



**GEETHANJALI COLLEGE OF ENGINEERING and TECHNOLOGY
(AUTONOMOUS)**

Cheeryal (V), Keesara (M), Medchal (Dist). Telangana- 501 301

(Approved by AICTE, NEW DELHI, Accredited by NBA)

www.geethanjaliinstitutions.com

2020-2021

COURSE FILE

Programming for Problem Solving-I

(Subject Code: 20CS11001)

I B.Tech. I Sem.



**DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING**

S.No	Contents	Page No.
1	Cover Page	3
2	Syllabus copy	4
3	Vision of the Department	6
4	Mission of the Department	7
5	PEOs, POs and PSOs	8
6	Course objectives and outcomes	16
7	Brief notes on the importance of the course and how it fits into the curriculum	16
8	Prerequisites if any	19
9	Instructional Learning Outcomes	19
10	Course mapping with POs	20
11	Class Time Table	30
12	Individual Time Table	50
13	Lecture schedule with methodology being used/adopted	53
14	Detailed notes	55
15	Additional topics	134
16	University Question papers of previous years	139
17	Assignment Questions	144
18	Question Bank	145
19	Unit wise Quiz Questions and long answer questions	149
20	Tutorial problems	162
21	Known gaps ,if any and inclusion of the same in lecture schedule	163
22	Discussion topics , if any	163
23	References, Journals, websites and E-links if any	163
24	Quality Measurement Sheets a. Course End Survey b. Teaching Evaluation	164
25	Student List	164
26	Group-Wise students list for discussion topics	164

Course coordinator

Program Coordinator

HOD

GEETHANJALI COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF CSE

Name of the Subject : Programming for Problem Solving-I

JNTU CODE: 20CS1101

Programme : UG

Branch: All Branches

Version No : 01

Year: I Year B. Tech

Updated on : 29/10/20

Semester: I SEM

No. of pages : 134

Classification status (Unrestricted / ~~Restricted~~)

Distribution List : Students, Library, Faculty

Updated by : 1) Name : Mrs C. Esther Varma

1) Name: Dr. Puja S Prasad

2) Design: Associate Professor

2) Design: Associate Professor

3) Sign :

3) Sign:

4) Date :

4) Date:

Verified by : 1) Name :

*** For Q.C Only.**

2) Sign :

1) Name :

3) Design : Professor

2) Sign :

4) Date :

3) Design :

4) Date :

Approved by : (HOD) 1) Name :

2) Sign :

3) Date :

2. SYLLABUS

Programming for Problem Solving-I

Code: 20CS11001

1st year Semester-1

L-2, C- 2

UNIT – I

Basics of Computers- Introduction to components of a computer system: disks, primary and secondary memory, processor, operating system, compilers

Logic Building: Flow chart, Algorithm, Pseudo code.

Introduction to Programming – Computer Languages, Creating and running programs, Program Development.

Introduction to the C Language – Background, C Programs, Identifiers, Data Types, Variables, Constants, Input/output functions.

Operators - Arithmetic, relational, logical, bitwise, conditional, increment/decrement, assignment etc., C program examples. Expressions, Precedence and Associativity, Expression Evaluation, Type conversions.

UNIT - II

Statements- Selection Statements (decision making) – if and switch statements with C program examples.

Repetition statements (loops) - while, for, do-while statements with C Program examples

Statements related to looping – break, continue, goto, Simple C Program examples.

UNIT - III

Functions-Designing Structured Programs, Functions, user defined functions, inter function communication, Standard functions, Scope, Storage classes-auto, register, static, extern, scope rules, type qualifiers, C program examples.

Recursion- recursive functions, Limitations of recursion, example C programs

UNIT -IV

Arrays – Concepts, using arrays in C, arrays and functions, Bubble Sort, Linear Search, two – dimensional arrays-matrix addition and matrix multiplication, Declaration of Multidimensional arrays, Pre-processor Directives, C program examples.

UNIT - V

Pointers – Introduction (Basic Concepts), Pointers for inter function communication, pointers to pointers, compatibility, void pointer, null pointer.

