // filename is myconfig,

```
// user-owner is root, group-owner is root,
// user-owner has read, write and execute permissions,
// group-owner and others have no permission
drwxr-xr-x peter devel www
// Directory,
// directory name is www,
// user-owner is peter, group-owner is devel,
// user-owner has list, write and access permissions,
// group-owner has list and access permissions,
// others (the world) have list and access permissions
```

File Permissions

A file is indicated by type of "-". For files:

- "r" (read) permits reading the file content.
- "w" (write) permits writing into the file.
- "x" (execute) indicates that the file is executable, i.e., a program file (or binary file).

Directory Permissions

A directory is indicated by type of "d". For directories, "r" shall be interpreted as list and "x" as access, as follows:

- "r" (list) permits listing of directory's contents (filenames and sub-directory names only) via listing command such as ls. Without "r" permission, you cannot issue "ls" command.
- "w" (write) permits writing into the directory, i.e., creating new files or sub-directories inside the directory.
- "x" (access) permits access into this directory (i.e., "cd" into the directory).

A directory holds two pieces of information for each file/sub-directory it contains: the filename/subdirectory-name and its inode number. I-node stores the attributes of the file/directory, including the disk block location. You can list the name and inode number via command "ls -i". For directories, "r" permission is needed to get the name; "x" permission is needed to get its inode number given the name, which is needed to enter the directory. For example, to issue "cat /home/peter/test/f1.txt", you need "x" permission for directories /, home, peter, test (so as to enter the directories); and "r" permission for file "f1.txt". No "r" permission is needed for the directories, as the names are known. "r" permission is needed for a directory for issuing ls command.

For production system, "x" is needed to enter (access) the directory. Very few programs need to list the directory contents. For development system, both "x" and "r" are needed to enter the directory and issue the ls command.

Change Mode (chmod)

chmod (change file mode) to change the file mode (i.e., permission), in the form of "ugoa±rwx" with "+" to add permission and "-" to remove; or "ugoa=rwx" to set the permission, where "u" for user-owner, "g" for group-owner, "o" for others, and "a" for all (i.e., u+g+o). For example,

```
// Grant executable mode for owner(u), group(g) for all *.sh files
// -c lists all the changes
$ chmod -c ug+x *.sh
// Remove write permission from group(g), others(o) for all *.txt files
$ chmod -c go-w *.txt
// All (u+g+o) read and execute
$ chmod -c a=rx *
```

You can also use three octal numbers (each for u, g and o) to represent the file permission. Each octal number carries 3 bits (xxx), corresponding to the read-write-execute permissions, where 4 (100B) for read, 2 (010B) for

write and 1 (001B) for execute. E.g., "700" is equivalent to "rwx------"; "664" is equivalent to "rw-rw-r--"; "775" is equivalent to "rwxrwxr-x". For example,

```
$ chmod -c 700 myfile.txt  // 700 = 111 000 000 (rwx------)
$ chmod -c 600 *.txt       // 600 = 110 000 000 (rw-------)
```

Change Owner/Group (chown, chgrp)

We can use command chown (change file owner/group) to change the owner of files, e.g.,

```
// Change owner for a file
$ chown peter test.txt
// Change owner of the current directory
$ chown peter .
// Also change the group to programmers for a directory
$ chown peter:programmers /myproject

// Change Group
$ chgrp programmers .
```

You can use option -R to recursively changing the files in the subdirectories.
In most systems, only superuser can run chown, not even the file owner. This is because Unix systems prevent users from "giving away" files (you can only chown to yourself, which does not require any chown command!)
The file owner can run the chgrp if he belongs to the target group (again, you cannot "give away" files).

umask

umask (filemode creation mask) controls the permissions for new files and directory created. Recall that permissions can be represented by 3 octal numbers, each representing rwx for u (user), g (group) and o (others) respectively, e.g., 660 for (rw- rw- ---); 775 for (rwx rwx r-x).
umask is also represented in 3 octal numbers, with 1 to *disable* a certain permission. For example, suppose the umask is 022 (--- -w- -w-), it will disables the w for group and others, when a new file/directory is created. In other words, a new file/directory could have (rwx r-x r-x), or lesser permissions; it will never have w for group and others.
Each program has its own umask. For "Terminal" application, you can set the umask in one of the login scripts (such as "~/.profile") with the following line:

```
umask 022
```

You can also issue the command umask to display/change the current umask setting.
(For apache, the umask can be set in /etc/apache2/envvars.)

setuid and setgid permission for executable-file

When you execute a program (having "x" file permission) with setuid (Set User ID upon execution) or setgid (Set Group ID upon execution), it takes on the file owner's (or group's) privilege. For example, if a setuid program is owned by root, and executable by all, it will be run in root privilege, even when invoked by a less privilege user. setuid and setgid can be dangerous! Use them with great care!
You can use chmod command to enable setuid/setgid:

1.6 disk utilities

A disk utility is a utility program that allows a user to perform various functions on a computer disk, such as disk partitioning and logical volume management, as well as multiple smaller tasks such as changing drive letters and other mount points, renaming volumes, disk checking, and disk formatting, which are otherwise handled separately by multiple other built-in commands.

- **du**: Print disk file space usage.
- **df**: Report file system disk space usage.
- **ulimit:** It stands for 'user limit' and contains a value, which signifies the largest file that can be created by the user in the file system.

- **find:** You The find command is recursively examines a directory tree to look for file matching some criteria, and then takes some action on the selected files or locating files.

Syn: $find pathlist selection_criteria action. For example,

```
Ex: $ find -name "*.c"
  // Find all filename ending with .c, in current directory and subdirectories

$ find -iname "*.c"
  // option -i for case insensitive

$ find /usr -name "*.c"
  // Search /usr and its subdirectories
```

mount/unmount: A super user may extend the file system by using the 'mount' utility.
Syntax : mount –o options [device name directory]
unmount device name
The file system built on the floppy disk can be linked into the existing file system on the hard disk using the 'mount' command. Once mounted we can create files and directories in the new file system and treat it as a normal directory existing in a file system.

2. Exercises on Process, Text, network and backup utilities

Everything in Unix is a *file* - from data files, executable programs, to input and output devices. Files are organized in *directories* (aka *folders*). The directories are organized in a hierarchical *tree* structure, starting from the *root* directory, denoted by "/". There is only one root directory for the entire Unix's file system. A directory may contain sub-directories and files.
Unix is a *multi-user* system. Some files are used solely by the system; some are shared by all users; while some belong to a particular user.

`Disk Drives and Root Directory`
Windows systems has a concept of drive (e.g., C drive and D drive). Each drive begins with a root directory (e.g., `c:\`, `d:\`), resulting in multiple root directories.
Unix has no concept of drive and has a single root for the entire file system. The drives are mounted under the file system at a specific directory.

2.1 System and Application Software Directories

A Unix system has these directories for *system and application software* available to all users:

- /lib, /bin, /sbin: System libraries, binaries and superuser's binaries. Binaries are executable programs (having *executable* file attribute). Libraries are supporting codes for programs. There are two types of libraries, static libraries (.a for archives in Unix, .lib in Windows) and dynamic libraries (.so for shared objects in Unix, .dll for dynamic link libraries in Windows). Static library codes are linked into the program; whereas shared library codes are loaded during runtime and can be shared by many programs.
- /usr: Application Software (contrast to System Software in /bin, /lib, and /sbin). It contains sub-directories such as /usr/bin, /usr/lib, /usr/include, /usr/share/man (man pages), /usr/share/doc (documentation) and etc.
- /usr/local: Locally installed application software which are not part of the distribution. It contains sub-directories such as /usr/lcoal/bin, /usr/local/lib, /usr/local/include, and etc.
- /opt: optional application software package.
- /etc: System-wide configuration files, such as fstab (file system table for disks, CD drives, and storage devices), passwd (list of users), and sudoers (list of users with superuser access). (It is called "et cetera" in early days to mean additional things to bin and lib.)

- /var: Variable (changing) files, such as /var/logs (for log files), /var/spool (for printer spool files), /var/mail (for mail), /var/tmp, and etc.
- /root: superuser's home directory.
- /dev: file representation of devices. A special /dev/null (or the null device) is a special file that discards all data written to it but reports that the write operation succeeded. It provides no data to any process that reads from it, yielding EOF immediately.
- /mnt: file system mount point for (fixed) hard disks, CD drive, etc.
- /media: for removable media, such as external drive and USB drive.
- /sys: real-time information on devices used by the kernel.
- /boot: boot loader files and Linux kernel.
- /proc: process information.
- /srv: services.
- /tmp: System's temporary directory.

2.1 Processes utilites

Unix is a multi-process, multi-user operating system. It supports many processes concurrently.

You can use ps to list all the processes:

```
$ ps          // Print processes of current user
$ ps -e       // Print all processes
$ ps -ef      // Print all processes in full-listing
$ ps aux      // Same as above (in old BSD options)

// Search for processes
$ ps aux | grep mysqld   // Print MySQL server process
$ ps -ef | grep mysqld   // same as above
$ ps -ef | grep tomcat
$ ps -ef | grep $USER   // of current user, same as ps -f
```

[TODO] pid, Sample full-listing of ps

To terminal a process, you can issue kill command with the process ID, or job ID.

```
$ kill pid      // Kill a particular possess with the given processID
$ kill -9 pid   // Force kill
```

2.2 Text Processing Utilities

head: Displaying the beginning of a file.

Ex: $head -10 newfile
The first 10 lines of newfile are displayed.

tail: Displaying the end of a file.
The *tail* command is the counter part of the head command, displays the end of the file. By default *head* and *tail* commands display first and last 10 lines of a file respectively.

Ex: $tail -3 emp
It displays the last 3 lines in emp file.
The disadvantage of head and tail is that they cannot display a range of lines.

sort: This command can be used for sorting the contents of a file. Apart form sorting files, sort can merge multiple sorted files and store the result in the specified output file. While sorting the sort command bases its comparisons on the first character in each line in the file. If the first character of two lines is same then the second character in each line is compared and so on. It sorts the spaces the tabs first, then the punctuation marks followed by numbers, uppercase letters and lowercase letters in that order.

This simplest form of sort command would be
$sort file1
This would sort the contents of file1 and display the sorted output on the screen.

nl: line numbering:This is for numbering the lines.

Example:
$nl file1
1 chairman
2 special officer
3 principal
4 head
5 lecturer

uniq: locating repeated lines:

Ex:$uniq mp1
Uniq simply fetches one copy of each record and writes it to the standard output.

grep - Print lines matching the pattern

You can use grep to find lines matching a search pattern. The syntax is:

$ grep [*options*] *pattern* [*file*...]
for Ex:$ grep search-word filename
$ grep search-word file1 file2 file3
$ grep -r search-word directoryname // -r (or -R) for recursive

cut:: Like sort, cut is also a filter. It cuts or picks up a given number of character or fields from the specified file cut is slitting a file vertically cut identifies both columns and fields.

Ex: a) $cut –c 5-10 16-20 file1

Displays the columns from 5 to 10 and 16 to 20 of file1

b) $cut –c -3 5-10 16-20 30- file1

The expression 30- indicates column number 30 to the end of the line. Similarly, -3 is the same as 1-3.

Files often don't contain fixed length records, in which case, it is better to cut fields rather than columns. Two options need to be used here –d (delimiter) for the field delimiter and –f (field) for specifying the field list.

paste: pasting files:
It is a special type of 'concatenation' in that it pastes files vertically rather than horizontally.

Ex:$ paste file1 file2

Then the fields of file2 are concatenated to fields of file1. By default the delimiter is tab. The delimiter of our choice is set by '-d'(delimiter) option.

$cut –d "|" –f 1,4 –shortlist | paste –d "|" –cutlist

Join: A command that extracts common lines from two sorted files.
Syntax : join [options] filename1 filename2
One line of o/p is created for each line in the two files that match, based on option.
This command joins the common lines found in filename1 and filename2; if filename1 is not specified, *join* reads from the standard input.
wc: It counts the number of lines, words and characters in the specified file or files. It comes with the options *-l, -w* and *–c* which allow the user to obtain the number of lines, words or characters individually or in any desired combination.

Ex: $wc –lc file1 file2

file1 20 571

file2 30 804

The file file1 constitutes 20 lines and 571 characters similarly for the file2.

The wc command is capable of accepting input directly from the keyboard. By entering wc without any arguments, it waits for the user to type in the input. On terminating input (using ctrl d), the appropriate counts are displayed for the input.

Note: Filters are: *cat, pg, more, head, tail, grep, sort, wc, nl, uniq, cut, paste* etc.

tee:

Ex: who | tee logfile | sort

Here, the output of who becomes the standard input of tee. tee now sends one copy of the input to sort through one pipeline, whereas the other copy is stored in a file called logfile.

$who | tee file1 file2 | sort

Here, the output of who is stored in file1 and file2 and one copy is sent to sort.

If we want to store the output of who in file1 and file2, display the same output on the screen and store the sorted output in file3, then we write

$who | tee file1 file2 /dev/tty3a | sort >file3

If we wish to append the output of tee to a file, -a option is used with it as shown below:

$cat file1 file2 | tee –a file3 | more

In this pipeline the output of cat(contents of the files file1 and file2) are appended to the existing contents of file3. Another copy of output of cat is sent to more for displaying on the screen.

comm: finding what is common: *comm* requires two sorted files, and compares each line of the first file with it's corresponding line in the second.

Ex : $cat file1 $cat file2 $cat file3

Raghu.V	Varun.T	Raghu.V
Shiva.P	Tharun.Q	Shiva.P
Roja.R	Raghu.V	Gopi.N
Gopi.S	Roja.R	Sekhar.S

$comm file1 file3

Raghu.V

Shiva.P

Gopi.N

Roja.R

Gopi.S Sekhar.S

comm can also produce selective output, using options -1, -2 or -3. To drop a particular column, simply use it's column number with the '-' sign. We can also combine options and display only those lines that are common.

$comm -3 file1 file2 $comm -12 file1 file2

cmp: comparing two files:
Whether two files are identical or not is determined through *cmp, diff and comm*. commands.

Ex: $cmp chap1 chap2

chap1 chap2 differ :char 9,line1

The two files are compared byte by byte and the location of the first mismatch(in the 9th character of first line) is echoed to the screen. cmp, when invoked without options, doesnot bother about possible subsequent mismatches.

The –l(list) option gives a detailed list of the byte number and the differing bytes in octal for each character that differs in both files:

$cmp –l file1 file3

17 162 147

30 12 56

If the two files are identical, cmp displays no message, but simply returns the $ prompt.

diff: converting one file to other:

diff is used to display file differences. It tells us which lines in one file have to be changed to make the two files identical.

Ex: $diff file1 file3

3,4c3,4 #change line 3 of file1

< roja.r #}replace these

< gopi.s #}lines

- - - - - #with

> gopi.n #}these

> sekhar.s #}two lines

diff uses certain special symbols and instructions to indicate the changes that are required to make two files identical.

Maintaining several versions of a file: diff has an extremely useful option for the system administrator the –e option. If we have ten versions of a file differing only nominally, we now need to keep only one in full. For the others, we can keep only the differences, which help us conserve disk space.

tr (translate):
The *tr* (translate) command is a simple filter designed to replace one or more characters in given files with one or more characters. The syntax is:

Ex: tr [-cds] [in-string] [out-string]

The '-s' option substitutes all of the specified characters with another specified character and displays the results.

2.3 Hard Links and Symbolic Links

A link is a special file that references another file or directory. A link serves as an alias, which can be used to access the linked file/directory. For example, In Ubuntu, the link vi (in /usr/bin) references /etc/alternatives/vi.
Symlink: A symbolic link (or symlink or soft link) maintains a reference (not a direct pointer) to a file or a directory. A symlink is a file that stores the path to another file/directory. If the referenced file is removed, the symlink will be referencing an non-existent file.

To create a symbolic link, use command ln with option -s:

```
$ ln -s file-or-dir-name symlink-name
```

In "ls -l", symlinks are identified via the "symlink -> referenced file/dir".
Hard Link: A hard link is an *additional* pointer to the file's inode (physical location). You can view the inode of files via "ls -i". A file can be accessed via any hard link. It is available as long as there is at least one hard link left. The "hard link count" for a file is reflected in command "ls -l" (as illustrated in the earlier example).
To create a hard link, use command ln:

```
$ ln filename hard-link-name
```

Symlinks are more commonly-used. Symlinks can span file systems; while hard links work only in the same file system. Hard links usually work for file (hard link to directory could lead to inconsistency); while symlinks work for both file and directory. Hard links are also much hard to maintain, as there is little clue on where are the files, other than the link count.

You can remove a link (hard or soft) via rm command, just like any file. A hard-linked file is only deleted from the file system when there is no link to it.

Windows does not support symlink until Windows 7?!

2.4 Network Utilities

ftp: Copying files between non-unix hosts. It is the file transfer protocol.

```
Ex: $ftp source1        #open ftp connection to source1
Name :
Password :
```

Distributing processing:
Some hosts supply limited password less accounts with user ID's like 'user' so those explorers can roam the network without causing any harm.
Three utilities for distributing access are:

1. rlogin : Which allows you to log into a remote Unix host.
2. rsh : Which allows you to execute a command on a remote Unix host, and
3. telnet : Which allows you to execute commands on any remote host that has a telnet server.

Of these, 'telnet' is the most flexible, since there are other systems in addition to Unix that support telnet servers.

rlogin : Remote logins:To logon to the remote host, we use rlogin.
Syntax : rlogin –ec [-l useID] hostName
'rlogin' attempts to log you into the remote host *hostName*. If you don't supply a user ID by using the '-l' option, your local user ID is used during the login process.
If the remote host is not set as an equivalent of your local host in your "$Home/.rhost" file, your are asked for your password on the remote host.
Once you are connected, your local shell goes to sleep the remote shell starts to execute. When you are finished with the remote login shell, terminate it in the normal fashion(ctrl D) and your local shell will then awaken.
There are a few special "escape commands" each is preceded by the escape character, which is a tilde (~) by default, you may change this escape character by following the '-e' option with the preferred escape character.

```
Ex : $rlogin user2      #remote login
     Last login: Tue  Dec  20  17:23:51  from Varun
```

```
user2 % date
Wed  Dec  21  18:50:47  CDT  2005
user2 % ^D      #terminate the remote login shell
connection closed.
$-              #back home again at Varun
```

Here, we have logged into remote host 'user2' from local host 'Varun'.

Remote connection: *'telnet'* allows you to communicate with any remote host on the Internet that has a '*telnet*' server.

Syntax : telnet [host[port]]

telnet establishes a two way connection with a remote port. If you supply a host name, but not a port specifier, you are automatically connected to a *telnet* server on the specified host, which typically allows you to log into the remote machine. If you don't even supply a host name, *'telnet'* goes directly into command mode (like *ftp*).

arp : Address Resolution Protocol:

The arp command is used to manipulate system arp cache.

$arp options hostname

Options:

-d - deletes hostname

-a - displays entry of given host

-n - shows numerical address

In a networked system when a packet arrives at router machine then the IP address to Ethernet address mapping is needed. This is achieved by arp protocol. Theses mappings are stored in arp cache such that next time another packet arrives the same ip address then its Ethernet or physical address is calculated by carrying out a lookup operation on the arp cache. With arp command we can modify, view, delete the entries of this cache.

2.5 Tape Archive (tar) and ZIP Compression (gzip, bzip2)

The relevant file formats and utilities are:

- .gz: A file compressed via gzip utility; which can be uncompress via gunzip utility.
- .tar: A tar (tar archive) file, or tarball, is a collection of many files into a single file, used for distribution or backup. It is created via tar utility with option c (create); and can be extracted viatar with option x (extract).
- .tar.gz: A compressed tar file with gzip - the most popular format for software distribution and backup.

The Tape Archive (tar) Utility

You can use the tar utility to create, list, and extract .tar.gz files, as follows:

```
// "Create" a compressed archive of the given files and directories
$ tar czvf <compressed-filename>.tar.gz <directory1> <directory2> <file1> <file2> ...
  // Option "c" to create; "z" to compress; "v" for verbose; "f" for archive filename
```

Notes: to process .tar files without gzip compression (instead of .tar.gz), use the above commands minus the 'z' option.

gzip/gunzip Compression

gzip/gunzip (GNU zip) is an older compression utility. The resultant file type is .gz. gzip is still the most popular form for software distribution over the Internet. The man pages are also kept in gzip format. For example,

bzip2/bunzip2 Compression

bzip2/bunzip2 is a newer compression utility, which is more efficient than gzip, but not as popular. The resultant file type is .bz2. For example,

```
$ bzip2 -v listing.txt        // Compress to listing.txt.bz2 (-v for verbose)
$ bunzip2 listing.txt.bz2     // Uncompress
```

3. Exercises on Shell Programming

3.1 Displaying files

Write a shell script called displaying type extension file that prompts user to enter. The script shall read the type extension files and display all file names to output stream. For example,

```
Input:
Enter directory name: cfiles
Output:
f1.txt f2.txt add.c fibo.c
```

Hints

```
// For keyboard input
/**
 * 1. Prompt directory name
 * 2. Change to given directory name
 * 3. Using ls command to display .txt and .c files
 */
    echo directory name
    read dir   // Scan the keyboard for input
    cd $dir   // check the given file is directory or not if directory then change to new directory
    // displaying files
    ls *.txt *.c // ls command to list all files
```

Try

Write a shell script called displaying type extension file with detailed information (filename, permissions, no of links, owner name, group name, file created. The script shall read the type extension files and display all file names to output stream.

3.2 moving files

Write a shell script called moving files from one directory to another directory that prompts user to enter source directory and destination directory. The script shall read the directory files and move all files from source directory to destination directory. For example,

```
Input:
enter the  Existing directory
olddir
"enter new directory"
newlydir
Output:
f1.text  f2.c  g1.txt  g2.c  g3.txt  g4.c  mvf.sh  newlydirf1.txt f2.txt add.c fibo.c
```

Hints

```
// For keyboard input
/**
 * 1. Prompt source directory name
 * 2. Prompt destination directory name
 * 3. Test given directory name directory and existing or not
 * 4. Using mv command move files from source to destination directory
 */
```

```
mv ~/$dir/* ~/$ndir
cd $ndir // change directory to new directory
ls// ls command to list all files
```

Try

Write a shell script called moving all c extension files from one directory to another directory that prompts user to enter source directory, type extension files and destination directory. The script shall read the directory files and move all c type extension files from source directory to destination directory.

3.3 displaying logged in users

Write a shell script to display all the users who are currently logged in after a specified time. The script shall read the date and time, users who are logged in to linux server then displays users who logged in after specified time. For example,

```
Input:
enter the  Existing directory
olddir
"enter new directory"
newlydir
Output:
f1.text  f2.c  g1.txt  g2.c  g3.txt  g4.c  mvf.sh  newlydirf1.txt f2.txt add.c fibo.c
```

Hints

```
// For keyboard input
/**
 * 1. Prompt login time
 * 2. Use who command to list who are logged in linux server
 * 3. Extract time and user name
 * 4. Test the time with given time if condition is true then display time and username
 */
  echo "enter time to list specified users who login after specified time"
  read time1
  for i in `who|tr -s " " "|"|cut -d "|"-f1`
  do
  t=`w $i|tr -s " " ":"|tail -1|cut -d ":" -f4`
  for s in $t
  do
  if [ $time1 -le $s ]
  then
  echo $i $t
  fi
  done
  done
    cd $ndir // change directory to new directory
    ls// ls command to list all files
```

3.4 Wishing user based on time

Write a Shell Program to wish the user based on the login time. The script shall read the time of user logged in time and wish the user based on time. For example,

```
Input:
read the time of particular user
Output:
Good afternoon
```

Hints

```
// For keyboard input
/**
 * 1. Prompt source directory name
 * 2. Prompt destination directory name
 * 3. Test given directory name directory and existing or not
 * 4. Using mv command move files from source to destination directory
 */
     h=`date +%H`
    if [ $h -lt 12 ] wish good morning
     elif [ $h -lt 18 ] wish Good afternoon
     else wish Good evening
```

3.5 Searching for specified word

Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it. The script shall read the word and filenames and display all lines other than of matching word. For example,

```
Input:
sh we2.sh we2.sh
enter the word
echo
Output:
The lines displayed other than given word
```

Hints

```
// For keyboard input
/**
 * 1. Prompt the word
 * 2. Prompt the file name
 * 3. Read all files of given directory
 * 4. Display all lines other than given word
 */
grep -v "$word" $fname
else
echo "file doesnot exist"
//test given file is existing or not if existing display between lines
```

Try

Write a shell script that displaying all lines containing a specified word in one or more files supplied as arguments to it.

3.6 Displaying between lines

Write a shell script to read starting line and ending line of a file and display the lines in between them. For example,

```
Input:
enter file name
we1.sh
enter starting line number
2
enter ending line number
```

5
Output:
The lines displayed between line 2 and line 5 are

Hints

```
// For keyboard input
/**
 * 1. Prompt starting line
 * 2. Prompt ending line
 * 2. Find difference between two lines
 * 3. Using head and tail command of given file and display all between lines
 */
    d=`expr $el - $sl`
    head -$el $fname|tail -$d// test given file is existing or not if existing display between lines
```

Try

Write a shell script called displaying last 10 lines of given input file. The script shall read the line number and display all last lines.

4. Exercises on Shell Programming (Input, Decision and Loop)

4.1 Add2Integer (Input)

Write a shell script called Add2Integers that prompts user to enter two integers. The script shall read the two integers as int; compute their sum; and print the result. For example,

Enter first integer: **8**
Enter second integer: **9**
The sum is: 17

Hints

```
// For keyboard input
/**
 * 1. Prompt user for 2 integers
 * 2. Read inputs as "int"
 * 3. Compute their sum in "int"
 * 4. Print the result
 */
    // Put up prompting messages and read inputs as "int"
    echo enter a value and b value
    read a b   // Scan the keyboard for input
    // Compute sum
    sum=`expr $a + $b` or sum=$((a + b))

    // Display result
    echo "The sum is: $sum”   // Print with newline
```

Try

Write a shell script called Swap2Integers that prompts user for two integers. The script shall read the inputs as int, save in two variables called number1 and number2; swap the contents of the two variables; and print the results.

4.2 SumProductMinMax3 (Arithmetic & Min/Max)

Write a shell script called SumProductMinMax3 that prompts user for three integers. The script shall read the inputs as int; compute the sum, product, minimum and maximum of the three integers; and print the results. For example,

```
Enter 1st integer: 8
Enter 2nd integer: 2
Enter 3rd integer: 9
The sum is: 19
The product is: 144
The min is: 2
The max is: 9
```

Hints

```
// Prompt and read inputs as "int"
echo enter 3 numbers to find sum product min and max
read number1 number2 number3   // Scan the keyboard
// Compute sum and product use expr keyword
sum = ......
product = ......

// Compute min
// The "coding pattern" for computing min is:
// 1. Set min to the first item
// 2. Compare current min with the second item and update min if second item is smaller
// 3. Repeat for the next item
min = $number1        // Assume min is the 1st item
if [ $number2 -lt  $min ]   // Check if the 2nd item is smaller than current min
then
  min = $number2;     // Update min if so
else if [ $number3 -lt $min ]
then
// Continue for the next item
  min = $number3;
fi
fi
// Compute max - similar to min
......

// Print results
......
```

Try

1. Write a shell script called SumProductMinMax5 that prompts user for five integers. The script shall read the inputs as int; compute the sum, product, minimum and maximum of the five integers; and print the results. Use five int variables: number1, number2, ..., number5 to store the inputs.

4.3 IncomeTaxCalculator (Decision)

The progressive income tax rate is mandated as follows:

Taxable Income	Rate (%)
First $20,000	0
Next $20,000	10
Next $20,000	20
The remaining	30

For example, suppose that the taxable income is $85000, the income tax payable is $20000*0% + $20000*10% + $20000*20% + $25000*30%.

Write a shell script called **IncomeTaxCalculator** that reads the taxable income (in int). The script shall calculate the income tax payable (in double); and print the result rounded to 2 decimal places. For examples,

```
Enter the taxable income: $41234
The income tax payable is: $2246.80

Enter the taxable income: $67891
The income tax payable is: $8367.30

Enter the taxable income: $85432
The income tax payable is: $13629.60

Enter the taxable income: $12345
The income tax payable is: $0.00
```

Hints

```
// Declare constants first (variables may use these constants)
// The keyword "final" marked these as constant (i.e., cannot be changed).
// Use uppercase words joined with underscore to name constants
TAX_RATE_ABOVE_20K = 0.1
TAX_RATE_ABOVE_40K = 0.2
TAX_RATE_ABOVE_60K = 0.3
// Compute tax payable in "double" using a nested-if to handle 4 cases
if [ $taxableIncome -le 20000 ]         // [0, 20000]
then
  taxPayable = ......
else if [ $taxableIncome -le 40000)   // [20001, 40000]
then
  taxPayable = ......
else if [ $taxableIncome -le 60000 ]  // [40001, 60000]
then
  taxPayable = ......
else                          // [60001, ]
  taxPayable = ......
fi
// Alternatively, you could use the following nested-if conditions
// but the above follows the table data
//if [ $taxableIncome -gt 60000 ]         // [60001, ]
//then   ......
// else if [ $taxableIncome -gt 40000 ]   // [40001, 60000]
// then  ......
```

```
// else if [ $taxableIncome -gt 20000 ]   // [20001, 40000]
// then  ......
// else                          // [0, 20000]
//   ......
//
// Print results
Echo "The income tax payable is:
```

Try

Suppose that a 10% tax rebate is announced for the income tax payable, capped at $1,000, modify your program to handle the tax rebate. For example, suppose that the tax payable is $12,000, the rebate is $1,000, as 10% of $12,000 exceed the cap.

4.4 PensionContributionCalculatorWithSentinel (Decision & Loop)

Write a shellscript called PensionContributionCalculatorWithSentinel which shall repeat the calculations until user enter -1 for the salary. For examples,

```
Enter the monthly salary (or -1 to end): $5123
Enter the age: 21
The employee's contribution is: $1024.60
The employer's contribution is: $870.91
The total contribution is: $1895.51

Enter the monthly salary (or -1 to end): $5123
Enter the age: 64
The employee's contribution is: $384.22
The employer's contribution is: $461.07
The total contribution is: $845.30

Enter the monthly salary (or -1 to end): $-1
bye!
```

Hints

```
// Read the first input to "seed" the while loop
SENTINEL=-1
echo "Enter the monthly salary (or -1 to end): $"
read salary
while [ $salary -ne $SENTINEL]
do
  // Read the remaining
  echo "Enter the age:"
  read age
done
  ......
  ......

  // Read the next input and repeat
  Echo "Enter the monthly salary (or -1 to end): $")
  read salary
```

4.5 SalesTaxCalculator (Decision & Loop)

A sales tax of 7% is levied on all goods and services consumed. It is also mandatory that all the price tags should include the sales tax. For example, if an item has a price tag of $107, the actual price is $100 and $7 goes to the sales tax.

Write a shell script using a loop to continuously input the tax-inclusive price (in double); compute the actual price and the sales tax (in double); and print the results rounded to 2 decimal places. The script shall terminate in response to input of -1; and print the total price, total actual price, and total sales tax. For examples,

```
Enter the tax-inclusive price in dollars (or -1 to end): 107
Actual Price is: $100.00, Sales Tax is: $7.00
Enter the tax-inclusive price in dollars (or -1 to end): 214
Actual Price is: $200.00, Sales Tax is: $14.00
Enter the tax-inclusive price in dollars (or -1 to end): 321
Actual Price is: $300.00, Sales Tax is: $21.00
Enter the tax-inclusive price in dollars (or -1 to end): -1
Total Price is: $642.00
Total Actual Price is: $600.00
Total Sales Tax is: $42.00
```

Hints

```
// Declare constants
SALES_TAX_RATE = 0.07;
SENTINEL = -1;       // Terminating value for input
// Declare variables
totalPrice = 0.0, totalActualPrice = 0.0, totalSalesTax = 0.0  // to accumulate
......
// Read the first input to "seed" the while loop
echo "Enter the tax-inclusive price in dollars (or -1 to end): "
read price

while [ price -ne SENTINEL ]
do
  // Compute the tax
  ......
  // Accumulate into the totals
  ......
  // Print results
  ......
  // Read the next input and repeat
  echo "Enter the tax-inclusive price in dollars (or -1 to end):"
  echo $price
done
// print totals
......
```

4.6 ReverseInt (Loop with Modulus/Divide)

Write a shell script that prompts user for a positive integer. The script shall read the input as int; and print the "reverse" of the input integer. For examples,

```
Enter a positive integer: 12345
The reverse is: 54321
```

Hints

Use the following *coding pattern* which uses a while-loop with repeated modulus/divide operations to extract and drop the last digit of a positive integer.

```
// Extract and drop the "last" digit repeatably using a while-loop with modulus/divide operations
while [ $inNumber -gt 0 ]
  inDigit=`expr $inNumber % 10` // extract the "last" digit
  // Print this digit (which is extracted in reverse order)
  ......
  inNumber=`expr $isNumber / 10`          // drop "last" digit and repeat
}
......
```

Try

Write a shell script that prompts user for a positive integer. The script shall read the input as int; compute and print the sum of all its digits.

4.7. Amicable Numbers

Two different numbers are said to be so Amicable numbers if each sum of divisors is equal to the other number. Amicalble Numbers are: (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368). For example,

```
Enter 1st number: 228
Enter 2nd number: 220
The numbers are Amicable Numbers.
```

Hints

```
220 and 284 are Amicable Numbers.
Divisors of 220 = 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110
1+2+4+5+10+11+20+22+44+55+110 = 284
Divisors of 284 = 1, 2, 4, 71, 142
1+2+4+71+142 = 220
```

Try

Write a shell script called **TimeTable** that prompts user for the size (a positive integer in int); and prints the multiplication table as shown:

```
Enter the size: 10
 *|  1  2  3  4  5  6  7  8  9 10
-------------------------------------------
 1|  1  2  3  4  5  6  7  8  9 10
 2|  2  4  6  8 10 12 14 16 18 20
 3|  3  6  9 12 15 18 21 24 27 30
 4|  4  8 12 16 20 24 28 32 36 40
 5|  5 10 15 20 25 30 35 40 45 50
 6|  6 12 18 24 30 36 42 48 54 60
 7|  7 14 21 28 35 42 49 56 63 70
 8|  8 16 24 32 40 48 56 64 72 80
 9|  9 18 27 36 45 54 63 72 81 90
10| 10 20 30 40 50 60 70 80 90 100
```

4.8. Capricorn Number

A number is called Capricorn or Kaprekar number whose square is divided into two parts in any conditions and parts are added, the additions of parts is equal to the number, is called Capricorn or Kaprekar number. For example,

```
Enter a number : 45
45 is a Capricorn/Kaprekar number
Enter a number : 297
297 is a Capricorn/Kaprekar number
Enter a number : 44
44 is not a Capricorn/Kaprekar number
```

Hints

```
Number = 45
(45)2  = 2025

All parts for 2025:
202    + 5   =    207 (not 45)
20     + 25  =    45
2+  025    =    27 (not 45)
```

From the above we can see one combination is equal to number so that 45 is Capricorn or Kaprekar number.

Try

Write a shell script to generate and show all Kaprekar numbers less than 1000.

5. Exercises on Simulating commands - I

5.1 simulating cat command

Write a program called simulating cat command that reads the file name and displays the file content to output stream.

```
Input:
Enter the filename: file1
Output:
# displays the content of given file1 to output stream
```

Hints

```
#include<unistd.h>
// open function used to open existing file
open(argv[1],O_RDONLY);// file name read from command line arguments
while((n=read(fd,&buf,1))>0)// while loop to read content from given file character by
{                    character
 write(1,&buf,1);// Use write() to display characters to output stream


      ......
}      // repeat the while loop upto end of file
```

Try

Rewrite the above program to append new content to existing file.

5.2 simulating cp command

Write a program called simulating cp command that create duplicate file of exiting file, reads the file name and writes the file content to new file, repeats the procedure up to end of file.

```
Input:
Reads existing file
Output:
# creates duplicate file and copy content from source file to new file
```

Hints

```
#include<unistd.h>
// open function used to open existing file
fd1=open(argv[1],O_RDWR);// read existing file
fd2=open(argv[2],O_RDWR);//create or open new file
while((n=read(fd,&buf,1))>0)// while loop to read content from given file character by
{                    character
 write(fd2,buf,sizeof(buf));// Use write() to write characters to new file


      .....
 }      // repeat the while loop up to end of file
```

Try

Rewrite the above program to simulate cp -R command to recursive copy (including hidden files.

5.3 Simulating rm command

Write a program called simulating rm command that remove exiting file or group of files of given directory, reads the file name or group of filenames and directory name then removes that related files from given directory using rm command.

```
Input:
Reads existing file
Output:
# removes source file
```

Hints

```
#include<unistd.h>
//system function executes linux commands
system(cd dir) //move to specific directory
remove("abc.txt");// remove function deletes existing file
```

Try

Rewrite the above program to simulate rm -i command for interactive remove existing file or group of files.

5.4 Simulating ls command

Write a program called simulating ls command using low level system calls, that reads particular directory and displays the list files, subdirectories and executable files to output stream.

```
Input:
Enter the filename: file1
Output:
# displays the content of given file1 to output stream
```

Hints

```
#include<unistd.h>
// opendir function used to open existing directory
DIR *dp;
struct dirent *p;
dp=opendir("dir1");// opens dir is directory name
while((p=readdir(dp))!=NULL) // while loop to read files from given directory
    {
```

```
    printf("%d\t",p->d_ino); // displays inode number of file
    printf("%s\n",p->d_name);// displays name of the file
        ......
  }      // repeat the while loop upto end of directory file
    close(dir); //closes open directory
```

Try

Rewrite the above program to implement ls -l command to display long listing files.

6. Exercises on Simulating commands - II

6.1 Simulating head command

Write a program called simulating head command using system calls, that reads existing file name and displays the first 10 lines content of file to output stream.

```
Input:
gcc headcmd.c -o headcmd
./headcmd
Output:
# displays the content of given file1 first 10 lines to output stream
```

Hints

```
#include<unistd.h>
 // reads file name from command line arguments
 open(argv[1],O_RDONLY);");// open function opens file for read only purpose
 while(lseek(read_fd,offset, SEEK_SET) < statFd.st_size)
  // lseek function to move cursor to particular location of given file
  {
    if(read(read_fd, &lu, 1) != -1)
    {// reads line by line
          . . . .
       printf("%c",lu);// displays content line by line
          ......
    }      // repeat the while loop upto condition fail
     close(fd); //closes open file
  }
```

Try

Rewrite the above program to implement head -20 command to display top 20 lines.

6.2 Simulating tail command

Write a program called simulating tail command using system calls, that reads existing file name and displays the last 10 lines content of file to output stream.

```
Input:
gcc tailcmd.c -o tailcmd./headcmd
```

./tailcmd file

Output:

```
# displays the content of given file last 10 lines to output stream
```

Hints

```
#include<unistd.h>
 // reads file name from command line arguments
/* Allocate space for tail buffer */
 tail = calloc(count, sizeof(char *));
 for (i = 0; i < count; i++) {
   tail[i] = calloc(MAX_LINE_LEN, sizeof(char));
 }
/* Fill circular tail buffer until EOF */
while (fgets(tail[tailX], MAX_LINE_LEN, stdin) != NULL) {
   tailX = (tailX + 1) % count;
   if (tailX == headX) {
     headX = (headX + 1) % count;
   }
 }
/* Display tail */
i = tailX;
do {
  printf("%s", tail[i]);
  i = (i + 1) % count;
  } while (i != tailX);
return 0;
    }
```

Try

Rewrite the above program to implement tail command to display last lines with option, option is number of lines take as input.