Pointer Applications - Arrays and Pointers, Pointer Arithmetic and arrays, passing an array to a function.

Memory allocation functions – malloc(), calloc(), realloc(), free().

Array of pointers, pointers to functions, C program examples.

TEXT BOOKS:

1. Computer Science: A Structured Programming Approach Using C, B.A.Forouzan and R.F. Gilberg, Third Edition, Cengage Learning.

REFERENCE BOOKS:

1. Raptor-A flow charting Tool <http://raptor.martincarlisle.com>
2. The C Programming Language, B.W. Kernighan and Dennis M.Ritchie, PHI.
3. Schaum's Outline of Programming with C, Byron Gottfried, McGraw-Hill.
4. Programming in C. P. Dey and M Ghosh , Oxford University Press.
5. Problem Solving and Program Design in C, J.R. Hanly and E.B. Koffman, 7th Edition, Pearson education.

3. VISION OF THE DEPARTMENT – CSE

To produce globally competent and socially responsible computer science engineers contributing to the advancement of engineering and technology which involves creativity and innovation by providing excellent learning environment with world class facilities.

VISION OF THE DEPARTMENT -IT

The department of Information Technology endeavours to bring out technically competent, socially responsible technocrats through continuous improvement in teaching learning processes and innovative research practices.

VISION OF THE DEPARTMENT – ECE

To impart quality technical education in Electronics and Communication Engineering emphasizing analysis, design/synthesis and evaluation of hardware/ embedded software, using various Electronic Design Automation (EDA) tools with accent on creativity, innovation and research thereby producing competent engineers who can meet global challenges with societal commitment.

VISION OF THE DEPARTMENT – EEE

To provide excellent Electrical and Electronics education by building strong teaching and research environment

VISION OF THE DEPARTMENT – CE

To provide the competent and qualitative professionals with innovative ideas in Civil Engineering.

VISION OF THE DEPARTMENT – ME

The Mechanical Engineering department strives to be recognized globally for outstanding education and research, imparting quality education, churning well-qualified engineers, who are creative, innovative, and entrepreneurial, solving problems for societal development.

4. MISSION OF THE DEPARTMENT – CSE

1. To be a centre of excellence in instruction, innovation in research and scholarship, and service to the stake holders, the profession, and the public.
2. To prepare graduates to enter a rapidly changing field as a competent computer science engineer.
3. To prepare graduate capable in all phases of software development, possess a firm understanding of hardware technologies, have the strong mathematical background necessary for scientific computing, and be sufficiently well versed in general theory to allow growth within the discipline as it advances.
4. To prepare graduates to assume leadership roles by possessing good communication skills, the ability to work effectively as team members, and an appreciation for their social and ethical responsibility in a global setting.

MISSION OF THE DEPARTMENT –IT

1. Inculcate into the students, the technical and problem-solving skills to ensure their success in their chosen profession.
2. Impart the essential skills like teamwork, lifelong learning etc. to the students which make them globally acceptable technocrats.
3. Facilitate the students with strong fundamentals in basic sciences, mathematics and Information Technology areas to keep pace with the growing challenges in field.
4. Enrich the faculty with knowledge in the frontiers of the Information Technology area.

MISSION OF THE DEPARTMENT – ECE

1. To impart quality education in fundamentals of basic sciences, mathematics, electronics and communication engineering through innovative teaching-learning processes.
 2. To facilitate Graduates define, design, and solve engineering problems in the field of Electronics and Communication Engineering using various Electronic Design Automation (EDA) tools.
 3. To encourage research culture among faculty and students thereby facilitating them to be creative and innovative through constant interaction with R & D organizations
-

and Industry.

4. To inculcate teamwork, imbibe leadership qualities, professional ethics and social responsibilities in students and faculty.

MISSION OF THE DEPARTMENT – EEE

1. To offer high quality graduate program in Electrical and Electronics education and to prepare students for professional career or higher studies.
2. The department promotes excellence in teaching, research, collaborative activities and positive contributions to society

MISSION OF THE DEPARTMENT – CE

To offer the best quality education in under graduation and post-graduation, research guidance, professional consultancy and manpower training as well as leadership in civil engineering.

MISSION OF THE DEPARTMENT – ME

1. Imparting quality education to students to enhance their skills and make them globally competitive.
2. Prepare graduates to engage in life-long learning, possess intellectual capabilities, serving society with a strong commitment to their profession, meeting technical challenges and exhibiting ethical responsibility for societal development.
3. Conduct quality research, create opportunities for students and faculty to showcase their talent, disseminate knowledge, and promote community development, leading to peace and harmony in society.

PEO's, PO's & PSO's

PROGRAM EDUCATIONAL OBJECTIVES (PEO's)

CSE & CSE (AI&ML,DS,CS,IOT)

1. To provide graduates with a good foundation in mathematics, sciences and engineering fundamentals required to solve engineering problems that will facilitate them to find employment in industry and / or to pursue postgraduate studies with an appreciation for lifelong learning.

2. To provide graduates with analytical and problem-solving skills to design algorithms, other hardware / software systems, and inculcate professional ethics, inter-personal skills to work in a multi-cultural team.
3. To facilitate graduates to get familiarized with the art software / hardware tools, imbining creativity and innovation that would enable them to develop cutting-edge technologies of multi-disciplinary nature for societal development.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's) – IT

1. Exhibit sound knowledge in the fundamentals of Information Technology and apply practical experience with programming techniques to solve real world problems.
2. To inculcate professional behavior with strong ethical values, leadership qualities, innovative thinking and analytical abilities into the student.
3. To empower the student with the qualities of effective communication, teamwork, continuous learning attitude, leadership and proficiency in cutting edge technologies needed for a successful Information Technology professional.
4. Imbibe sound knowledge in mathematics, basic sciences, first principles to form a strong base for the student to keep up with the growing challenges in the field of Information Technology.

PROGRAM EDUCATIONAL OBJECTIVES - ECE

1. To prepare students with excellent comprehension of basic sciences, mathematics and engineering subjects facilitating them to gain employment or pursue postgraduate studies with an appreciation for lifelong learning.
2. To train students with problem solving capabilities such as analysis and design with adequate practical skills that are Program Specific wherein they demonstrate creativity and innovation that would enable them to develop state of the art equipment and technologies of multidisciplinary nature for societal development.
3. To inculcate positive attitude, professional ethics, effective communication and interpersonal skills which would facilitate them to succeed in the chosen profession exhibiting creativity and innovation through research and development both as team member and as well as leader.

PROGRAM EDUCATIONAL OBJECTIVES – EEE

1. To prepare students with excellent comprehension of mathematics, Basic Sciences and Engineering subjects facilitating them to find gainful employment or pursue post graduate program with an appreciation for lifelong learning.
2. To inculcate problem solving capabilities in students with analysis, design and practical skills that are Program Specific which would facilitate them to exhibit creativity and innovation that would enable them to develop modern equipment with emerging technologies of multidisciplinary nature for societal development.
3. To inculcate positive attitude, professional ethics, effective communication and interpersonal skills which would facilitate them to succeed in the chosen profession through research and development both as team member and as well as leader.

PROGRAM EDUCATIONAL OBJECTIVES – CE

1. Provide a strong foundation in Mathematics, Basic Sciences and Engineering fundamentals to the students, enabling them to excel in the various careers in Civil Engineering.
2. Impart necessary theoretical and practical background in Civil Engineering to the students, so that they can effectively compete with their contemporaries in the National / International level.
3. Motivate and prepare the Graduates to pursue higher studies, Research and Development, thus contributing to the ever-increasing academic demands of the country.
4. Enrich the students with strong communication and interpersonal skills, broad knowledge and an understanding of multicultural and global perspectives, to work effectively in multidisciplinary teams, both as leaders and team members.

PROGRAM EDUCATIONAL OBJECTIVES – ME

1. To prepare students with strong fundamental knowledge in basic sciences mathematics, engineering courses which would facilitate them find gainful employment with a sense of appreciation to pursue life-long learning for professional development.