7. Exercises on Simulating commands - III

7.1 Simulation of mv command

Write a program called **simulation of mv command**, which prompts user for source file and destination file to rename file. The output shall look like:

```
Input:
$gcc mvcmd.c -o mvcmd
$/mvcmd xx movefile
Output:
#renames source file to destination file.
```

Hints

```
    #include<unistd.h>
     // Define variables
     int main()
     {
        ........
     open(argv[1],O_RDONLY);// opens existing file
     creat(argv[2],S_IWUSR);// creates new file
     rename(fd1,fd2); // rename source file with new file
     unlink(argv[1]);//removes existing file
        ........
     }
```

Try

Rewrite the above program to implement mv command to move files from source directory to new directory.

7.2 Simulation of nl command

Write a program called simulation of nl command, which prompts the user for a file name and read line by line and display line number before each line. For example,

```
Input: $ gcc nlprg.c -o nlprg
$ ./nlprg sample1
  1 aaaaaaaaaa
  2 sssssssssss
  3 ddddddddddd
  4 fffffffffff
  5 ggggggggggg
```

Hints

```
 #include<unistd.h>
 // Define variables
  int main()
 {
  while(lseek(read_fd,offset, SEEK_SET) < statFd.st_size)
  {
      if(read(read_fd, &lu, 1) != -1)
      {
        printf("%c",lu);
        offset++;
          if(lu=='\n')
          {
          printf("  %d",++counter);          //counter++;
          }
      }
              . . . . .
    }
 }
```

Try

Rewrite the above program to implement nl command to display special symbol before each line.

8. Exercises on Signal Handling

8.1 Signal handler function with SIGINT

Write a program called signal handler to catch SIGINT, SIGINT signal which interrupts the process by pressing ctrl + c. For example,

```
Input
$ gcc signalpg.c -o sp
Output:
$ ./sp
received SIGINT
```

Hints

```
// Declare variables
#include<unistd.h>
void sig_handler(int signo) // function definition
 {
   if (signo == SIGINT)
   printf("received SIGINT\n");
 }
 int main(void)
 {        .....
    if (signal(SIGINT, sig_handler) == SIG_ERR) // function calling
    printf("\ncan't catch SIGINT\n");
    // A long long wait so that we can easily issue a signal to this process
    while(1)
    sleep(1 );            .....
 }
```

8.2 Signal handler function with SIGDFL

Write a program called signal handler to catch SIGINT, SIGINT signal which interrupts the process by pressing ctrl + c, SIG_DFL signal do default action i.e. terminates the current process. For example,

```
Input
$ gcc signalpg.c -o sp
Output:
$ ./sp
#Press ctrl + c, Process terminates automatically.
```

Hints

```
// Declare variables
#include<unistd.h>
void sig_handler(int signo) // function definition
 {
    if (signo == SIGINT)
    printf("received SIGINT\n");
 }
 int main(void)
 {        .....
    if (signal(SIGINT, SIG_DFL) == SIG_ERR) //default action
    printf("\ncan't catch SIGINT\n");
    // A long long wait so that we can easily issue a signal to this process
    while(1)
    sleep(1 );            .....
  }
```

8.3 Signal handler function with SIGKILL

Write a program called signal handler to catch SIGINT, SIGINT signal which interrupts the process by pressing ctrl + c, SIG_DFL signal do default action i.e. terminates the current process. For example,

```
Input
$ gcc signalpg.c -o sp
Output:
```

```
$ ./sp
#Press ctrl + DEL, Process ignores SIGKILL signal.
```

Hints

```
#include<unistd.h>
void sig_handler(int signo) // function definition
 {
   if (signo == SIGINT)
   printf("received SIGINT\n");
  }
 int main(void)
 {       ......
   if (signal(SIGKILL, SIG_IGN) == SIG_ERR) //ignoring signal
   printf("\ncan't catch SIGINT\n");
   // A long long wait so that we can easily issue a signal to this process
   while(1)
   sleep(1 );
   ......
  }
```

Try

Rewrite the above program to generate SIGABRT signal and call function to display message process aborted.

9. Exercises on Inter Process Communication (IPC)

9.1 One way communication using pipe

Write a program to implement one way communication using pipes. The program shall execute pipe function, stores data at one end reads the data from other end, fork function creates new process and exchange data between related process. For examples,

```
Input:
$ gcc pipepg.c -o pp
$./pp hello
Ouput:world
```

Hints

Use the following *coding pattern* which uses a pipe, fork function to exchange data between to related process.

```
 # one way communication using pipes
 int main(void)
  {
 # declare variables
 if (pipe(fd) < 0) //execute pipe function to create pipe
 printf("pipe error");
 if ((pid = fork()) < 0) // creates new process is called child process
 {
 printf("fork error");
 }
 else if (pid > 0) // parent process writes the data to pipe one end
  { /* parent */
     . . . .
```

```
    write(fd[1], "hello world\n", 12);
  } else
  { /* child */
  . . . .
  read(fd[0], line, MAXLINE); // child process reading data from pipe
  write(STDOUT_FILENO, line, n); // displaying data to output stream
 }
```

Try

Rewrite the above program to implement two way communication using pipes. The program shall execute two pipe functions, one fork function exchange data between related process between parent and child process.

9.2 One way communication using fifo function

Write a program to implement one way communication using fifo function. The program shall execute mkfifo function, creates fifo file, stores data in fifo file by one process and other process has to read by other process, it exchanges data between un-related process. For example,

```
Input:
$ gcc producer.c -o producer          #first window
$gcc consumer.c  -o consumer          # second window
Output:
$ ./producer                        #first window
$./consumer                          # second window
Producer:
Producer sent: hai                    #first window
Consumer read: hai                    # second window
Producer sent: good morning           #first window
Consumer read: good morning           # second window
Producer sent: welcome                #first window
Consumer read: welcome                # second window
$./pp hello
Ouput:world
```

Hints

Use the following *coding pattern* which uses a mkfifo function to create fifo file and exchange data between to un-related process.

```
# one way communication using fifos
//FIFO or Unnamed(Child Process)
#include<unistd.h>
int main()
{
 //declare variables
 open(FIFO_NAME, O_RDONLY);//Open existing fifo file
 while(1) //repeat while loop upto reading end of file
{
 n=read(fd, r, MAXSIZE);
 if(n > 0)
 printf("\nConsumer read: %s", r);
}
}

//FIFO or Unnamed(Parent Process)
```

```
#include<unistd.h>
int main()
{
// declare variables
fifoid=mkfifo(FIFO_NAME, 0755); // create fifo file
while(1)
{
 // write content to fifo file
 read(0, w, MAXSIZE);
 n=write(fd, w, MAXSIZE);
 if(n > 0)
 printf("\nProducer sent: %s", w);
}
}
```

Try

Rewrite the above program to implement two way communication using fifos.

10. Exercises on Message Queues

10.1 storing messages in Message queues (sender)

Write a program (sender.c) to create a message queue with read and write permissions to write 3 messages to it with sequence order. For example,

Input:
$gcc msend.c -o msend
Output:
$./msend
Enter the message to send: hi
Enter the message to send: hello, how are you
Enter the message to send: bye

Hints

Create message queue or open an existing using msgget() which returns a common key in order to gain access to the queue and communicating between different process using msgsnd() and msgrcv(). Each message is given an identification or type so that processes can select the appropriate message.

```
#include<unistd.h>
#include<string.h>
int main()
{
//declaring variables
struct
{
 long mtype;
 char mtext[15]; //declaring message structure
 }message;
 msgget((key_t)10,IPC_CREAT|0666);  //create message queue with key 10
 msgsnd(qid,&message,len+1,0); //store messages in existing message queue
}
```

Try

Rewrite the above program (sender.c) to create a message queue with read and write permissions to write 3 messages to it with different priority numbers.

10.2 Retrieving messages from message queues (receiver)

Write a C program (receiver.c) that receives the messages (from the above message queue as specified and displays them to output stream. For example

```
Input:
$ $gcc mrecv.c -o mrecv
Output
$./mrecv
Message received from sender is: hi
Message received from sender is: hello, how are you
Message received from sender is: bye
```

Hints

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by msgget(). New messages are added to the end of a queue by msgsnd().Messages are fetched from a queue by msgrcv(). Fetch the messages in a first-in, first-out order. All processes can exchange information through access to a common system message queue. Process must share a common key in order to gain access to the queue in the first place.

```
#include<sys/ipc.h>
#include<sys/msg.h>
main()
{
//declaring variables
struct
{
long mtype;
char mtext[15]; //declaring message structure
}buff;
msgget((key_t)10,IPC_CREAT|0666);
        //accessing existing message queue with key 10
if(msgrcv(qid,&buff,15,i,0)==-1)//retrieve messages from message queue
printf("Message received from sender is %s\n",buff.mtext);
  //displaying messages to output stream
}
```

Try

Rewrite the above program (receiver.c) to access existing message queue with read and write permissions to retrieve and display 3 messages to output stream with different priority numbers.

11. Exercises on Shared Memory

11.1 Sharing memory segment between processes

To write a program to implement inter process communication using shared memory. The program creates shared memory segment and stores data, multiple users can access data at the same time. For example:

```
Input:cc shmem.c
Output:./a.out
Write Data :hai
child: Data read from memory: hai
parent: Data written in memory: hai
```

Hints

Shared memory is the fastest method of interprocess communication (IPC) under Linux and other Unix-like systems. The system provides a shared memory segment which the calling process can map to its address space. After that, it behaves just like any other part of the process's address space due to which as soon as the first process writes data in the shared memory segment, it becomes available to the second process..

```
#include <sys/ipc.h>
#include <sys/shm.h>
int main()
{
   // ftok to generate unique key
   key_t key = ftok("shmfile1",10); //generates key
   int child=fork();
if(!child)
{
   // shmget returns an identifier in shmid
   int shmid = shmget(key,1024,0666|IPC_CREAT);
   // shmat to attach to shared memory
   char *str = (char*) shmat(shmid,(void*)0,0);
   printf("Write Data : ");
   gets(str);
   printf("parent: Data written in memory: %s\n",str);
}
   else
   {    // ftok to generate unique key
   key_t key = ftok("shmfile1",10);

   // shmget returns an identifier in shmid
   int shmid = shmget(key,1024,0666|IPC_CREAT);
   // shmat to attach to shared memory
   char *str = (char*) shmat(shmid,(void*)0,0);
   printf("child :Data read from memory: %s\n",str);
   //detach from shared memory
   shmdt(str);
   // destroy the shared memory
   shmctl(shmid,IPC_RMID,NULL);
  }
}
```

Try

Rewrite the above program to implement inter process communication using shared memory without fork function. The program creates shared memory segment and stores data (sender process), multiple users can access data (receiver process) at the same time. For example:

12. Exercises on Socket Programming

12.1 echo Client Server Program using TCP elementary functions

Write client and server programs (using c) for interaction between server and client processes using TCP Elementary functions. For example,

```
Input:
$ gcc TCP_server.c -o server          # first window
$gcc TCP_client.c -o client           # second window
Output:
$ ./server                        # first window
```

```
Listening                    # first window
$./client                    # second window
Data received: Hello World          # second window
```

Hints

Socket used to communication between unrelated process which are connected in the network. TCP is connection-oriented protocol. Data send and receive in TCP (Transmission Control Protocol) is reliable. Sockets are the "virtual" endpoints of any kind of network communications done between 2 hosts over in a network. A socket is a combination of IP address and port on one system. So on each system a socket exists for a process interacting with the socket on other system over the network. Three way handshake is the procedure that is followed to establish a TCP connection between two remote hosts.

```
##server side program
#include <sys/socket.h>
#include <netinet/in.h>
 int main(){
 // declare variables
 struct sockaddr_in serverAddr;
 struct sockaddr_storage serverStorage;
 socklen_t addr_size;
 /*---- Create the socket. The three arguments are: ----*/
 /* 1) Internet domain 2) Stream socket 3) Default protocol*/
 welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);
 /*---- Configure settings of the server address struct ----*/
 /* Address family = Internet */
 serverAddr.sin_family = AF_INET;
 /* Set port number, using htons function to use proper byte order */
 serverAddr.sin_port = htons(7891);
 /* Set IP address to localhost */
 serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
 /* Set all bits of the padding field to 0 */
 memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
 /*---- Bind the address struct to the socket ----*/
 bind(welcomeSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
 /*---- Listen on the socket, with 5 max connection requests queued ----*/
 if(listen(welcomeSocket,5)==0)
   printf("Listening\n");
 else
   printf("Error\n");
 /*---- Accept call creates a new socket for the incoming connection ----*/
 addr_size = sizeof serverStorage;
 newSocket = accept(welcomeSocket, (struct sockaddr *) &serverStorage, &addr_size);
 /*---- Send message to the socket of the incoming connection ----*/
 strcpy(buffer,"Hello World\n");
 send(newSocket,buffer,13,0);
 return 0;
}
##client side program
int main()
{
 //declaring variables
 struct sockaddr_in serverAddr;
 socklen_t addr_size;
 /*---- Create the socket. The three arguments are: ----*/
```

```
  /* 1) Internet domain 2) Stream socket 3) Default protocol*/
  clientSocket = socket(PF_INET, SOCK_STREAM, 0);
  /*---- Configure settings of the server address struct ----*/
  /* Address family = Internet */
  serverAddr.sin_family = AF_INET;
  /* Set port number, using htons function to use proper byte order */
  serverAddr.sin_port = htons(7891);
  /* Set IP address to localhost */
  serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
  /* Set all bits of the padding field to 0 */
  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
  /*---- Connect the socket to the server using the address struct ----*/
  addr_size = sizeof serverAddr;
  connect(clientSocket, (struct sockaddr *) &serverAddr, addr_size);
  /*---- Read the message from the server into the buffer ----*/
  recv(clientSocket, buffer, 1024, 0);
  /*---- Print the received message ----*/
  printf("Data received: %s",buffer);
  return 0;
}
```

Try

Write client and server programs (using c) for interaction between server and client processes using TCP Elementary functions client request particular file and server responds with file content.

12.2 Client Server Program using UDP elementary functions

Write client and server programs (using c) for interaction between server and client processes using UDP Elementary functions to convert given string to upper case.

```
Input:
$ gcc UDP_server.c -o server                    # first window
$gcc UDP_client.c -o client                     # second window
Output:
$ ./server                                      # first window
$./client                                       # second window
Type a sentence to send to server:              # second window
good morning                                    # second window
You typed: good morning                         # second window
Received from server: GOOD MORNING              # first window
Type a sentence to send to server:              # second window"
```

Hints

UDP there is no need of system calls listen and accept since it is a connectionless protocol

```
 /************* UDP SERVER CODE *******************/
#include <sys/socket.h>
int main()
{
  //declare variables
  struct sockaddr_in serverAddr, clientAddr;
  struct sockaddr_storage serverStorage;
```

```
  socklen_t addr_size, client_addr_size;
  /*Create UDP socket*/
  udpSocket = socket(PF_INET, SOCK_DGRAM, 0);
  /*Configure settings in address struct*/
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(7891);
  serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
  /*Bind socket with address struct*/
  bind(udpSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
  /*Initialize size variable to be used later on*/
  addr_size = sizeof serverStorage;
  while(1)
{
/* Try to receive any incoming UDP datagram. Address and port of
      requesting client will be stored on serverStorage variable */
    nBytes = recvfrom(udpSocket,buffer,1024,0,(struct sockaddr *)&serverStorage, &addr_size);
/*Convert message received to uppercase*/
    for(i=0;i<nBytes-1;i++)
       buffer[i] = toupper(buffer[i]);
/*Send uppercase message back to client, using serverStorage as the address*/
    sendto(udpSocket,buffer,nBytes,0,(struct sockaddr *)&serverStorage,addr_size);
  }
  return 0;
}
/************* UDP CLIENT CODE *******************/
#include <sys/socket.h>
int main()
{
  //declare variables
  struct sockaddr_in serverAddr;
  socklen_t addr_size;
  /*Create UDP socket*/
  clientSocket = socket(PF_INET, SOCK_DGRAM, 0);
  /*Configure settings in address struct*/
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(7891);
  serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
  /*Initialize size variable to be used later on*/
  addr_size = sizeof serverAddr;
  while(1)
  {
    printf("Type a sentence to send to server:\n");
    fgets(buffer,1024,stdin);
    printf("You typed: %s",buffer);
    nBytes = strlen(buffer) + 1;
    /*Send message to server*/
    sendto(clientSocket,buffer,nBytes,0,(struct sockaddr *)&serverAddr,addr_size);
    /*Receive message from server*/
    nBytes = recvfrom(clientSocket,buffer,1024,0,NULL, NULL);
    printf("Received from server: %s\n",buffer);
```

```
    }
}
```

Try

Write client and server programs (using c) for interaction between server and client processes using UDP Elementary functions client sends number to server and server runs reverse of given number and responds with result.

V. TEXT BOOKS:

7. Sumitabha Das, "*Your Unix The Ultimate Guide"*, Tata McGraw-Hill, New Delhi, India, 2007.
8. B. A. Forouzan and R. F. Gilberg, "*Unix and Shell Programming"*, Cengage Learning.

VI. REFERENCE BOOKS:

1. Robert Love, "*Linux System Programming"*, O'Reilly, SPD.
2. Stephen G. Kochan, Patrick Wood, "*Unix Shell Programming"*, Sams publications, 3rd Edition, 2007.
3. T. Chan, "*Unix System Programming using C++",* Prentice Hall India, 1999.

VII. WEB REFERENCES:

1. http://spoken-tutorial.org/tutorialsearch/?search_foss=Linux&search_language=English
2. https://www.redhat.com/en/files/resources/en-rhel-whats-new-in-rhel-712030417.pdf
3. http:// www.tutorialspoint.com/unix/
4. http://cse09-iiith.virtual-labs.ac.in

VIII. MATERIALS ONLINE

1. Course Template
2. Lab Manual

ESSENCE OF INDIAN TRADITIONAL KNOWLEDGE

III Semester: Common for all branches

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC10	Mandatory	-	-	-	-	-	-	-
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: Nil		

Prerequisite: No Prerequisites

COURSE OBJECTIVES:

The course should enable the students to:

I. Understand the concept of Traditional knowledge and its importance
II. Know the need and importance of protecting traditional knowledge.
III. Know the various enactments related to the protection of traditional knowledge.
IV. Understand the concepts of Intellectual property to protect the traditional knowledge

MODULE-I	INTRODUCTION TO TRADITIONAL KNOWLEDGE

Define traditional knowledge, nature and characteristics, scope and importance, kinds of traditional knowledge, the physical and social contexts in which traditional knowledge develop, the historical impact of social change on traditional knowledge systems. Indigenous Knowledge (IK), characteristics, traditional knowledge vis-à-vis indigenous knowledge, traditional knowledge Vs western knowledge traditional knowledge vis-à-vis formal knowledge

MODULE-II	PROTECTION OF TRADITIONAL KNOWLEDGE

Protection of traditional knowledge: The need for protecting traditional knowledge Significance of TK Protection, value of TK in global economy, Role of Government to harness TK.

MODULE-III	LEGAL FRAMEWORK AND TK

A: The Scheduled Tribes and Other Traditional Forest Dwellers (Recognition of Forest Rights) Act, 2006, Plant Varieties Protection and Farmer's Rights Act, 2001(PPVFR Act);

B: The Biological Diversity Act 2002 and Rules 2004, the protection of traditional knowledge bill, 2016. Geographical indicators act 2003.

MODULE-IV	TRADITIONAL KNOWLEDGE AND INTELLECTUAL PROPERTY

Systems of traditional knowledge protection, Legal concepts for the protection of traditional knowledge, Certain non IPR mechanisms of traditional knowledge protection, Patents and traditional knowledge, Strategies to increase protection of traditional knowledge, global legal FORA for increasing protection of Indian Traditional Knowledge.

MODULE-V	TRADITIONAL KNOWLEDGE IN DIFFERENT SECTORS:

Traditional knowledge and engineering, Traditional medicine system, TK and biotechnology, TK in agriculture, Traditional societies depend on it for their food and healthcare needs, Importance of conservation and sustainable development of environment, Management of biodiversity, Food security of the country and protection of TK. 139.

Text Books:

1. Traditional Knowledge System in India, by Amit Jha, 2009.
2. Traditional Knowledge System and Technology in India by Basanta Kumar Mohanta and Vipin Kumar Singh Pratibha Prakashan 2012.

Reference Books:

1. Traditional Knowledge System in India by Amit Jha Atlantic publishers, 2002.
2. "Knowledge Traditions and Practices of India" Kapil Kapoor1, Michel Danino2

FOUNDATIONS OF CYBER SECURITY

III Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC02	Core	3	1	0	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisites: There are no prerequisites to take this course.

I. COURSE OVERVIEW:

This course provides learners with a baseline understanding of common cyber security threats, Vulnerabilities and risks. An overview of how basic cyber attacks are constructed and applied to real systems is also included. Examples include simple Unix kernel hacks, Internet worms, and Trojan horses in software utilities. Network attacks such as distributed denial of service (DDOS) and botnet-attacks are also described and illustrated using real time examples from the past couple of decades.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The various types of cyber-attacks and cyber-crimes.
II. The threats and risks within context of the cyber security.
III. An overview of the cyber laws and concepts of cyber forensics.
IV. The defensive techniques against these attacks.

III. SYLLABUS:

Module I - Introduction to Cyber Security

Basic Cyber Security Concepts, layers of security, Vulnerability, threat, Harmful acts, Internet Governance – Challenges and Constraints, Computer Criminals, CIA Triad, Assets and Threat, motive of attackers, active attacks, passive attacks, Software attacks, hardware attacks, Cyber Threats-Cyber Warfare, Cyber Crime, Cyber terrorism, Cyber Espionage, etc., Comprehensive Cyber Security Policy.

Module II - Cyber Space and the Law & Cyber Forensics

Introduction, Cyber Security Regulations, Roles of International Law. The INDIAN Cyberspace, National Cyber Security Policy. Introduction, Historical background of Cyber forensics, Digital Forensics Science, The Need for Computer Forensics, Cyber Forensics and Digital evidence, Forensics Analysis of Email, Digital Forensics Lifecycle, Forensics Investigation, Challenges in Computer Forensics.

Module III - Cyber Crime : Mobile and Wireless Devices

Introduction, Proliferation of Mobile and Wireless Devices, Trends in Mobility, Credit card Frauds in Mobile and Wireless Computing Era, Security Challenges Posed by Mobile Devices, Registry Settings for Mobile Devices.

Authentication service Security, Attacks on Mobile/Cell Phones, Organizational security Policies and Measures in Mobile Computing Era, Laptops.

Module IV - Cyber Security: Organizational Implications

Introduction: cost of cybercrimes and IPR issues, web threats for organizations, security and privacy implications, social media marketing: security risks and perils for organizations, social computing and the associated challenges for organizations.

Module V - Cybercrime: Mini-Cases And Examples

Official Website of Maharashtra Government Hacked, Indian Banks Lose Millions of Rupees, Parliament Attack, Pune City Police Bust Nigerian Racket, e-mail spoofing instances. Mini-Cases: The Indian Case of online Gambling, An Indian Case of Intellectual Property Crime, Financial Frauds in Cyber Domain.

IV. TEXT BOOKS:

1. Nina Godbole and SunitBelpure, “Cyber Security Understanding Cyber Crimes,Computer Forensics and Legal Perspectives” , Wiley publications.
2. B.B.Gupta, D.P.Agrawal, HaoxiangWang, “Computer and Cyber Security: Principle s, Algorithm, Applications, and Perspectives”, CRC Press, ISBN 9780815371335, 2018.

V. REFERENCE BOOKS:

1. James Graham, Richard Howard and Ryan Otson, “Cyber Security Essentials”, CRC Press.
2. Chwan-Hwa(john) Wu,J. David Irwin, “Introduction to Cyber Security”, CRC Press T&F Group.

DESIGN AND ANALYSIS OF ALGORITHMS

IV Semester: CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC13	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: Programming for Problem Solving, Data Structures

I. COURSE OVERVIEW:

Design and analysis of algorithms is the process of finding the computational complexity of algorithms. It helps to design and analyze the logic on how the algorithm will work before developing the actual code for a program. It focuses on introduction to algorithm, asymptotic complexity, sorting and searching using divide and conquer, greedy method, dynamic programming, backtracking, branch and bound. NP-hard and NP-complete problems. The applications of algorithm design are used for information storage, retrieval, transportation through networks, and presentation to users.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Assess how the choice of data structures and algorithm design methods impacts the performance of programs.
II. Solve problems using data structures such as binary search trees, and graphs and writing programs for these solutions.
III. Choose the appropriate data structure and algorithm design method for a specified application.
IV. Solve problems using algorithm design methods such as the greedy method, divide and conquer, dynamic programming, backtracking, and branch and bound and writing programs for these solutions.

III. SYLLABUS:

MODULE – I: INTRODUCTION

Algorithm: Pseudo code for expressing algorithms; Performance analysis: Space complexity, time complexity; Asymptotic notations: Big O notation, omega notation, theta notation and little o notation, amortized complexity; Divide and Conquer: General method, binary search, quick sort, merge sort, Strassen‘s matrix multiplication.

MODULE – II: SEARCHING AND TRAVERSAL TECHNIQUES

Disjoint set operations, union and find algorithms; Efficient non recursive binary tree traversal algorithms, spanning trees; Graph traversals: Breadth first search, depth first search, connected components, biconnected components.

MODULE – III: GREEDY METHOD AND DYNAMIC PROGRAMMING

Greedy method: The general method, job sequencing with deadlines, knapsack problem, minimum cost spanning trees, single source shortest paths.

Dynamic programming: The general method, matrix chain multiplication optimal binary search trees, 0/1 knapsack problem, single source shortest paths, all pairs shortest paths problem, the travelling salesperson problem.

MODULE – IV: BACKTRACKING AND BRANCH AND BOUND

Backtracking: The general method, the 8 queens problem, sum of subsets problem, graph coloring, Hamiltonian cycles; Branch and bound: The general method, 0/1 knapsack problem, least cost branch and bound solution, first in first out branch and bound solution, travelling salesperson problem.

MODULE – V: NP-HARD AND NP-COMPLETE PROBLEMS

Basic concepts: Non-deterministic algorithms, the classes NP - Hard and NP, NP Hard problems, clique decision problem, chromatic number decision problem, Cook's theorem.

IV. TEXT BOOKS:

1. Ellis Horowitz, Satraj Sahni, Sanguthevar Rajasekharan, "Fundamentals of Computer Algorithms", Universities Press, 2nd Edition, 2015.
2. Alfred V. Aho, John E. Hopcroft, Jeffrey D, "The Design And Analysis Of Computer Algorithms", Pearson India, 1st Edition, 2013.

V. REFERENCE BOOKS:

1. Levitin A, "Introduction to the Design and Analysis of Algorithms", Pearson Education, 3rd Edition, 2012.
2. Goodrich, M. T. R Tamassia, "Algorithm Design Foundations Analysis and Internet Examples", John Wileyn and Sons, 1st Edition, 2001.
3. Base Sara Allen Vangelder, "Computer Algorithms Introduction to Design and Analysis", Pearson, 3rd Edition, 1999.

VI. WEB REFERENCES:

1. http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
2. http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
3. http://www.facweb.iitkgp.ernet.in/~sourav/daa.html

DATABASE MANAGEMENT SYSTEMS

IV Semester: CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC05	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: Programming for Problem Solving, Data Structures

I. COURSE OVERVIEW:

The purpose of this course is to provide a clear understanding of fundamentals with emphasis on their applications to create and manage large data sets. It highlights on technical overview of database software to retrieve data from n database. The course includes database design principles, normalization, concurrent transaction processing,, security, recovery and file
organization techniques.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Understand the role of database management system in an organization and learn the database concepts.
II. Design databases using data modeling and Logical database design techniques.
III. Construct database queries using relational algebra and calculus and SQL.
IV. Understand the concept of a database transaction and related concurrent, recovery facilities.
V. Learn how to evaluate a set of queries in query processing.

III. SYLLABUS:

MODULE – I: CONCEPTUAL MODELING INTRODUCTION

Introduction to Data bases: Purpose of Database systems, view of data, data models, Database languages, Database users, various components of overall DBS architecture, various concepts of ER model, basics of Relational Model.

MODULE – II: RELATIONAL APPROACH

Relational algebra and calculus: Relational algebra, selection and projection, set operations, renaming, joins, division, examples of algebra queries, relational calculus: Tuple relational calculus, Domain relational calculus, expressive power of algebra and calculus.

MODULE – III: SQL QUERY - BASICS , RDBMS - NORMALIZATION

SQL – Data Definition commands, Queries with various options, Mata manipulation commands, Views, Joins, views, integrity and security; Relational database design: Pitfalls of RDBD, Lossless join decomposition, functional dependencies, Armstrong axioms, normalization for relational databases 1st, 2nd and 3rd normal forms, Basic definitions of MVDs and JDs, 4th and 5th normal forms.

MODULE – IV: TRANSACTION MANAGEMENT

Transaction processing: Transaction concept, transaction State, implementation of atomicity and durability, concurrent executions, serializability, recoverability.
Concurrency Control: Lock-based protocols, timestamp-based protocols, validation-based protocols, multiple granularity, multiversion schemes, deadlock handling.
Recovery: Failure classification, storage structure, recovery and atomicity, Log-Based recovery, shadow paging, recovery with concurrent transactions buffer management.

MODULE – V: DATA STORAGE AND QUERY PROCESSING

Data storage: Overview of physical storage media, magnetic disks, storage access, file organization, organization of records in files.
Indexing and Hashing: Basic concepts: Ordered indices, B+-tree index files, B-tree index files, static hashing, Dynamic Hashing, Comparison of Ordered Indexing and Hashing.
Query Processing: Overview, measures of query cost.

IV. TEXT BOOKS:

1. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, "Database System Concepts", McGraw-Hill, 6th Edition, 2017.

V. REFERENCE BOOKS:

1. Ramez Elmasri, Shamkant B. Navathe, "Fundamental Database Systems", Pearson Education, 6th Edition, 2014.
2. Raghu Ramakrishnan, "Database Management System", Tata McGraw-Hill Publishing Company, 3rd Edition, 2007.
3. Hector Garcia Molina, Jeffrey D. Ullman, Jennifer Widom, "Database System Implementation", Pearson Education, United States, 1st Edition, 2000.
4. Peter Rob, Corlos Coronel, "Database System, Design, Implementation and Management", Thompson Learning Course Technology, 5th Edition, 2003.

VI. WEB REFERENCES:

1. https://www.youtube.com/results?search_query=DBMS+onluine+classes
2. http://www.w3schools.in/dbms/
3. http://beginnersbook.com/2015/04/dbms-tutorial/

COMPUTER NETWORKS

IV Semester: IT, CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC06	Core	3	1	0	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisites: There are no prerequisites to take this course.

I. COURSE OVERVIEW

The main emphasis of this course is on the organization and management of local area networks (LANs) wide area networks (WANs). The course includes learning about computer network organization and implementation, obtaining a theoretical understanding of data communication and computer networks. Topics include layered network architectures, addressing, naming, forwarding, routing, communication reliability, the client-server model, and web and email protocols. The applications of this course are to design, implement and maintain a basic computer networks.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The fundamental knowledge of statement notations and logical connectives which are used to convert English sentences into logical expressions.
II. The effective use of combinatory principles for calculating probabilities and solving counting problems
III. The characteristics of generating functions for finding the solution of linear homogeneous recurrence relations.
IV. The effective use of graph theory in subsequent fields of study such as computer networks, and algorithms for solving real world engineering problems.

III. COURSE SYLLABUS:

MODULE-I INTRODUCTION

Introduction: Networks, network types, internet history, standards and administration; Network models: Protocol layering, TCP/IP protocol suite, the OSI model Transmission media: Introduction, guided media, unguided media; Switching: Introduction, circuit switched networks, packet switching.

MODULE-II DATA LINK LAYER

Introduction: Link layer addressing; Error detection and correction: Cyclic codes, checksum, forward error correction; Data link control: DLC services, data link layer protocols, media access control: Random access, virtual LAN.

MODULE-III NETWORK LAYER

Network layer design issues, routing algorithms, congestion control algorithms, quality of service, and internetworking. The network layer in the internet: IPv4 addresses, IPv6, internet control protocols, OSPF(Open Shortest Path First), IP (Internet Protocol)

MODULE-IV TRANSPORT LAYER

The transport service, elements of transport protocols, congestion control; The internet transport protocols:UDP (User Datagram Protocol), TCP (Transport Control Protocol), performance problems in computer networks, network performance measurement.

MODULE-V APPLICATION LAYER

Introduction, client server programming, WWW (World Wide Web) and HTTP (Hyper Text Transfer Protocol), FTP (File Transfer Protocol), E-mail, telnet, DNS (Domain Naming System), SNMP (Simple Network Management Protocol).

IV Text Books:

1. Behrouz A. Forouzan, “Data Communications and Networking”, Tata McGraw-Hill, 5th Edition, 2012.
2. Andrew S. Tanenbaum, David.j.Wetherall, “Computer Networks”, Prentice-Hall, 5th Edition, 2010.

V Reference Books:

1. Douglas E. Comer, "Internetworking with TCP/IP ", Prentice-Hall, 5th Edition, 2011.
2. Peterson, Davie, Elsevier, "Computer Networks", 5th Edition,2011
3. Comer, "Computer Networks and Internets with Internet Applications", 4th Edition, 2004.

BUSINESS ECONOMICS AND FINANCIAL ANALYSIS

IV Semester: CSE / CSIT / CSE(DS), CSE(CS)

V Semester: AE / CE / EEE | VI Semester: ECE / ME / IT / CSE(AI&ML)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC13	Core	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

The course is designed in such a way that it gives an overview of concepts of Economics. Managerial Economics enables students to understand micro environment in which markets operate how price determination is done under different kinds of competitions. Financial Analysis gives clear idea about concepts, conventions and accounting procedures along with introducing students to fundamentals of ratio analysis and interpretation of financial statements. Break Even Analysis is very helpful to the Business Concern for Decision Making, controlling and forward Strategic Planning. Ratio analysis gives an idea about financial forecasting, financial planning, controlling the business and decision making.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The concepts of business economics and demand analysist o helps in optimal decision making in business environment**.**

II. The functional relationship between Production and factors of production and able to compute breakeven point to illustrate the various uses of breakeven analysis.

III. The features, merits and demerits of different forms of business organizations existing in the modern business environment and market structures.

IV. The concept of capital budgeting and allocations of the resources through capital budgeting methods and compute simple problems for project management.

V. Various accounting concepts and different types of financial ratios for knowing financial positions of business concern.

III. COURSE OBJECTIVES:

MODULE – I: INTRODUCTION AND DEMAND ANALYSIS (07)

Definition, nature and scope of business economics; Demand analysis; Demand determinants, law of demand and its exceptions; Elasticity of demand: Definition, types, measurement and significance of elasticity of demand, demand forecasting, factors governing demand forecasting.

MODULE – II: PRODUCTION AND COST ANALYSIS (10)

Production function; Isoquants and isocosts, MRTS, least cost combination of inputs, Cobb-Dougles production function, internal and external economies of scale, cost analysis; Cost concepts: Break even analysis (BEA), determination of break-even point (simple problems), managerial significance.

MODULE – III: MARKETS AND NEW ECONOMIC ENVIRONMENT (08)

Types of competition and markets, features of perfect competition, monopoly and monopolistic competition, price-output determination in case of perfect competition and monopoly business.

Features and evaluation of different forms of business organizations: Sole proprietorship, partnership, joint stock company, public enterprises and their types.

MODULE – IV: CAPITAL BUDGETING (10)

Capital and its significance, types of capital, estimation of fixed and working capital requirements, methods and sources of raising capital, capital budgeting: features of capital budgeting proposals; Methods of capital budgeting: Payback period, accounting rate of return(ARR), net present value method and internal rate of return method (simple problems).

MODULE – V: INTRODUCTION TO FINANCIAL ACCOUNTING AND FINANCIAL ANALYSIS (10)

Financial accounting objectives, functions, importance; Accounting concepts and accounting conventions -double-entry book keeping, journal, ledger, trial balance; Final accounts: Trading account, profit and loss account and balance sheet with simple adjustments; Financial analysis: Analysis and interpretation of liquidity ratios, activity ratios, capital structure ratios and profitability ratios (simple problems), Du Pont chart.

IV. TEXT BOOKS:

1. Aryasri, "Managerial Economics and Financial Analysis", TMH publications, 4th Edition, 2012.
2. M. Kasi Reddy, Saraswathi, "Managerial Economics and Financial Analysis", PHI Publications, New Delhi, 2nd Edition, 2012.
3. Varshney, Maheswari, "Managerial Economics", Sultan Chand Publications, 11th Edition, 2009.

V. REFERENCE BOOKS:

1. S. A. Siddiqual, A. S. Siddiqual, "Managerial Economics and Financial Analysis", New Age International Publishers, Hyderabad, Revised 1st Edition, 2013.
2. S. N. Maheswari, S. K. Maheswari, "Financial Accounting", Vikas publications, 3rd Edition, 2012.
3. J. V. Prabhakar Rao, P. V. Rao, "Managerial Economics and Financial Analysis", Maruthi Publishers, Reprinted Edition, 2011.
4. Vijay Kumar, Appa Rao, "Managerial Economics and Financial Analysis", Cengage Publications, 1st Edition, Paperback, 2011.

VI. WEB REFERENCES:

1. https:// www.slideshare.net/glory1988/managerial-economics-and- financial analysis
2. https:// thenthata.web4kurd.net/mypdf/managerial-economics-and- financial analysis
3. https:// bookshallcold.link/pdfread/managerial-economics-and-financial analysis
4. https:// www.gvpce.ac.in/syllabi/Managerial Economics and financial analysis

EXPERIENTIAL ENGINEERING EDUCATION (EXEED) – FABRICATION / MODEL DEVELOPMENT

IV Semester: **Common for all branches**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC14	Foundation	2	0	0	1	30	70	100
Contact Classes: 28	Tutorial Classes: Nil	Practical Classes: 28				Total Classes: 28		

Prerequisite: **There are no prerequisites to take this course**

I. COURSE OVERVIEW:

This course provide the environment to develop high-tech, ecological and socially responsible products from concept and design to production. The course covers hands-on learning in product and industrial design and product development with product lifecycle management.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The design thinking process and Identify opportunities through customer needs analysis.
II. Product specifications based on customer needs that are desirable, feasible, and viable through applied creativity.
III. The Implementation techniques for planning and executing a prototype design services.

WEEK NO	TOPIC
WEEK – I	Introduction To Product Design
WEEK – II	Design Thinking Skills
WEEK – III	Identifying Customer Needs
WEEK – IV	Product Specifications
WEEK – V	Applied Creativity
WEEK – VI	Prototyping
WEEK – VII	Design Of Services
WEEK –VIII	Product Architecture
WEEK - IX	Financial Analysis
WEEK - X	Design For Environment
WEEK - XI	Product Development Process
WEEK - XII	Reverse Engineering
WEEK - XIII	Value Engineering
WEEK - XIV	Assessment

DATABASE MANAGEMENT SYSTEMS LABORATORY

IV Semester: CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC07	Core	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes:36		

Prerequisite: No Prerequisites

I. OURSE OVERVIEW:

The purpose of this course is to provide a clear understanding of fundamentals with emphasis on their applications to create and manage large data sets. It highlights on technical overview of database software to retrieve data from n database. The course includes database design principles, normalization, concurrent transaction processing,, security, recovery and file organization techniques.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Implement the basic knowledge of SQL queries and relational algebra.
II. Construct database models for different database applications.
III. Apply normalization techniques for refining of databases.
IV. Practice various triggers, procedures, and cursors using PL/SQL.

III. COURSE SYLLABUS:

Week – 1: CREATION OF TABLES

1. Create a table called Employee with the following structure

Name	Type
Empno	Number
Ename	Varchar2(20)
Job	Varchar2(20)
Mgr	Number
Sal	Number

a. Add a column commission with domain to the Employee table.
b. Insert any five records into the table.
c. Update the column details of job
d. Rename the column of Employ table using alter command.
e. Delete the employee whose empno is 19.