2. To inculcate problem solving skills in students, imbibing creativity and innovation which would enable them to develop modern machinery involving cutting edge technologies of multidisciplinary nature for societal development.
3. To develop critical thinking with an aptitude to conduct research and development, instill professional ethics, develop effective communication and interpersonal skills with positive attitude to contribute significantly towards their chosen profession, thereby supporting community development.

PROGRAM OUTCOMES (Common to all branches)

Engineering Graduates would be able to:

PO 1:Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2: 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.

PO 3: 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.

PO 4: 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.

PO 5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO 6: 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.

PO 7: 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.

PO 8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10: 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.

PO 11: 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.

PO 12: 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..

PROGRAM SPECIFIC OUTCOMES (PSO's) - (CSE)

PSO 1: Demonstrate competency in Programming and problem-solving skills and apply these skills in solving real world problems

PSO 2: Select appropriate programming languages, Data structures and algorithms in combination with modern technologies and tools, apply them in developing creative and innovative solutions

PSO 3: Demonstrate adequate knowledge in emerging technologies

PROGRAM SPECIFIC OUTCOMES (PSO's) - (CSE-AI&ML)

PSO 1: Demonstrate competency in Programming and problem-solving skills and apply those skills in solving computing problems

PSO 2: Select appropriate programming languages, Data structures and algorithms in combination with modern technologies and apply them in developing innovative solutions

PSO 3: Demonstrate adequate knowledge in the concepts and techniques of artificial intelligence and machine learning, apply them in developing intelligent systems to solve real world problems

PROGRAM SPECIFIC OUTCOMES (PSO's) - (CSE-DS)

PSO 1: Demonstrate competency in Programming and problem-solving skills and apply those skills in solving computing problems.

PSO 2: Select appropriate programming languages, Data structures and algorithms in combination with modern technologies and apply them in developing innovative solutions.

PSO 3: Apply techniques of data modelling, analysis and visualization which include statistical techniques to solve real world problems delivering actionable insights for decision making.

PROGRAM SPECIFIC OUTCOMES (PSO's) - (CSE-CS)

PSO 1: Demonstrate competency in Programming and problem-solving skills and apply those skills in solving computing problems.

PSO 2: Select appropriate programming languages, Data structures and algorithms in combination with modern technologies and apply them in developing innovative solutions.

PSO 3: Apply cryptographic algorithms for ensuring cyber security as per cyber laws, demonstrate awareness of all the security related issues and employ state of the art technologies to protect the digital assets of an organization.

PROGRAM SPECIFIC OUTCOMES (PSO's) - (CSE-IoT)

PSO 1: Demonstrate competency in Programming and problem-solving skills and apply those skills in solving computing problems.

PSO 2: Select appropriate programming languages, Data structures and algorithms in combination with modern technologies and apply them in developing innovative solutions.

PSO 3: Demonstrate an ability in using IoT devices and protocols to develop IoT based solutions for real world problems.

PROGRAM SPECIFIC OUTCOMES (PSO's) –IT

PSO1: To inculcate algorithmic thinking and problem-solving skills, applying different programming paradigms.

PSO2: Develop an ability to design and implement various processes/methodologies/practices employed in design, validation, testing and maintenance of software products

PROGRAM SPECIFIC OUTCOMES (PSO's) - ECE

PSO 1: An ability to design an Electronic and Communication Engineering system, component, or process and conduct experiments, analyze, interpret data and prepare a report with conclusions to meet desired needs within the realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability and sustainability.

PSO 2: An ability to use modern Electronic Design Automation (EDA) tools, software and electronic equipment to analyze, synthesize and evaluate Electronics and Communication Engineering systems for multidisciplinary tasks.

PROGRAM SPECIFIC OUTCOMES (PSO's) - EEE

PSO1: An ability to simulate and determine the parameters like voltage profile and current ratings of transmission lines in Power Systems.

PSO2: An ability to understand and determine the performance of electrical machines namely, speed, torque, efficiency etc.