2. Create department table with the following structure.

Name	Type
Deptno	Number
Deptname	Varchar2(20)
location	Varchar2(20)

a. Add column designation to the department table.
b. Insert values into the table.
c. List the records of emp table grouped by deptno.
d. Update the record where deptno is 9.
e. Delete any column data from the table.

3. Create a table called Customer table

Name	Type
Cust name	Varchar2(20)
Cust street	Varchar2(20)
Cust city	Varchar2(20)

a. Insert records into the table.
b. Add salary column to the table.
c. Alter the table column domain.
d. Drop salary column of the customer table.
e. Delete the rows of customer table whose cust_cit is 'hyd'.

4. Create a table called branch table.

Name	Type
Branch name	Varchar2(20)
Branch city	Varchar2(20)
asserts	Number

a. Increase the size of data type for asserts to the branch.
b. Add and drop a column to the branch table.
c. Insert values to the table.
d. Update the branch name column.
e. Delete any two columns from the table.

5. Create a table called sailor table

Name	Type
Sid	Number
Sname	Varchar2(20)
rating	Varchar2(20)

a. Add column age to the sailor table.
b. Insert values into the sailor table.
c. Delete the row with rating >8.
d. Update the column details of sailor.
e. Insert null values into the table.

6. Create a table called reserves table

Name	Type
Boat id	Integer
sid	Integer
day	Integer

a. Insert values into the reserves table.
b. Add column time to the reserves table.
c. Alter the column day data type to date.
d. Drop the column time in the table.
e. Delete the row of the table with some condition.

Week – 2: QUERIES USING DDL AND DML

1. a. Create a user and grant all permissions to the user.
   b. Insert the any three records in the employee table and use rollback. Check the result.
   c. Add primary key constraint and not null constraint to the employee table.
   d. Insert null values to the employee table and verify the result.
2. a. Create a user and grant all permissions to the user.
   b. Insert values in the department table and use commit.
   c. Add constraints like unique and not null to the department table.
   d. Insert repeated values and null values into the table.
3. a. Create a user and grant all permissions to the user.
   b. Insert values into the table and use commit.
   c. Delete any three records in the department table and use rollback.
   d. Add constraint primary key and foreign key to the table.
4. a. Create a user and grant all permissions to the user.
   b. Insert records in the sailor table and use commit.
   c. Add save point after insertion of records and verify save point.
   d. Add constraints not null and primary key to the sailor table.
5. a. Create a user and grant all permissions to the user.
   b. Use revoke command to remove user permissions.
   c. Change password of the user created.
   d. Add constraint foreign key and not null.
6. a. Create a user and grant all permissions to the user.
   b. Update the table reserves and use savepoint and rollback.
   c. Add constraint primary key , foreign key and not null to the reserves table
   d. Delete constraint not null to the table column.

Week – 3: QUERIES USING AGGREGATE FUNCTIONS

1. a. By using the group by clause, display the enames who belongs to deptno 10 along with average salary.
   b. Display lowest paid employee details under each department.
   c. Display number of employees working in each department and their department number.
   d. Using built in functions, display number of employees working in each department and their department name from dept table. Insert deptname to dept table and insert deptname for each row, do the required thing specified above.
   e. List all employees which start with either B or C.
   f. Display only these ename of employees where the maximum salary is greater than or equal to 5000.
2. a. Calculate the average salary for each different job.
   b. Show the average salary of each job excluding manager.
   c. Show the average salary for all departments employing more than three people.
   d. Display employees who earn more than the lowest salary in department 30
   e. Show that value returned by sign (n) function.
   f. How many days between day of birth to current date.
3. a. Show that two substring as single string.
   b. List all employee names, salary and 15% rise in salary.
   c. Display lowest paid emp details under each manager
   d. Display the average monthly salary bill for each deptno.
   e. Show the average salary for all departments employing more than two people.
   f. By using the group by clause, display the eid who belongs to deptno 05 along with average salary.
4. a. Count the number of employees in department 20
   b. Find the minimum salary earned by clerk.
   c. Find minimum, maximum, average salary of all employees.
   d. List the minimum and maximum salaries for each job type.
   e. List the employee names in descending order.
   f. List the employee id, names in ascending order by empid.
5. a. Find the sids ,names of sailors who have reserved all boats called “INTERLAKE
   Find the age of youngest sailor who is eligible to vote for each rating level with at least two such sailors.
   b. Find the sname , bid and reservation date for each reservation.

c. Find the ages of sailors whose name begin and end with B and has at least 3 characters.
d. List in alphabetic order all sailors who have reserved red boat.
e. Find the age of youngest sailor for each rating level.

6. a. List the Vendors who have delivered products within 6 months from order date.
   b. Display the Vendor details who have supplied both Assembled and Sub parts.
   c. Display the Sub parts by grouping the Vendor type (Local or Non Local).
   d. Display the Vendor details in ascending order.
   e. Display the Sub part which costs more than any of the Assembled parts.
   f. Display the second maximum cost Assembled part.

Week – 4: PROGRAMS ON PL/SQL

1. a. Write a PL/SQL program to swap two numbers.
   b. Write a PL/SQL program to find the largest of three numbers.
2. a. Write a PL/SQL program to find the total and average of 6 subjects and display the grade.
   b. Write a PL/SQL program to find the sum of digits in a given number.
3. a. Write a PL/SQL program to display the number in reverse order.
   b. Write a PL / SQL program to check whether the given number is prime or not.
4. a. Write a PL/SQL program to find the factorial of a given number.
   b. Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns radius and area.
5. a. Write a PL/SQL program to accept a string and remove the vowels from the string. (When 'hello' passed to the program it should display 'Hll' removing e and o from the world Hello).
   b. Write a PL/SQL program to accept a number and a divisor. Make sure the divisor is less than or equal to 10. Else display an error message. Otherwise Display the remainder in words.

Week – 5: PROCEDURES AND FUNCTIONS

1. Write a function to accept employee number as parameter and return Basic +HRA together as single column.
2. Accept year as parameter and write a Function to return the total net salary spent for a given year.
3. Create a function to find the factorial of a given number and hence find NCR.
4. Write a PL/SQL block o pint prime Fibonacci series using local functions.
5. Create a procedure to find the lucky number of a given birth date.
6. Create function to the reverse of given number.

Week – 6: TRIGGERS

1. Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

   CUSTOMERS table:

ID	NAME	AGE	ADDRESS	SALARY
1	Alive	24	Khammam	2000
2	Bob	27	Kadappa	3000
3	Catri	25	Guntur	4000
4	Dena	28	Hyderabad	5000
5	Eeshwar	27	Kurnool	6000
6	Farooq	28	Nellur	7000

2. Creation of insert trigger, delete trigger, update trigger practice triggers using the passenger database.
   Passenger(Passport_ id INTEGER PRIMARY KEY, Name VARCHAR (50) Not NULL, Age Integer Not NULL, Sex Char, Address VARCHAR (50) Not NULL);

   a. Write a Insert Trigger to check the Passport_id is exactly six digits or not.
   b. Write a trigger on passenger to display messages '1 Record is inserted', '1 record is deleted', '1 record is updated' when insertion, deletion and updation are done on passenger respectively.

Insert row in employee table using Triggers. Every trigger is created with name any trigger have same name must be replaced by new name. These triggers can raised before insert, update or delete rows on data base. The main difference between a trigger and a stored procedure is that the former is attached to a table and is only fired when an INSERT,

3. UPDATE or DELETE occurs.
4. Convert employee name into uppercase whenever an employee record is inserted or updated. Trigger to fire before the insert or update.
5. Trigger before deleting a record from emp table. Trigger will insert the row to be deleted into table called delete _emp and also record user who has deleted the record and date and time of delete.
6. Create a transparent audit system for a table CUST_MSTR. The system must keep track of the records that are being deleted or updated.

Week – 7: PROCEDURES

1. Create the procedure for palindrome of given number.
2. Create the procedure for GCD: Program should load two registers with two Numbers and then apply the logic for GCD of two numbers. GCD of two numbers is performed by dividing the greater number by the smaller number till the remainder is zero. If it is zero, the divisor is the GCD if not the remainder and the divisors of the previous division are the new set of two numbers. The process is repeated by dividing greater of the two numbers by the smaller number till the remainder is zero and GCD is found.
3. Write the PL/SQL programs to create the procedure for factorial of given number.
4. Write the PL/SQL programs to create the procedure to find sum of N natural number.
5. Write the PL/SQL programs to create the procedure to find Fibonacci series.
6. Write the PL/SQL programs to create the procedure to check the given number is perfect or not.

Week – 8: CURSORS

1. Write a PL/SQL block that will display the name, dept no, salary of fist highest paid employees.
2. Update the balance stock in the item master table each time a transaction takes place in the item transaction table. The change in item master table depends on the item id is already present in the item master then update operation is performed to decrease the balance stock by the quantity specified in the item transaction in case the item id is not present in the item master table then the record is inserted in the item master table.
3. Write a PL/SQL block that will display the employee details along with salary using cursors.
4. To write a Cursor to display the list of employees who are working as a Managers or Analyst.
5. To write a Cursor to find employee with given job and deptno.
6. Write a PL/SQL block using implicit cursor that will display message, the salaries of all the employees in the 'employee' table are updated. If none of the employee's salary are updated we get a message 'None of the salaries were updated'. Else we get a message like for example, 'Salaries for 1000 employees are updated' if there are 1000 rows in 'employee' table.

Week – 9: CASE STUDY: BOOK PUBLISHING COMPANY

A publishing company produces scientific books on various subjects. The books are written by authors who specialize in one particular subject. The company employs editors who, not necessarily being specialists in a particular area, each take sole responsibility for editing one or more publications.

A publication covers essentially one of the specialist subjects and is normally written by a single author. When writing a particular book, each author works with on editor, but may submit another work for publication to be supervised by other editors. To improve their competitiveness, the company tries to employ a variety of authors, more than one author being a specialist in a particular subject for the above case study, do the following:

1. Analyze the data required.
2. Normalize the attributes.

Create the logical data model using E-R diagrams.

Week – 10: CASE STUDY GENERAL HOSPITAL

A General Hospital consists of a number of specialized wards (such as Maternity, Pediatric, Oncology,etc). Each ward hosts a number of patients, who were admitted on the recommendation of their own GP and confirmed by a consultant employed by the Hospital. On admission, the personal details of every patient are recorded. A separate register is to be held to store the information of the tests undertaken and the results of a prescribed treatment. A number of tests may be conducted for each patient. Each patient is assigned to one leading consultant but may be examined by another doctor, if required. Doctors are specialists in some branch of medicine and may be leading consultants for a number of patients, not necessarily from the same ward. For the above case study, do the following.

1. Analyze the data required.

Normalize the attributes. Create the logical data model using E-R diagrams.

Week – 11: CASE STUDY: CAR RENTAL COMPANY

A database is to be designed for a car rental company. The information required includes a description of cars, subcontractors (i.e. garages), company expenditures, company revenues and customers. Cars are to be described by such data as: make, model, year of production, engine size, fuel type, number of passengers, registration number,

purchase price, purchase date, rent price and insurance details. It is the company policy not to keep any car for a period exceeding one year. All major repairs and maintenance are done by subcontractors (i.e. franchised garages), with whom CRC has long-term agreements. Therefore the data about garages to be kept in the database includes garage names, addresses, range of services and the like. Some garages require payments immediately after a repair has been made; with others CRC has made arrangements for credit facilities. Company expenditures are to be registered for all outgoings connected with purchases, repairs, maintenance, insurance etc. Similarly the cash inflow coming from all sources: Car hire, car sales, insurance claims must be kept of file. CRC maintains a reasonably stable client base. For this privileged category of customers special credit card facilities are provided. These customers may also book in advance a particular car. These reservations can be made for any period of time up to one month. Casual customers must pay a deposit for an estimated time of rental, unless they wish to pay by credit card. All major credit cards are accepted. Personal details such as name, address, telephone number, driving license, number about each customer are kept in the database. For the above case study, do the following:

1. Analyze the data required.
   Normalize the attributes. Create the logical data model using E-R diagrams.

Week – 12: CASE STUDY: STUDENT PROGRESS MONITORING SYSTEM

A database is to be designed for a college to monitor students' progress throughout their course of study. The students are reading for a degree (such as BA, BA (Hons) M.Sc., etc) within the framework of the modular system. The college provides a number of modules, each being characterized by its code, title, credit value, module leader, teaching staff and the department they come from. A module is coordinated by a module leader who shares teaching duties with one or more lecturers. A lecturer may teach (and be a module leader for) more than one module. Students are free to choose any module they wish but the following rules must be observed: Some modules require pre- requisites modules and some degree programmes have compulsory modules. The database is also to contain some information about students including their numbers, names, addresses, degrees they read for, and their past performance i.e. modules taken and examination results. For the above case study, do the following:

1. Analyze the data required.
2. Normalize the attributes.
3. Create the logical data model i.e., ER diagrams.
4. Comprehend the data given in the case study by creating respective tables with primary keys and foreign keys wherever required.
5. Insert values into the tables created (Be vigilant about Master- Slavetables).
6. Display the Students who have taken M.Sc course.
7. Display the Module code and Number of Modules taught by each Lecturer.
8. Retrieve the Lecturer names who are not Module Leaders.
9. Display the Department name which offers ‘English’ module.
10. Retrieve the Prerequisite Courses offered by every Department (with Department names).
11. Present the Lecturer ID and Name who teaches ‘Mathematics’.
12. Discover the number of years a Module is taught.
13. List out all the Faculties who work for ‘Statistics’ Department.
14. List out the number of Modules taught by each Module Leader.
15. List out the number of Modules taught by a particular Lecturer.
16. Create a view which contains the fields of both Department and Module tables. (Hint- The fields like Module code, title, credit, Department code and its name).
17. Update the credits of all the prerequisite courses to 5. Delete the Module ‘History’ from the Module table.

IV. REFERENCE BOOKS:

1. Ramez Elmasri, Shamkant, B. Navathe, “Database Systems”, Pearson Education, 6th Edition, 2013.
2. Peter Rob, Carles Coronel, “Database System Concepts”, Cengage Learning, 7th Edition, 2008.

V. WEB REFERENCES:

1. http://www.scoopworld.in

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

IV Semester: **CSE / CSIT / CSE (CS)**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC15	Core	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

Prerequisites: **Programming for Problem Solving, Data Structures**

I. COURSE OVERVIEW:

Design and analysis of algorithms is the process of finding the computational complexity of algorithms. It helps to design and analyze the logic on how the algorithm will work before developing the actual code for a program. It focuses on introduction to algorithm, asymptotic complexity, sorting and searching using divide and conquer, greedy method, dynamic programming, backtracking, branch and bound. NP-hard and NP-complete problems. The applications of algorithm design are used for information storage, retrieval, transportation through networks, and presentation to users.

II. COURSE OBJECTIVES:

The students will try to learn :

I. The problem analysis and design the solution for the given problem.
II. The suitable algorithm for the given real world problem.

III. COURSE SYLLABUS:

WEEK-1: QUICK SORT

Sort a given set of elements using the quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the 1st to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

WEEK – 2: MERGE SORT

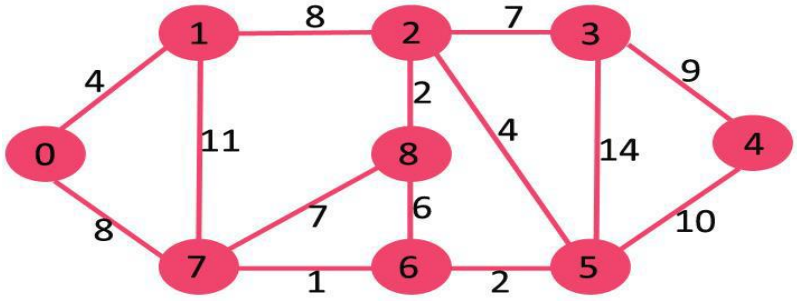
Implement merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

WEEK – 3: KNAPSACK PROBLEM

Implement 0/1 Knapsack problem using Dynamic Programming.

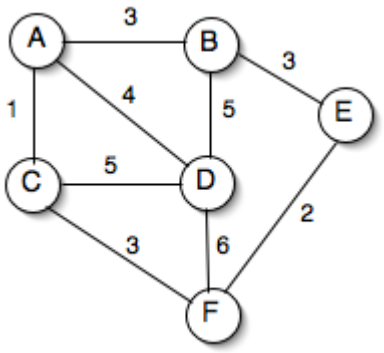
WEEK – 4: SHORTEST PATHS ALGORITHM

From a given vertex in a weighted connected graph, find shortest paths from 0 to other vertices using Dijkstra‘s algorithm.



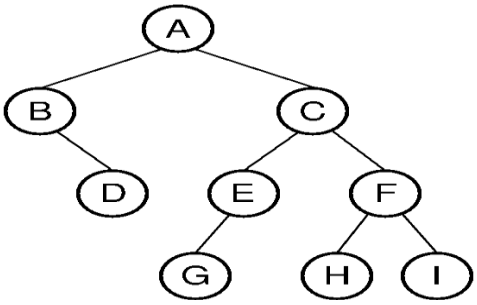
WEEK – 5: MINIMUM COST SPANNING TREE

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.



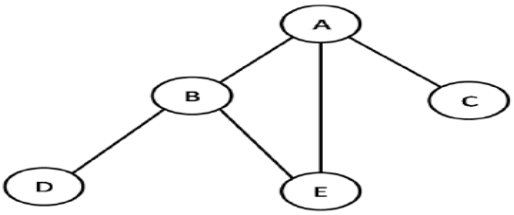
WEEK – 6: TREE TRAVESRSALS

Perform various tree traversal algorithms for a given tree.

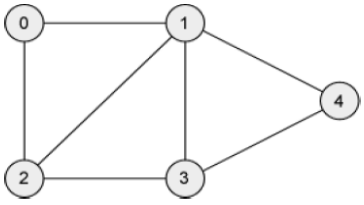


WEEK – 7: GRAPH TRAVERSALS

a. Print all the nodes reachable from a given starting node in a digraph using BFS method.



b. Check whether a given graph is connected or not using DFS method.



WEEK – 8: SUM OF SUB SETS PROBLEM

Find a subset of a given set S = {sl, s2,....., sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1,8}.A suitable

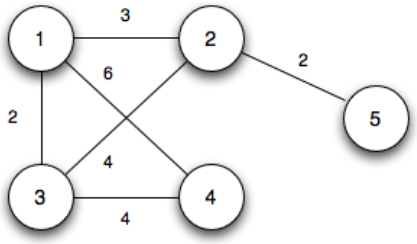
message is to be displayed if the given problem instance doesn't have a solution.

WEEK – 9: TRAVELLING SALES PERSON PROBLEM

Implement any scheme to find the optimal solution for the Traveling Sales Person problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

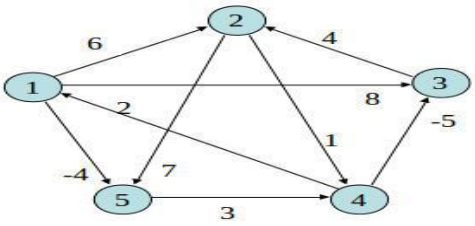
WEEK 10: MINIMUM COST SPANNING TREE

Find Minimum Cost Spanning Tree of a given undirected graph using Prim‘s algorithm.



WEEK – 11: ALL PAIRS SHORTEST PATHS

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.



	1	2	3	4	5
1	0	6	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	3	0

WEEK – 12: N QUEENS PROBLEM

Implement N Queen's problem using Back Tracking.

IV. REFERENCE BOOKS:

1. Levitin A, “Introduction to the Design and Analysis of Algorithms”, Pearson Education, 2008.
2. Goodrich, M.T. R Tomassia, “Algorithm Design foundations Analysis and Internet Examples”, John Wiley and Sons, 2006.
3. Base Sara, Allen Van Gelder, “Computer Algorithms Introduction to Design and Analysis”, Pearson, 3rd Edition, 1999.

V. WEB REFERENCE:

1. http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
2. http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
3. http://www.facweb.iitkgp.ernet.in/~sourav/daa.html

COMPUTER NETWORKS LABORATORY

IV Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC12	Foundation	0	0	2	1	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 24				Total Classes:24		

Prerequisite: There are no prerequisites to take this course.

I. COURSE OVERVIEW:

This course is to provide students with an overview of the concepts and fundamentals of computer networks. Topics to be covered include: data communication concepts and techniques in a layered network architecture, communications switching and routing, types of communication, network congestion, network topologies, network configuration and management, network model components, layered network models (OSI reference model, TCP/IP networking architecture) and their protocols, various types of networks (LAN, MAN, WAN and Wireless networks) and their protocols.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Modern network architectures from a design and performance perspective.
II. The basics and challenges of network communication.
III. The network programming using TCP/IP.
IV. The operation of the protocols that are used inside the Internet..

III. COURSE SYLLABUS:

Week – 1: DATA LINK LAYER - BIT STUFFING

Write a python program to implement the data link layer framing methods such as bit stuffing.

Week – 2: DATA LINK LAYER - CHARACTER STUFFING

Write a python program to implement the data link layer framing method such as character stuffing.

Week – 3: DATA LINK LAYER - CHARACTER COUNT

Write a python program to implement data link layer framing method character count.

Week – 4: CRC

Write a python program to implement on a data set characters the three CRC polynomials – CRC 12, CRC 16, and CRC CCIP.

Week – 5: SHORTEST PATH ROUTING ALGORITHM

a. Write a python program to implement Dijkstra's Algorithm to compute the shortest path through a given path.
b. Write a python program to find the shortest path between vertices using bellman-ford algorithm.

Week – 6: DISTANCE VECTOR ROUTING ALGORITHM

Write a python program to take an example subnet graph with weights indicating delay between nodes. Now obtain Routing table art each node using distance vector routing algorithm.

Week – 7: BROADCAST TREE

Write a python program implement broadcast tree for a given subnet hosts.

Week – 8: ENCRYPTING DES

Write a python program to implement that to take a 64 bit playing text and encrypt the same using DES algorithm.

Week – 9: DECRYPTING DES

Write a python program to implement to break the above DES coding.

Week – 10: RSA ALGORITHM

Write python program to implement RSA algorithm encrypts a text data and decrypt the same.

Week – 11: FTP PROTOCOL

Write python program simulate to find the number of packets dropped.

Week – **12: UDP PROTOCOL**

a. Write a python program for UDP protocol.

b. Write a python program using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

IV.REFERENCE BOOKS:

1. Behrouz Forouzan, "Introduction to Data Communications and Networking", Tata McGraw Hill, 5th Edition, 2015.
2. Stallings, "Data and Computer Communications", PHI, 10th Edition, 2015.
3. William Schewber ,"Data Communication", McGraw Hill, 1987.
4. Tanenbaum , "Computer Networks", PHI, 5rd Edition, 2011.

V.WEB REFERENCES

1. http://www.cse.iitk.ac.in/users/dheeraj/cs425/
2. http://www.tcpipguide.com/free/t_OSIReferenceModelLayers.htm
3. http://iit.qau.edu.pk/books/Data%20Communications%20and%20Networking%20By%20Behrouz%20A.Forouzan.pdf
4. http://www.networkdictionary.com/protocols/osimodel.php
5. www.niecdelhi.ac.in
6. https://www.linkedin.com/in/achin-jain-85061412

INDIAN CONSTITUTION

IV Semester: CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC14	Mandatory	-	-	-	-	-	-	-
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: Nil		

Prerequisite: No Prerequisites

I. COURSE OVERVIEW:

The Constitution of India is the supreme law of India. The document lays down the framework that demarcates fundamental political code, structure, procedures, powers, and duties of government institutions and sets out fundamental rights, directive principles, and the duties of citizens. It is the longest written constitution of any country on earth.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The need for constitution.
II. The fundamental duties and rights of the citizens of India.
III. The role and amendments of constitution in a democratic society.
IV. The directive principles of state policy and their significance.
V. The key features of the constitution, union government and state government.

MODULE-I	HISTORY OF MAKING OF THE INDIAN CONSTITUTION

History of Making of the Indian Constitution: Introduction to the constitution of India, the making of the constitution and salient features of the constitution.

MODULE-II	PHILOSOPHY OF THE INDIAN CONSTITUTION

Philosophy of the Indian Constitution: Preamble Salient Features, Contours of Constitutional Rights & Duties: Fundamental Rights, Right to Equality, Right to Freedom, Right against Exploitation, Right to Freedom of Religion, Cultural and Educational Rights, Right to Constitutional Remedies, Directive Principles of State Policy, Fundamental Duties, Amendment of the constitutional powers and procedures.

MODULE-III	UNION GOVERNMENT

Union Government: Union Government, Union Legislature (Parliament), Lok Sabha and Rajya Sabha (with powers and functions), president of India (with powers and functions), Prime minister of India (With powers and functions), Union judiciary (Supreme court), Jurisdiction of the supreme court.

MODULE-IV	STATE GOVERNMENT

State Government: State Government, State legislature (Legislative Assembly/ Vidhan Sabha, Legislative council/ Vidhan parishad), powers and functions of the state legislature, State executive, Governor of the state (with powers and functions), The chief Minister of the state (with powers and functions), State Judiciary (High courts)

MODULE-V	ELECTION COMMISSION

Election Commission: Election Commission: Role and Functioning, Chief Election Commissioner and Election Commissioners, State Election Commission: Role and Functioning, Institute and Bodies for the welfare of SC/ST/OBC and women.

III. Text Books:

1. M.V. Pylee, Indian Constitution Durga Das Basu, Human Rights in Constitutional Law, Prentice – Hall of India Pvt Ltd. New Delhi.
2. Noorani, A.G., (South Asia Human Rights Documentation Centre), Challenges to Civil Right), Challenges to Civi Rights Guarantees in India, Oxford University Press 2012.
3. The constitution of India, P.M.Bakshi, Universal Law Publishing Co.,
4. The Constitution of India, 1950 (Bare Act), Government Publication.
5. Dr. S. N. Busi, Dr. B. R. Ambedkar framing of Indian Constitution, 1st Edition, 2015.

Reference Books:
1. M. P. Jain, Indian Constitution Law, 7th Edn., Lexis Nexis, 2014. 2. D.D. Basu, Introduction to the Constitution of India, Lexis Nexis, 2015.

WEB APPLICATION DEVELOPMENT

V Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC09	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There is no prerequisite to take this course

I. COURSE OVERVIEW:

This course introduces students to create concurrently a web app and a native app (for Android and iOS) with React Native and React Native Web. It covers HTML5 for structuring and presenting content on the World Wide Web. CSS3 being used to format structured content. To create a dynamic and interactive experience for the user it covers JAVASCRIPT. How build the applications using React concepts such as JSX, REDUX.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The characteristics, systematic methods, model for developing web applications.
II. The fundamentals of HTML and CSS to design static and dynamic web pages.
III. The concepts of client - server programming with JavaScript, Servlets, JSX.
IV. The MVC architecture, about React and built single and multiple page applications using REACT with REDUX.

III. COUSRSE SYLLABUS:

MODULE – I: INTRODUCTION TO WEB APPLICATION AND HYPERTEXT MODELLING (10)

Introduction to web application, Basics of hypertext modelling, hypertext structure modelling concepts, access modelling concepts, relation to content modelling, presentation modelling, relation to hypertext modelling, customization modelling, relation to content, hypertext, and presentation modelling. Basics of HTML5 and web design, creating tables, HTML forms, styles and classes to your web pages, web page layouts with CSS, introduction to responsive web design with CSS3 and HTML5.

MODULE – II: BUILD INTERFCAES USING BOOTSTARP (08)

Introduction to web design from an evolutionary perspective, user interface design through bootstrap, containers, tables, jumptrons, list, cards, carousal, navigation, modals, flex and forms, responsive web page design, basic UI grid structure.

MODULE – III: INTERACTIVE USER INFERFACE AND WEB APPLICATION DEVELOPMENT (10)

JavaScript variable naming rules, data types, expressions and operators, pattern matching with regular expressions, managing web page styles using JavaScript and CSS, script forms, introduction to AJAX.

Introduction to web design from an evolutionary perspective, create a native and web app, JSX, class and function components, props, state, lifecycle methods, and hooks.

MODULE – IV:UI BINDING LIBRARY FOR REACT (09)

Introduction to client-side routing using React Router, global state management and transitions using REDUX, server side rendering and testing using Jest, Enzyme and more. Web Development Using REACT is delivered both in a blended learning and self-paced mode.

MODULE – V: CONNECT TO AN EXTERNAL API (08)

REDUX store using the official create store function, REDUX toolkit has a configure store API, loading state for that particular API, adding an API service as a middleware, example uses create REACT App.

IV. TEXT BOOKS:

1. AlokRanjanAbhilashaSinha, RanjitBattewad, “JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and JavaScript”, 1st Edition, 2020.
2. Alex Banks and Eve Porcello, “Learning React: Functional Web Development with React and Redux”, 2017.

V. REFERENCE BOOKS:

1. Adam BoduchandRoy Derks, “React and React Native: A complete hands-on guide to modern web and mobile

development with React.js", 3rd Edition, 2020.
2. W Hans Bergsten, "Java Server Pages", O'Reilly, 3rd Edition, 2003.
3. D. Flanagan, "Java Script", O'Reilly, 6th Edition, 2011.
4. Jon Duckett, "Beginning Web Programming", WROX, 2nd Edition, 2008.

VI. WEB REFERENCES:

1. https://www.codecademy.com/learn/paths/web-development/
2. https://nptel.ac.in/courses/106/105/106105084/
3. https://medium.com/@aureliomerenda/create-a-native-web-app-with-react-native-web-419acac86b82
4. https://www.coursera.org/learn/react-native
5. https://desirecourse.net/react-native-and-redux-course-using-hooks/

OBJECT ORIENTED SOFTWARE ENGINEERING

V SEMESTER: CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC19	Core	3	1	0	4	30	70	100
Contact Classes: 45	Total Tutorials: 15	Total Practical Classes: Nil				Total Classes: 60		

Prerequisite: There is no prerequisite to take this course

I. COURSE OVERVIEW:

This course presents the concepts, methods and techniques necessary to efficiently capture software requirements in use cases and transform them into detailed designs. It combines instruction on the Unified Software Development Process (UP), object-oriented methodologies and the Unified Modeling Language. In this course, students learn how to apply the UML notation in the context of an iterative, use case-driven, architecture-centric process. They are also exposed to an advanced CASE tool that allows the rapid development of UML diagrams and promotes an agile workflow by synchronizing changes in the various models and the code.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The object-oriented concepts along with their applicability contexts.
II. The software development process models and coding standards.
III. The knowledge of testing methods and comparison of various testing techniques.
IV. The underlying concepts and standards of quality and knowledge about software quality assurance group.
V. The modeling techniques to model different perspectives of object-oriented software design

III. COURSE SYLLABUS

MODULE –I: INTRODUCTION TO SOFTWARE ENGINEERING (09)

Introduction to software engineering, software development process models, agile development, project and process, project management, process and project metrics, object-oriented concepts, principles and methodologies.

MODULE –II: PLANNING AND SCHEDULING (09)

Software requirements specification, software prototyping, software project planning, scope, resources, software estimation, empirical estimation models, planning, risk management, software project scheduling, object-oriented estimation and scheduling.

MODULE –III: ANALYSIS (09)

Analysis modeling, data modeling, functional modeling and information flow, behavioral modeling, structured analysis, object-oriented analysis, domain analysis.

Object-oriented analysis process, object relationship model, object behaviour model, design modeling with UML.

MODULE –IV: DESIGN (09)

Design concepts and principles, design process, design concepts, modular design, design effective modularity, introduction to software architecture, data design, transform mapping, transaction mapping, object-oriented design, system design process, object design process.

MODULE –V: IMPLEMENTATION, TESTING AND MAINTENANCE (09)

Top-down, bottom-up, object-oriented product implementation and integration. Software testing methods, white box, basis path, control structure, black box, unit testing, integration testing, validation and system testing, testing tools, software maintenance and reengineering.

IV. TEXT BOOKS:

1. Ivar Jacobson, "Object Oriented Software Engineering: A Use Case Driven Approach", Pearson India, 1st Edition, 2002.
2. Bernd Bruegge, Allen H. Dutoit, "Object-Oriented Software Engineering: Using UML, Patterns and Java", Pearson New International Edition, 3rd Edition, 2013.

V. REFERENCE BOOKS:

1. Roger. S. Pressman and Bruce R. Maxim, "Software Engineering – A Practitioner's Approach", McGraw Hill, 7th Edition, 2015.
2. Ian Sommerville, "Software Engineering", Pearson Education, New Delhi, 8th Edition, 2011.
3. Bill Brykczynski, Richard D. Stutz, "Software Engineering- Project Management", Wiley-IEEE Computer Society, 2nd Edition, 2000.
4. Craig Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development", Pearson Education, 3rd Edition, 2008.
5. Jalote P, "An Integrated Approach to Software Engineering", Narosa Publishers, New Delhi, 3rd Edition 2013.

VI. WEB REFERENCES:

1. https://onlinecourses.nptel.ac.in/noc19_cs52/preview
2. https://ece.iisc.ac.in/~parimal/2019/ml.html
3. https://www.springer.com/gp/book/9780387848570
4. https://www.cse.iitb.ac.in/~sunita/cs725/calendar.html
5. https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/
6. https://cs.nyu.edu/~mohri/mlu11/

THEORY OF COMPUTATION

V Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC04	Core	3	1	0	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisites: Discrete Mathematical Structures

I. COURSE OVERVIEW:

This course focuses on infinite languages in finite ways, and classify machines by their power to recognize .It includes finite automata, regular grammar, push down automata ,context free grammars, and Turing machines It is applicable in designing phrasing and lexical analysis of a compiler, genetic programming and recursively enumerable languages.

II. OBJECTIVES:

The students will try to learn:

I. The fundamental knowledge of automata theory which is used to solve computational problems.
II. The reorganization of context free language for processing infinite information using push down automata.
III. The computer based algorithms with the help of an abstract machines to solve recursively enumerable problems.

III. SYLLABUS:

MODULE –I: FINITE AUTOMATA

Fundamentals: Alphabet, strings, language, operations; Introduction to finite automata: The central concepts of automata theory, deterministic finite automata, nondeterministic finite automata, an application of finite automata, finite automata with and without epsilon transitions, Conversion of NFA to DFA, Moore and Melay Machines.

MODULE –II: REGULAR LANGUAGES

Regular sets, regular expressions, identity rules, constructing finite automata for a given regular expression, conversion of finite automata to regular expressions, pumping lemma of regular sets, closure properties of regular sets (proofs not required), regular grammars-right linear and left linear grammars, equivalence between regular linear grammar and finite automata, inter conversion.

MODULE –III: CONTEXT FREE GRAMMARS

Context free grammars and languages: Context free grammar, derivation trees, sentential forms, right most and leftmost derivation of strings, applications.
Ambiguity in context free grammars, minimization of context free grammars, Chomsky normal form, Greibach normal form, pumping lemma for context free languages, enumeration of properties of context free language (proofs omitted).

MODULE –IV: PUSHDOWN AUTOMATA

Pushdown automata, definition, model, acceptance of context free language, acceptance by final state and acceptance by empty stack and its equivalence, equivalence of context free language and pushdown automata, inter conversion;(Proofs not required); Introduction to deterministic context free languages and deterministic pushdown automata.

MODULE –V: TURING MACHINE

Turing machine: Turing machine, definition, model, design of Turing machine, computable functions, recursively enumerable languages, Church's hypothesis, counter machine, types of Turing machines (proofs not required), linear bounded automata and context sensitive language, Chomsky hierarchy of languages.

IV.TEXT BOOKS:

1. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, "Introduction to Automata, Theory, Languages and Computation"|, Pearson Education, 3rd Edition, 2007.

IV. REFERENCE BOOKS:

1. John C Martin, –Introduction to Languages and Automata Theory|, Tata McGraw-Hill, 3rd Edition, 2017.
2. Daniel I.A. Cohen, –Introduction to Computer Theory|, John Wiley & Sons, 2nd Edition, 2004.

V. WEB REFERENCES:

1. https://www.tutorialspoint.com/automata_theory/index.htm
2. https://www.iitg.ernet.in/dgoswami/Flat-Notes.pdf

NETWORK SECURITY

V Semester: CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC03	Core	3	1	0	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisite: Computer Networks

I.COURSE OVERVIEW:

The purpose of this course is to provide understanding of the main issues related to security in modern networked computer systems. This covers underlying concepts and foundations of computer security, basic knowledge about security-relevant decisions in designing IT infrastructures, techniques to secure complex systems and practical skills in managing a range of systems, from personal laptop to large-scale infrastructures.

II.COURSE OBJECTIVES:

The students will try to learn:

1. The Fundamental practices, policies, technologies and standards in providing security on network.
2. The TCP/IP networking mechanism to diagnose the security problems in network.
3. The different network and communication protocols presence in the network to apply some security factors.

III.COURSE SYLLABUS:

MODULE-I: INTRODUCTION ON NETWORKING AND SECURITY (10)

Access Control and Site Security- Virtual Local Area Network (VLAN), Demilitarized zone (DMZ)attacks, services mechanisms. Attack Methods–TCP/IP, Internetworking, Security problems in TCP/IP protocol suite, BGP security attacks, DNS Cache poisoning, Denial of Service(DoS) attacks, Distributed Denial of Service (DDoS) attacks, IP Trace back attacks.

MODULE-II: REAL-TIME COMMUNICATION SECURITY (08)

Introduction to TCP/IP protocol stack – Implementation layers for security protocols and implications-Psec: A Hand ESP-IP sec: IKE-SSL/TLS –Distribution lists –Establishing keys-Privacy, Source Authentication, Message Integrity, Non-Repudiation, Proof of Submission, Proof of Delivery, Message Flow Confidentiality, Anonymity–Packet filters – Application level gate ways.

MODULE-III: INTERNET CONTROL MESSAGE PROTOCOL (ICMP) (09)

ICMP Messages - Attacks Using ICMP Messages - reconnaissance scanning - ICMP sweep-Trace route - firewalk - inverse mapping - OS finger printing - exploiting systems.

ICMP- ICMP Route Redirect - ICMP informational messages - ICMP Router Discovery Messages - ICMP Floods - Smurf-Keeping Access Covering the Tracks.

MODULE-IV: ELECTRONIC MAIL SECURITY (09)

Pretty Good Privacy–PGP services–Transmission and Reception of PGP Messages– PGP message generation–PGP message reception.

MODULE-V WEB SECURITY (09)

Threats on the web –Secure Socket Layer and Transport Layer Security: SSL architecture– SSL record protocol–Handshake protocols.

IV. TEXT BOOKS:

1. W. Stallings, “Cryptography and Network Security: Principles and Practice”, Boston: Prentice Hall, 5th Edition, 2010.
2. A.Das and C.Veni Madhavan, “Public-key Cryptography: Theory and Practice”, New Delhi, India: Pearson Education India, 2009.

DATABASE SECURITY

V Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC04	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There is no prerequisite to take this course

I.COURSE OVERVIEW:

Database security has a great impact on the design of today's information systems. This course will provide an overview of database security concepts and techniques and discuss new directions of database security in the context of Internet information management. The topics will cover database application security models, database and data auditing, XML access control, trust management and privacy protection.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The fundamentals of security related to database system.
II. The security mechanisms to solve the problems.
III. The essentials of secure software design.
IV. The various types of attacks and intruder detection system.
V. The secure database model for new generations.

III. COURSE SYLLABUS:

MODULE-II NTRODUCTION AND SECURITY MODEL-I (09)

Introduction to databases security problems in databases security controls conclusions; Security models: Introduction access matrix model; Take-grant model; Acten model; PN model; Hartson and Hsiao's Model; Fernandez's model Bussolati and Martella's model for distributed databases.

MODULE-II SECURITY MODEL-II AND SECURITY MECHANISMS (09)

Security models 2: Bell and LaPadula's model; Bib‘s model; Dion's model; Sea view model; Jajodia and Sandhu's model; The lattice model for the flow control conclusion; Security mechanisms: User identification/authentication; Memory protection; Resource protection; Control flow mechanisms isolation security functionalities in someoperating systems; Trusted computer system evaluation criteria.

MODULE-III SECURITY SOFTWARE DESIGN (09)

Introduction: A methodological approach to security software design; Secure operating system.

Design secure DBMS; Design security packages database security design.

MODULE-IV STATISTICAL DATABASE PROTECTION AND INTRUSION DETECTION SYSTEMS (08)

Discovery introduction statistics concepts and definitions; Types of attacks; Inference controls evaluation criteria for control comparison; Introduction IDES system; RETISS system; ASES system.

MODULE-V MODELS FOR THE PROTECTION OF NEW GENERATION DATABASE SYSTEMS-1&DATABASE SYSTEMS-2 (10)

Models for the protection of new generation database Systems-1: A model for the protection of frame basedsystems; A model for the protection of object-oriented systems: SORION model for the protection of object-oriented databases; models for the protection of new generation database systems-2: The orion model, Jajodia and Kogan's model; A model for the protection of active databases conclusions.

IV. TEXT BOOKS:

1. Hassan A, Afyouni, “Database Security and Auditing: Protecting Data Integrity and Accessibility**”,** Cengage Learning, 1st Edition, 2009.
2. Maria Grazia Fugini, Silvana Castano, Giancarlo Martella, “Database Security”, Pearson Education, 1st Edition, 1994.

V. REFERENCE BOOKS:

1. Alfred Basta, Melissa Zgola, “Database Security”, Cengage Learning, 1st Edition, 2012.

VI. WEB REFERENCES:

1. http://www.applicure.com/blog/database-security-best-practice
2. https://docs.oracle.com/cd/B19306_01/network.102/b14266/apdvntro.htm#DBSEG12000
3. http://www.cse.msu.edu
4. http://cms.gcg11.ac.in/

NETWORK PROGRAMMING AND MANAGEMENT

V Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC05	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Computer Networks

I.COURSE OVERVIEW:

This course teaches the methods and tools used for network programming and management. Topics include: Socket programming which tells about TCP and UDP sockets, network protocols;It also explains how information is secured and managed in network. Finally, it gives information about advanced sockets which includes IPV4,IPV6 interoperability.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The basic concepts of connection oriented communication over network.
II. The concepts of multiplexing in client server environment.
III. The functions and protocols needed for connection less communication over networks.
IV. The management concepts and practical issues of simple network management protocols.

III. COURSE SYLLABUS:

MODULE-I: ELEMENTARY TCP SOCKETS (09)

Introduction to socket programming, overview of TCP/IP protocols, introduction to Sockets, socket address structures, byte ordering functions, address conversion functions, elementary TCP sockets, socket, connect, bind, listen, accept, read, write, close functions, iterative server, concurrent server.

MODULE-II: APPLICATION DEVELOPMENT (09)

TCP echo server, TCP echo client, posixsignal handling, server with multiple clients; Boundary conditions: Server process crashes, server host crashes, server crashes and reboots, server shutdown, I/O multiplexing, I/O Models, select function, shutdown function, TCP echo server (with multiplexing), poll function, TCP echo client (with multiplexing).

MODULE-III: SOCKET OPTIONS, ELEMENTARY UDP SOCKETS (09)

Socket options, getsocket and setsocket functions, generic socket options, IP socket options, ICMP socket options, TCP socket options, elementary UDP sockets, UDP echo server, and UDP echo client.

Multiplexing TCP and UDP sockets, domain name system, and gethostbyname function, Ipv6 support in DNS, gethostbyadr function, getservbyname and getserv by port functions.

MODULE-IV: ADVANCED SOCKETS (09)

Ipv4 and Ipv6 interoperability, threaded servers, thread creation and termination, TCP echo server using threads, mutexes, condition variables, raw sockets, raw socket creation, raw socket input, raw socket output, ping program, trace route program.

MODULE-V: SIMPLE NETWORK MANAGEMENT (09)

SNMP network management concepts, SNMP management information, standard MIB‘s, SNMPv1 protocol and practical issues, introduction to RMON, SNMPv2 andSNMPv3.

IV. TEXT BOOKS:

1. W. Richard Stevens, “UNIX Network Programming Vol-I”, Pearson Education, 3rd Edition, 2008.
2. Mani Subramanian, “Network Management: Principles and Practice”, Addison Wesley, 1st Edition, 2001.

V. REFERENCE BOOKS:

1. D.E. Comer, “Internetworking with TCP/IP Vol- III”, (BSD Sockets Version), Pearson Education, 2nd Edition, 2003.
2. William Stallings, “SNMP, SNMPv2, SNMPv3 and RMON 1 and 2”, Addison Wesley, 3rd Edition, 1999.

VI. WEB REFERENCES:

1. https://notes.shichao.io/unp/ch4/
2. https://books.google.co.in/books?isbn=8184317565
3. https://docs.oracle.com/cd/E19683-01/817-0573/transition-tbl-16/index.html
4. https://docs.oracle.com/cd/E26502_01/html/E35299/sockets-22932.html

SOFTWARE DEFINED NETWORKS

V SEMESTER: CSE(CS)

Course Code	Category	Hours /Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC06	Elective	3	0	0	3	30	70	100
Contact Classes:45	Total Tutorials: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There is no prerequisite to take this course

I.COURSE OVERVIEW:

This course emphasizes on different control planes in Centralized and Distributed systems. Network programmability and the concepts related to Datacenters are explained. It also gives information about Network function virtualization and the topology used. Finally this course tells about how to build a software defined network using network virtulization.

II.COURSE OBJECTIVES:

The students will try to learn:

I. The introduction of distributed control planes.
II. The architectures, algorithms, protocols and applications of data center networks
III. The concepts of Software defined network framework.
IV. The network virtualization for handling bigdata.

III. COURSE SYLLABUS:

MODULE -I: CENTRALIZED AND DISTRIBUTED CONTROL PLANES (09)

Introduction, distributed control planes; Centralized control planes open flow: Introduction; Hybrid Approaches SDN Controllers: Introduction General Concepts Layer 3 Centric Plexxi Cisco OnePK.

MODULE -II: NETWORK PROGRAMMABILITY AND DATA CENTER CONCEPTS (09)

Network programmability: Introduction, the management interface, the application-network divide, modern programmatic interfaces, I2RS, modern orchestration; Data center concepts and constructs: Introduction, the multitenant data center, the virtualized multitenant data center, SDN solutions for the data center network, LANs, EVPN, VxLan, NVGRE.

MODULE -III: NETWORK FUNCTION VIRTUALIZATION AND NETWORK TOPOLOGY (09)

Network function virtualization: Introduction, virtualization and data plane I/O, services engineered path, service locations and chaining, NFV at ETSI, Non-ETSI NFV Work.

Network topology and topological information abstraction: Introduction, network topology, traditional methods, LLDP, BGP-TE/LS, ALTO, I2RS topology.

MODULE -IV: BUILDING AN SDN FRAMEWORK (09)

Building an SDN framework: Introduction, build code first; ask questions later, the Juniper SDN framework, IETF SDN framework(s), open daylight controller/framework, policy, use cases for bandwidth scheduling, manipulation, and calendaring: introduction, bandwidth calendaring, big data and CSPF, expanding topology, use cases for data center overlays, big data, and network function virtualization, introduction, data center orchestration, puppet (DevOps Solution).

MODULE -V: NETWORK FUNCTION VIRTUALIZATION (NFV) (09)

Network Function Virtualization (NFV): Optimized big data, use cases for input traffic monitoring; Classification and triggered actions: Introduction, the firewall, firewalls as a service, network access control replacement, extending the use case with a virtual firewall, feedback and optimization, intrusion detection/threat mitigation.

IV TEXT BOOKS:

1. Thomas D. Nadeau, Ken Gray, "Software Defined Networks An Authoritative Review of Network Programmability Technologies", O'Reilly Media Publisher , 2nd Edition, 2013.

V REFERENCE BOOKS:

Paul Goransson, Chuck Black, Morgan Kaufmann,"Software Defined Networks: A Comprehensive Approach", 1st Edition, 2014.

VI. WEB REFERENCES:

1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf
2. http://www.menog.org/presentations/menog-15/341-MENOG_SDN_April.pdf

SECURITY COMPUTING

V SEMESTER: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC07	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: Computer Networks

I.COURSE OVERVIEW:

This course covers fundamental issues and first principles of security and information assurance. The course will look at the security policies, models and mechanisms related to confidentiality, integrity, authentication, identification, and availability issues related to information and information systems. Different types of Security in networks and in Databases is known.

II.COURSE OBJECTIVES:

The students will try to learn:

I. The security managing techniques in different domains.
II. The vulnerabilities in programs and how to overcome them.
III. The different kinds of security threats in networks databases and the different solutions available.
IV. The models and standards for security.

III.COURSE SYLLABUS:

MODULE-I: ELEMENTARY CRYPTOGRAPHY

Terminology and background – substitution ciphers – transpositions – making good encryption algorithms- data encryption standard- AES encryption algorithm – public key encryption – cryptographic hash functions – keyexchange – digital signatures – certificates.

MODULE –II: PROGRAM SECURITY

Secure programs – non-malicious program errors – viruses – targeted malicious code – controls against program threat – control of access togeneral objects – user authentication – good coding practices – open web application security project top 10 flaws – common weakness enumeration top 25 most dangerous software errors.

MODULE –III: SECURITY IN NETWORKS

Threats in networks – encryption – virtual private networks – PKI – SSH – SSL– IPSec – content integrity – access controls – wireless security.

Honeypots –Traffic flow security – firewalls – intrusion detection systems – secure e-mail.

MODULE –IV: SECURITY IN DATABASES

Security requirements of database systems – Reliability and Integrity in databases – two phase update – redundancy/internal consistency – recovery– concurrency/consistency – monitors – sensitive data – types of disclosures –inference.

MODULE –V: SECURITY MODELS AND STANDARDS

Secure SDLC – Secure application testing – security architecture models –trusted computing base – bell-LaPadula confidentiality model – Biba integrity model – Graham-denning access control model – Harrison-Ruzzo-Ulman model– secure frameworks – COSO – CobiT – compliances – PCI DSS – security standards - ISO 27000 family of standards – NIST.

IV.TEXT BOOKS:

1. Charles P. Pfleeger, Shari Lawrence Pfleeger, “Security in Computing”, Pearson Education, 4th Edition, 2007.
2. Michael Whitman, Herbert J. Mattord, “Management of Information Security”, Course Technology, 3rd Edition 2010.

V.REFERENCE BOOKS:

1. William Stallings, “Cryptography and Network Security: Principles and Practices”, Prentice Hall, 5th Edition 2010.
2. Michael Howard, David LeBlanc, John Viega, “24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them”, First Edition, Mc Graw Hill Osborne Media, 2009..
3. Matt Bishop, “Computer Security: Art and Science”, First Edition, Addison-Wesley, 2002.

VI.WEB REFERENCES:

1. https://www.owasp.org/index.php/Top_10_2010
2. https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml
3. http://cwe.mitre.org/top25/index.html.

EXPERIENTIAL ENGINEERING EDUCATION (ExEEd) – PROJECT BASED LEARNING

V Semester: Common for all branches

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC20	Foundation	2	-	-	1	30	70	100
Contact Classes: 36	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 36		

Prerequisite: There are no prerequisites to take this course

I. COURSE OVERVIEW:

Project-based learning (PBL) is collaborative, learner-centered instructional approach where students work in groups to construct their knowledge using modern tools. It often requires students to collaborate, design, revise, and share their ideas and experiences with authentic audiences and supportive peer groups rather than collect resources, organize work, and manage long-term activities. Project-Based Learning begins with the assignment of tasks that will lead to the problem identification, modeling, simulation and analyzing the results.

II. COURSE OBJECTIVES:

The students will try to learn:

I. To emphasize learning activities that is long-term, interdisciplinary and student-centric.
II. To inculcate independent learning by problem solving with social context.
III. To engages students in rich and authentic learning experiences.
IV. To provide every student the opportunity to get involved either individually or as a group so as to develop team skills and learn professionalism.

III. COURSE SYLLABUS

I. Defining the Problem
II. Gathering requirements
III. Design / *Mode*ling
IV. Implementation
V. Testing
VI. Report

WEB APPLICATION DEVELOPMENT LABORATORY

V Semester : CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC10	Core	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

Prerequisites: Data structures and Object oriented programming

I. COURSE OVERVIEW:

This course will give you the basic background, terminology and fundamental concepts that you need to understand in order to build modern web applications. This course introduces students to developing web applications. This course presents the basics of HTML5 and CSS3, Other HTML tags for Web application Development. Learn to create links in HTML, Uses of HTML forms. Introduction to the use of Reactstrap for Bootstrap 4-based responsive UI design. React router and its use in developing single-page applications, designing controlled forms. Redux and use it to develop React-Redux powered applications, client-server communication and the use of REST API on the server side, React primitives render to native platform UI. It introduces the usage of the front-end framework Bootstrap and exposes students to basic security mechanisms for server-side web application development.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Programming skills in Html5, CSS3, Bootstrap 4.
II. Developing skills of Web Applications user interactions using JavaScript (i.e. ES6+).
III. Web application Development Database with React and React Native.

III. COURSE SYLLABUS:

Week – 1: HTML LAYOUTS AND LINKS

a. Develop a web application to control over different layouts.
b. Create a webpage with HTML describing your department use paragraph and list tags.
c. Apply various colors to suitable distinguish key words, also apply font styling like italics, underline and two other fonts to words you find appropriate, also use header tags.
d. Create links on the words e.g. “Wi-Fi” and “LAN” to link them to Wikipedia pages.

Week – 2: WEB APLLICATION DESIGN FORMTTING

a. Develop a web application with background banner image and navigation menus.
b. Develop a web application with responsive images.
c. Develop a web application using left menu.
d. Develop setting to change the theme of entire web Application.

Week – 3: INTRODUCTION TO RESPONSIVE INTERFACE USING BOOTSRAP.

a. Write code for developing responsive web application with Admin panel and tables with static data.

Week – 4:BUIDLING INTERFACES USING JAVASCRIPT

a. Set up the Folder Structure.
b. Write the Model code and initialize the application.
c. Implement the list objects and use cases.
d. Implement the create object use case.
e. Implement the update object use case.

Week – 5: INTRODUCTION TO INTERATIVE FORMS AND AJAX DATA BINIDNG

a. Developing Web Page Styles using JavaScript and CSS,
b. Develop Script interactive forms
c. Data binding using Ajax.

Week – 6: REACT ENVIRONMENT SETUP

a. Setting up development environment.
b. Integration with Existing Apps.
c. Running on Device.
d. Debugging
e. Testing
f. Write source code using Typescript.

Week – 7: PROGRAMMING WITH REACT

a. Basics Interactive examples.
b. Function Components and Class Components
c. React Native Fundamental, Handling Text Input,
d. Using a scroll View, using List View.
e. Platform Specific Code.

Week – 8: BUILD A DRUNKEN SNAKE GAME USING HOOKS

a. Introduction and scaffolding the project.
b. Components, Props and Styles.
c. State and Lifecycle Events.
d. Extended Game Functionality.
e. Finishing up and Deployment.

Week – 9: PHP SESSIONS BOX React FOR DATA VISUALIZATION

a. Introduction and scaffolding the Project.
b. Pages and Layout.
c. Working with an API, CSS-in-JS.
d. Dynamic Pages and React Hooks.
e. Custom React Hooks, Dynamic CSS-in-JS.
f. Finishing up and Deployment.
g. Optimization and PWA.

Week – 10:CHAT APPLICATION

a. Firebase Environment. Introduction and Scaffolding the project.
b. Private and Public pages, Context API.
c. Creating Side bar and Dashboard
d. Creating and displaying Chat Rooms.
e. Creating Layout for Chat page.

Week – 11: CHAT APPLICATION API RESPONSES

a. Context API Problem-solution for the chat messages.
b. Denormalization of the data to be stored in app.
c. Displaying chat feed for Interactive UI along with Real time user presence.

Week – 12: DATABASES HANDLING

a. Role Based Access.
b. Messages Likes and deletion.
c. File and Audio Chat Messages
d. Extended Chat Features and Deployment

V. REFERENCE BOOKS:

1. Adam BoduchandRoy Derks, “React and React Native: A Complete Hands-on Guide to Modern Web and Mobile Development with React.js”, 3rd Edition, 2020.
2. W Hans Bergsten, “Java Server Pages”, O’Reilly, 3rd Edition, 2003.
3. D. Flanagan, “Java Script”, O’Reilly, 6th Edition, 2011.
4. Jon Duckett, “Beginning Web Programming”, WROX, 2nd Edition, 2008.

VI. WEB REFERENCES:

1. https://www.codecademy.com/learn/paths/web-development/
2. https://nptel.ac.in/courses/106/105/106105084/
3. https://medium.com/@aureliomerenda/create-a-native-web-app-with-react-native-web-419acac86b82

4. https://www.coursera.org/learn/react-native
5. https://desirecourse.net/react-native-and-redux-course-using-hooks/

NETWORK SECURITY LABORATORY

V Semester: CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC08	Core	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

Prerequisite: Python Programming

I.COURSE OVERVIEW:

The purpose of this course is to provide understanding of the main issues related to security in modern networked computer systems. This covers underlying concepts and foundations of computer security, basic knowledge about security-relevant decisions in designing IT infrastructures, techniques to secure complex systems and practical skills in managing a range of systems, from personal laptop to large-scale infrastructures.

II.COURSE OBJECTIVES:

The students will try to learn:

1. The different packet crafting techniques using different Networking tools.
2. The different network Script programmes to measure the performance of Network.
3The understanding of different Protocols that measure the scope and lifetime of network.

III.COURSE SYLLABUS:

Week – 1:
Implement UDP and TCP Packet crafting techniques using HPING3

Week – 2:
Demonstrate the understanding the nmap and Zenmap tool.

Week – 3:
Scanning network of the target about half open and full open scan.

Week – 4:
Scanning network of the target using scan techniques

Week – 5:
Scanning the network of the target using host discovery content.

Week – 6:
Scanning the network of the target using the port specification.

Week – 7:
Scanning the network of the target using the Service and version detection

Week – 8:
Scanning the network of the target using OS detection

Week – 9:
Scanning the network of the target using the Timing and Performance

Week – 10:
Scanning the network of the target using the NSE scripts

Week – 11:
Scanning the network of the target using Firewall/IDS evasion and spoofing

Week – 12:
Checking for the livesystems using Angry IP scanner tool

IV.TEXT BOOKS:

1. W. Stallings, "Cryptography and Network Security: Principles and Practice", Boston: Prentice Hall, 5th Edition, 2010.
2. A.Das and C.Veni Madhavan, "Public-key Cryptography: Theory and Practice", New Delhi, India: Pearson Education India, 2009.

COMPETITIVE PROGRAMMING USING GRAPH ALGORITHMS

V SEMESTER: Common for CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)

CourseCode	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC22	Skill	-	-	-	-	-	-	-
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: Nil		

I. COURSE OVERVIEW:

Graph theory is a study of graphs, trees and networks. Topics that will be discussed include Euler formula, Hamilton paths, planar graphs and coloring problem; the use of trees in sorting and prefix codes; useful algorithms on networks such as shortest path algorithm, minimal spanning tree algorithm and min-flow max-cut algorithm.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The concepts of graph theory along with applications.
II. The connected graphs and algorithms for traversability.
III. The directed graphs and planar graphs along with applications.

III.COURSE SYLLABUS:

MODULE–I:FUNDAMENTAL CONCENTS OF GRAPH THEORY

Review of graph theory and its real time applications, definitions of graphs and digraphs with examples ,properties ,practice problems, konigsberg bridge problem , shortest path problem, matrix representation of graphs and sub graphs , incidence and adjacency matrices explanation, problems on incidence and adjacency matrices, complete graphs , regular graphs ,petersen graph , handshaking lemma, problems, bipartite graphs , isomorphism of graphs, applications, definitions of degree, complement, finite graph, induced graph ,explanation, problems, Petersen graph, handshaking lemma, examples, commonly used terminology, definitions and explanation, union, sum, cartesian product, composition of graphs, casestudy: graph theoretic model of the lan problem, comparative study of graph isomorphism applications.

MODULE–II:CONNECTED GRAPHS AND SHORTEST PATHS

Definitions of walks, trails, paths, cycles, distance , explanation, problems, cut, vertices and cut, edges, blocks, explanation, practice problems, definitions of vertex degree, girth, automorphism, counting, explanation, practice problems, connectivity, weighted graphs and shortest paths, explanation, Dijkstra‟S shortest path algorithm, explanation, Floyd-Warshall shortest path algorithm, real time applications of walks, paths, cycles and blocks, casestudy: Friend'ssuggestions in facebook,.

MODULE–III:TRAVERSABILITY

Eulerian Path , Eulerian Cycles , Eulerian Graph , Application Problem, Hamiltonian Path , Hamiltonian Graph , RealTimeApplicationProblem,Dirac'sTheorem,Applications,Fleury'sAlgorithmForFindingEulerianPathsOrCycles,Traveling Salesman Problem – Explanation, Chinese Postman Problem – Explanation, Definitions Of Tree , Spanning Tree , , Tree Enumeration –Explanation, Trees In Computer Science , Rooted Tree , Binary Tree – Explanation, Definitions Of Cut Edge , Bond , Centers And Centroids , Steiner Trees –Explanation, Properties Of Tree – Problems, Cayley's Theorem Proof And Application , Prufer Codes, Cayley's Formula , Recursion Formula – Problems, Case Study: Planning Bus Route To Pick Up Students, Real Time Application: Railway Network Connector Problem.

MODULE–IV:DIRECTED GRAPHS

directed paths, tournaments and cycles, explanation, practice problems, Eulerian digraph, Euler Tours, Hamilton cycles – problems, connectivity and strongly connected digraphs – explanation, cyclic graphs, acyclic graphs, directed acyclic graphs (dag) –problems, graphic sequence-functional graph, weakly connected graph – strongly connected graph– explanation, topological sorting algorithm– explanation, casestudy: topological sort for sentence ordering.

MODULE–V:PLANARITY

Definitions of plane, planar graphs, non-planar graphs, outer planar graphs–explanation, combinational versus geometric graphs, detection of planarity–explanation, maximal planar graphs , nonplanarity of k5 and k3,3 –explanation, Kuratowski's theorem , explanation – applications, planar dual , definition –problems, Euler's formula, tailt's conjecture , planar embedding of trees and graphs , definitions – explanation, genus, thickness and crossing number of graphs – explanation, planar separability theorem – explanation, case study: design thesnake and ladder game, application of graph theory in data structures, operating system, computer hardware, information retrival, image processing, artificial intelligence.

IV. TEXTBOOKS:

1. DouglasB.West, “Introduction to Graph Theory”, Pearson Singapore, 2nd Edition, 2000.
2. ArthurBenjamin, Gary Chartrand, and Ping Zhang, “The Fascinating World of Graph Theory”, Princeton University Press, 2015.

V.REFERENCEBOOKS:

1. Frank Harary, “Graph Theory”, CRC Press, 2018.
2. PaulZeitz, “The Art and Craft of Problem Solving”, Wiley, 3rd Edition, 2016.
3. NarsinghDeo, “Graph Theory with Applications to Engineering & Computer Sciences”, Prentice Hall, 1974.

PENETRATION TESTING AND CYBER OPERATIONS

VI Semester: CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC09	Core	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Foundations of Cyber security

I.COURSE OVERVIEW:

This course mainly focuses on port scanning and web application scanning. It also gives information about different attacks like password attacks and detection of vulnerabilities. This covers wireless security and penetration tools like Trace routes and Neo trace. Information about Database access and security at different levels is also defined.

II.COURSE OBJECTIVES:

The students will try to learn:

I. The tools that can be used to perform information gathering.
II. The various attacks in various domains of cyberspace.
III. The various operating systems and wireless environment.
IV. The vulnerability assessment can be carried out by means of automatic tools or manual investigation.
V. The vulnerabilities associated with various network applications and database system.

III.COURSE SYLLABUS:

MODULE 1:INFORMATION GATHERING AND DETECTING VULNERABILITIES (09)

Open source intelligence gathering – port scanning – nessus policies – web application scanning manual analysis- traffic capturing.

MODULE 2:ATTACKS AND EXPLOITS (09)

Password Attacks Client side Exploitation Social Engineering – By passing Antivirus Applications.
Metasploit Payloads Open php My Admin-Buffer overflow: Windows and Linux, Web scanning exploits, port scanning exploits, SQL exploits.

MODULE 3: WIRELESS SECURITY (09)

Wired vs wireless Privacy Protocols – Wireless Frame Generation Encryption Cracking Tools-Wireless DoS Attacks

MODULE 4:COMMON VULNERABILITY ANALYSIS OF APPLICATION PROTOCOLS (09)

Simple Mail Transfer Protocol – File Transfer Protocol – Trivial File Transfer Protocol-Hyper Text Transmission Protocol- ICMP SMURF-UDP-DNS-PING-SYN.

MODULE 5:PENETRATION TOOLS AND DATABASE SECURITY (09)

Trace routes, Neo trace, What web. Database Security: Access control in database systems – Inference control-Multilevel database security.

IV. TEXT BOOKS

1. GeorgiaWeidman, “Penetration Testing: A Hands on Introduction to Hacking”, No Startch Press, 1st Edition, 2014.
2. B.Singh, H. Joseph and Abhishek Singh, “Vulnerability Analysis and Defense for the Internet”, Springer, 2008.

V. REFERENCE BOOKS

1. Rafay Baloch, “Ethical Hacking and Penetration Testing Guide”, CRC Press, 2015,
2. Dr.Patrick Engebretson, “The Basics of Hacking and Penetration Testing”, Syngress Publications Elseveir, 2013.
3. Prakhar Prasad, “Mastering Modern Web Penetration Testing”, Packtet Publishing, 2016.

DATA MINING AND KNOWLEDGE DISCOVERY

VI SEMESTER: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACIC01	Core	3	1	0	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes:60		

Prerequisite: There are no prerequisites to take this course

I. COURSE OVERVIEW:

Data mining refers to extracting or mining knowledge from large amounts of data. It emphasizes various techniques and algorithms used to explore, analyze and leverage data and turn it into valuable and actionable information. It includes data warehousing and data mining functionalities such as association mining, classification, clustering and outlier analysis. The techniques are used to tackle data centric applications in various domains such as financial analysis, telecommunication industry, intrusion detection, and complex data mining applications in stream, web, text, spatial and other scientific applications.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The scope and essentiality of data warehousing and mining.
II. The analyses of data, choosing relevant models and algorithms for respective applications.
III. The process and mining of complex datatypes such as streams, spatial, web and multimedia.
IV. The research perspectives towards advances in data mining.

III. COURSE SYLLABUS:

MODULE – I: KNOWLEDGE DISCOVARY (09)

Data Mining definition, knowledge discovery in data (KDD), kinds of data can be mined, kinds of patterns / data mining functionalities, technologies, applications, issues in data mining. data objects and attribute types, basic statistical descriptions of data, data visualization, measuring data similarity and dissimilarity.

MODULE – II: DATA PREPROCESSING (08)

Data Preprocessing: Data quality, major tasks in data preprocessing, data cleaning, data integration and transformation, data reduction, data discretization.

MODULE – III: DATA WAREHOUSING AND ONLINE ANALYTICAL PROCESSING (09)

Data warehouse concepts, differences between operational database systems and data warehouses, a multitiered architecture; Data Warehouse Models: Enterprise warehouse, data mart, and virtual warehouse, extraction, transformation, and loading, metadata repository, a multidimensional data model; Schemas for Multidimensional Data Models: Stars, snowflakes, and fact constellations, dimensions, measures, OLAP operations, a starnet query model for querying multidimensional databases.

Business Analysis framework for data warehouse design, data warehouse design process, data warehouse implementation, indexing OLAP data, OLAP server architectures, data generalization by attribute, oriented induction

MODULE – IV: MINING FREQUENT PATTERNS AND CLASSIFICATION (10)

Market basket analysis, frequent itemsets, closed itemsets, and association rules, frequent itemset mining methods; Apriori algorithm, generating association rules from frequent itemsets, improving the efficiency of Apriori Pattern-Growth Approach. Classification: Basic concepts, decision tree induction, Bayesian belief networks, classification by back propagation, support vector machines, classification using frequent patterns, lazy learners, other classification methods, model evaluation and selection, techniques to improve classification accuracy.

MODULE –V: CLUSTERING AND RESEARCH FRONTIERS (09)

Cluster Analysis, Partitioning methods, hierarchical methods, density-based methods, grid based methods, evaluation of clustering.

Mining Complex Types of Data: Mining Sequence Data: Time-series, symbolic sequences, and biological sequences, mining graphs and networks.

IV. TEXT BOOKS:

1. Jiawei Han, MichelineKamber, “Data Mining, Concepts and Techniques”, Morgan Kaufmann Publishers, Elsevier, 3rd Edition, 2012.
2. Alex Berson, Stephen J.Smith, “Data warehousing Data mining and OLAP”, Tata McGraw-Hill, 2nd Edition, 2007.

V. REFERENCE BOOKS:

1. Arum K Pujari, “Data Mining Techniques”, Universities Press, 3rd Edition, 2005.
2. PualrajPonnaiah, “Data Warehousing Fundamentals”, Wiley, Student Edition, 2004.
3. Ralph Kimball, “The Data Warehouse Life Cycle Toolkit”, Wiley, Student Edition, 2006.
4. VikramPudi, P Radha Krishna, “Data Mining”, Oxford University, 1st Edition, 2007.

VI. WEB REFERENCES:

1. http://www.anderson.ucla.edu
2. https://www.smartzworld.com
3. http://iiscs.wssu.edu

COMPILER DESIGN

VI Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC40	Core	3	1	0	4	30	70	100
Contact Classes: 45	**Total Tutorials: 15**	**Total Practical Classes: Nil**				**Total Classes: 60**		

Prerequisite: Theory of Computation

I. COURSE OVERVIEW:

The course will teach the students to the fundamental concepts and techniques used for building a simple compiler. It focuses on both theory and practice, we will use a sample language to explore the lexical, syntactic and semantic structures of programming languages, and learn to use those structures in implementing a demonstrative compiler. The main objective of this course is to introduce the major concept areas of language translation and compiler design and to develop an awareness of the function and complexity of modern compilers. This course describes the theory and practice of compilation, in particular, the lexical analysis, parsing and code generation and optimization phases of compilation, and design a compiler for a concise programming language.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The basic principles of compiler design, and its various constituent parts, algorithms and data structures required to be used in the compiler.
II. The code generation algorithms to get the machine code for the optimized code.
III. Identify different methods of lexical analysis.
IV. The various parsers and develop appropriate parser to produce parse tree representation of the input.
V. Exercise and reinforce prior programming knowledge with a non-trivial programming project to construct a compiler.

III. COURSE SYLLABUS:

MODULE –I : INTRODUCTION TO COMPILERS (08)

Introduction to compilers: Definition of compiler, interpreter and its differences, the phases of a compiler; Lexical Analysis: Role of lexical analyzer, input buffering, recognition of tokens, finite automata, regular Expressions, from regular expressions to finite automata, pass and phases of translation, bootstrapping, LEX, lexical analyzer generator.

MODULE –II: SYNTAX ANALYSIS (09)

Syntax Analysis: Parsing, role of parser, context free grammar, derivations, parse trees, ambiguity, elimination of left recursion, left factoring, eliminating ambiguity from dangling-else grammar; Types of parsing: Top-down parsing, backtracking, recursive-descent parsing, predictive parsers, LL (1) grammars. Bottom-up parsing: Definition of bottom-up parsing, handles, handle pruning, stack implementation of shift-reduce parsing, conflicts during shift-reduce parsing, LR grammars, LR parsers-simple LR, canonical LR and Look Ahead LR parsers, YACC, automatic parser generator.

MODULE –III: SYNTAX-DIRECTED TRANSLATION AND INTERMEDIATECODE GENERATION (10)

Syntax-Directed Translation: Syntax directed definitions, construction of syntax trees, S-attributed and L, attributed definitions; Syntax Directed Translation schemes.

Intermediate code generation: Intermediate forms of source programs– abstract syntax tree, polish notation and three address code, types of three address statements and its implementation, syntax directed translation into three address code, translation of simple statements, Boolean expressions and flow-of-Control statements.

MODULE –IV: TYPE CHECKING AND RUN TIME ENVIRONMENT(09)

Type checking: Definition of type checking, type expressions, type systems, static and dynamic checking of types, specification of a simple type checker; Run time environments: Source language issues, storage organization, storage-allocation strategies, access to nonlocal data on the stack, garbage collection, symbol tables.

MODULE –V: CODE OPTIMIZATION AND CODE GENERATION(09)

Code optimization: The principle sources of optimization, optimization of basic blocks, loops in flow graphs, peephole optimization; Code Generation: Issues in the Design of a Code Generator, The Target Language, addresses in the Target Code, Basic Blocks and Flow Graphs, Optimization of Basic Blocks, A Simple Code Generator, register

allocation and assignment, DAG representation of basic blocks.

IV. TEXT BOOKS:

1. Alfred V.Aho, Ravi Sethi, Jeffrey D, Ullman, "Compilers–Principles, Techniques and Tools", Pearson Education, 2nd Edition, 2006.

V. REFERENCE BOOKS:

1. Kenneth C. Louden, Thomson, "Compiler Construction–Principles and Practice", PWS Publishing, 1st Edition, 1997.
2. Andrew W. Appel, "Modern Compiler Implementation C", Cambridge University Press, Revised Edition, 2004.

VI. WEB REFERENCES:

1. www.vssut.ac.in/lecture_notes/lecture1422914957.pdf
2. http://csenote.weebly.com/principles-of-compiler-design.html
3. http://www.faadooengineers.com/threads/32857,Compiler-Design-Notes-full-book-pdf-download
4. https://www.vidyarthiplus.com/vp/thread,37033.html#.WF0PhlMrLDc

AGILE SOFTWARE DEVELOPMENT APPROACHES

VI Semester: CSE(CS)

Course Code	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACIC04	Elective	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				TotalClasses:45		

Prerequisite: Object Oriented Software Engineering

I. COURSEOVERVIEW:

Software industry is going crazy on agile methods. It is rapidly becoming the choice for software development where requirements are unpredictable or is expected to change over time. This course will help you gain knowledge on what is agile? Why agile is better suited for these situations?. It also cover some of the most common agile frameworks like scrum and XP in depth. This course also makes student learn the fundamental principles and practices associated with each of the agile development methods. To apply the principles and practices of agile software development on a project of interest and relevance to the student.

II. COURSEOBJECTIVES:

The students will try to learn:

I. The tradeoffs in selecting a software engineering method.
II. The practices and philosophies of agile methods.
III. The fundamental principles of scrum.
IV. The in depth of Ability to understand and apply Extreme Programming.
V. How to tailor an agile method to the needs of the project.

III. COURSE SYLLABUS:

MODULE–I: INTRODUCTION AGILE SOFTWARE DEVELOPMENT(09)

Basics and Fundamentals of Agile Process Methods, Values of Agile, Principles of Agile, stakeholders, Challenges Lean Approach: Waste Management, Kaizen and Kanban, add process and products add value. Roles related to the lifecycle, differences between Agile and traditional plans, differences between Agile plans at different lifecycle phases. Testing plan links between testing, roles and key techniques, principles, understand as a means of assessing the initial status of a project/ How Agile helps to build quality.

MODULE–II:AGILE AND SCRUM PRINCIPLES (09)

Agile Manifesto, Twelve Practices of XP, Scrum Practices, Applying Scrum. Need of scrum, working of scrum, advanced Scrum Applications, Scrum and the Organization, scrum values.

MODULE– III:AGILE PRODUCT MANAGEMENT (09)

Communication, Planning, Estimation Managing the Agile approach Monitoring progress, Targeting and motivating the team, managing business involvement, Escalating issue. Quality, Risk, Metrics and Measurements, Managing the Agile approach Monitoring progress.

Targeting and motivating the team, managing business involvement and Escalating issue Agile Requirements: User Stories, Backlog Management. Agile Architecture: Feature Driven Development. Agile Risk Management: Risk and Quality Assurance, Agile Tools.

MODULE– IV:AGILE TESTING (09)

Agile Testing Techniques, Test-Driven Development, User Acceptance Test Agile Review: Agile Metrics and Measurements, The Agile approach to estimating and project variables, Agile Measurement, Agile Control: The 7 control parameters. Agile approach to Risk, The Agile approach to Configuration Management, The AternPrinciples, AternPhilosophy,The rationale for using Atern, Refactoring, Continuous integration, Automated Build Tools.

MODULE– V:SOFTWARE TESTING AND RELIABILITY (09)

Scaling Agile for large projects: Scrum of Scrums, Team collaborations, Scrum, Estimate a Scrum Project, Track Scrum Projects, Communication in Scrum Projects, Best Practices to Manage Scrum. Laboratory Work: Exploring the tools related to Agile Development and approached and develop small projects using this technology.

IV. TEXTBOOKS:

1. Robert C. Martin, "Agile Software Development, Principles, Patterns, and Practices" Alan Apt Series (2011).
2. Succeeding with Agile: "Software Development Using Scrum", Pearson (2010).

V. REFERENCE BOOKS:

1. Mark C. Layton, "Agile Project Management For Dummies", 3rd Edition, 2013.

SOFTWARE PROJECT MANAGEMENT

VI Semester: CSE(CS)

Course Code	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACIC05	Elective	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite: Object Oriented Software Engineering

I. COURSEOVERVIEW:

The main goal of software development projects is to create a software system with a predetermined functionality and quality in a given time frame and with given costs. For achieving this goal. models are required for determining target values and for continuously controlling these values. This course focuses on principles, techniques, methods & tools for model-based management of software projects. Assurance of product quality and process adherence (quality assurance), as well as experience-based creation & improvement of models (process management).

II. COURSE OBJECTIVES:

The students will try to learn:

I. The specific roles within a software organization as related to project and process management
II. The basic infrastructure competences (e.g., process modeling and measurement).
III. The basic steps of project planning, project management. Quality assurance, and process management and their relationships.

III. COURSE SYLLABUS:

MODULE–I:CONVENTIONAL SOFTWARE MANAGEMENT (09)

The waterfall model, conventional software Management performance. Evolution of Software Economics: Software Economics. Pragmatic software cost estimation.

MODULE–II:IMPROVING SOFTWARE ECONOMICS (09)

Reducing Software product size, improving software processes, improving team effectiveness. Improving automation, Achieving required quality, peer inspections. The old way and the new- The principles of conventional software engineering. Principles of modem software management, transitioning to an iterative process.

MODULE– III:LIFE CYCLE PHASES (09)

Engineering and production stages, inception. Elaboration, construction, transition phases. Artifacts of the process: The artifact sets. Management artifacts, Engineering artifacts, programmatic artifacts.

Model based software architectures: A Management perspective and technical perspective.

MODULE– IV:PROJECT ORGANIZATIONS (09)

Project Organizations Line-of- business organizations, project organizations, evolution of organizations, process automation. Project Control and process instrumentation the seven-core metrics, management indicators, quality indicators, life-cycle expectations, Pragmatic software metrics, metrics automation.

MODULE– V:CASE STUDIES (09)

CCPDS-R Case Study and Future Software Project Management Practices Modern Project Profiles, Next-Generation software Economics, Modern Process Transitions

IV. TEXTBOOKS:

1. Walker Royce, “Software Project Management”, Pearson Education, 6th Edition, 2000.
2. Bob Hughes & Mike Cotterell, “Software Project Management”, Tate McGraw H, 4th Edition, 2000.

V. REFERENCE BOOKS:

1. Andrew SteIbian 8 Jennifer Greene, “Applied Software Project Management”, O’Reilly. 2006.
2. Jennifer Greene & Andrew Steliman, “Head First PMP”, O RoiHy, 2007.
3. Richard H. Thayer & Edward Yourdon, “Software Engineering Project Management”, Wiley India, 2nd Edition, 2004.
4. Jim Highsniith, “Ale Project Management”, Pearson Education, 2004.