PSO3: An ability to apply electrical engineering and management principles to Power Projects

PROGRAM SPECIFIC OUTCOMES (PSO's) - CE

PSO 1: Apply knowledge in core areas of Civil Engineering such as Structural, Geotechnical, Water Resources, Transportation and Environmental Engineering to Civil Engineering practice

PSO 2: Utilize Civil Engineering principles that are appropriate to produce detailed drawings, design reports, quantity and cost estimates, specifications, contracts and other documents appropriate for the design, construction, operations and maintenance of Civil Engineering projects.

PSO 3: Shall interact and collaborate with stakeholders; execute quality construction works applying Civil Engineering tools namely, Total Station, Global Positioning System (GPS), ArcGIS, AutoCAD, STAAD and other necessary tools.

PROGRAM SPECIFIC OUTCOMES (PSO's) - ME

PSO 1: Apply Continuity, Energy and Momentum equations to mechanical systems, design and perform experiments in all fields of mechanical engineering

PSO 2: Able to analyze, design and develop/model mechanical and its allied systems using software tools such as AUTOCAD, ANSYS, Creo etc.

PSO 3: Able to design layouts for process and manufacturing industry taking into consideration optimization of resources for effective operation and maintenance.

COURSE OBJECTIVES AND OUTCOMES

Course Objectives

Develop ability to

1. Solve problems by developing flowcharts using Raptor tool.
2. Understand the concepts of variables, constants, basic data types and input and output statements in C programming language.
3. Understand the use of sequential, selection and repetitive statements in algorithms implemented using C programming language.
4. Understand structured design by implementing programs with functions to solve complex problems.
5. Understand the concepts related to arrays and pointers along with dynamic memory allocation using C programming language.

Course Outcomes (COs):

After completion of the course, student would be able to

- CO1. Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.
- CO2. Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.
- CO3. Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.
- CO4. Write C programs using 1D and 2D arrays.
- CO5. Write C programs using pointers and also with dynamic memory allocation.

7. Brief notes on the importance of the course and how it fits into the curriculum

a. What role does this course play within the Program?

- This course strengthens logical thinking and logical reasoning of the students.
- This is a fundamental course which helps in understanding the subsequent courses, namely, Computer programming – II, Java, Data structures etc.

b. How is the course unique or different from other courses of the Program?

- This is the only course wherein students learn the basic concepts of computer programming language, logic design, which is used in majority of the future courses and their applications.
- This course is fundamental to any engineering graduate.

c. What essential knowledge or skills should they gain from this experience?

- Students will acquire the capabilities to write and understand algorithms, flow charts. These can be used to write programs in any computer programming language.
- Students acquire capabilities of sequential, branching and looping techniques used in programming languages apart from basics. These techniques are the basis of any further engineering courses.

d. What knowledge or skills from this course will students need to have mastered to perform well in future classes or later (Higher Education / Jobs)?

- All the design, logical thinking and basics of writing computer programs for future research and solutions to complex computer related problems.

e. Why is this course important for students to take?

- In order to understand the basics of logical thinking, logical reasoning, virtual thinking of solutions to complex computer related problems, this course serves as a fundamental, to all branches of engineering.

f. What is/are the prerequisite(s) for this course?

- Basic Mathematical Knowledge and an inclination to think logically.

g. When students complete this course, what do they need know or be able to do?

- Able to design, analyze, evaluate and implement algorithms and flowcharts using RAPTOR Tool.
- Able to design, analyze, evaluate and implement basic programs using “C” Language.

h. Is there specific knowledge that the students will need to know in the future?

- In future, students have to apply these concepts in the design of systems that use programming languages.

i. Are there certain practical or professional skills that students will need to apply in the future?

- YES. Most of the mini and major projects are generally based on computer programming languages and computer-based software tools.

j. Five years from now, what do you hope students will remember from this course?

- Analytical thinking and logical reasoning and techniques employed in the analysis and design of solutions to given logical problem statements.

k. What is it about this course that makes it unique or special?

- It is the only fundamental course that facilitates students in the attainment of all levels of Bloom's taxonomy.

l. Why does the program offer this course?