SOFTWARE ARCHITECTURE AND DESIGN PATTERNS

VI Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACIC06	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Object Oriented Software Engineering

I. COURSE OVERVIEW:

This course is to introduce software architecture and design patterns to create software architecture and to analyze it. This course covers software architecture, patterns, architectural patterns, and pattern system. The main objective is to introduce the student to architecture of software and design patterns. Upon completion of this course the student will Get an idea on envisioning architecture, creating architecture and analyzing architecture.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The challenges of advanced software design and the issues associated with large-scale software architectures, frameworks, patterns and components.
II. The tools and techniques that may be used for the automatic analysis and evaluation of software.
III. The need for software architecture and the principles of the classic architectural styles.
IV. Major approaches to automated software analysis achievable through static and dynamic analysis.

III. COURSE SYLLABUS:

MODULE-I: INTRODUCTION TO SOFTWARE ARCHITECTURE (08)

Software architecture: What software architecture is and what it is not, architectural structures and views, architectural patterns, importance of software architecture, inhibiting or enabling a system‘s quality attributes.

MODULE –II: PATTERNS (09)

Patterns: Introduction about pattern, what makes a pattern, pattern categories, relationship between patterns, pattern description, patterns and software architecture, summary.

MODULE –III: PATTERNS AND SOFTWARE ARCHITECTURE (09)

Patterns and software architecture: Introduction, patterns in software architecture, enabling techniques for software architecture, non-functional properties of software architecture.

Architectural Patterns: Introduction, layers, pipes and filters, black board; Distributed Systems: Broker-interactive systems: Model-view controller, presentation-abstraction control.

MODULE –IV: ARCHITECTURAL PATTERNS (10)

Architectural patterns: Adaptable systems, micro-kernel, reflection design patterns, structural decomposition, master-slave, access control, proxy.

MODULE –V: PATTERN SYSTEMS (09)

Pattern systems: Introduction to pattern system, pattern classification, pattern selection, pattern systems as implementation guidelines.

IV. TEXT BOOKS:

1. Len Bass, Paul Clement, Rick Kazman, “Software Architectures in Practice”, Pearson, 3rd Edition, 2013.
2. Frank Buschmann, RegineMeunier, Hans Rohnert, Peter Sommerlad, Michael Stal, “Pattern Oriented Software Architecture: A System of Patterns”, John Wiley and Sons, Volume 1, Reprinted, 2001.

V. REFERENCE BOOKS:

1. Alan Shalloway, James R Trott, “Design Patterns Explained, A New Perspective on Object Oriented Design”, Addison-Wesley, 2nd Edition, 2005.
2. Mary Shaw and David Garlan, “Software Architecture: Perspectives on an Emerging Discipline”, PHI Learning, 2nd Edition, 1996.

3. James W. Cooper, “Java Design Patterns- a Tutorial”, Addison-Wesley Professional, 2nd Edition, 2000.
4. Eric Freeman, Elisabeth Freeman, “Head First Design Patterns”, O‘Reilly Publications, 1st Edition, 2004.

VI. WEB REFERENCES:

1. http://www.ece.ubc.ca/~matei/EECE417/BASS/ch02lev1sec4.html
2. https://msdn.microsoft.com/en-in/library/ee658117.aspx.
3. http://www.openloop.com/softwareEngineering/patterns/designPattern/
4. http://xyuan.myweb.cs.uwindsor.ca/311/Lec11.pdf

SOFTWARE RELIABILITY

VI Semester: CSE(CS)

Course Code	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACIC07	Elective	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				TotalClasses:45		

Prerequisite: Object Oriented Software Engineering

I. COURSE OVERVIEW:

The course will introduce the students to techniques and tools for improving the reliability of software systems. It provides an overview of the main types of software reliability techniques and discuss their respective strengths and weaknesses. It will present in detail a selection of these techniques and associated tools, focusing on recent directions in both research and practice. Students can also know how to develop reliable software system and also the fault handling and failure forecasting techniques in software systems. Students can also understand different time dependent and time independent software reliability models and also learn the reliability models are needed for software systems.

II. COURSEOBJECTIVES:

The students will try to learn:

I. The engineering techniques for developing and maintaining reliable software systems.
II. How to measure the reliability of software systems.
III. About fault prevention, fault removal, fault tolerance and failure forecasting in software systems.
IV. The different time dependent and time independent software reliability models and design reliability models for software systems.

III. COURSE SYLLABUS:

MODULE–I:INTRODUCTION AND OPERATIONAL PROFILE (09)

The Need for Reliable Software, Software Reliability Engineering Concepts, Basic definitions, Software practitioners biggest problem, software reliability engineering approach, software reliability engineering process, defining the product, Reliability concepts, software reliability and hardware reliability, developing operational profiles, applying operational profiles, learning operations and run concepts.

MODULE–II:SOFTWARE RELIABILITY CONCEPTS (09)

Defining failure for the product, common measure for all associated systems, setting system failure intensity objectives, determining develop software failure intensity objectives, software reliability strategies, failures, faults and errors, availability, system and component reliabilities and failure intensities, predicting basic failure intensity.

MODULE– III: SOFTWARE RELIABILITY MODELING SURVEY (09)

Components of Software Estimations, Estimation methods, Problems associated with estimation, Key project factors that influence estimation.

Size Estimation: Two views of sizing, Function Point Analysis, Mark II FPA, Full Function Points, LOC Estimation, Conversion between size measures

MODULE– IV: SOFTWARE METRICS FOR RELIABILITY ASSESSMENT (09)

Introduction, Static Program Complexity, Dynamic Program Complexity, Software Complexity and Software Quality, Software Reliability Modeling.

MODULE– V: SOFTWARE TESTING AND RELIABILITY (09)

Introduction, Overview of Software Testing, Operational profiles, Time/Structure Based Software Reliability Estimation, Benefits and approaches of SRE, SRE during requirements phase, SRE during

implementation phase, SRE during Maintenance phase.

IV. TEXTBOOKS:

1. Michael R. Lyu, "Handbook of Software Reliability Engineering", published by IEEE Computer Society Press and McGraw-Hill Book Company.
2. John D. Musa, "Software Reliability Engineering", Tata McGraw-Hill, 2nd Edition, 2004.

V. REFERENCE BOOKS:

1. Patric D. T. O connor, "Practical Reliability Engineering", John Wesley & Sons, 4th Edition 2003.
2. Anderson and PA Lee, "Fault Tolerance Principles and Practice", PHI, 1981.
3. Pradhan D K, "Fault Tolerant Computing-Theory and Techniques", Vol 1 and Vol 2, Prentice hall, 1986.
4. E. Balagurusamy, "Reliability Engineering", Tata McGraw Hill, 1994.

SOFT SKILLS AND INTERPERSONAL COMMUNICATION

OE – I: VI Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT
OE –II: VII Semester: ECE / EEE
OE – III: VIII Semester: AERO / MECH / CIVIL

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC15	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

The objectives of the Soft Skills and Interpersonal Communication are to give each student a realistic perspective of work and work expectations, to help formulate problem solving skills, to guide students in making appropriate and responsible decisions, to create a desire to fulfill individual goals, and to educate students about unproductive thinking, self-defeating emotional impulses, and self- defeating behaviors.

II. COURSE OBJECTIVES:

The students will try to learn:

I. How to communicate in a comprehensible English accent and pronunciation.
II. The four language skills i.e., Listening, Speaking, Reading and Writing effectively.
III. The art of interpersonal communication skills to avail the global opportunities.
IV. The understanding of soft skills resulting in an overall grooming of the skills.

III. SYLLABUS

MODULE-I: SOFT SKILLS (09)

Soft Skills: An Introduction – Definition and Significance of Soft Skills; Process, Importance and Application of Soft Skills, Discovering the Self; Setting Goals; Positivity and Motivation: Developing Positive Thinking and Attitude.

MODULE –II: EFFECTIVENESS OF SOFT SKILLS (09)

Developing interpersonal relationships through effective soft skills; Define Listening, Speaking, Reading and Writng skills; Barriers to Listening, Speaking, Reading and Writing; Essential formal writing skills; Public Speaking: Skills, Methods, Strategies and Essential tips for effective public speaking.

MODULE-III: ORAL AND AURAL SKILLS (09)

Vocabulary:
Sounds of English vowels sounds and constant sounds, Word Accent and connected speech- contractions, questions tags, Listening for information, Taking notes while listening to lectures (use of Dictionary).

Group Discussion: Importance, Planning, Elements, Skills, Effectively disagreeing, Initiating.

MODULE-IV: VERBAL AND NON-VERBAL COMMUNICATION (09)

Interpersonal communication-verbal and nonverbal etiquette; Body language, grapevine, Postures, Gestures, Facial expressions, Proximity; Conversation skills, Critical thinking, Teamwork, Group Discussion, Impact of Stress; Measurement and Management of Stress.

MODULE-V: INTERPERSONAL COMMUNICATION (09)

Significance; Effectiveness of writing; Organizing principles of Paragraphs in documents; Writing introduction and conclusion; Techniques for writing precisely; Letter writing; Formal and Informal letter writing; E-mail writing, Report Writing.

IV. TEXT BOOKS:

Handbook of English for Communication (Prepared by Faculty of English, IARE)

V. REFERENCE BOOKS:

1. Dorch, Patricia. What Are Soft Skills? New York: Execu Dress Publisher, 2013.

2. Kamin, Maxine. Soft Skills Revolution: A Guide for Connecting with Compassion for Trainers, Teams, and Leaders. Washington, DC: Pfeiffer & Company, 2013.
3. Klaus, Peggy, Jane Rohman & Molly Hamaker. "The Hard Truth about Soft Skills", London: HarperCollins E-books, 2007.
4. Stein, Steven J. & Howard E. Book. "The EQ Edge: Emotional Intelligence and Your Success" Canada: Wiley & Sons, 2006
5. Suresh Kumar. English for Success. Cambridge University Press IndiaPvt.Ltd.2010.
6. Dorling Kindersley. Communication Skills & Soft Skills - An Integrated Approach. India Pvt. Ltd. 2013.

VI. **WEB REFERENCES:**

1. www.edufind.com
2. www.myenglishpages.com
3. http://grammar.ccc.comment.edu
4. http://owl.english.prudue.edu

VII. E-Text Books:

1. http://bookboon.com/en/communication-ebooks-zip
2. http://www.bloomsbury-international.com/images/ezone/ebook/writing-skills-pdf.pdf
3. https://americanenglish.state.gov/files/ae/resource_files/developing_writing.pdf
4. http://learningenglishvocabularygrammar.com/files/idiomsandphraseswithmeaningsandexamplespdf.pdf
5. http://www.robinwood.com/Democracy/General Essays/CriticalThinking.pdf

CYBER LAW AND ETHICS

OE – I: VI Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT
OE –II: VII Semester: ECE / EEE
OE –III: VIII Semester: AERO / MECH / CIVIL

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC16	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COUSRE OVERVIEW

This course consists of a sustained study of ethical and legal issues that arise in relation to employment in the public and private sectors, including allocation of resources, corporate and social responsibility, relationships, and discrimination. The main focus of this course will be on the ethical and legal standards governing information technology. New technology creates ethical challenges for individuals around the globe, and applies to most persons regardless of whether they are employed in the information technology field or a more traditional occupation. The study of this course provides a framework for making ethical decisions that professionals are likely to encounter in the workplace. This course will not only focus on ethics but on the legal, economic, social, cultural and global impacts of decisions that are made in the context of professional occupations.

II. COUSRE OBJECTIVES:

The students will try to learn:

I. The key terms and concepts in cyber society, cyber ethics.
II. The fundamentals of Cyber Law
III. The importance of nine P's in ethics.
IV. The artificial intelligence and Blockchain ethics.

III. SYLLABUS

MODULE-I: CYBER SOCIETY (09)

Definitions, Specificities of the Cyberspace, Dimensions of Cyber Ethics in Cyber Society, Fourth Industrial Revolution, Users' Motivations in Cyber-Space, Core Values and Virtues, Old Values or Eschatological Vision?, Cyber Ethics by Norms, Laws and Relations Artificial Intelligence Ethics: "AI for Good", Cyber-Capitalism: Cyber-Ethics as Business Ethics.

MODULE-II: CYBER LAW AND CYBER ETHICS (09)

Cyber Law and Cyber Ethics

The importance of cyber law, the significance of cyber ethics, cyber crime is unethical and illegal, ethics education has positive impact, the need for cyber regulation based on cyber ethics, very dangerous times.

MODULE-III: ETHICS IN THE INFORMATION SOCIETY, THE NINE P'S (09)

Principles: ethical values, participation: access to knowledge for all, people: community, identity, gender, generation, education, profession: ethics of information professions, privacy: dignity, data mining, security.

Piracy: intellectual property, cybercrime, protection: children and young people, power: Economic power of technology, media and consumers, policy: ethics of regulation and freedom.

MODULE-IV: DISRUPTIVE CYBER TECHNOLOGIES AND AI ETHICS (09)

Disruptive Cyber Technologies and Ethics -I

Artificial: negative moral judgment?, artificial: ethically positive innovation?, intelligence: action-oriented ability, creation story: human beings responsibility, the commandment to love and artificial intelligence;

Artificial Intelligence Ethics: Top nine ethical issues in artificial intelligence, five core principles to keep AI ethical, ethics should inform AI, but which ethics?

MODULE-V: DISRUPTIVE CYBER TECHNOLOGIES AND ETHICS –II (09)
Disruptive Cyber Technologies and Ethics -II
BLOCKCHAIN ETHICS:
Blockchain definition and description, Blockchain anonymity and privacy: ethical, no possibility to be forgotten, Blockchain for voting, Blockchain for transparent trade tracing, Blockchain energy: environmental impact, decentralized or majority-owned, ethically more benefits or dangers, future jobs in cyber society.

IV. TEXT BOOKS:

1. Christoph Stuckelberger, Pavan Duggal, "Cyber Ethics 4.0 Serving Humanity with Values", Globethics.net Global Series, 2018.

V. REFERENCE BOOKS:

1. Dr. Farooq Ahmad, Cyber Law in India, Allahbad Law Agency- Faridabad.
2. J.P. Sharma, SunainaKanojia, Cyber Laws
3. Harish Chander , Cyber Laws and IT Protection.

VI. E-REFERENCE:

https://www.globethics.net/documents/4289936/13403236/Ge_Global_17_web_isbn9782889312641.pdf/

ECONOMIC POLICIES IN INDIA

OE – I: VI Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT
OE –II: VII Semester: ECE / EEE
OE –III: VIII Semester: AERO / MECH / CIVIL

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC17	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW

The objective of this course is to provide a broad sweep of the concept, structure and trends in the Indian economy in a roughly chronological manner. It begins with a review of the evolution of the Indian economy during colonial rule and introduces the roots of Indian underdevelopment. This course is designed to acquaint the students in a comprehensive manner with different aspects of Indian economy. The policy issues and measure to understand economic initiatives for improving economic development and growth, agriculture and industry, planning of the different sectors of the economy and the place of Indian economy in the international level particularly after economic reforms and covered.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The economic development elements and its measures
II. The inside knowledge on monetary policy and its importance in economic development
III. The importance of fiscal policies in promoting the economy
IV. The policies and practices in resource base infrastructure
V. The industrial and exit policies related to the industries

III. SYLLABUS

MODULE-I: INTRODUCTION ECONOMIC DEVELOPMENT AND ITS DETERMINANTS (09)

Approaches to economic development and its measurement – sustainable development; Role of State, market and other institutions; Indicators of development – PQLI, Human Development Index (HDI), gender development indices.

MODULE-II: MONEY, BANKING AND PRICES (09)

Analysis of price behavior in India; Financial sector reforms; Interest rate policy; Review of monetary policy of RBI; Money and capital markets; Working of SEBI in India.

MODULE-III: FISCAL POLICY AND PUBLIC FINANCES (09)

Fiscal federalism – Centre-State financial relations; Finances of central government; Finances of state governments; Parallel economy; Problems relating to fiscal policy; Fiscal sector reforms in India.

MODULE-IV: RESOURCE BASE AND INFRASTRUCTURE (09)

Energy; social infrastructure – education and health; Environment; Regional imbalance; Issues and policies in financing infrastructure development. Policies and Performance in Industry Growth; productivity; diversification; small scale industries; public sector; competition policy; foreign investment.

MODULE-V: THE INDUSTRIAL AND EXIT POLICIES (09)

Industrial policy; Public Sector enterprises and their performance; Problem of sick units in India; Privatization and disinvestment debate; Growth and pattern of industrialization; Small-scale sector; Productivity in industrial sector; Exit policy – issues in labour market reforms; approaches for employment generation.

IV. TEXT BOOKS:

1. The Wealth of Nations-Adam Smith, introduction by Alan B Krueger.
2. The Strength of Economic Development by Albert Hirschman.
3. Money, Banking and Public Finance by Dr. V.C.Sinha
4. Government of India, Economic Survey (Annual), Ministry of Finance, New Delhi.
5. Jain, a. K. (1986), Economic Planning in India, Ashish Publishing House, New Delhi.

V. REFERENCE BOOKS:

1. Ahluwalia, I. J. and I. M. D Little (Eds.) (1999), India's Economic Reforms and Development (Essays in honour of Manmohan Singh), Oxford University Press, New Delhi.
2. Bardhan, P. K. (9th Edition) (1999), The Political Economy of Development in India, Oxford University Press, New Delhi.
3. Bawa, R. s. and P. S. Raikhy (Ed.) (1997), Structural Changes in Indian Economy, Guru Nanak Dev University Press, Amritsar.
4. Brahmananda, P. R. and V. R. Panchmukhi (Eds.) (2001), Development Experience in the Indian Economy: Inter-State Perspectives, Book well, Delhi.
5. Chakravarty, S. (1987), Development Planning: The Indian Experience, Oxford University Press, New Delhi.
6. Dantwala, M. L. (1996), Dilemmas of Growth: The Indian Experience, Sage Publications, New Delhi.
7. Datt, R. (Ed.) (2001), Second Generation Economic Reforms in India, Deep & Deep Publications, New Delhi.

VI. WEB REFERENCE:

1. Parikh, K. S. (1999), India Development Report – 1999-2000, Oxford University Press, New Delhi8.
2. Reserve Bank of India, Report on Currency and Finance, (Annual).
3. Sandesara, J. c. (1992), Industrial Policy and Planning, 1947-19919 : Tendencies, Interpretations and Issues, Sage Publications, New Delhi.

GLOBAL WARMING AND CLIMATE CHANGE

OE – I: VI Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT
OE –II: VII Semester: ECE / EEE
OE –III: VIII Semester: AERO / MECH / CIVIL

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC18	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW

This course aims to address the whole complexity of climate change as an issue, by bringing together the science, impacts, economics, abatement technologies, and policy solutions. The course will address several important questions like what is the scientific basis for our understanding of climate change, and in what ways is that scientific basis uncertain. What changes in climate might we expect over the coming centuries? What would be the impacts of these changes in climate for human well-being and the natural world? What are the sources of emissions of greenhouse gases? What technologies exist or might be developed to allow us to slow climate change, and what international policy solutions might be necessary or preferred?

II. COURSE OBJECTIVES:

The students will try to learn:

I. The importance of Ozone layer in the atmosphere.
II. The comprehend composition of atmosphere.
III. The impacts of climate change on ecosystem.
IV. The initiatives taken by different countries to reduce emission of greenhouse gases.

III. SYLLABUS:

MODULE – I: EARTH'S CLIMATE SYSTEM (09)

Role of ozone in environment, Ozone layer – Ozone depleting gases, Green House Effect – Radioactive effects of Greenhouse gases, The Hydrological cycle, Green House Gases and Global Warming, Carbon Cycle.

MODULE –II: ATMOSPHERE AND ITS COMPONENTS (09)

Importance of Atmosphere – Physical and chemical characteristics of Atmosphere, Vertical structure of the atmosphere, Composition of the atmosphere, Atmospheric stability, Temperature profile of the atmosphere, Lapse rates, Temperature inversion, Effects of inversion on pollution dispersion.

MODULE – III: IMPACTS OF CLIMATE CHANGE (09)

Causes of Climate change: Changes of Temperature in the environment, Melting of ice pole, sea level rise, Impacts of Climate Change on various sectors – Agriculture, Forestry and Ecosystem, Water Resources, Human Health, Industry, Settlement and Society.

Methods and Scenarios, Projected Impacts for different regions, Uncertainties in the projected impacts of Climate Change, Risk of Irreversible Changes.

MODULE – IV: OBSERVED CHANGES AND ITS CAUSES (09)

Climate change and Carbon credits, CDM – Initiatives in India-Kyoto Protocol, Paris Convention - Intergovernmental Panel on Climate change, Climate Sensitivity and Feedbacks. The Montreal Protocol – UNFCCC – IPCC – Global Climate Models (GCM) - Evidences of Changes in Climate and Environment- on a Global scale and in India.

MODULE – V: CLIMATE CHANGE AND MITIGATION MEASURES (09)

Clean Development Mechanism, Carbon Trading – Examples of future clean technology, Biodiesel – Natural Compost, Eco-friendly plastic, Alternate Energy –Hydrogen, Bio-fules, Solar Energy, Wind and Hydroelectric Power. Mitigation Efforts in India and Adaptation funding. Key Mitigation Technologies and Practices –

Energy Supply, Transport, Buildings, Industry, Agriculture, Forestry – Carbon sequestration, Carbon capture and storage (CCS), Waste (MSW & Bio-waste, Biomedical, Industrial waste) – International and Regional cooperation.

IV. TEXT BOOKS:

1. Dr. Sushil Kumar Dash, "Climate Change: An Indian Perspective (Environment and Development)", Cambridge University Press India Pvt Ltd, 2007.
2. Adaptation and mitigation of climate change – Scientific Technical Analysis, Cambridge University Press, Cambridge, 2006.

V. REFERENCE BOOKS:

1. Atmospheric Science, J.M. Wallace and P.V Hobbs, Elsevier/ Academic Press, 2006.
2. "Climate Change and Climate Variability on Hydrological Regimes", Jan C. Van Dam, Cambridge University Press, 2003.

VI. E-TEXT BOOKS

1. https://www.worldcat.org/title/encyclopedia-of-global-warming-climate-change/oclc/805580328
2. https://libguides.nus.edu.sg/c.php?g=433566&p=2955835

INTELLECTUAL PROPERTY RIGHTS

OE – I: VI Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT
OE –II: VII Semester: ECE / EEE
OE –III: VIII Semester: AERO / MECH / CIVIL

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC19	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COUSRE OVERVIEW:

The course will cover the philosophy of intellectual property rights, various technical and legal dimensions of IPR, and implications of IPR for growth and development of science, along with the various socio-economic and ethico-legal consequences of IPR on economic development. Students can also get disseminate knowledge on Design, Geographical Indication (GI), Plant Variety and Layout Design Protection and their registration aspects and also aware about current trends in IPR and Govt. steps in fostering IPR.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The knowledge in world trade organization and agreements between nations.
II. The intellectual property with international trade agreements.
III. The different types of intellectual property rights.
IV. The different laws in protection of intellectual property rights and its implementation.

III. SYLLABUS:

MODULE- I: INTRODUCTION (10)

General agreement on tariffs and trade (GATT) eight rounds: Uruguay round, world trade organization: structure, technology transfer, dispute resolution mechanism, Doha declaration world trade organization agreements including trade related intellectual properties rights and trade related investment measures.

MODULE- II: WORLD INTELLECTUAL PROPERTY ORGANIZATION (08)

Paris convention, Bern convention, Budapest treaty, Madrid agreement, huge agreement.

MODULE- III: PATENTS (09)

Historical background of intellectual property rights, introduction, definition and classification of intellectual property, patents, patentable and non-patentable inventions. Legal requirements for patents, types of patent applications.

Patent document: specification and claims, important procedural aspects, management of intellectual property rights assets and intellectual property portfolio, commercial exploitation of intellectual property.

MODULE- IV: DESIGNS AND GEOGRAPHICAL INDICATIONS (10)

Designs: basic requirements, procedure, convention application term, date, geographical indication: definition, what can be registered, who can apply, rights, term, restrictions.

MODULE- V: TRADEMARK AND COPYRIGHTS (08)

Definition, classification of trademarks, classifications of goods and services, Vienna classification, trademarks procedure, trademarks enforcement: infringement and passing off, remedies, copyrights, term of copyrights, and procedure of copyright assignment of copyright, copyright infringement remedies.

IV. TEXT BOOKS:

1. P. K. Vasudeva,World Trade Organization: Implications on Indian Economy, Pearson Education,2015.
2. P.KrishnaRao, WTO, Text and cases, Excel Books, 2015.
3. Carlos M.Correa- Intellectual property rights, The WTO and Developing countries-Zed books.

V. REFERENCE BOOKS:

1. Caves, Frankel, Jones, World Trade and Payments-An Introduction, Pearson4. Education, 2015.

2. Carlos M.Correa- Intellectual property rights, The WTO and Developing countries-Zed books.
3. Peter-Tobias stoll, Jan busche, Katrianarend- WTO- Trade –related aspects of IPR- Library of Congress.

VI. WEB REFERENCES:

1. http://www.ebooks directory.com
2. http://Campus guides.lib.utah.edu

VII. E-Text Books:

1. http://www.bookboon.com
2. http://www.freemagagement.com
3. http://www.emeraldinsight.com

ENTREPRENEURSHIP

OE – I: VI Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT
OE –II: VII Semester: ECE / EEE
OE –III: VIII Semester: AERO / MECH / CIVIL

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSC20	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

This course aims to provide students with an understanding of the nature of enterprise and entrepreneurship and introduces the role of the entrepreneur, will inculcate the knowledge of government supporting programs. Apart from this, students learn about the women entrepreneurs and success stories of women entrepreneurs, gain the knowledge of project management and profitability appraisal, focus on importance of training the new entrepreneurs as well as existing entrepreneurs. The students can also acquire necessary knowledge and skills required for organizing and carrying out entrepreneurial activities, for analysing and understanding business situations in entrepreneurs act and to master the knowledge necessary to plan entrepreneurial activities. The objective of the course is, to develop the ability of analysing various aspects of entrepreneurship – especially of taking over the risk, and the specificities as well as the pattern of entrepreneurship development and, finally, to contribute to their entrepreneurial and managerial potentials.

II. COUSRE OBJECTIVES:

The students will try to learn:

I. The Entrepreneurial process and also inspire them to be Entrepreneurs.
II. The key steps in the elaboration of business idea.
III. The stages of the entrepreneurial process and the resources needed for the successful development of entrepreneurial ventures.

III. SYLLABUS:

MODULE-I: UNDERSTANDING ENTREPRENEURIAL MINDSET (09)

The revolution impact of entrepreneurship- The evolution of entrepreneurship - Functions of Entrepreneurs – types of entrepreneurs -Approaches to entrepreneurship- Process approach- Role of entrepreneurship in economic development- Twenty first century trends in entrepreneurship.

MODULE-II: INDIVIDUAL ENTREPRENEURIAL MIND-SET AND PERSONALITY (09)

The entrepreneurial journey Stress and the entrepreneur - the entrepreneurial ego - Entrepreneurial motivations- Motivational cycle – Entrepreneurial motivational behavior – Entrepreneurial competencies. Corporate Entrepreneurial Mindset, the nature of corporate entrepreneur- conceptualization of corporate entrepreneurship Strategy-sustaining corporate entrepreneurship.

MODULE-III: LAUNCHING ENTREPRENEURIAL VENTURES (09)

Opportunities identification- Finding gaps in the market place – techniques for generating ideas-entrepreneurial Imagination and Creativity- the nature of the creativity process - Innovation and entrepreneurship.

Methods to initiate Ventures- Creating new ventures-Acquiring an Established entrepreneurial venture-Franchising- advantage and disadvantages of Franchising.

MODULE-IV: LEGAL CHALLENGES OF ENTREPRENEURSHIP (09)

Intellectual property protection - Patents, Copyrights - Trademarks and Trade secrets - Avoiding trademark pitfalls. Feasibility Analysis - Industry and competitor analysis - Formulation of the entrepreneurial Plan- The challenges of new venture start-ups, developing an effective business model – Sources of finance - Critical factors for new venture development - The Evaluation process.

MODULE-V: STRATEGIC PERSPECTIVES IN ENTREPRENEURSHIP (09)

Strategic planning - Strategic actions strategic positioning- Business stabilization - Building the adaptive firms - Understanding the growth stage – Internal growth strategies and external growth strategies, Unique managerial concern of growing ventures. Initiatives by the Government of India to promote entrepreneurship, Social and women entrepreneurship.

IV. TEXT BOOKS:

1. D F Kuratko and T V Rao, "Entrepreneurship- A South-Asian Perspective", Cengage Learning, 2012.
2. Bruce R. Barringer/ R.Duane Ireland, "Entrepreneurship Successfully Launching New Ventures", Pearson, 4th Edition, 2015.
3. S.S.Khanka, Entrepreneurship Development, S. Chand Publications, 2015.

V. REFERENCE BOOKS:

1. Stuart Read, Effectual Entrepreneurship, Routledge, 2013.
2. Rajeev Roy, Entrepreneurship, Oxford publications, 2nd Edition, 2012.
3. Nandan .H, Fundamentals of Entrepreneurship, PHI, 2013.

EXPERIENTIAL ENGINEERING EDUCATION (ExEEd) – RESEARCH BASED LEARNING

VI Semester: Common for all branches

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC27	Foundation	2	-	-	1	30	70	100
Contact Classes: 36	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 36		

Prerequisite: There are no prerequisites to take this course

I. COURSE OVERVIEW:

Research-based learning (RBL) presents as an alternative learning model that can develop the critical thinking skills. The research-based learning is conducted under constructivism which covers four aspects: learning which constructs student's understanding, learning through developing prior knowledge, learning which involves social interaction process, and meaningful learning which is achieved through real-world experience. The major focus is to engage students in the inquiry process where they formulate questions, conduct investigations, apply information and media to learning, and generate products that illustrate learning. The 5E learning cycle adopted for RBL leads students through five phases: Engage, Explore, Explain, Elaborate, and Evaluate which results in greater benefits concerning student's ability for scientific inquiry.

II. COURSE OBJECTIVES:

The students will try to learn:

I. To provide an opportunity for the students to engage in solving the real-world problems.
II. To introduce the overall process of research from its inception to the report.
III. To create the environment for multi-disciplinary research.
IV. Comprehend the role of ethics in research

III. COURSE SYLLABUS

I. What is Research?
II. Identifying Problem Statement
III. Overview of research-literature
IV. Planning activities, clarifying methods/methodologies
V. Experimentation
VI. Hypothesis testing
VII. Undertaking investigation and analyzing the data
VIII. Interpretation and consideration of results
IX. Presentation of replication studies

PENETRATION TESTING AND CYBER OPERATIONS LABORATORY

VI Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC10	Core	1	0	2	2	30	70	100
Contact Classes: 12	Tutorial Classes:	Practical Classes: 36				Total Classes: 48		

Prerequisite: Foundations of Cyber security

I. COURSE OVERVIEW:

The process of assessing an application or infrastructure for vulnerabilities in an attempt to exploit those vulnerabilities, and circumvent or defeat security features of system components through rigorous manual testing.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The tools that can be used to perform information gathering.
II. The Various attacks in various domains of cyberspace.
III. About exploits in various operating systems and wireless environment.
IV. How vulnerability assessment can be carried out by means of automatic tools or manual investigation.
V. The vulnerabilities associated with various network applications and database system.

III. COURSE SYLLABUS:

Week – 1:
Use Google and Whois for Reconnaissance.

Week – 2:
1. Use CryptTool to encrypt and decrypt passwords using RC4 algorithm.
2. Use Cain and Abel for cracking Windows account password using Dictionary attack and to decode wireless network passwords

Week – 3:
Use TraceRoute, ping, ifconfig, netstat Command

Week – 4:
To perform ARP poisoning

Week – 5:
Use Nmap scanner to perform port scanning of various forms – ACK, SYN, FIN, NULL, XMAS..

Week – 6:
Use WireShark sniffer to capture network traffic and analyze.

Week – 7:
Simulate persistent Cross Site Scripting attack.

Week – 8:
Session impersonation using Firefox

Week – 9:
Session impersonation using Tamper Data add-on

Week – 10:
Perform SQL injection attack.

Week – 11:
Create a simple keylogger using Python

Week – 12:
Use Metasploit to exploit the data

V. REFERENCE BOOKS:

1. RafayBaloch, “Ethical Hacking and Penetration Testing Guide”, CRC Press, 2015.
2. Dr.Patrick Engebretson, “The Basics of Hacking and Penetration Testing”, Syngress Publications Elseveir, 2013.
3. Prakhar Prasad, “Mastering Modern Web Penetration Testing”, Packt Publishing, 2016.
4. Gilberto Najera Gutierrez, “Kali LinuxWeb Penetration Testing”, Cookbook, 2016.
5. Robert Svensson, “From Hacking to Report Writing: An Introduction to Security and Penetration Testing”, 2016.

DATA MINING AND KNOWLEDGE DISCOVERY LABORATORY

VI SEMESTER: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACIC08	Core	1	0	2	2	30	70	100
Contact Classes: 12	Tutorial Classes: Nil	Practical Classes: 36				Total Classes:48		

Prerequisite: Python Programming

I. COURSE OVERVIEW:

The course covers the exploratory analysis of data and algorithms for data mining applications. Topics covered including manipulation of data, working with different data objects, applications of data mining on real time data sets using association rule generation with frequent item sets, classification, clustering algorithms with the help of the python programming.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The Data Object Exploration and visualization
II. The pre-processing on new and existing datasets.
III. Frequent item set generation and association rules on transactional data.
IV. The data model creation by using various classification and clustering algorithms.
V. The data models accuracy analysis by varying the sample size.

III. COURSE SYLLABUS:

Week – 1: DATA OBJECT, MATRIX OPERATIONS USING NUMPY

Introduction to Python libraries for Data Mining: NumPy, SciPy, Pandas, Matplotlib, Scikit, Learn
Write a Python program to do the following operations:
Library: NumPy

a. Create multidimensional arrays and find its shape and dimension
b. Create a matrix full of zeros and ones
c. Reshape and flatten data in the array
d. Append data vertically and horizontally
e. Apply indexing and slicing on array
f. Use statistical functions on array, Min, Max, Mean, Median and Standard Deviation

Week – 2: DATA OBJECT, MATRIX OPERATIONS USING NUMPY

Write a Python program to do the following operations:
Library: NumPy

a. Dot and matrix product of two arrays
b. Compute the Eigen values of a matrix
c. Solve a linear matrix equation such as 3 * x0 + x1 = 9, x0 + 2 * x1 = 8
d. Compute the multiplicative inverse of a matrix
e. Compute the rank of a matrix
f. Compute the determinant of an array

Week – 3: EXPLORATION ANDVISULIZATION OF DATA

Write a Python program to do the following operations:
Data set: brain_size.csv
Library: Pandas

a. Loading data from CSV file
b. Compute the basic statistics of given data, shape, no. of columns, mean

c. Splitting a data frame on values of categorical variables
d. Visualize each attribute

Week – 4: EXPLORATION OF DATA, CORRILATION

Write a python program to load the dataset and understand the input data
Dataset: Pima Indians Diabetes Dataset
Library: Scipy

a. Load data, describe the given data and identify missing, outlier data items.
b. Find correlation among all attributes.
c. Visualize correlation matrix.

Week – 5: DATA PREPROCESSING – HANDLING MISSING VALUES

Write a python program to impute missing values with various techniques on given dataset.

a. Remove rows/ attributes
b. Replace with mean or mode
c. Write a python program to Perform transformation of data using Discretization (Binning) and normalization (MinMaxScaler or MaxAbsScaler) on given dataset.

Week – 6: ASSOCIATION RULE MINING, APRIORI

Write a python program to find rules that describe associations by using Apriori algorithm between different products given as 7500 transactions at a French retail store.
Libraries: NumPy, SciPy, Matplotlib, Pandas
Dataset: https://drive.google.com/file/d/1y5DYn0dGoSbC22xowBq2d4po6h1JxcTQ/view?usp=sharing

a. Display top 5 rows of data
b. Find the rules with min_confidence: 0.2, min_support= 0.0045, min_lift=3, min_length=2

Week – 7: CLASSIFICATION – DECISION TREE

Classification of Bank Marketing Data
The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The dataset provides the bank customers' information. It includes 41,188 records and 21 fields. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit (variable y).
Libraries: Pandas, NumPy, Sklearn, Seaborn
Write a python program to

a. Explore data and visualize each attribute
b. Predict the test set results and find the accuracy of the model
c. Visualize the confusion matrix
d. Compute precision, recall, F-measure and support.

Week – 8: CLASSIFICATION – DECISION TREE

Dataset: The data set consists of 50 samples from each of three species of Iris: Iris setosa, Iris virginica and Iris versicolor. Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.
Libraries: import numpy as np
Write a python program to

a. Calculate Euclidean Distance.
b. Get Nearest Neighbors
c. Make Predictions

Week – 9: CLASSIFICATION – DECISION TREE

Write a python program to

a. build a decision tree classifier to determine the kind of flower by using given dimensions.
b. train with various split measures (Gini index, Entropy and Information Gain)

c. Compare the accuracy

Week – 10:**CLASSIFICATION – BAYESIAN NETWORK**

Predicting Loan Defaulters:

A bank is concerned about the potential for loans not to be repaid. If previous loan default data can be used to predict which potential customers are liable to have problems repaying loans, these "bad risk" customers can either be declined a loan or offered alternative products.

Dataset: The stream named bayes_bankloan.str, which references the data file named bankloan.sav. These files are available from the Demos directory of any IBM® SPSS® Modeler installation and can be accessed from the IBM SPSS Modeler program group on the Windows Start menu. The bayes_bankloan.str file is in the streams directory.

a. Build Bayesian network model using existing loan default data
b. Visualize Tree Augmented Naïve Bayes model

Predict potential future defaulters and looks at three different Bayesian network model types (TAN, Markov, Markov-FS) to establish the better predicting model.

Week – 11: CLASSIFICATION – SUPPORT VECTOR MACHINES (SVM)

A multi, class classification problem on the cancer dataset. This dataset is computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

The dataset comprises 30 features (mean radius, mean texture, mean perimeter, mean area, mean smoothness, mean compactness, mean concavity, mean concave points, mean symmetry, mean fractal dimension, radius error, texture error, perimeter error, area error, smoothness error, compactness error, concavity error, concave points error, symmetry error, fractal dimension error, worst radius, worst texture, worst perimeter, worst area, worst smoothness, worst compactness, worst concavity, worst concave points, worst symmetry, and worst fractal dimension) and a target (type of cancer).

Use:scikit, learn

Build a model to classify the cancer data as malignant (harmful) and benign (not harmful) using SVM. This model can use the features of cancer patients data and to give an early indication of whether their samples might be benign or malignant.

Hint: Refer UCI Machine Learning Repository for data set or load from scikit learn library

Week – 12: CLUSTERING – K- MEANS

Predicting the titanic survive groups:

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships. One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

Libraries: Pandas, NumPy, Sklearn, Seaborn, Matplotlib

Write a python program

a. to perform preprocessing.
b. to perform clustering using k-means algorithm to cluster the records into two i.e., the ones who survived and the ones who did not.

IV. REFERENCE BOOKS:

Robert Layton, “Learning Data Mining with Python”,Packt Publishing, 2015.

V. WEB REFERENCES:

1. https://www.dataquest.io/blog/sci-kit-learn-tutorial/
2. https://www.ibm.com/support/knowledgecenter/en/SS3RA7_sub/modeler_tutorial_ddita/modeler_tutorial_ddita,gentopic1.html

3. https://archive.ics.uci.edu/ml/datasets.php/
4. https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
5. https://pythonprogramming.net/k-mean-titanic-dataset-machine-learning-tutorial/
6. https://newoutlook.it/download/python/learning-data-mining-with-python.pdf

GOPROGRMMING

VI SEMESTER: Common for CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC28	Skill	-	-	-	-	-	-	-
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: Nil		

Prerequisite: Programming for Problem Solving using C or Python Programming

I. COURSEOVERVIEW:

The Go Programming. The document lays down the framework that demarcates fundamentals of Go Programming, Installation and Runtime Environment, Strings, Control Structures, Functions, Arrays and Slices, Maps, Packages Structs, Interfaces and methods of Go Programming

II. COURSEOBJECTIVES:

The students will try to learn:

I. The basics of Go Programming.
II. The syntax and semantics of control structures and Functions.
III. How to implement arrays and Maps.
IV. How to implement packages, structs and discuss about various and methods and the importance of interfaces.

III.COURSE SYLLABUS:

MODULE-I: INTRODUCTION TO GO PRGRAMMING (10)

Languages that influenced Go, programming in Go, installation and runtime environment, editors, IDE's and othertools;Basicconstructsandelementarydatatypes:Filenames,keywords,identifiers,constants,variables,elementarytypesandoperator,strings,timesanddates,pointers

MODULE-II:CONTROL STRUCTURES AND FUNCTIONS (09)

Control structures: The if else construct, the switch keyword, for construct, break / continue, use of labels with break and continue and goto. Functions: Parameters and return value, passing a variable number of parameters, deferand tracing, built-infunctions, recursive functions; Applying closures: A function returning another function.

MODULE-III: ARRAYS AND MAPS(09)

Arrays and Slices: Declaration and initialization, multidimensional arrays, slices, passing a slice to a function, difference between new() and make(), multi dimensional slice.

Maps: Declaration, initialization and make, deleting an element, a slice of maps, sorting a map, inverting a map.

MODULE-IV: PACKAGES ANDSTRUCTS(10)

Packages: Overview of the standard library, the regexp package, locking and the sync package, custom packages and visibility, using godoc for your custom packages, using go install for installing custom packages custom packages: map structure, go install and go test, using git for distribution and installation; Structs: Definition of astruct,creatingastructvariablewithafactorymethod,custompackageusingstructs,structs with tags.

MODULE-V: METHODS AND INTERFACES(10)

Methods: Definition of method, difference between a function and a method, pointer or value as receiver, methods and not-exported fields, methods on embedded types and inheritance, how to embed functionality in a type, multiple inheritance, universal methods and method naming; Interfaces: Definition of interface, interface embedding interface(s), how to detect and convert the type of an interface variable, type assertion, the type switch, testing if a value implements an interface, using method sets with interfaces.

IV. TEXTBOOKS:

1. Alan A. A. Donovan and Brian W.Kernighan, "The Go Programming Language", Addison-Wesley Professional; 1st Edition, 2015.
2. Jay McGavren, "Head First Go: A Brain-Friendly Guide For Go Programming", O'Reilly, 2019.

DIGITAL FORENSICS

VII Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC11	Core	3	1	0	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisite: There are no prerequisites to take this course

I. COURSE OVERVIEW:

This course provides learners with a baseline understanding of common cyber security threats, Vulnerabilities and risks. An overview of how basic cyber attacks are constructed and applied to real systems is also included. Examples include simple Unix kernel hacks, Internet worms, and Trojan horses in software utilities. Network attacks such as distributed denial of service (DDOS) and botnet-attacks are also described and illustrated using real time examples from the past couple of decades.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The various types of cyber-attacks and cyber-crimes.
II. The threats and risks within context of the cyber security.
III. The overview of the cyber laws and concepts of cyber forensics.
IV. The defensive techniques against these attacks.

III. COURSE SYLLABUS

MODULE-I: INTRODUCTION TO DIGITAL FORENSICS (09)

Introduction: Computer Forensics Fundamentals – Benefits of forensics, computer crimes, computer forensics evidence and courts, legal concerns and private issues. Types of Computer Forensics Technology – Types of Computer Forensics Systems – Vendor and Computer Forensics Services.

MODULE-II: DATA ACQUISITION AND INCIDENT RESPONSE (09)

Computer forensics evidence and capture: Data Recovery – Evidence Collection and Data Seizur E-Duplication and Preservation of Digital Evidence-Computer Image Verification and Authentication, conducting and investigations.

MODULE-III: MEMORY FORENSICS (09)

Computer forensic analysis: Discover of Electronic Evidence-Identification of Data – Reconstructing Past Events – Fighting against Macro Threats.

Information Warfare Arsenal – Tactics of the Military – Tactics of Terrorist and Rogues – Tactics of Private Companies.

MODULE-IV: INFORMATION WARFARE (09)

Information warfare: Arsenal – Surveillance Tools – Hackers and Theft of Components – Contemporary Computer Crime-Identity Theft and Identity Fraud – Organized Crime &Terrorism – Avenues Prosecution and Government Efforts – Applying the First Amendment to Computer Related Crime-The Fourth Amendment and other Legal Issues.

MODULE-V: DIGITAL FORENSIC CASES (09)

Computer forensic cases: Developing Forensic Capabilities – Searching and Seizing Computer Related Evidence – Processing Evidence and Report Preparation – Future Issues.

IV. TEXT BOOKS:

1. John R. Vacca, “Computer Forensics: Computer Crime Scene Investigation”, Cengage Learning, 2nd Edition, 2005. (CHAPTERS 1 – 18).
2. Marjie T Britz, “Computer Forensics and Cyber Crime: An Introduction”, Pearson Education, 2nd Edition, 2008. (CHAPTERS 3 – 13).
3. Warren G. Kruse II and Jay G. Heiser, “Computer Forensics: Incident Response Essentials”, Addison Wesley, 2002.

V. REFERENCE BOOKS:

1. MariE-Helen Maras, "Computer Forensics: Cybercriminals, Laws, and Evidence", Jones & Bartlett Learning; 2nd Edition, 2014.
2. Chad Steel, "Windows Forensics", Wiley, 1st Edition, 2006.
3. Majid Yar, "Cybercrime and Society", SAGE Publications Ltd, Hardcover, 2nd Edition, 2013.
4. Robert M Slade, "Software Forensics: Collecting Evidence from the Scene of a Digital Crime", Tata McGraw Hill, Paperback, 1st Edition, 2004.

CLOUD SECURITY

VII Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC12	Core	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

Prerequisite: Computer Networks

I. COURSE OVERVIEW:

This course provides learners with a baseline understanding of common cyber security threats, Vulnerabilities and risks. An overview of how basic cyber attacks are constructed and applied to real systems is also included. Examples include simple Unix kernel hacks, Internet worms, and Trojan horses in software utilities. Network attacks such as distributed denial of service(DDOS) and botnet-attacks are also described and illustrated using real time examples from the past couple of decades.

II. COURSE OBJECTIVES:

The students will try to learn:

1. The various types of cyber-attacks and cyber-crimes.
2. The threats and risks within context of the cyber security.
3. The overview of the cyber laws and concepts of cyber forensics.
4. The defensive techniques against these attacks

III. COURSE SYLLABUS

MODULE-I INTRODUCTION TO CLOUD SECURITY

Characteristics of cloud Security, Types of Cloud Computing Services, Separation of Responsibilities in cloud, Cloud Deployment models.

MODULE-II CLOUD COMPUTING THREATS

Cloud Provider Acquisition, Management Interface Compromise, Network Management Failure, Authentication Attacks, Data breach/Loss, Insufficient Due Diligence, Malicious Insiders.

MODULE-III CLOUD COMPUTING ATTACKS

Container vulnerabilities, Kubernetes Vulnerabilities, Service Hijackng, Wrapping Attack, Man-in-the Cloud Attack, Cloud Hopper Attack, Cloud brone attack.

MODULE-IV CLOUD COMPUTING SECURITY

Cloud Security Control layers, Cloud Security Considerations, Kubernetes vulnerabilities and Security, Server less Security Risks and Solutions.

MODULE-V CLOUD SECURITY MANAGEMENT

Address various data, application, platform and infrastructure needs. Managing disparate cloud services can be challenging, Organizations need a sound strategy that protects corporate assets.

IV. TEXT BOOKS:

1. Nina Godbole and Sunit Belpure," Cyber Security Understanding Cyber Crimes, Computer Forensics and Legal Perspectives", Wiley Pubications, 2014.
2. B.B.Gupta, D.P. Agrawal, Haoxiang Wang, "Computer and Cyber Security:Principles, Algorithm, Applications, and Perspectives", CRC Press, ISBN 9780815371335,2018.

V. REFERENCE BOOKS:

1. James Graham, Richard Howard and Ryan Otson ,"Cyber Security Essentials", CRC Press..
2. Chwan-Hwa(john) Wu,J. David Irwin , "Introduction to Cyber Security" , CRC Press T&F Group.

PRINCIPLES OF ARTIFICIAL INTELLIGENCE

VII Semester: CSE(CS)

Course Code	Category	Hours /Week			Credits	MaximumMarks		
		L	T	P	C	CIA	SEE	Total
AITC26	Elective	3	0	0	3	30	70	100
ContactClasses:45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite: Data Structures, LAC

I. COURSEOVERVIEW:

Artificial Intelligence has emerged as an increasingly impactful discipline in science and technology. AI applications are embedded in the infrastructure of many products and industries search engines, medical diagnoses, speech recognition, robot control, web search, advertising and even toys.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Gain a historical perspective of AI and its foundations.
II. The basic principles of AI toward problem solving, inference, knowledge representation, and learning.
III. Investigate applications of AI techniques in intelligent agents, expert systems, artificial neural networks and other machine learning models.
IV. The AI development tools such as Prolog (AI language), expert system shell, and/or data miningtool.
V. The current scope, potential, limitations, and implications of intelligent systems.

III. COURSE SYLLABUS:

MODULE-I INTRODUCTION OF AI AND KNOWLEDGE REPRESENTATION (08)

Definition of AI, The AI Problems, The Underlying Assumption, AI Techniques, The Level of the Model, Criteria for Success, The importance of AI, Early works in AI, AI and Related fields, The Foundations of Artificial Intelligence, The History of Artificial Intelligence. Defining the Problem as a State Space Search, Production Systems, Problem Characteristics, Production System Characteristics, Issues in the Design of Search Programs.

Knowledge Representation Issues: Representations and Mappings, Approaches to Knowledge Representation, Issues in Knowledge Representation.

AI Languages and Tools: Lisp, Prolog, CLIPS.

MODULE-II FIRST ORDER LOGIC AND INFERENCE(10)

Using Predicate Logic: Representing Simple Facts in Logic, Representing Instance and ISA Relationships, Computable Functions and Predicates, Properties of Wff, Clausal Forms, Conversion to clausal forms, Resolution.

Representing Knowledge Using Rules: Procedural Versus Declarative Knowledge, Logic Programming, Forward Versus Backward Reasoning, Matching, Control Knowledge.

MODULE-III SEARCH TECHNIQUES (08)

Heuristic Search Techniques: Generate-and-Test, Hill Climbing, Best-first Search, A* algorithm, AO*algorithm, Problem Reduction, And-Or search, Constraint Satisfaction, Means-ends Analysis.

Adversarial Search and Game Playing: Optimal Decision in Games, The minimax algorithm, Alpha-Beta pruning, Iterative Deepening, Expect Imax search.

MODULE-IV HANDLING UNCERTANITY (10)

Symbolic Reasoning Under Uncertainty: Introduction to Nonmonotonic Reasoning, Logics for Nonmonotonic Reasoning, Implementation Issues, Augmenting Problem-solver.

Statistical Reasoning: Probability and Bayes 'Theorem, Certainty Factors and Rule-based Systems, Bayesian Networks, Dempster-Shafer Theory, Fuzzy Logic.

MODULE-V PLANNING, LEARNING AND EXPERT SYSTEMS (09)

Planning: Overview, An Example Domain: The Blocks World, Components of a Planning System, Goal Stack Planning, Nonlinear Planning Using Constraint Posting, Hierarchical Planning, Reactive Systems.

Learning: What is learning, Rote learning, learning by taking Advice, learning from example: Induction, Explanation based learning (EBL), Discovery, Clustering, Analogy, Neural net and genetic learning, Reinforcement learning.

Expert System: Representing and Using Domain Knowledge, Expert System Shells, Explanation, Knowledge Acquisition, Expert System Architectures, Rule based systems, Non production system, knowledge acquisition.

IV. TEXT BOOKS:

1. Elaine Rich, Kevin Knight, & Shivashankar B Nair, "Artificial Intelligence", McGraw Hill, 3rdEdition,2019.
2. DanW. Patterson, "Introduction to AI and Expert Systems", Prentice Hall, 2007.

V. REFERENCE BOOKS:

1. Nils J.Nilsson, "Principles of Artificial Intelligence", Narosa Publishing House, 1990.
2. Stuart Russell and Peter Norvig, "Artificial Intelligence A Modern Approach", Pearson Education, 2nd Edition, 2010.
3. VS Janakiraman K, Sarukesi Gopalakrishnan, "Foundations of Artificial Intelligence & Expert Systems", Macmillan.

VI. WEB REFERENCES:

1. http://www.youtube.com/playlist?list=PLD52D2B739E4D1C5F
2. NPTEL: ArtificialIntelligence,https://nptel.ac.in/courses/106105077/
3. http://www.udacity.com/
4. http://www.library.thinkquest.org/2705/
5. http://www.ai.eecs.umich.edu/

VII. E-TEXT BOOKS:

1. http://www.stpk.cs.rtu.lv/sites/all/.../Artificial%20Intelligence%20A%20Modern%20Approach.pdf
2. http://www.bookboon.com/en/artificial-intelligence-ebooks
3. http://www.onlineprogrammingbooks.com/ai-and-robotics
4. http://www.e-booksdirectory.com

MACHINE LEARNING

VII Semester: CSE(CS)

Course Code	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC27	Elective	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite: Linear Algebra and Calculus

I. COURSE OVERVIEW:

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. ... Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning is a field of study that looks at using computational algorithms to turn empirical data into usable models. The machine learning field grew out of traditional statistics and artificial intelligences communities.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The concepts of machine learning and related algorithms.
II. The dimensionality problems using linear discriminates.
III. The various statistical models for analyzing the data.
IV. The clustering algorithms for unlabeled data.

III. COURSE SYLLABUS:

MODULE–I: TYPES OF MACHINE LEARNING (09)

Concept learning: Introduction, version spaces and the candidate elimination algorithm; Learning with trees: Constructing decision trees, CART, classification example.

MODULE–II: ARTIFICIAL NEURAL NETWORKS (09)

Introduction, neural network representations, Appropriate problems for neural network learning, perceptions

Multi-layer networks and the Back propagation algorithms, An illustrative example: Face recognition, advanced topics in Artificial neural networks.

MODULE– III: BAYESIAN LEARNING (09)

Averages, variance and covariance, the Gaussian; The bias-variance tradeoff, Bayesian learning: Introduction, Bayes theorem, Bayes optimal classifier, naïve Bayes classifier, Gibs algorithm, Bayesian belief networks.

The EM algorithm Computational learning theory: Introduction, probably learning an approximately correctly hypothesis sample complexity for finite hypothesis spaces, sample complexity for infinite hypothesis spaces, the mistake bound model of learning

MODULE- IV: INSTANCE BASED LEARNING (09)

Introduction, k-nearest neighbor algorithm, locally weighted regression, Radial basis functions case-based learning, remarks on lazy and eager learning. Genetic Algorithms, genetic operators; Genetic programming.

MODULE- V: INDUCTIVE ANDREINFORCEMENTLEARNING (09)

Motivation, Inductive analytical approaches to learning, using prior knowledge to alter the search objective. Introduction, The learning Task.Q Learning, On deterministic Rewards and Actions, Temporal Difference programming, Generating from examples, relationship to dynamic programming

IV. TEXT BOOKS:

1. Tom M. Mitchell, "Machine Learning ", McGraw Hill, 1st Edition, 2013.
2. Stephen Marsland, “Machine Learning- An Algorithmic Perspective “, CRC Press, 1st Edition, 2009.

V. REFERENCE BOOKS:

1. Margaret H Dunham, "Data Mining", Pearson Edition, 2nd Edition, 2006.
2. Galit Shmueli, Nitin Rel, Peter C Bruce, “Data Mining for Business Intelligence”, John Wiley and Sons, 2nd Edition,2007.
3. Rajjal Shinghal, “Pattern Recognition and Machine Learning”, Springer-Verlag, New York, 1st Edition, 2006.

VI. WEB REFERENCES:

1. https://www.oracle.com/in/cloud/application-development
2. http://computingcareers.acm.org/?page_id=12
3. http://en.wikibooks.org/wiki/cloudapplication

VII. E-TEXT BOOKS:

1. http://www.acadmix.com/eBooks_Download

DATA HANDLING AND VISUALIZATION

VII Semester: CSE(CS)

Course Code	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC28	Elective	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite:

I. COURSEOVERVIEW:

This course is all about data visualization, the art and science of turning data into readable graphics. We'll explore how to design and create data visualizations based on data available and tasks to be achieved. This process includes data modeling, data processing (such as aggregation and filtering), mapping data attributes to graphical attributes, and strategic visual encoding based on known properties of visual perception as well as the task(s) at hand. Students will also learn to evaluate the effectiveness of visualization designs, and think critically about each design decision, such as choice of color and choice of visual encoding. Students will create their own data visualizations, and learn to use Open Source data visualization tools, especially D3.js.

II.COURSEOBJECTIVES:

The students will try to learn:

I. How to design and create data visualizations.
II. The exploratory data analysis using visualization.
III. The visual presentations of data for effective communication.
IV. The perception and cognition to evaluate visualization design alternatives.
V. How to design and evaluate color palettes for visualization based on principles of perception.

III. SYLLABUS:

MODULE- I : DATA VISUAL REPRESENTATION (08)

Introduction of visual perception, visual representation of data, Gestalt principles, information overloads.

MODULE – II: VISUALIZATION (10)

Creating visual representations, visualization reference model, visual mapping, visual analytics, Design of visualization applications.

MODULE – III: DATA DIMENSIONAL(10)

Classification of visualization systems, Interaction and visualization techniques misleading, Visualization of one, two and multi-dimensional data, text and text documents.

Input for Visualization: Data and Tasks, Loading and Parsing Data with D3.js.

MODULE – IV: DATA VISUALIZATION TYPES (08)

Visualization of groups, trees, graphs, clusters, networks, software, Metaphorical visualization, Histograms Aggregating Data with Group-By, Hexbin Mapping, Cross filtering.

MODULE – V: DATA MAPS (10)

Visualization of volumetric data, vector fields, processes and simulations, Visualization of maps, geographic information, GIS systems, collaborative visualizations, evaluating is utilizations.

IV. TEXT BOOKS:

1. Ward, Grinstein Keim, "Interactive Data Visualization: Foundations, Techniques, and Applications", Natick: A K Peters, Ltd, 1st Edition, 2014.

V. REFERENCE BOOKS:

1. E. Tufte, "The Visual Display of Quantitative Information", Graphics Press.

ADVANCED SOCIAL, TEXT AND MEDIA ANALYTICS

VII Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACAC09	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Data Mining and Knowledge Discovery

I. COURSE OVERVIEW:

This course enables the students to understand advanced social media, text and social media analytics,and their potential impact. Determine how to leverage social media for better services and understand usability metrics, mining and social media metrics.

II. COURSE OBJECTIVES:

The students will try to learn:

I. An overview of common text mining and social media data analytic activities.
II. The complexities of processing text and network data from different data sources.
III. How to solve complex real-world problems for sentiment analysis and Recommendation systems.
IV. The learners to develop skills required for analyzing the effectiveness of social media.

III. COURSE SYLLABUS:

MODULE-I: INTRODUCTION TO SOCIAL MEDIA ANALYTICS (SMA) (09)

Overview of social media, Social media landscape, Need for SMA, SMA in Small organizations, SMA in large organizations, Need of using analytics, current analytics platforms, Essentials of Social graphs, Social Networks, Models, Information Diffusion in Social Media.

MODULE –II: KEY PERFORMANCE INDICATOR/ METRICS (09)

Understand the discipline of social analytics, Aligning social objectives with business goals, Identify common social business objectives, developing KPIs; Standard vs Critical metrics. PULSE metrics (Page views, Uptime, Latency, Seven-day active users) on business and technical Issues, HEART metrics (Happiness, Engagement, Adoption, Retention, and Task success) on user behavior issues; Bounce rate, exit rate, conversion rate, engagement, Measuring Macro & micro conversions, On-site web analytics, off-site web analytics, the goal-signal-metric process.

MODULE –III: MINING TEXTUAL DATA(09)

Text Representation- tokenization, stemming, stop words, TF-IDF, Feature Vector Representation, NER,N-gram modeling, Text Clustering, Text Classification, Topic Modeling-LDA,HDP.

Text Extraction: Introduction, Rapid automatic keyword extraction: candidate keywords, keyword scores, adjoining keywords, extracted keywords, Benchmark evaluation: precision and recall, efficiency, stoplist generation, Evaluation on new articles.

MODULE –IV: MINING TWITTER AND MINING FACEBOOK (09)

Why is twitter all the rage? exploring twitter's api, fundamental twitter terminology, creating a twitter api connection, exploring trending topics, searching for tweets, analyzing the 140 character, extracting tweet entities, analyzing tweets and tweet entities with frequency analysis, computing the lexical diversity of tweets, examining patterns in retweets, visualizing frequency data with histograms. analyzing fan pages, examining friendships, and more overview, exploring facebook's social graph api, understanding the social graph api, understanding the open graph protocol, analyzing social graph connections, analyzing facebook pages, examining friendships.

MODULE –V: DATA MINING IN SOCIAL MEDIA AND SOCIAL NETWORKS (09)

Introduction, data mining in a nutshell, social media, motivations for data mining in social media, data mining methods for social media, data representation, data mining - a process, social networking sites: illustrative examples, the blogosphere: illustrative examples, related efforts, ethnography and netnography, event maps. Introduction, keyword search, query semantics and answer ranking, keyword search over xml and relational data, keyword search over graph data, classification algorithms, clustering algorithms, transfer learning in heterogeneous networks.

IV. TEXT BOOKS

1. Matthew A. Russell, "Mining of Social Web", O'Reilly; 2nd Edition (8 October 2013), ISBN-13: 978-1449367619.
2. Charu C Agarwal, "Social Network Data Analytics", Springer; 2011 Edition (1 October 2014), 978-1489988935
3. Matthew A. Russell, "Mining the Social Web", O'Reilly Media, 2nd Edition, 2013.

V. REFERENCE BOOKS:

1. Hand, Mannila, and Smyth, "Principles of Data Mining", Cambridge, MA: MIT Press, 2001. ISBN: 026208290X.
2. Tom Tullis, Bill Albert, "Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics", Morgan Kaufmann; 1 Edition (28 April 2008).
3. Jim Sterne, "Social Media Metrics: How to Measure and Optimize Your Marketing Investment", John Wiley &Sons (16 April 2010).
4. Brian Clifton, "Advanced Web Metrics with Google Analytics", John Wiley & Sons; 3rd Edition (30 Mar 2012).

WEB REFERENCES:

1. https://www.webpages.uidaho.edu/~stevel/504/Mining-the-Social-Web-2nd-Edition.pdf
2. http://pa.cm1911.com/Files/Subject/Advances%20in%20Social%20Network%20Mining%20and%20Analysis.pdf

VII. E-TEXT BOOKS:

1. https://ocw.mit.edu/courses/sloan-school-of-management/15-599-workshop-in-it-collaborative-innovation networks-fall-2011/lecture-notes/MIT15_599F11_lec04.pdf
2. https://www.cambridgeassessment.org.uk/Images/465808-big-data-and-social-media-analytics.pdf

PRINCIPLES OF COMPUTER SECURITY

VII Semester: CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC13	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite: There is no prerequisite to take this course

I. COURSE OVERVIEW:

The purpose of the cyber security principles is to provide strategic guidance on how organizations can protect their systems and data from cyber threats. These cyber security principles are grouped into four key activities: govern, protect, detect and respond. Govern: Identifying and managing security risks. Protect: Implementing security controls to reduce security risks. Detect: Detecting and understanding cyber security events. Respond: Responding to and recovering from cyber security incidents.

II. COURSE OBJECTIVES:

The students will try to learn:

1. The broad set of technical, social and political aspects of computer security.
2. The operational and organizational security aspects.
3. The fundamentals of cryptography and purpose of intrusion detection system.

III. COURSE SYLLABUS:

MODULE – I:INTRODUCTION TO SECURITY TRENDS

The Computer Security Problem - Targets and Attacks - Approaches to Computer Security - Ethics - Basic Security Terminology - Security Models

MODULE – II: OPERATIONAL AND ORGANIZATIONAL SECURITY

Policies, Procedures, Standards, and Guidelines - Security Awareness and Training - Interoperability Agreements - The Security Perimeter - Physical Security - Environmental Issues - Wireless - Electromagnetic Eavesdropping - People—A Security Problem - People as a Security Tool.

MODULE – III:CRYPTOGRAPHY

Cryptography in Practice - Historical Perspectives - Algorithms - Hashing Functions - Symmetric Encryption.

Asymmetric Encryption - Quantum Cryptography- Cryptography Algorithm Use.

MODULE – IV: AUTHENTICATION AND REMOTE ACCESS

User, Group, and Role Management - Password Policies - Single Sign-On - Security Controls and Permissions - Preventing Data Loss or Theft - The Remote Access Process - Remote Access Methods.

MODULE – V:INTRUSION DETECTION SYSTEMS

History of Intrusion Detection Systems - IDS Overview - Network-Based IDSs - Host-Based IDSs- Intrusion Prevention Systems - Honeypots and Honeynets – Tools.

IV. TEXT BOOKS:

1. W.A.Coklin, G.White, “Principles of Computer Security”, McGraw Hill, 4th Edition,2016
2.William Stallings, “Cryptography and Network Security Principles and Practices”, 7thEdition,Pearson

V. REFERENCE BOOKS:

1. Achyut S. Godbole,“Web Technologies: TCP/IP, Web/Java Programming, and Cloud Computing”, Tata McGraw Hill Education, 2013.

VI. WEB REFERENCES:

1. https://www.newhorizons.com/promotions/cybersecurity-ebooks
2. https://www.coursera.org/learn/introduction-cybersecurity-cyber-attacks#syllabus.

CYBER SECURITY TECHNIQUES AND TOOLS

VII Semester: **CSE (CS)**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC14	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisites: Foundations of Cyber Security

I. COURSE OVERVIEW:

Learn the history of cyber security, types and motives of cyber attacks to further your knowledge of current threats to organizations and individuals. Key terminology, basic system concepts and tools will be examined as an introduction to the cyber security field.

II. COURSE OBJECTIVES:

The students will try to learn:

1. The cyber issues in real world by various means.
2. The installation of VMware and inspect kali linux.
3. The metasploit framework for hacking and assess the security in mobile devices.

III. COURSE SYLLABUS:

MODULE – I: CYBER ISSUES

Window Password Hacking and Cracking – Steganography - Hiding Secret Message - Anonymous Call, Message and Email Header Analysis - Access Darknet or Darkweb Using TOR : Anonymous Browsing - Access Darknet or Darkweb Using TOR : Anonymous Browsing.

MODULE – II: VIRTUAL LAB SET UP

Installing VMware -Setting Up Kali Linux - Target Virtual Machines - Creating the Windows XP Target - Setting Up the Ubuntu 8.10 Target - Creating the Windows 7 Target.

MODULE – III: KALI LINUX

Linux Command Line - The Linux File system - User Privileges - File Permissions - Editing Files- Data Manipulation - Managing Installed Packages.

Processes and Services - Managing Networking - Netcat: The Swiss Army Knife of TCP/IP Connections - Automating Tasks with cron Jobs.

MODULE – IV: METASPLOIT FRAMEWORK

Starting Metasploit - Finding Metasploit Modules - Setting Module Options - Payloads - Types of Shells - Setting a Payload Manually - Msfcli - Creating Standalone Payloads with Msfvenom - Using an Auxiliary Module

MODULE – V: MOBILE HACKING

Mobile Attack Vectors - The Smartphone Pentest Framework - Remote Attacks - Client-Side Attacks - Malicious Apps - Mobile Post Exploitation

IV. TEXT BOOKS:

1. Gautam Kumawat, “Ethical Hacking & Cyber Security Course : A Complete Package”, Udemy Course, 2017.
2. Georgia Weidman ,” Penetration testing A Hands-On Introduction to Hacking”, no starch press, 2014.

V. REFERENCE BOOKS:

1. Charles P. Pfleeger Shari Lawrence Pfleeger Jonathan Margulies, “Security in Computing”, 5th Edition , Pearson Education , 2015.

ETHICAL HACKING AND SYSTEM DESIGN

VII Semester: CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC15	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisites: Foundations of Cyber Security

I. COURSE OVERVIEW:

This course combines an ethical hacking methodology to better help students secure their systems. Students are introduced to common countermeasures that effectively reduce and/or mitigate attacks.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The various information security tools given different target systems in different environments;
II. The tools interrelate with each other in an overall penetration testing process;
III. The common ethical hacking methodology to carry out a penetration test;
IV. The penetration testing and ethical hacking fit into a comprehensive enterprise information security program; Implement countermeasures for various types of attacks.

III. COURSE SYLLABUS:

MODULE – I: INTRODUCTION TO ETHICAL HACKING

Introduction-Ethical hacking Terminology-types of hacking technologies-phases of ethical hacking-Foot printing-Social Engineering-Scanning and enumeration

MODULE – II: SYSTEM HACKING

Understanding the password hacking techniques-Rootkits-Trojans-Backdoors-Viruses and worms-sniffers-denial of service-Session hijacking.

MODULE – III: TCP/IP OVERVIEW CONCEPTS AND PORT SCANNING

Overview of TCP/IP-IP addressing-numbering systems- Introduction to port scanning-types of port scans.

Port scanning tools-ping sweeps- understanding scripting-enumeration.

MODULE – IV: DESKTOP AND SERVER OS VULNERABILITIES

Windows OS vulnerabilities-tools for identifying vulnerabilities in windows-Linux OS vulnerabilities-vulnerabilities of embedded OS.

MODULE – V: NETWORK PROTECTION SYSTEMS

Understanding routers-understanding firewalls-risk analysis tools for firewalls- understanding intrusion and detection and prevention systems-honeypots.

IV. TEXT BOOKS:

1. Michael T. Simpson, Kent Backman, James Corley, “Hands-On Ethical Hacking and Network Defense”, 2016.
2. Steven DeFino, Barry Kaufman, Nick Valenteen, “Official Certified Ethical Hacker Review Guide”, 2015.

V. REFERENCE BOOKS:

1. The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy (Syngress Basics Series).

VI. WEB REFERENCES:

1. https://www.nationalcyberwatch.org/resource/ethical-hacking-systems-defense-national-cyberwatch-center-edition/

CYBER PHYSICAL SYSTEMS

VII Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC16	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite: Computer Networks

I. COURSE OVERVIEW:

Cyber-physical systems, which consist of physical systems tightly integrated and/or controlled by software, are ubiquitous in many safety critical domains, including automotive, avionics, railways, healthcare, atomic energy, power, and industrial automation. The principles of design and implementation of cyber-physical systems are remarkably different from that of other embedded systems because of the tight integration of real valued and dense time real time systems with software based discrete automated control. The course aims to expose the student to real world problems in this domain and provide a walk through the design and validation problems for such systems. Applications for CPS research are far reaching and span medical devices, smart buildings, vehicle systems, and mobile computing. The application domain for this course will be cyber-physical vehicle systems though techniques are more broadly applicable. Current literature, techniques, theories, and methodologies will be reviewed and discussed.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The basic design, architecture and design principles of cyber physical systems.
II. The fundamental concepts of cryptography for ensuring security of cyber- physical systems
III. The sources of vulnerability in a cyber physical system systematically via attack surfaces.
IV. The various modeling formalisms for CPS, such as hybrid automata, state-space methods, etc.

III. COURSE SYLLABUS:

MODULE – I: INTRODUCTION TO CYBER PHYCIAL SYSTEM (09)

Motivation and examples of CPS e.g. Energy, Medical and Transportation cyber physical systems; Key design drivers and quality attributes of CPS. Attributes of high confidence CPS.

MODULE – II: CYBER PHYSICAL SYSTEM DESIGN (09)

Continuous systems modeling; Discrete time system modeling; Finite state machine; Extended state machines; Hybrid system modeling; Classes of Hybrid Systems.

MODULE – III: ANALYSIS AND VERIFICATION (09)

Basic concepts of embedded systems; Embedded Processors; Input-outputs; Invariants and Temporal Logic; Linear Temporal Logic;

Equivalence and Refinement; Development of models from specifications; Reachability analysis and Model Checking.

MODULE - IV:CYBER PHYSICAL SYSTEM MODELLING (9)

Modeling experiments in continuous, discrete and hybrid system; Verification of model using different techniques; Sensitivity analysis of Models; Sensitivity analysis of Hybrid Models; Scheduling in embedded system.

MODULE – V: SECURITY IN CYBER PHYSICAL SYSTEMS (09)

Security issues of Industrial Control Systems; Integrity attacks on SCADA systems; Model based technique to detect integrity attacks on sensors; threat model and its effect on control scheme;

countermeasure for detecting such attacks; watermarking scheme; Design of observers under sensor and actuator attacks; design of observer for distributed environment under different attacks; applications of swarms of UAVs; Control design with denial service attack; case studies

IV. TEXT BOOKS:

1. R. Rajkumar, D. de. Niz and M. Klein, "Cyber Physical Systems", Addision-Wesely, 2017.
2. E.A.Lee and S AShesia, (2018), "Embedded System Design: A Cyber-Physical Approach", Second Edition, MIT Press.
3. A.Platzer, "Logical Foundations of Cyber Physical Systems", Springer, 2017.

V. REFERENCE BOOKS:

1. F. Pasqualetti, F. Dörfler and F. Bullo, "Attack Detection and Identification in Cyber-Physical Systems", in IEEE Transactions on Automatic Control, vol. 58, no. 11, pp. 2715-2729, Nov. 2013.
2. H. Fawzi, P. Tabuada and S. Diggavi, "Secure Estimation and Control for Cyber-Physical Systems Under Adversarial Attacks", in IEEE Transactions on Automatic Control, vol. 59, no. 6, pp. 1454-1467, June 2014.
3. Yilin Mo, RohanChabukswar and Bruno Sinopoli, "Detecting Integrity Attacks on SCADA Systems" in IEEE Transactions on Control System Technology, Vol. 22, No. 4, 2014.
4. F. Pasqualetti, F. Dörfler and F. Bullo, "Control Theoretic methods for Cyber Physical Security", in IEEE Control System Magazine, pp. 110-127, Feb. 2015.

VI. WEB REFERENCES:

1. https://www.nist.gov/el/cyber-physical-systems
2. https://www.cs.cmu.edu/~aplatzer/course/fcps14/fcps14.pdf
3. https://www.eecs.umich.edu/courses/eecs571/lectures/lecture2-intro-of-CPS.pdf

INDUSTRIAL AUTOMATION AND CONTROL

OE –II: VII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AEEC29	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

This course provides an exposure to technology of industrial automation and control as widely seen in across a range of industries. It contains a wide range of topics from the advantages and architecture of automation systems, measurement systems including sensors and signal conditioning, discrete and continuous variable control systems, programmable logic controllers, SCADA & DCS systems, CNC machines and actuators.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The functionality of the basic elements of industrial automation systems and the fundamental principles of operation of numerous instruments and machines.
II. The various control techniques employed in process automation including programmable logic controllers.
III. The substantial applications of automation systems and analyze real-life problems from an automation perspective based on engineering and cost-oriented thinking.

III. SYLLABUS

MODULE-I: ELECTROCHEMISTRY AND CORROSION (09)

Automation: Introduction to industrial automation and control architecture of industrial automation system, Levels of Automation, Types of Automation, Automation in Production System,

Measurement Systems: Specifications, temperature measurement, pressure and force measurement, displacement and speed measurement, signal conditioning circuits, errors and calibration.

MODULE –II: PROGRAMMABLE LOGIC CONTROL SYSTEMS (09)

Programmable logic control systems: introduction to sequence or logic control and programmable logic controllers, the software environment and programming of PLCs, formal modeling of sequence control specifications. Programming, programming of PLCs: sequential function charts, the PLC hardware environment, Industrial Stamping Process case study.

MODULE-III: PROCESS CONTROL (09)

Process control: Introduction to process control, PID control, controller tuning, implementation of PID controllers, special control structures, feed forward and ratio control special control structures: predictive control, control of systems with inverse response.

MODULE –IV: CNC MACHINES AND ACTUATORS (09)

CNC machines and actuators: Introduction to computer numerically controlled machines, control valves, hydraulic actuation systems, principle and components, directional control valves, switches and gauges, industrial hydraulic circuits.

MODULE –V: LARGE SCALE CONTROL SYSTEM (09)

SCADA: Introduction, SCADA Architecture, Different Communication Protocols, Common System Components, Supervision and Control, HMI, RTU and Supervisory Stations, Trends in SCADA, Security Issues

DCS: Introduction, DCS Architecture, Local Control (LCU) architecture, LCU languages, LCU - Process interfacing issues, communication facilities, configuration of DCS, displays, redundancy concept - case studies in DCS.

IV. TEXT BOOKS:

1. Madhu Chanda Mitra, Samarjit Sen Gupta, "Programmable Logic Controllers and Industrial Automation: An Introduction", Penram International Publishing (India) Pvt. Ltd., 1st Edition, 2008.
2. K Krishnaswamy, S Vijayachitra, "Industrial Instrumentation", New Age Publications, 1st Edition, 2010.
3. Rajesh Mehra, Vikrant Vij, "PLCs & SCADA: Theory and Practice", Laxmi publications, 2nd Edition, 2016.
4. Stuart A. Boyer: SCADA-Supervisory Control and Data Acquisition, Instrument Society of America Publications, USA, 1999

V. REFERENCE BOOKS:

1. AK Gupta, S K Arora, "Industrial Automation and Robotics", Laxmi Publications, 2nd Edition, 2013.
2. Jon Stenerson, "Industrial Automation and Process Control", Prentice Hall, 1st Edition, 2002.
3. Gordon Clarke, Deon Reynders, Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems, Newnes Publications, Oxford, UK,2004.

VI. WEB REFERENCES:

1. https://nptel.ac.in/courses/108/105/108105088/
2. https://oleumtech.com/what-is-scada
3. https://www.plc-scada-dcs.blogspot.com/p/downloads.html
4. https://1lib.in/s/industrial%20automation%20and%20control

ARTIFICIAL NEURAL NETWORKS

OE –II: VII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AEEC30	**Elective**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	-	-	3	30	70	100
Contact Classes: 45	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 45**		

I. COURSE OVERVIEW:

This course introduces and relates the basic concept of neural networks. The course will provide a current and coherent view of artificial neural networks. The neural net algorithms will be discussed to understand contemporary neuro computing technology. It emphasizes mathematical analysis of neural networks, methods for training networks, and application of networks to practical problems. Neural network implementation will be discussed to understand contemporary neuro computing and soft-computing techniques. Students in computer science, engineering, and behavioural science can obtain immediate benefits to visualize new approaches and interdisciplinary perspectives.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The biological neural network and to model equivalent neuron models
II. The architecture, learning algorithm and issues of various feed forward and feedback neural networks.
III. Different neural networks of various architectures both feed forward and feedbackward.
IV. How to perform the training of neural networks using various learning rules.
V. The testing of neural networks and do the perform analysis of these networks for various pattern recognition applications.

III. SYLLABUS

MODULE-I: INTRODUCTION TO ANN (09)

A Neural Network, Human Brain, Models of a Neuron, Neural Networks viewed as Directed Graphs, Network Architectures, Knowledge Representation, Artificial Intelligence and Neural Networks; Learning Process: Error Correction Learning, Memory Based Learning, Hebbian Learning, Competitive, Boltzmann Learning, Credit Assignment Problem, Memory, Adaption, Statistical Nature of the Learning Process

MODULE –II: PERCEPTRON (09)

Single Layer Perceptron: Adaptive Filtering Problem, Unconstrained Organization Techniques, Linear Least Square Filters, Least Mean Square Algorithm, Learning Curves, Learning Rate Annealing Techniques, Perceptron: convergence theorem, Relation Between Perceptron and Bayes Classifier for a Gaussian Environment; Multilayer Perceptron: Back Propagation Algorithm XOR Problem, Heuristics, Output, Representation and Decision Rule, Computer Experiment, Feature Detection.

MODULE-III: BACK PROPAGATION (09)

Back Propagation: Back Propagation and Differentiation, Hessian Matrix, Generalization, Cross Validation, Network Pruning Techniques, Virtues, and Limitations.

Back Propagation Learning, Accelerated Convergence, Supervised Learning.

MODULE –IV: SELF-ORGANIZATION MAPS (09)

Two Basic Feature Mapping Models, Self-Organization Map, SOM Algorithm, Properties of Feature Map, Computer Simulations, Learning Vector Quantization, Adaptive Patter Classification.

MODULE –V: DYNAMICAL SYSTEMS (09)

Neuro Dynamics: Dynamical Systems, Stability of Equilibrium States, Attractors, Neuro Dynamical Models, Manipulation of Attractors as a Recurrent Network Paradigm Hopfield Models – Hopfield Models, Computer Experiment.

IV. TEXT BOOKS:

1. Simon Haykin, "Neural Networks a Comprehensive Foundations", Prentice Hall India, 2nd Edition, 2003.

V. REFERENCE BOOKS:

1. B. Yegnanarayana, "Artificial Neural Networks", Prentice Hall of India Private Limited, 2005
2. Li Min Fu,"Neural Networks in Computer Intelligence", Tata McGraw Hill, 3rd Edition, 2003
3. James A Freeman David M S Kapura, "Neural Networks", Pearson Education, 2004.
4. Jacek M. Zurada,"Introduction to Artificial Neural Systems", JAICO Publishing House,1st Edition, 2006.

VI. WEB REFERENCES:

1. https:// www.en.wikipedia.org/wiki/ neural networks
2. https://www.jaicobooks.com/j/PDF%20HED/J-878%20Artificial%20Neural%20Systems.pdf
3. https://www.abebooks.co.uk/book-search/title/an-introduction-to-fuzzy-control/system.pdf

RENEWABLE ENERGY SOURCES

OE –II: VII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AEEC31	**Elective**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	-	-	3	30	70	100
Contact Classes: 45	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 45**		

I. COURSE OVERVIEW:

The course presents the various sources of renewable energy including wind, solar, and biomass as potential sources of energy and investigates the contribution they can make to the energy profile of the nation. The technology used to harness these resources will be presented. Discussions of economic, environment, politics and social policy are integral components of the course.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The knowledge on role of power electronics for renewable energy.
II. The power conditioning schemes for grid connected systems.
III. The skills in designing wind, solar systems and their integration.

III. COURSE SYLLABUS

MODULE-I: INTRODUCTION (10)

Introduction: Causes of Energy Scarcity, Solution to Energy Scarcity, Factors Affecting Energy Resource Development, Energy Resources and Classification, Renewable Energy – Worldwide Renewable Energy Availability, Renewable Energy in India. Energy from Sun: Sun- earth Geometric Relationship, Layer of the Sun, Earth – Sun Angles and their Relationships, Solar Energy Reaching the Earth's Surface, Solar Thermal Energy Applications

MODULE –II: SOLAR SYSTEMS(10)

Solar Thermal Energy Collectors: Types of Solar Collectors, Configurations of Certain Practical Solar Thermal Collectors, Material Aspects of Solar Collectors, Concentrating Collectors, Parabolic Dish – Stirling Engine System, Working of Stirling or Brayton Heat Engine, Solar Collector Systems into Building Services, Solar Water Heating Systems, Passive Solar Water Heating Systems, Applications of Solar Water Heating Systems, Active Solar Space Cooling, Solar Air Heating, Solar Dryers, Crop Drying, Space Cooing, Solar Cookers, Solar pond. Solar Cells: Components of Solar Cell System, Elements of Silicon Solar Cell, Solar Cell materials, Practical Solar Cells, I – V Characteristics of Solar Cells, Efficiency of Solar Cells, Photovoltaic Panels, Applications of Solar Cell Systems

MODULE-III: HYDROGEN, WIND AND GEO-THERMAL SYSTEMS (09)

Hydrogen Energy: Benefits of Hydrogen Energy, Hydrogen Production Technologies, Hydrogen Energy Storage, Use of Hydrogen Energy, Advantages and Disadvantages of Hydrogen Energy, Problems Associated with Hydrogen Energy. Wind Energy: Windmills, Wind Turbines, Wind Resources, Wind Turbine Site Selection.

Geothermal Energy: Geothermal Systems, Classifications, Geothermal Resource Utilization, Resource Exploration, Geothermal Based Electric Power Generation, Associated Problems, environmental Effects.
Solid waste and Agricultural Refuse: Waste is Wealth, Key Issues, Waste Recovery Management Scheme, Advantages and Disadvantages of Waste Recycling, Sources and Types of Waste, Recycling of Plastics.

MODULE –IV: BIOMASS SYSTEMS (08)

Biomass Energy: Biomass Production, Energy Plantation, Biomass Gasification, Theory of Gasification, Gasifier and Their Classifications, Chemistry of Reaction Process in Gasification, Updraft, Downdraft and Cross-draft Gasifiers, Fluidized Bed Gasification, Use of Biomass Gasifier, Gasifier Biomass Feed Characteristics, Applications of Biomass Gasifier, Cooling and Cleaning of Gasifiers. Biogas Energy: Introduction, Biogas and its Composition, Anaerobic Digestion, Biogas Production, Benefits of Biogas, Factors Affecting the Selection of a Particular Model of a Biogas Plant, Biogas Plant Feeds and their Characteristics. Tidal Energy: Introduction, Tidal Energy Resource, Tidal Energy Availability, Tidal Power Generation in India, Leading Country in Tidal Power Plant Installation, Energy Availability in Tides, Tidal Power Basin, Turbines for Tidal Power, Advantages and Disadvantages of Tidal Power, Problems Faced in Exploiting Tidal Energy

MODULE –V: PV WATER PUMPING AND GRID INTERFACE (08)

Sea Wave Energy: Introduction, Motion in the sea Waves, Power Associated with Sea Waves, Wave Energy Availability, Devices for Harnessing Wave Energy, Advantages and Disadvantages of Wave Power. Ocean Thermal Energy: Introduction, Principles of Ocean Thermal Energy Conversion (OTEC), Ocean Thermal Energy Conversion plants, Basic Rankine Cycle and its Working, Closed Cycle, Open Cycle and Hybrid Cycle, Carnot Cycle, Application of OTEC in Addition to Produce Electricity, Advantages, Disadvantages and Benefits of OTEC

IV. TEXT BOOKS:

1. G.D Rai, "Non conventional Energy Sources", Khanna Publications, 3rd Edition, 2008.
2. John Twidell and Tony Weir / E &F.N.Spon, "Renewable Energy Resources", Special Indian Edition, 3rd Edition, 2007.
3. G.N.Tiwari and M.K.Ghosal, "Renewable Energy Resources Basic Principles and Applications", Narosa Edition, 2nd Edition, 2004.
4. S.P. Sukhatme, "Solar Energy - Principles of Thermal Collection and Storage," TMH, 4th Edition, 2017.

V. REFERENCE BOOKS:

1. Daniel, Hunt. V, "Wind Power – A Hand Book of WECS", Van Nostrend Co., Newyork, 1998.
2. K.Khendelwal& S.S. Mahdi, "Biogas Technology - A Practical Hand Book", McGraw-Hill
3. Mukund. R. Patel, "Wind and Solar Power Systems", CRC Press, 1999.

VI. WEB REFERENCES:

1. http://ecoursesonline.iasri.res.in/course/view.php?id=524

BASIC ELECTRONIC ENGINEERING

OE –II: VII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AECC38	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I.COURSE OVERVIEW:

This course provides circuit analysis to design high frequency amplifiers and wave shaping circuits using discrete components. It covers multistage amplifiers, power amplifiers, feedback concepts, sampling gates and multivibrators. Analog electronics are widely used in radio and audio equipment and in many applications where signals are derived from analog sensors and transducers.

II.COURSE OBJECTIVES:

The students will try to learn:

I. The Fundamental knowledge of the operational principles and characteristics of semiconductor devices and their applications.
II. The response of a linear wave shaping circuits of low pass and high pass filters
III. The generation of nonlinear oscillations by using regenerative feedback circuit for multivibrators.

III. COURSE SYLLABUS:

MODULE –I: TRANSISTOR BIASING AND STABILIZATION (09)

Bias Stability, Fixed Bias, Collector to Base bias, Self-Bias, Bias Compensation using Diodes and Transistors

MODULE –II JUNCTION FIELD EFFECT TRANSISTOR (09)

Construction, Principle of Operation, Pinch-Off Voltage, Volt- Ampere Characteristic, Comparison of BJT and FET, Biasing of FET, FET as Voltage Variable Resistor, MOSFET Construction and its Characteristics in Enhancement and Depletion modes.

MODULE –III: FEEDBACK AMPLIFIERS (09)

Concepts of feedback, classification of feedback amplifiers, general characteristics of negative feedback amplifiers

Effect of feedback on amplifier characteristics, voltage series, voltage shunt, current series and current shunt feedback configurations.

MODULE –IV: OSCILLATORS (09)

Condition for Oscillations, RC type Oscillators-RC phase shift and Wien-bridge Oscillators, LC type Oscillators, generalized analysis of LC oscillators, hartley and colpitts Oscillators, frequency and amplitude stability of Oscillators, crystal Oscillator

MODULE –V: LINEAR WAVE SHAPING AND SAMPLING GATES (09)

Linear wave shaping circuits: High pass RC and low pass RC circuits, response to step and square inputs with different time constants, high pass RC circuit as a differentiator, low pass RC circuit as an integrator. Sampling gates: basic operating principle of sampling gate, uni and bi directional sampling gates.

IV.TEXT BOOKS

1. Jacob Millman , “Electronic Devices and Circuits”, McGraw Hill Education, 2017.
2. Robert L. Boylestead, Louis Nashelsky, “Electronic Devices and Circuits Theory”, Pearson, 11th Edition, 2009
3. A Anand Kumar, “Pulse and Digital Circuits”, PHI learning, 2nd Edition, 2005.

V. REFERENCE BOOKS

1. David A. Bell, “Electronic Devices and Circuits”, Oxford, 5th Edition, 1986.
2. Robert L. Boylestead, Louis Nashelsky, “Electronic Devices and Circuits Theory”, Pearson Education, 11th Edition, 2009.
3. Millman J., Taub, “Pulse, Digital and Switching Waveforms”, Tata McGraw-Hill, 2nd Edition, 2007.

VI.WEB REFERENCES

1. http://www.web.eecs.utk.edu
2. https://everythingvtu.wordpress.com
3. http://nptel.ac.in/
4. http://www.iare.ac.in

VII. E-TEXT BOOKS

1. http://www.bookboon.com/
2. http://www.jntubook.com
3. http://www.smartzworld.com
4. http://www.archive.org

PRINCIPLES OF COMMUNICATIONS

OE –II: VII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AECC39	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I.COURSE OVERVIEW:

This course intended to provide the concepts of analog communications. It covers modulation schemes, single and double sideband schemes. The applications include in all personal communications and transmission of signal to long distances without attenuation.

II.COURSE OBJECTIVES:

The students will try to learn:

I. The performance of analog modulation schemes in time and frequency domains
II. The performance of analog communication systems
III. The characteristics of pulse amplitude modulation, pulse position modulation and pulse code modulation systems.

III. COURSE SYLLABUS:

MODULE –I: AMPLITUDE MODULATION (08)

Introduction, Amplitude Modulation: Time & Frequency – Domain description, Switching modulator, Envelop detector.

MODULE –II DOUBLE SIDE BAND-SUPPRESSED CARRIER MODULATION (09)

Time and Frequency – Domain description, Ring modulator, Coherent detection, Costas Receiver, Quadrature Carrier Multiplexing.

MODULE –III: SINGLE SIDE–BAND AND VESTIGIAL SIDEBAND METHODS OF MODULATION (09)

SSB Modulation, VSB Modulation, Frequency Translation, Frequency- Division Multiplexing,

Theme Example: VSB Transmission of Analog and Digital Television.

MODULE –IV: ANGLE MODULATION (09)

Basic definitions, Frequency Modulation: Narrow Band FM, Wide Band FM, Transmission bandwidth of FM Signals, Generation of FM Signals, Demodulation of FM Signals, FM Stereo Multiplexing, Phase–Locked Loop: Nonlinear model of PLL, Linear model of PLL, Nonlinear Effects in FM Systems. The Super heterodyne Receiver

MODULE –V: DIGITAL REPRESENTATION OF ANALOG SIGNALS (10)

Introduction, Why Digitize Analog Sources?, The Sampling process, Pulse Amplitude Modulation, Time Division Multiplexing, Pulse-Position Modulation, Generation of PPM Waves, Detection of PPM Waves, The Quantization Process, Quantization Noise, Pulse–Code Modulation: Sampling, Quantization, Encoding, Regeneration, Decoding, Filtering, Multiplexing

IV.TEXT BOOKS:

1. Simon Haykins & Moher, "Communication Systems", John Willey, India Pvt. Ltd, 5th Edition, 2010. ISBN 978 – 81 – 265 – 2151 – 7.

V. REFERENCE BOOKS:

1. B. P. Lathi, "Modern Digital and Analog Communication Systems", Oxford University Press., 4th Edition, 2010.
2. Simon Haykins, "An Introduction to Analog and Digital Communication", John Wiley India Pvt. Ltd., 2008, ISBN 978–81–265–3653–5.
3. H.Taub & D.L.Schilling, "Principles of Communication Systems", TMH, 2011.

4. Harold P.E, Stern Samy and A.Mahmond, “Communication Systems”, Pearson Edition, 2004.
5. R.P.Singh and S.Sapre, “Communication Systems: Analog and Digital”, TMH, 2nd Edition, 2004.

VI.WEB REFERENCES

1. http://www.web.eecs.utk.edu
2. https://everythingvtu.wordpress.com
3. http://nptel.ac.in/
4. http://www.iare.ac.in

VII. E-TEXT BOOKS

1. http://www.bookboon.com/
2. http://www.jntubook.com
3. http://www.smartzworld.com
4. http://www.archive.org

EMBEDDED SYSTEMS

OE –II: VII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AECC40	**Elective**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	-	-	3	30	70	100
Contact Classes: 45	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 45**		

I. COURSE OVERVIEW:

This course allows students to learn the fundamentals of embedded system hardware and firmware design. It focus on embedded system design process, embedded C, interfacing modules, software development tools for debugging and testing of embedded applications, ARM & SHARC processor architectures and memory organization. It provides hands-on experience on implementation of embedded application prototype design using embedded C.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The fundamental concepts of embedded computing, embedded C, RTOS and embedded software tools for implementing embedded systems.
II. The Embedded the software development tools for debugging and testing of embedded applications, architectures of ARM and SHARC processors.
III. The interface with external environments using sensors, actuators and communication in distributed embedded systems.

III. SYLLABUS:

MODULE –I: EMBEDDED COMPUTING (08)

\Definition of embedded system, embedded systems vs. general computing systems, history of embedded systems, complex systems and microprocessor, classification, major application areas, the embedded system design process, characteristics and quality attributes of embedded systems, formalisms for system design,design examples.

MODULE –II: INTRODUCTION TO EMBEDDED C AND APPLICATIONS (09)

C looping structures, register allocation, function calls, pointer aliasing, structure arrangement, bit fields,unaligned data and endianness, inline functions and inline assembly, portability issues; Embedded systems programming in C, binding and running embedded C program in Keil IDE, dissecting the program, building the hardware; Basic techniques for reading and writing from I/O port pins, switch bounce; Applications: Switch bounce, LED interfacing, interfacing with keyboards, displays, D/A and A/D conversions, multiple interrupts, serial data communication using embedded C interfacing.

MODULE –III: RTOS FUNDAMENTALS AND PROGRAMMING (09)

Operating system basics, types of operating systems, tasks and task states, process and threads, multiprocessing and multitasking, how to choose an RTOS ,task scheduling, semaphores and queues, hard real-time scheduling considerations, saving memory and power.

Task communication: Shared memory, message passing, remote procedure call and sockets; Task synchronization: Task communication synchronization issues, task synchronization techniques, device drivers.

MODULE –IV: EMBEDDED SOFTWARE DEVELOPMENT TOOLS (09)

Host and target machines, linker/locators for embedded software, getting embedded software into the target system; Debugging techniques: Testing on host machine, using laboratory tools, an example system.

MODULE –V: INTRODUCTION TO ADVANCED PROCESSORS (10)

Introduction to advanced architectures: ARM and SHARC, processor and memory organization and instruction level parallelism; Networked embedded systems: Bus protocols, I2C bus and CAN bus; Internet-Enabled systems, design example-Elevator controller.

IV. TEXT BOOKS

1. Shibu K.V, "Introduction to Embedded Systems", Tata McGraw Hill Education Private Limited,2nd Edition, 2009.
2. Raj Kamal, "Embedded Systems: Architecture, Programming and Design", Tata McGraw-Hill Education, 2nd Edition, 2011.
3. Andrew Sloss, Dominic Symes, Wright, "ARM System Developer's Guide Designing and Optimizing System Software", 1st Edition, 2004.

V. REFERENCE BOOKS

1. Wayne Wolf, "Computers as Components, Principles of Embedded Computing Systems Design", Elsevier, 2nd Edition, 2009.
2. Dr. K. V. K. K. Prasad, "Embedded / Real-Time Systems: Concepts, Design & Programming", Dreamtech publishers, 1st Edition, 2003.
3. Frank Vahid, Tony Givargis, "Embedded System Design", John Wiley & Sons, 3rd Edition, 2006.
4. Lyla B Das, "Embedded Systems", Pearson Education, 1st Edition, 2012.
5. David E. Simon, "An Embedded Software Primer", Addison-Wesley, 1st Edition,1999.
6. Michael J. Pont, "Embedded C", Pearson Education, 2nd Edition, 2008.

VI. WEB REFERENCES

1. https://www.smartzworld.com/notes/embedded-systems-es/
2. http://notes.specworld.in/embedded-systems-es/
3. http://education.uandistar.net/jntu-study-materials
4. http://www.nptelvideos.in/2012/11/embedded-systems.html

VII. E-TEXT BOOKS

1. https://www.scribd.com/doc/233633895/Intro-to-Embedded-Systems-by-Shibu-Kv
2. http://www.ee.eng.cmu.ac.th/~demo/think/_DXJSq9r3TvL.pdf
3. https://www.scribd.com/doc/55232437/Embedded-Systems-Raj-Kamal
4. https://docs.google.com/file/d/0B6Cytl4eS_ahUS1LTkVXb1hxa00/edit
5. http://www.ecpe.nu.ac.th/ponpisut/22323006-Embedded-c-Tutorial-8051.pdf

DIGITAL FORENSICS LABORATORY

VII Semester: **CSE (CS)**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC17	Core	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

Prerequisite: Foundations of Cyber security

I.COURSE OVERVIEW:

The process of assessing an application or infrastructure for vulnerabilities in an attempt to exploit those vulnerabilities, and circumvent or defeat security features of system components through rigorous manual testing.

II.COURSE OBJECTIVES:

The students will try to learn:

I. The tools that can be used to perform information gathering.
II. The attacks in various domains of cyberspace.
III. The exploits in various operating systems and Wireless environment.
IV. The vulnerability assessment can be carried out by means of automatic tools or manual investigation.
V. The vulnerabilities associated with various network applications and database system.

III.COURSE SYLLABUS:

Week – 1:

Install and configure information security devices

Week – 2:

Security assessment of information security systems using automated tools.

Week – 3:

1. Vulnerability Identification and Prioritization
2. Working with Exploits

Week – 4:

1. Password Cracking
2. Web Application Security Configuration

Week – 5:

Patch Management

Week – 6:

Bypassing Antivirus Software

Week – 7:

Static Malware Analysis

Week – 8:

Dynamic Malware Analysis

Week – 9:

Penetration Testing

Week – 10:

Using Metasploit to exploit

Week – 11:

1. MySQL SQL Injection

2. Risk Assessment

Week - 12

1. Information security incident Management
2. Exhibit Security Analyst Role

IV. REFERENCE BOOKS:

1. MariE-Helen Maras, "Computer Forensics: Cybercriminals, Laws, and Evidence", Jones & Bartlett Learning; 2nd Edition, 2014.
2. Chad Steel, "Windows Forensics", Wiley, 1st Edition, 2006.
3. Majid Yar, "Cybercrime and Society", SAGE Publications Ltd, Hardcover, 2nd Edition, 2013.
4. Robert M Slade, "Software Forensics: Collecting Evidence from the Scene of a Digital Crime", Tata McGraw Hill, Paperback, 1st Edition, 2004.

CLOUD SECURITY LABORATORY

VII Semester: CSE (CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC18	Core	0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

Prerequisite: Computer Networks

I. COURSE OVERVIEW:

The process of assessing an application or infrastructure for vulnerabilities in an attempt to exploit those vulnerabilities, and circumvent or defeat security features of system components through rigorous manual testing.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The tools that can be used to perform information gathering.
II. The various attacks in various domains of cyberspace.
III. The exploits in various operating systems and wireless environment.
IV. The vulnerability assessment can be carried out by means of automatic tools or manual investigation.
V. The vulnerabilities associated with various network applications and database system.

III. COURSE SYLLABUS:

Week – 1:
Program to create one grid resource with three machines

Week – 2:
Program to create one or more Grid users. A Grid user contains one or more Gridlets

Week – 3:
Program to shows how two GridSim entities interact with each other ; main(ie example3) class creates Gridletsand sends them to the other GridSim entities, i.e. Test class

Week – 4:
Program shows how a grid user submits its Gridlets or tasks to one grid resource entity

Week – 5:
Program to show how a grid user submits its Gridlets or task to many grid resource entities

Week – 6:
Program to show how to create one or more grid users and submits its Gridlets or task to many grid resource entities

Week – 7:
Program to creates one Grid resource with three machines

Week – 8:
Building a Cloud using own cloud and LAMP server

Week – 9:
Implement to perform S3 bucket enumeration

Week – 10:
Implement to perform and exploiting misconfigured S3 buckets

Week –11:

Implement to perform escalating privileges of a target IAM user account by exploiting misconfigurations in a user policy

Week – 12:

Securing own cloud from Malicious File Uploads using CalmAV

IV. TEXT BOOKS:

1. Nina Godbole and Sunit Belpure," Cyber Security Understanding Cyber Crimes, Computer Forensics and Legal Perspectives", Wiley Pubications, 2014.
2. B.B.Gupta, D.P. Agrawal, Haoxiang Wang, "Computer and Cyber Security:Principles, Algorithm, Applications, and Perspectives", CRC Press, ISBN 9780815371335,2018.

V. REFERENCE BOOKS:

1. James Graham, Richard Howard and Ryan Otson ,"Cyber Security Essentials", CRC Press..
2. Chwan-Hwa(john) Wu,J. David Irwin , "Introduction to Cyber Security" , CRC Press T&F Group.

PROJECT WORK - I

VII Semester: Common for all branches								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACCC19	**Project**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	4	2	30	70	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 36**				**Total Classes: 36**		

The object of Project Work I is to enable the student to take up investigative study in the broad field of Computer Science and Engineering(CS), either fully theoretical/practical or involving both theoretical and practical work to be assigned by the Department on an individual basis or two/three students in a group, under the guidance of a Supervisor. This is expected to provide a good initiation for the student(s) in R&D work. The assignment to normally include:

1. Survey and study of published literature on the assigned topic;
2. Working out a preliminary Approach to the Problem relating to the assigned topic;
3. Conducting preliminary Analysis / Modeling / Simulation/Experiment/Design/Feasibility;
4. Preparing a Written Report on the Study conducted for presentation to the Department;
5. Final Seminar, as oral Presentation before a departmental committee.

SYSTEMS PROGRAMMING

VIII Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC36	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Computer Networks

I. COURSE OVERVIEW:

This course provides information about language processors and introduces you to design and implementation of various types of software, such as assemblers, macros, loaders and linkers. Apart from these, this course also includes compilers, aspects of compilation, memory allocation, compilation of expression and control structure, code optimization and interpreters.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The process of execution of High-Level Language programs
II. The working principles of compilers and parsers
III. The basic principles of designing system software.
IV. Implementation of system software for any application.

III. COURSE SYLLABUS:

MODULE-I: LANGUAGE PROCESSORS (08)

Introduction, Language Processing Activity, Fundamental of Language Processing, Fundamental of Language Specification, Language Processor Development tool.

MODULE –II: COMPILERS (09)

Compilers: Aspects of Compilation, various phases of a compiler, Memory Allocation, Compilation of Expressions, Compilation of Control Structures, Code Optimization.

MODULE –III: ASSEMBLERS, LINKER AND LOADER (09)

Assemblers: Elements of Assembly Language Programming, Assembly Scheme, Pass Structure of Assembler, Design of Assembler, Data structure, Format of Database, Algorithm, Look for modularity, Table processing: Searching and sorting, A Single Pass Assembler for IBM PC.

Reallocation and Linking Concept, Design of Linker, Self Reallocation Programs, Linking of Overlays, Loader, Absolute Loader, Reallocating Loader, Direct Linking Loader, Design of Loader

MODULE –IV: MACRO LANGUAGE AND MACRO PROCESSOR (10)

Macro Instructions, Features of Macro facility, Macro Instruction arguments, Conditional macro-Expression, Macro calls within micros, Macro Instructions defining Macros, Implementation of Two pass Algorithm.

MODULE –V: TEXT EDITOR AND DEBUGGING SYSTEM (09)

Introduction, Editing features, Type of Editor and user interface, Structure of editor, Editor design and evaluation, Editors function in computing environments, Error rates, Interactive debugging system, Debugging Functions and Capabilities, Type of bugs, Debugging techniques.

IV. TEXT BOOKS:

1. John J. Donovan, “System Programming”, McGraw-Hill Education.
2. D. M. Dhamdhere, “System Software and Operating System”, Tata McGraw-Hill.

V. REFERENCE BOOKS:

1. Leland L. Black, System Software -An Introduction to System Programming, Addison Wesley.
2. A.V. Aho, R. Sethi and J D. Ullman, “Compilers-Principles, Techniques and Tools”, Pearson Education.

VI. Web References:

1. http://web.mit.edu/jjd/www/documents/
2. https://trello.com/c/1Y8thzAT/35-john-j-donovan-systems-programmingpdf

HUMAN COMPUTER INTERACTION (UI & UX)

VIII Semester: CSE(CS)

Course Code	Category	Hours /Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACDC12	Elective	3	0	0	3	30	70	100
Contact Classes:45	Total Tutorials: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Computer Networks

I. COURSE OVERVIEW:

This course is an introduction to Human-Computer Interaction (HCI), a discipline concerned with the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them. The course considers the inherently multi- and interdisciplinary nature of HCI and situates various HCI issues in the organizational and societal contexts. It introduces theories of human psychology, principles of computer systems and user interfaces designs, a methodology of developing effective HCI for information systems, and issues involved in using technologies for different purposes. It is intended to give students an overview of the entire HCI field by covering most aspects of it. This course will thus provide a background for students to practice system design, selection, installation, evaluation, and use with the knowledge of human characteristics, interaction styles, use context, task characteristics, and design processes.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The Essentials of designing interactive systems
II. The different Techniques for designing interactive systems
III. The Contexts for designing interactive systems
IV. The important aspects of implementation of human-computer interfaces.
V. Identify the various tools and techniques for interface analysis, design, and evaluation.

III. SYLLABUS:

MODULE -I: ESSENTIALS OF DESIGNING INTERACTIVE SYSTEMS (09)

Designing interactive systems: a fusion of skills: The variety of interactive systems - The concerns of interactive systems design-Being digital-The skills of the interactive systems designer-Why being human-centered is important; The process of human-centered interactive systems design: Introduction- Developing personas and scenarios- Using scenarios throughout design - A scenario-based design method.

MODULE -II: TECHNIQUES FOR DESIGNING INTERACTIVE SYSTEMS (09)

Understanding: Understanding requirements- Participative design- Interviews- Questionnaires- Probes- Card sorting techniques-Working with groups - Fieldwork: observing activities in situ - Artefact collection and 'desk work'; Envisionment: Finding suitable representations- Basic techniques- Prototypes- Envisionment in practice; Design Introduction-Conceptual design- Metaphors in design- Conceptual design using scenarios - Physical design-Designing interactions; Evaluation Introduction -Expert evaluation -Participant-based evaluation - Evaluation in practice -Evaluation: further issues

MODULE -III: VISUAL INTERFACE DESIGN, MULTIMODAL INTERFACE DESIGN (09)

Visual interface design: Introduction, Graphical user interfaces, interface design guidelines, psychological principles and interface design, information design, visualization.

Multimodal interface design: Introduction, interacting in mixed reality, using sound at the interface, tangible interaction, gestural interaction and surface computing

MODULE -IV: CONTEXTS FOR DESIGNING INTERACTIVE SYSTEMS (09)

Designing websites 3io: Introduction, website development, the information architecture of websites, navigation design for websites; Case study: designing the Robert Louis Stevenson website; Social media: Introduction, background ideas, Social networking, Sharing with others, the developing web; Collaborative environments: Introduction, issues for cooperative working, technologies to support cooperative working, collaborative virtual

environments; Case study: Developing a collaborative tabletop application.

MODULE -V: UBIQUITOUS COMPUTING, MOBILE COMPUTING, WEARABLE COMPUTING (09)

Ubiquitous computing: Information spaces, blended spaces, home environments, navigating in wireless sensor networks; Mobile computing: Introduction, context awareness, understanding in mobile computing, designing for mobiles, evaluation for mobile computing; Wearable computing Introduction: Smart materials,material design, from materials to implants.

IV. TEXTBOOKS:

1. David R. Benyon, "Designing Interactive Systems: A Comprehensive Guide to HCI, UX and Interaction Design", Pearson; 3rd Edition, 2013
2. James Cabrera, "Modular Design Frameworks: A Projects-based Guide for UI/UX Designers", Apress, 1st Edition, 2017.
3. Alan Dix, Janet Finlay, Gregory Abowd, Russell Beale, "Human Computer Interaction", Pearson Education, 3rd Edition, 2004.

V. REFERENCEBOOKS:

1. Ben Schneiderman, "Designing the User Interface", Pearson Education Asia, 3rd Edition, 2013.
2. Prece, Rogers, Sharps, "Interaction Design", Wiley Dreamtech.
3. SorenLauesen, "User Interface Design", Pearson Education.
4. D. R. Olsen, "Human –Computer Interaction", Cengage Learning.
5. Smith –Atakan, "Human –Computer Interaction", Cengage Learning

VI. WEB REFERENCES:

1. http://staff.fit.ac.cy/com.ph/vp/VP_Lecture_2.pdf
2. https://fac.ksu.edu.sa/nmalmobarak/course/41031
3. https://www.tutorialspoint.com/human_computer_interface/quick_guide.html

INTERNET SYSTEMS PROGRAMMIMG

VIII Semester: CSE(CS)

Course Code	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC37	Elective	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes:45		

Prerequisite: Web Application Developments

I. COURSE OVERVIEW:

An Internet application is a client/server application that uses standard Internet protocols for connecting the client to the server. You can use exactly the same techniques to create a true Internet application, which is available publicly through the World Wide Web, or to create an intranet application. An intranet application is one that runs on your corporate intranet and is only available to the staff in your corporation. Whenever we talk about Internet applications, we mean either true Internet applications or intranet applications. For an introduction to the World Wide Web, see the appendix Introduction to the World Wide Web.

II. COURSE OBJECTIVES:

The students will try to learn:

I. Front-end development technologies including HTML, CSS, JavaScript, and JQuery.
II. User experience design methodologies like separation of concerns, Ajax, and responsive web design.
III. The anatomy and use of web requests and responses, including the types and formats of data that comprises them.
IV. How a web server works and the facilities it utilizes to service client requests.
V. The fundamental concepts related to search engine optimization, web accessibility, and web analytics.

III. COURSE SYLLABUS:

MODULE -I: WEBSITE BASICS, HTML 5, CSS 3, WEB 2.0 (09)

Web Essentials: Clients, Servers and Communication, The Internet, Basic Internet protocols, World wide web, HTTP Request Message, HTTP Response Message, Web Clients, Web Servers, HTML5, Tables, Lists, Image, HTML5 control elements, Semantic elements, Drag and Drop, Audio, Video controls, CSS3, Inline, embedded and external style sheets, Rule cascading, Inheritance, Backgrounds, Border Images, Colors, Shadows, Text, Transformations, Transitions, Animations.

MODULE -II: CLIENT-SIDE PROGRAMMING (09)

Java Script: An introduction to JavaScript–JavaScript DOM Model, Date and Objects, Regular Expressions, Exception Handling, Validation, Built-in objects, Event Handling, DHTML with JavaScript, JSON introduction – Syntax – Function Files – Http Request – SQL.

MODULE -III: SERVER-SIDE PROGRAMMING (09)

Servlets: Java Servlet Architecture, Servlet Life Cycle, Form GET and POST actions, Session Handling, Understanding Cookies, Installing and Configuring Apache Tomcat Web Server, DATABASE CONNECTIVITY: JDBC perspectives, JDBC program example.

JSP: Understanding Java Server Pages, JSP Standard Tag Library (JSTL), Creating HTML forms by embedding JSP code.

MODULE - IV: PHP AND XML (09)

An introduction to PHP: PHP, Using PHP, Variables, Program control, Built-in functions, Form Validation, Regular Expressions, File handling – Cookies, Connecting to Database; XML: Basic XML, Document Type Definition, XML Schema DOM and Presenting XML, XML Parsers and Validation, XSL and XSLT Transformation, News Feed (RSS and ATOM).

MODULE -V: INTRODUCTION TO AJAX and WEB SERVICES (09)

AJAX: Ajax Client Server Architecture, XML Http Request Object-Call Back Methods; Web Services:

Introduction, Java web services Basics – Creating, Publishing, Testing and Describing a Web services (WSDL), Consuming a web service, Database Driven web service from an application –SOAP.

IV. TEXTBOOKS:

I. Deitel and Deitel and Nieto, "Internet and World Wide Web, How to Program", Prentice Hall, 5th Edition, 2011.
II. Uttam K.Roy, "WebTechnologies", Oxford University Press, 2011.

V.REFERENCE BOOKS:

1. Chris Bates, "Web Programming – Building Intranet Applications", Wiley Publications, 3rd Edition, 2009.
2. Jeffrey C and Jackson, "Web Technologies A Computer Science Perspective", Pearson Education, 2011.

VI. WEB REFERENCES:

1. https://www.tutorialspoint.com/internet_technologies/internet_overview.htm
2. https://www.geeksforgeeks.org/internet,and,web,programming/
3. http://www.cs.uakron.edu/~xiao/isp/index.html
4. https://nptel.ac.in/courses/106/105/106105084/
5. https://www.udemy.com/courses/development/
6. https://www.coursera.org/courses?query=web%20development

VII. E-TEXT BOOKS:

1. Gopalan N.P. and Akilandeswari J., "Web Technology", Prentice Hall of India, 2011

RUST PROGRAMMING

VIII Semester: CSE(CS)

CourseCode	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSC38	Elective	3	0	0	3	30	70	100
Contact Classes:45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There are no prerequisite to take this course

I. COURSE OVERVIEW:

The Rust programming language helps you write faster, more reliable software. High-level ergonomics and low-level control are often at odds in programming language design; Rust challenges that conflict. Through balancing powerful technical capacity and a great developer experience, Rust gives you the option to control low-level details (such as memory usage) without all the hassle traditionally associated with such control. Rust is often cited as a language of choice for systems programming by developers because it combines best-in-class speed with a very low resource usage while still offering the safety of a standard server language. Rust solves problems associated with C/C++ such as garbage collection and safety.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The fundamental programming concepts designed for high performance and safety.
II. The concepts of structs, enums and pattern matching and also the features of rust programming.
III. The various error handling mechanisms to solve real world problem.
IV. How to develop software and programs for apps and websites.

III. COURSE SYLLABUS:

MODULE–I: INTRODUCTION AND COMMON PROGRAMMING CONCEPTS (09)

Installation, Hello, World!, Hello, Cargo!, Programming a Guessing Game, Variables and Mutability Data Types Functions, Comments, Control Flow, What is Ownership?, References and Borrowing, The Slice Type.

MODULE–II: STRUCTS , ENUMS AND PATTERN MATCHING (09)

Defining and Instantiating Structs, An Example Program Using Structs, Method Syntax, an Enum,The match Control Flow Operator, Concise Control Flow with if let. Managing Growing Projects with Packages, Crates, and Modules Packages and Crates, Defining Modules to Control Scope and Privacy.

MODULE– III:ERROR HANDLING (09)

Common Collections: Storing Lists of Values with Vectors, Storing UTF-8 Encoded Text with Strings, Storing Keys with Associated Values in Hash Maps. Unrecoverable Errors with panic!, Recoverable Errors with Result.

Programming Errors: Propagating Errors, a shortcut for propagating Erors; can only be used in functions that return result, to panic! or not panic!. Guidelines for error handling. Generic types traits and lifetimes;
Removing duplication by extracting a function, generic daa types, Traits: Defining shared behaviour.
To panic! or Not To panic!, Generic Data Types.

MODULE- IV: AUTOMATED TESTS & OOP FEATURES OF RUST (09)

How to Write Tests, Controlling How Tests Are Run, Test Organization, Characteristics of Object-Oriented Languages, Using Trait Objects That Allow for Values of Different Types, Implementing an Object-Oriented Design Pattern

MODULE- V: ADVANCED FEATURES, BUILDING A MULTITHREADED WEB SERVER (09)

Unsafe Rust, Advanced Traits, Advanced Types, Advanced Functions and Closures, Macros, Final Project: Building a Multithreaded Web Server. Building a Single-Threaded Web Server, Turning Our Single-Threaded Server into a Multithreaded Server, Graceful Shutdown and Cleanup.

IV. TEXT BOOKS:

1. Steve Klabnik and Carol Nichols, “The Rust Programming Language”, with Contributions from the Rust

Community.
2. Rust Web Programming: A hands-on guide to developing fast and secure web apps with the Rust programing language ,Maxwell Fitton.

V. REFERENCE BOOKS:

1. Carlo Milanesi, “Beginning Rust: From Novice to Professional”, 2018.
2. Kevin Hoffman, “Programming Web Assembly with Rust Unified Development for Web, Mobile, and Embedded Applications”, 2015.

VI. WEB REFERENCES:

1. https://en.wikipedia.org/wiki/Rust_(programming_language)
2. https://doc.rust-lang.org/stable/reference/
3. http://web.mit.edu/rust-lang_v1.25/arch/amd64_ubuntu1404/share/doc/rust/html/reference/index.html
4. https://towardsdatascience.com/you-want-to-learn-rust-but-you-dont-know-where-to-start-fc826402d5ba

VII. E-BOOKS:

1. Jim Blandy & Jason Orendorff, “Programming Rust: Fast, Safe Systems Development, 2017.

HIGH PERFORMANCE COMPUTING

VIII Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC20	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There is no prerequisite to take this course

I.COURSE OVERVIEW:

This course introduces the fundamentals of high-performance and parallel computing. It is targeted to scientists, engineers, scholars, really everyone seeking to develop the software skills necessary for work in parallel software environments. This course include big-data analysis, machine learning, parallel programming, and optimization and the basics of Linux environments and bash scripting all the way to high throughput computing and parallelizing the code.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The basic concepts related to HPC architecture and parallel computing.
II. Emerging trends in computing technology.
III. The advantage of deploying computing technology.
IV. How to improve the quality of the programs that you write for execution on high performance computer systems.

III. COURSE SYLLABUS

MODULE-1: CLUSTER COMPUTING AND ITS ARCHITECTURE (09)

Ease of computing, scalable parallel computer architecture, towards low cost parallel computing motivation, windows opportunity, a cluster computer and its architecture, cluster classification, commodity components for clusters, network services / communication sw, cluster middleware and single systems image, resource management & scheduling (rms).

MODULE-II: CLUSTER SETUP AND ADMINISTRATION (09)

Introduction, setting up the cluster, security, system monitoring, system tuning.

MODULE-III: INTRODUCTION TO GRID AND ITS EVOLUTION (09)

Introduction to grid and its evolution: beginning of the grid, building blocks of grid.

Grid application and grid middleware, evolution of the grid: first, second & third generation.

MODULE-IV: INTRODUCTION TO CLOUD COMPUTING (09)

Defining clouds, cloud providers, consuming cloud services, cloud models - iaas, paas, saas, inside the cloud, administering cloud services, technical interface, cloud resources.

MODULE-V: NATURE OF CLOUD (09)

Tradition data center , cost of cloud data center , scaling computer systems , cloud work load , managing data on clouds public, private and hybrid clouds; cloud elements: infrastructure as a service, platform as a service, software as a service.

IV.TEXT BOOKS

1. Rajkumar Buyya, “High Performance Cluster Computing, Volume 1, Architecture and Systems”, Pearson Education.
2. Berman, Fox and Hey, Grid Computing, “Making the Global Infrastructure Reality”, Wiley India.
3. Hurwitz, Bllor, Kaufman, Halper, “Cloud Computing for Dummies”, Wiley India.

V. REFERENCE BOOKS

1. Anthony Velte, Toby Velte, Robert Elsenpeter, "Cloud Computing, A Practical Approach", McGraw Hill. 2nd Edition, 2020.
2. Rajkumar Buyya, Satish Narayana Sriram, "Fog and Edge Computing: Principles and Paradigms", McGraw Hill. 2nd Edition, 2020.

QUANTUM COMPUTING

VIII Semester: CSE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC21	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: Linear Algebra and Calculus

I.COURSE OVERVIEW:

Quantum computers have the potential to efficiently solve problems that are intractable for classical computers. This course will explore the foundation of quantum computing. This course covers the basics of quantum computation, and different topics that explore both the capabilities and limitations of quantum computers. The main motive of this course is to provide the student to an introduction to quantum computation. Much of the background, material related to the algebra of complex vector spaces and quantum mechanics is covered within the course.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The structural units of quantum computers of the future, forming an understanding of the differences between quantum bits and classical bits.
II. The basic quantum logical operations and algorithms for processing quantum information.
III. The basic knowledge about the practical use of quantum algorithms and quantum programming skills.

III. COURSE SYLLABUS

MODULE-I: QUBIT & QUANTUM STATES (09)

Quantum bits, bloch sphere representation of a qubit, multiple qubits,vector spaces. linear combination of vectors, uniqueness of a spanning set, basis & dimensions, inner products, orthonormaliy gram-schmidt ortho gonalization, bra-ket formalism, the cauchy-schwarez and triangle.

MODULE-II: MATRICES & OPERATORS (09)

Observables, the pauli operators, outer products, the closure relation, representation of operators using matrices, outer products & matrix representation, matrix representation of operators in two dimensional spaces, pauli matrix, hermitian unitary and normal operator, eigen values & eigen vectors, spectral decomposition, trace of an operator, important properties of trace, expectation value of operator, projection operator, positive operators,commutator algebra, heisenberg uncertainty principle, polar decomposition & singular values, postulates of quantum mechanics.

MODULE-III: TENSOR PRODUCTS AND DENSITY OPERATOR (09)

Representing composite states in quantum mechanics, computing inner products, tensor products of column vectors, operators and tensor products of matrices.

Density operator of pure & mix state, key properties, characterizing mixed state, practical trace & reduce density operator, density operator & bloch vector.

MODULE-IV: QUANTUM MEASUREMENT THEORY, NOISE AND ERROR CORRECTION (09)

Distinguishing quantum states & measures, projective measurements, measurement on composite systems, generalized measurements, positive operator-valued measures, graph states and codes, quantum error correction.

MODULE-V: QUANTUM ALGORITHMS (09)

Classical computation on quantum computers, Relationship between quantum and classical complexity classes. Deutsch's algorithm, Deutsch's-Jozsa algorithm, Shor factorization, Grover search.

IV TEXT BOOKS

1. M A Nielsen and I L Chuang, “Quantum Computation and Quantum Information”, Cambridge University Press.
2. Principles of Quantum Computation and Information, Vol. I: Basic Concepts, Vol II: Basic Tools and Special Topics, World Scientific.
3. P Kaye, R Laflamme and M Mosca, “An Introduction to Quantum Computing”, Cambridge University Press.

V REFERENCE BOOKS

1. Zdzis_law Meglicki , “Quantum Computing without Magic”.
2. DAVID McMAHON, “Quantum Computing”, Cambridge University Press.

UBIQUITOUS COMPUTING

VIII Semester: CSE(CS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC22	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There is no prerequisite to take this course

I.COURSE OVERVIEW:

The major focus of this is to course is to explore the high level facilities, system architecture and protocols of the ubiquitous system and apply data analytics to facilitate next generation computing and is to acquaint participants with some of the fundamental concepts and state-of-the-art research in the areas of ubiquitous computing.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The basics of ubiquitous computing, its properties applications and architectural design.
II. The various smart devices and services used in ubiquitous computing.
III. The role of sensors and actuators in designing real time applications using Ubicomp.
IV. The concept of human computer interaction in the context of Ubicomp.
V. The indebt about Ubicomp network with design issues and Ubicomp management.

III. COURSE SYLLABUS

MODULE-I: INTRODUCTION TO UBIQUITOUS COMPUTING (09)

Concept of Ubiquitous Computing and Advantages, Ubiquitous Computing Applications and Scope, Properties of Ubiquitous Computing, Modelling the Key Ubiquitous Computing Properties. Ubiquitous System Environment Interaction. Architectural Design for UbiCom Systems: Smart DEI Model.

MODULE-II: UBIQUITOUS COMPUTING SMART DEVICES AND SERVICES (09)

Smart Devices and Service properties, Smart mobile devices and Users, Mobile code, Smart Card Devices and Networks, Service Architecture Models. Service Provision Life-Cycle. Virtual Machines and Operating Systems, OS for Mobile Computers and Communicator Devices.

MODULE-III: ACTUATION AND CONTROL (09)

Tagging the Physical World, Sensors and Networks, Micro- Electro-Mechanical Systems (MEMS), Embedded Systems and Real-Time Systems. Programmable and PID type control system, Robots.

MODULE-IV: UBIQUITOUS COMPUTING PRIVACY (09)

Ubiquitous computing privacy definition, Solove's taxonomy of privacy, legal background, Interpersonal privacy, Ubicomp challenges to privacy: Collection scale, manner and motivation, data types, data accessibility; Case study of privacy solution such as Protecting RFID tags, ways of addressing privacy in Ubicomp.

MODULE-V: UBIQUITOUS COMMUNICATION AND MANAGEMENT (09)

Data Networks, Audio Networks, Wireless Data Networks, Ubiquitous Networks, Service oriented networks, network design issues; Configuration and Security management, Service oriented computer and information management, Context awareness.

IV.TEXT BOOKS

1. Stefan Poslad, "Ubiquitous Computing", Wiley, Student Edition.
2. John Krumm, "Ubiquitous Computing Fundamentals".

V. REFERENCE BOOKS

1. Yin-LengTheng and Henry B. L. Duh, "Ubiquitous Computing", IGI, 2nd Edition, 2010.
2. Adam Greenfield, "Everyware the Drawing age of Ubiquitous Computing", AIGA, 1st Edition, 2009.
3. Laurence T. Yeng, EviSyukur and Seng W. Loke, "Handbook on Mobile and Ubiquitous Computing", CRC, 2nd Edition.

EDGE COMPUTING

VIII Semester: CSE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC23	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

Prerequisite: There is no prerequisite to take this course

I.COURSE OVERVIEW:

The course covers various topics such as the evolution of computing industry, cloud computing basics and edge computing. The course also educates the students on the different vendor platforms, software services, standard bodies and open source communities available for edge computing. This course provides a comprehensive understanding of multiple types of edge computing deployments, different types of edge compute services such as CDN Edge, IoT Edge, and Multi-access Edge (MEC).

II. COURSE OBJECTIVES:

The students will try to learn:

I. The fundamental principle and architectures of edge computing.
II. Gow to design and implement Internet of Everything (IoE) applications through fog computing. architecture.
III. The knowledge to log the sensor data and to perform further data analytics.
IV. The how the data will storage will be done and computation in edge computing.

III. COURSE SYLLABUS

MODULE-I: IOT AND EDGE COMPUTING DEFINITION AND USE CASES (09)

Introduction to edge computing scenario's and use cases - edge computing purpose and definition, edge computing use cases, edge computing hardware architectures, edge platforms, edge vs fog computing, communication models - edge, fog and M2M.

MODULE-II: IOT ARCHITECTURE AND CORE IOT MODULES (09)

IoT Architecture and Core IoT Modules-A connected ecosystem, IoT versus machine-to-machine versus, SCADA, The value of a network and Metcalfe's and Beckstrom's laws, IoT and edge architecture, Role of an architect, Understanding Implementations with examples-Example use case and deployment, Case study – Telemedicine palliative care, Requirements, Implementation, Use case retrospective.

MODULE-III: RASPBERRYPI (09)

Introduction to RaspberryPi, about the RaspberryPi Board: Hardware Layout and Pinouts, Operating Systems on RaspberryPi, Configuring RaspberryPi, Programming RaspberryPi.

Connecting Raspberry Pi via SSH, remote access tools, interfacing DHT sensor with Pi, Pi as webserver, Pi camera, image & video processing using Pi.

MODULE-IV: IMPLEMENTATION OF MICROCOMPUTER RASPBERRY (09)

Implementation of microcomputer RaspberryPi and device interfacing, edge to cloud protocols protocols, MQTT, MQTT publish-subscribe, MQTT architecture details, MQTT state transitions, MQTT packet structure, MQTT data types, MQTT communication formats, MQTT 3.1.1 working example.

MODULE-V: EDGE COMPUTING WITH RASPBERRYPI (09)

Edge computing with RaspberryPi, industrial and commercial IoT and edge, edge computing and solutions.

IV. TEXT BOOKS

1. Perry Lea, “IoT and Edge Computing for Architects”, Packt Publishing, 2nd Edition 2020.
2. Simon Monk, “Raspberry Pi Cookbook”, O'Reilly Media, Inc., 3rd Edition, 2019.

V. REFERENCE BOOKS

1. Rajkumar Buyya, Satish Narayana Srirama, “Fog and Edge Computing: Principles and Paradigms”, Wiley Publication, 2019.
2. David Jensen, “Beginning Azure IoT Edge Computing: Extending the Cloud to the Intelligent Edge”,

FLIGHT CONTROL THEORY

OE –I: VI Semester: AERO / MECH / CIVIL
OE – III: VIII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AAEC30	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

Flight control system of an aircraft is instrumental in establishing stability of the aircraft through control surfaces. This course introduces the concepts of the control system theory such as transfer functions, step response and impulse response. This course covers stability, feedback and different techniques used for control systems analysis. The course emphasizes on the flight control systems, response analysis for control surface inputs and control augmentation systems such as autopilots.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The stability criteria to determine the stability of an aircraft, and specify the aircraft time-domain and frequency-domain response specifications.
II. The classical control theory in the frequency domain and modern control theory in the state- space are effectively mixed to provide the student with a modern view of systems theory.
III. The various control techniques for aircraft control systems, and study some feedback control applications.
IV. The controllability and observability of aerospace systems, and apply the modern control techniques to design enhanced flight control systems.

III. COURSE SYLLABUS:

MODULE-I: INTRODUCTION TO CONTROL SYSTEM (10)

Dynamical systems-principal constituents-input, output-process (plant)-block diagram representation. Inputs- control input, noise. Function of controls regulation (hold), tracking (command)-examples. Measure of effectiveness. Sensitivity of output to control input, noise and system parameters-robustness. Deterministic and stochastic control. Control in everyday life. The pervasiveness of control in nature, engineering and societal systems. The importance of study of control system. Need for stable, effective (responsive), robust control system. Modeling of dynamical systems by differential equations-system parameters. Examples from diverse fields. First and second order systems, higher order systems, single input single output systems, and multiple-input multiple-output.

MODULE –II: MATHEMATICAL MODELING OF DYNAMICAL SYSTEMS (10)

Control system performance- time domain description- output response to control inputs-- impulse and indicial response- characteristic parameters- significance- relation to system parameters- examples- first and second order linear systems, higher order systems. Synthesis of response to arbitrary input functions from impulse and indicial response. Review of Fourier transforms and Laplace transforms-inverse transforms- significance, applications to differential equations. 's' (Laplace) domain description of input- output relations- transfer function representation- system parameters- gain, poles and zeroes. Characteristic equation- significance- examples. Frequency and damping ratio of dominant poles. Relation of transfer functions to impulse response. Partial fraction decomposition of transfer functions-significance.

MODULE –III: STEADY STATE RESPONSE ANALYSIS (10)

System type, steady state error, error constants- overall system stability. Application of feedback in stability augmentation, control augmentation, automatic control-examples. Composition, reduction of

block diagrams of complex systems-rules and conventions. Control system components - sensors, transducers, servomotors, actuators, filters-modeling, transfer functions. Single-input single-output systems. Multiple input-multiple output systems, matrix transfer functions-examples. Types of control problems- the problem of analysis, control synthesis, system synthesis- examples- static control of aircraft.

Extension to dynamic control. System identification from input output measurements importance. Flight path stabilization, longitudinal control law design using back stepping algorithm. Experimental determination of system transfer functions by frequency response measurements. Example. Frequency domain description- frequency response- gain and phase shift- significance- representation asymptotic (Bode) plots, polar (Nyquist) plots, frequency transfer functions. Characteristic parameters corner frequencies, resonant frequencies, peak gain, and bandwidth- significance. First and second order systems- extension to higher order systems.

MODULE –IV: AIRCRAFT RESPONSE TO CONTROL (07)

Approximations to aircraft transfer functions, control surface actuators-review. Response of aircraft to elevator input, Response of aircraft to rudder input and Response of aircraft to aileron input to atmosphere. Need for automatic control. Auto pilots Stability augmentation systems-pitch damper and yaw damper.

MODULE –V: FLYING QUALITIES OF AIRCRAFT (08)

Reversible and irreversible flight control systems. Flying qualities of aircraft-relation to airframe transfer function. Pilot's opinion ratings. Flying quality requirements- pole-zero, frequency response and time- response specifications. Displacement and rate feedback determination of gains conflict with pilot input s resolution-control augmentation systems- Full authority fly-by-wire. Auto Pilot-Normal acceleration, Turn rate, Pitch rate Commands-Applications.

IV. TEXT BOOKS:

1. Kuo, B.C., “Automatic control of Aircraft and Missiles”, John Wiley Sons, New York, 1990.
2. Stevens B.L & Lewis F.L, “Aircraft control & Simulation”, John Wiley Sons, New York, 1992.

V. REFERENCE BOOKS:

1. Mc Lean, D., “Automatic Flight Control Systems”, Prentice Hall, 1990.
2. Bryson, A.E., “Control of Aircraft and Spacecraft”, Princeton University Press, 1994.
3. E H J Pallett, Shawn Coyle, “Automatic Flight Control”, 4th Edition, 2002.

VI. WEB REFERENCES:

1. https://www.e-booksdirectory.com/
2. https://www.aerospaceengineering.es/book/

VII. E-TEXT BOOKS:

1. https://books.google.co.in/books?isbn=1118870972
2. https://books.google.co.in/books?isbn=0387007261

AIRFRAME STRUCTURAL DESIGN

OE –I: VI Semester: AERO / MECH / CIVIL
OE – III: VIII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AAEC31	Elective	3	-	-	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

This course deals with fundamental aspects of an anatomy of aircraft and the current trends in airframe design. It includes the evolution of the aircraft and space industry, aerodynamics and performance of the aircraft with their applications. It compares and contrasts various thrust vector control mechanisms of different aircraft propulsion systems. It discusses various materials and its properties that are used for manufacturing different parts of an aircraft. This course enriches the knowledge of connection between theoretical and practical methods for performing the airframe design exercises

II. COURSE OBJECTIVES:

The student will try to learn:

I. The fundamental concepts of various airframe designs, aircraft propulsion systems and aerodynamic forces/moments acting on the aircraft and spacecraft under static and dynamic load conditions
II. The characteristics of stability and performance of an aircraft and the role of primary and secondary controls in longitudinal and lateral stability
III. The properties of different materials that are used in industries for manufacturing various components of an aircraft and spacecraft achieving specified stability requirements.
IV. The mathematical modeling of tailless aircraft, flapping wing aircraft and innovative designs in modern aircraft for future requirements.

II. COURSE SYLLABUS:

MODULE-I: HISTORY OF FLIGHT AND SPACE ENVIRONMENT (10)

Balloons and dirigibles, heavier than air aircraft, commercial air transport; Introduction of jet aircraft, helicopters, missiles; Conquest of space, commercial use of space; Different types of flight vehicles, classifications exploring solar system and beyond, a permanent presence of humans in space; Earth‘s atmosphere, the standard atmosphere; The temperature extremes of space, laws of gravitation, low earth orbit, microgravity, benefits of microgravity; Environmental impact on spacecraft, space debris; Planetary environments.

MODULE –II: INTRODUCTION TO AERODYNAMICS (10)

Anatomy of the airplane, helicopter; Understanding engineering models; Aerodynamic forces on a wing, force coefficients; Generating lift, moment coefficients; Aerodynamic forces on aircraft – classification of NACA airfoils, aspect ratio, wing loading, mach number, centre of pressure and aerodynamic centre, aerofoil characteristics-lift, drag curves; Different types of drag.

MODULE –III: FLIGHT VEHICLE PERFORMANCE AND STABILITY (09)

Performance parameters, performance in steady flight, cruise, climb, range, endurance, accelerated flight symmetric maneuvers, turns, sideslips, takeoff and landing.

Flight vehicle Stability, static stability, dynamic stability; Longitudinal and lateral stability; Handling qualities of the airplanes.

MODULE –IV: INTRODUCTION TO AIRPLANE STRUCTURES AND MATERIAL,POWERPLANT (08)

General types of construction, monocoque, semi-monocoque; Typical wing and fuselage structure; Metallic & non-metallic materials, use of aluminum alloy, titanium, stainless steel and composite materials; Basic ideas about engines, use of propellers and jets for thrust production; Principles of operation of rocket, types of rockets.

MODULE –V: SATELLITE SYSTEMS ENGINEERING HUMAN SPACE EXPLORATION (08)

Satellite missions, an operational satellite system, elements of satellite, satellite bus subsystems; Satellite structures, mechanisms and materials; Power systems; Communication and telemetry; Propulsion and station keeping; Space missions, mission objectives. Goals of human space flight missions, historical background, the Soviet and US missions; The mercury, Gemini, Apollo (manned flight to the moon), Skylab, apollo-soyuz, space Shuttle; International space station, extravehicular activity; The space suit; The US and Russian designs; Life support systems, flight safety; Indian effort in aviation, missile and space technology.

IV. TEXT BOOKS:

1. Newman D, "Interactive Aerospace Engineering and Design", McGraw-Hill, 1st Edition, 2002.
2. Anderson J. D, "Introduction To Flight", McGraw-Hill Education, 5th Edition, 2002.

V. REFERENCE BOOKS:

1. Kermode. A. C, "Flight without Formulae", McGraw Hill, 4th Edition, 1997.
2. Barnard R.H and Philpot. D.R, "Aircraft Flight", Pearson, 3rd Edition, 2004.
3. SwattonP.J, "Flight Planning", Blackwell Publisher, 6th Edition, 2002.

VI. WEB REFERENCES:

1. http://ase.sbu.ac.ir/FA/Staff/abbasrahi/Lists/Dars/Attachments/11/Vibrations%20of%20Continuous%20Systems.pdf
2. http://arc-test.aiaa.org/doi/book/10.2514/4.862458
3. http://arc-test.aiaa.org/doi/abs/10.2514/5.9781600862373.0719.0728

VII. E-TEXT BOOKS:

1. http://www.gregorypaulblog.com/structural-dynamics-in-aeronautical-engineering-aiaa-education-series.pdf
2. https://aerocastle.files.wordpress.com/2012/10/mechanical_vibrations_5th-edition_s-s-rao.pdf

INDUSTRIAL MANAGEMENT

OE –I: VI Semester: AERO / MECH / CIVIL
OE – III: VIII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AMEC34	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I.COURSE OVERVIEW:

The industrial management prepares engineers to design, improve, install, and operate the integrated systems of people, materials, and facilities needed by industry, commerce, and society. Industrial engineers solve problems that arise in the management of systems, applying the principles of engineering science, product/service and process design, work analysis, human factors principles, and operations research.The focus of this course is how to improve processes or design things that are more efficient and waste less money, time, raw resources, man-power and energy while following safety standards and regulations

II. COURSE OBJECTIVES:

The students will try to learn:

I. The production planning and control procedures to handle industrial disputes.
II. The Work study procedures and quality concepts to enhance more productivity
III. The significant exposure on some maintenance practices in industry for consistent productivity.

III.COURSE SYLLABUS:

MODULE-I: CONCEPTS OF INDUSTRIAL MANAGEMENT (9)

Principles of management- Growth of management thought, Functions of management, Principles of organization, Types of organization and committees.

MODULE –II: WORK STUDY (9)

Concept of productivity, Method Study - Basic steps in method study, Process charts, Diagrams, Principles of motion economy, Micro motionstudy, Therbligs, SIMO chart. Work Measurement - Stop watch procedure of time study, Performance rating, allowances, Work sampling, Simple problems.

MODULE –III: INVENTORY CONTROL (9)

Inventory Control: Inventory, Cost, Deterministic Models and Introduction to Supply Chain Management.

MODULE –IV: QUALITY CONTROL (9)

Quality Control: Process control, SQC, Control charts, Single, Double and Sequential Sampling, Introduction to TQM.

MODULE –V: DEMAND FORECASTING AND COST ESTIMATION (9)

Demand Forecasting and cost Estimation: Characteristics of Forecasts, Forecasting Horizons, Steps to Forecasting, Forecasting Methods, Seasonal Adjustments, Forecasting Performance Measures, Cost Estimation, Elements of cost, Computation of Material Variances Break-Even Analysis.

IV. TEXT BOOKS:

1. O.P. Khanna, “Industrial Engineering and Management”, Khanna Publishers.
2. T.R. Banga and S.C.Sarma, “Industrial Engineering and Management Science”, Khanna Publishers.

V. REFERENCE BOOKS:

1. Ralph M Barnes, “Motion and Time Study”, John Willey & Sons Work Study lLO.
2. Ernest J McCormick, “Human factors in Engineering & Design”, TMH.
3. Paneer Selvam, “Production & Operation Management”, PHI.
4. NVS Raju, “Industrial Engineering Management”, Cengage Learning.

VI. REFERENCE BOOKS:

1. https://nptel.ac.in/courses/112/107/112107142/#
2. https://nptel.ac.in/courses/112/107/112107143/#

ELEMENTS OF MECHANICAL ENGINEERING

OE –I: VI Semester: AERO / MECH / CIVIL
OE – III: VIII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AMEC35	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 45		

I.COURSE OVERVIEW:

The main aim of this course to impart mechanical engineering fundamental basics to allied engineering students so that they have minimum understanding of mechanical system, equipment and process.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The fundamentals of mechanical systems.
II. The significance of mechanical engineering and apply in different fields of engineering.
III. The various applications of engineering materials for designing different engineering components.

III. COURSE SYLLABUS:

MODULE-I: SOURCES OF ENERGY, BASIC CONCEPTS OF THERMODYNAMICS (9)

Sources of Energy : Introduction and application of energy sources like fossil fuels, hydel, solar, wind, nuclear fuels and bio-fuels; environmental issues like global warming and ozone depletion.

Basic concepts of Thermodynamics: Introduction, states, concept of work, heat, temperature; Zeroth, 1st, 2nd and 3rd laws of thermodynamics. Concept of internal energy, enthalpy and entropy (simple numericals).

MODULE –II: BOILER AND TURBINES(9)

Boilers: Introduction to boilers, classification, Lancashire boiler, Babcock and Wilcox boiler. Introduction to boiler mountings and accessories (no sketches).

Turbines: Hydraulic Turbines-Classification and specification, Principles and operation of Pelton wheel turbine, Francis turbine and Kaplan turbine (elementary treatment only).

Hydraulic Pumps: Introduction, classification and specification of pumps, reciprocating pump and centrifugal pump, concept of cavitations and priming.

MODULE –III: PROPERTIES, COMPOSITION AND INDUSTRIAL APPLICATIONS OF ENGINEERING MATERIALS(9)

Metals-Ferrous: cast iron, tool steels and stainless steels and nonferrous: aluminum, brass, bronze. Polymers -Thermoplastics and thermosetting polymers. Ceramics -Glass, optical fiber glass, cermets. Composites -Fiber reinforced composites, Metal Matrix Composites, Smart materials -Piezoelectric materials, shape memory alloys, semiconductors and insulators.

Joining Processes: Soldering, Brazing and Welding Definitions. Classification and methods of soldering, brazing and welding. Brief description of arc welding, oxy-acetylene welding, TIG welding, and MIG welding.

MODULE –IV: MACHINE TOOLS(9)

Lathe -Principle of working of a center lathe. Parts of a lathe. Operations on lathe –Turning, Facing, Knurling, Thread Cutting, Drilling, Taper turning by Tailstock offset method and Compound slide swiveling method, Specification of Lathe.

Milling Machine-Principle of milling, types of milling machines. Working of horizontal and vertical milling machines. Milling processes -plane milling, end milling, slot milling, angular milling, form milling, straddle milling, and gang milling.

MODULE –V: INTRODUCTION TO ADVANCED MANUFACTURING SYSTEMS (9)

Computer Numerical Control (CNC): Introduction. Components of CNC, open loop and closed loop systems, advantages of CNC, CNC Machining centers and Turning centers.

Robots: Robot anatomy, joints and links, common robot configurations. Applications of Robots in material handling, processing and assembly and inspection

IV.TEXT BOOKS

1. V. K. Manglik, "Elements of Mechanical Engineering", Prentice Hall, 1st Edition, 2013.
2. Mikell P. Groover, "Automation, Production Systems and CIM", Prentice Hall, 4th Edition, 2013

V.REFERENCE BOOKS:

1. S. Trymbaka Murthy, "A Text Book of Elements of Mechanical Engineering", University Press, 4th Edition, 2006.
2. K. P. Roy, S. K. Hajra Choudary, Nirjhar Roy, "Element of Mechanical Engineering", Media Promoters & Publishers, 7th Edition, 2012.
3. Pravin Kumar, "Basic Mechanical Engineering", Pearson, 1st Edition, 2013

VI.WEB REFERENCES:

1. http://www.nptel.ac.in/courses/112107144/
2. http://www.nptel.ac.in/courses/112101098/download/lecture-37.pdf

MODERN CONSTRUCTION MATERIALS

OE –I: VI Semester: AERO / MECH / CIVIL
OE – III: VIII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACEC30	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Total Tutorials: Nil	Total Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

This course provides the scientific basis for the understanding and development of construction materials. It serves as a foundation course for post-graduate students interested in careers involving research, teaching and/or construction engineering, as well as marketing, decision making, innovation and specification related to construction materials.

II. COURSE OBJECTIVES:

The student will try to learn:

I. The concept of modern water proofing and insulating materials in constructions.
II. Importance of composites and chemicals in production of modern concrete.
III. The types of concrete and their constituents and properties.
IV. The impact of building construction on society and demonstrate awareness of contemporary issues.

III.COURSE SYLLABUS:

MODULE-I: STONES – BRICKS – CONCRETE BLOCKS (09)

Stone as building material, Criteria for selection, Tests on stones, Deterioration and Preservation of stone work, Bricks, Classification, Manufacturing of clay bricks, Tests on bricks Compressive Strength, Water Absorption, Efflorescence, Bricks for special use, Refractory bricks, Cement, Concrete blocks, Light-weight concrete blocks.

MODULE-II: LIME – CEMENT – AGGREGATES – MORTAR (09)

Lime, Preparation of lime mortar, Cement, Ingredients, Manufacturing process, Types and Grades, Properties of cement and Cement mortar, Hydration, Compressive strength, Tensile strength, Fineness, Soundness and consistency, Setting time, Industrial byproducts, Fly ash, Aggregates, Natural stone aggregates, Crushing strength, Impact strength, Flakiness Index, Elongation Index, Abrasion Resistance, Grading, Sand Bulking.

MODULE-III: CONCRETE (09)

Concrete, Ingredients, Manufacturing Process, Batching plants, RMC, Properties of fresh concrete, Slump, Flow and compaction Factor, Properties of hardened concrete, Compressive, Tensile and shear strength.

Modulus of rupture, Tests, Mix specification, Mix proportioning, BIS method, High Strength Concrete and HPC, Self compacting Concrete, Other types of Concrete, Durability of Concrete.

MODULE-IV: TIMBER AND OTHER MATERIALS (09)

Timber, Market forms, Industrial timber, Plywood, Veneer, Thermacole, Panels of Laminates, Steel, Aluminum and Other Metallic Materials, Composition, Aluminium composite panel, Uses: Market forms, Mechanical treatment, Paints, Varnishes, Distempers, Bitumens.

MODULE-V: MODERN MATERIALS (09)

Glass, Ceramics, Sealants for joints, Fibre glass reinforced plastic, Clay products, Refractories, Composite materials, Types, Applications of laminar composites, Fibre textiles, Geomembranes and Geotextiles for earth reinforcement.

IV.TEXT BOOKS:

1. W.D. Callister, John Wiley, "Materials Science and Engineering: An Introduction", John Wiley & Sons, Inc. 1994.
2. P.C. Varghese, "Building Materials", Prentice-Hall India, 2005.

V. REFERENCE BOOKS:

1. V. Raghavan, "Materials Science and Engineering", Prentice Hall, 1990.
2. R.A. Higgins, "Properties of Engineering Materials", Industrial Press, 1994.
3. Eds. J.M. Illston and P.L.J. Domone, "Construction Materials: Their nature and behaviour", Spon Press, 3rd Edition, 2002

VI.WEB REFERENCES:

1. https://www.scribd.com/document/394619658/Material-Science-and-Engineering-V-Raghavan-pdf
2. https://files.isec.pt/DOCUMENTOS/SERVICOS/BIBLIO/INFORMA%C3%87%C3%95ES%20ADICIONAIS/Materials-for-engineers-5ed_Higgins.pdf

VII. E-TEXT BOOKS:

1. https://onlinecourses.nptel.ac.in/noc20_ce05/preview
2. http://kaizenha.com/wp-content/uploads/2016/04/Materials-Textbook-8th-Edition.pdf

DISASTER MANAGEMENT

OE –I: VI Semester: AERO / MECH / CIVIL
OE – III: VIII Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / CSIT / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACEC31	Elective	3	0	0	3	30	70	100
Contact Classes: 45	Total Tutorials: Nil	Total Practical Classes: Nil				Total Classes: 45		

I. COURSE OVERVIEW:

The Disaster management provides a fundamental understanding of different aspects. It deals with the concepts and functions of disaster management to build competencies of professionals and development practitioners. It provides effective supporting environment by the governmental locating substantial resources for effective mitigation of disasters. It helps learners to apply the disaster mitigation strategies, preparedness for reducing damage intensity, loss of life and property.

II. COURSE OBJECTIVES:

The student will try to learn:

I. The concept of environmental hazards, disasters and various approaches dealing with the mitigation of disasters.
II. The knowledge on various types of environmental disasters and their impacts on human beings and nature.
III. The Different types of endogenous and exogenous hazards and their influence on human life and nature.
IV. The immediate response and damage assessment with information reporting and monitoring tools.

III.COURSE SYLLABUS:

MODULE-I: ENVIRONMENTAL HAZARDS AND DISASTERS (09)

Environmental hazards and disasters: meaning of environmental hazards, environmental disasters and environmental stress; concept of environmental hazards, environmental stress and environmental disasters, different approaches and relation with human ecology, landscape approach, ecosystem approach, perception approach, human ecology and its application in geographical researches.

MODULE-II: TYPES OF ENVIRONMENTAL HAZARDS AND DISASTERS (09)

Types of environmental hazards and disasters: Natural hazards and disasters, man induced hazards and disasters, natural hazards, planetary hazards/ disasters, extra planetary hazards/ disasters, planetary hazards, endogenous hazards, exogenous hazards.

MODULE-III: ENDOGENOUS HAZARDS (09)

Endogenous hazards, volcanic eruption, earthquakes, landslides, volcanic hazards/ disasters, causes and distribution of volcanoes, hazardous effects of volcanic eruptions, environmental impacts of volcanic eruptions.

Earthquake hazards/ disasters, causes of earthquakes, distribution of earthquakes, hazardous effects of, earthquakes, earthquake hazards in India, human adjustment, perception and mitigation of earthquake

MODULE-IV: EXOGENOUS HAZARDS (09)

Exogenous hazards/ disasters, infrequent events, cumulative atmospheric hazards/ disasters; Infrequent events: Cyclones , lightning , hailstorms; Cyclones: Tropical cyclones and local storms, destruction by tropical cyclones and local storms (causes, distribution human adjustment, perception and mitigation); Cumulative atmospheric hazards/ disasters: Floods, droughts, cold waves, heat waves floods; Causes of

floods, flood hazards India, flood control measures (human adjustment, perception and mitigation); Droughts: Impacts of droughts, drought hazards in India, drought control measures, extra planetary hazards/ disasters, man induced hazards /disasters, physical hazards/ disasters, soil erosion, Soil erosion: Mechanics and forms of soil erosion, factors and causes of soil erosion, conservation measures of soil erosion; Chemical hazards/ disasters: Release of toxic chemicals, nuclear explosion, sedimentation processes; Sedimentation processes: Global sedimentation problems regional sedimentation problems, sedimentation and environmental problems, corrective measures of erosion and sedimentation, biological hazards/ disasters, population explosion.

MODULE-V: EMERGING APPROACHES IN DISASTER MANAGEMENT (09)

Emerging approaches in Disaster Management, Three Stages

1. Pre, disaster stage(preparedness)
2. Emergency Stage
3. Post Disaster stage, Rehabilitation.

IV.TEXT BOOKS:

1. Pardeep Sahni, "Disaster Mitigation: Experiences and Reflections", PHI Learning Pvt. Ltd., 1st Edition, 2001.
2. J.Glynn,Gary W.HeinKe, "Environmental Science and Engineering", Prentice Hall Publishers, 2nd Edition, 1996.

V. REFERENCE BOOKS:

1. B. C. Punmia, Ashok K Jain and Arun K Jain, "Mechanics of Materials", Laxmi Publications Pvt. Ltd., New Delhi, 12th Edition, 2007.
2. R. Subramanian, "Strength of Materials", Oxford University Press, 2nd Edition, 2010.
3. D. S. Prakash Rao, "Strength of Materials A Practical Approach Vol.1", Universities Press (India) Pvt. Ltd., India, 3rd Edition, 2007.
4. J. M. Gere, S.P. Timoshenko, "Mechanics of Materials, SI units edition", CL Engineering, USA, 5th Edition, 2000.

VI. Web References:

1. https://www.google.co.in/?gfe_rd=cr&ei=,iAwWLiDIazv8we8_5LADA#q=disater+mangement
2. http://ndma.gov.in/images/policyplan/dmplan/National%20Disaster%20Management%20Plan%20May%202016.pdf
3. http://www.eib.europa.eu/attachments/pipeline/20080021_eia_en.pdf
4. http://www.ndmindia.nic.in/

VII. E-Text Books:

1. http://cbse.nic.in/natural%20hazards%20&%20disaster%20management.pdf\
2. http://www.digitalbookindex.org/_search/search010emergencydisastera.asp
3. http://www.icbse.com/books/cbse,ebooks,download

PROJECT WORK - II

VIII Semester: **Common for all branches**

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACCC24	Project	0	0	16	8	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 180				Total Classes: 180		

The object of Project Work II & Dissertation is to enable the student to extend further the investigative study taken up under CSE(CS) P1, either fully theoretical/practical or involving both theoretical and practical work, under the guidance of a Supervisor from the Department alone or jointly with a Supervisor drawn from R&D laboratory/Industry. This is expected to provide a good training for the student(s) in R&D work and technical leadership. The assignment to normally include:

1. In depth study of the topic assigned in the light of the Report prepared under CSE(CS) P1;
2. Review and finalization of the Approach to the Problem relating to the assigned topic;
3. Preparing an Action Plan for conducting the investigation, including team work;
4. Detailed Analysis / Modeling / Simulation / Design / Problem Solving / Experiment as needed;
5. Final development of product/process, testing, results, conclusions and future directions;
6. Preparing a paper for Conference presentation/Publication in Journals, if possible;
7. Preparing a Dissertation in the standard format for being evaluated by the Department.
8. Final Seminar Presentation before a Departmental Committee.

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

UNDERTAKING BY STUDENT / PARENT

"To make the students attend the classes regularly from the first day of starting of classes and be aware of the College regulations, the following Undertaking Form is introduced which should be signed by both student and parent. The same should be submitted to the Dean, Academic".

I, Mr. / Ms. ---------- joining I Semester / III Semester for the academic year 20 - 20 / 20 - 20 in Institute of Aeronautical Engineering, Hyderabad, do hereby undertake and abide by the following terms, and I will bring the ACKNOWLEDGEMENT duly signed by me and my parent and submit it to the Dean of Academic.

1. I will attend all the classes as per the timetable from the starting day of the semester specified in the institute Academic Calendar. In case, I do not turn up even after two weeks of starting of classes, I shall be ineligible to continue for the current academic year.
2. I will be regular and punctual to all the classes (theory/laboratory/project) and secure attendance of not less than 75% in every course as stipulated by Institute. I am fully aware that an attendance of less than 65% in more than 60% of theory courses in a semester will make me lose one year.
3. I will compulsorily follow the dress code prescribed by the college.
4. I will conduct myself in a highly disciplined and decent manner both inside the classroom and on campus, failing which suitable action may be taken against me as per the rules and regulations of the institute.
5. I will concentrate on my studies without wasting time in the Campus/Hostel/Residence and attend all the tests to secure more than the minimum prescribed Class/Sessional Marks in each course. I will submit the assignments given in time to improve my performance.
6. I will not use Mobile Phone in the institute premises and also, I will not involve in any form of ragging inside or outside the campus. I am fully aware that using mobile phone to the institute premises is not permissible and involving in Ragging is an offence and punishable as per JNTUH/UGC rules and the law.
7. I declare that I shall not indulge in ragging, eve-teasing, smoking, consuming alcohol drug abuse or any other anti-social activity in the college premises, hostel, on educational tours, industrial visits or elsewhere.
8. I will pay tuition fees, examination fees and any other dues within the stipulated time as required by the Institution / authorities, failing which I will not be permitted to attend the classes.
9. I will not cause or involve in any sort of violence or disturbance both within and outside the college campus.
10. If I absent myself continuously for 3 days, my parents will have to meet the HOD concerned/ Principal.
11. I hereby acknowledge that I have received a copy of IARE – UG20 Academic Rules and Regulations, Syllabus copy and hence, I shall abide by all the rules specified in it.

ACKNOWLEDGEMENT

I have carefully gone through the terms of the undertaking mentioned above and I understand that following these are for my/his/her own benefit and improvement. I also understand that if I/he/she fail to comply with these terms, shall be liable for suitable action as per Institute/JNTUH/AICTE/UGC rules and the law. I undertake that I/he/she will strictly follow the above terms.

Signature of Student with Date

Signature of Parent with Date
Name & Address with Phone Number