- This is the basic course for any engineering field.
- This course is a prerequisite for Programming for Problem Solving-II, Computational Mathematics Lab, Data structures etc.,

m. Why can't this course be "covered" as a sub-section of another course?

- It is not possible as it is the first and basic course offered in engineering branches.
- This course is not covered in any of the earlier studies (Intermediate or previous classes)

n. What unique contributions to students' learning experience does this course make?

- It helps in building logical thinking and analysis which is key to any engineering branch.
- It helps in executing mini and major projects that are generally based on computer programming languages and computer-based software tools, during the later years of the program.

o. What is the value of taking this course? How exactly does it enrich the program?

- This course plays a vital role in design and development of solutions to various problems in any discipline of Engineering which is so essential and useful to the development of society and this course also helps for the student's professional career growth in terms of professional career.

This course makes significant contributions to the following program outcomes:

- an ability to apply knowledge of mathematics, science, and engineering,
- an ability to design and conduct experiments, as well as to analyze and interpret data,
- an ability to design a system, component, or process to meet desired needs within realistic constraints
- an ability to identify, formulate, and solve engineering problems,

- an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice.

p. What are the major career options that require this course?

Programming for Problem Solving-II, Computational Mathematics Lab, Data Structures and any computer related courses that are advanced courses for which Programming for Problem Solving-I course is a pre-requisite and is key for many careers in engineering and technology. Specific occupations that employ this course include:

- Any field that relates to computer programming.
- Embedded systems designer
- Digital systems designer/engineer
- CAD/CAM
- Software tools used in Architecture and Civil Engineering fields

8. PREREQUISITES IF ANY

- Mathematics of 10+2 helps in understanding the logic required.

9. INSTRUCTIONAL LEARNING OUTCOMES

Upon completing this course, it is expected that a student will be able to do the following:

- Program development skills:** Draw a flowchart and write an algorithm for a given problem.
- Coding in C:** Write a program in C language for a given problem, a flowchart, or an algorithm.
- Modular approach:** Can divide a program into several modules and make them work together

10. COURSE MAPPING WITH PEO's AND PO's

Mapping of Course with Programme Educational Objectives

S.No	Course component	Code	Course	Semester/Year	PEO 1	PEO 2	PEO 3
1	Computer Science	20CS1101	Programming for Problem Solving- I	I Year I Sem	√	√	

Mapping of Course outcomes with Programme outcomes: CSE

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	Computer Science Engineering
Programming for Problem Solving –I																
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2	3	2	-	
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2	3	2	-	
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2	3	2	-	
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2	3	2	-	

CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			3	3	2	-	
---	---	---	--	---	---	--	--	---	---	--	--	---	---	---	---	--

Mapping of Course outcomes with Programme outcomes: (CSE-AI&ML)

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	
Programming for Problem Solving -I																
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2	3	2	-	
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2	3	2	-	
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2	3	2	-	
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2	3	2	-	
CO 5: <u>Write C programs using pointers</u> and also with dynamic memory allocation.	3	2		2	2			2	2			2	3	2	-	

Computer Science Engineering (CSE-AI&ML)

Mapping of Course outcomes with Programme outcomes: (CSE-DS)

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	Computer Science Engineering(CSE-DS)
Programming for Problem Solving -I																
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2	3	2	-	
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2	3	2	-	
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2	3	2	-	
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2	3	2	-	
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2	3	2	-	

Mapping of Course outcomes with Programme outcomes: (CSE-CS)

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	Computer Science Engineering(CSE-CS)
Programming for Problem Solving -I																
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2	3	2	-	
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2	3	2	-	
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2	3	2	-	
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2	3	2	-	
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2	3	2	-	

Mapping of Course outcomes with Programme outcomes: (CSE-IoT)

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	
	Programming for Problem Solving -I															Computer Science Engineering (CSE-IoT)
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2	3	2	-	
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2	3	2		
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2	3	2	-	
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2	3	2	-	
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2	3	2	-	

Mapping of Course outcomes with Programme outcomes: IT

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2	Information Technology
Programming for Problem Solving –I															
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2	2	2	
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2	2	2	
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2	2	2	
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2	2	2	
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2	2	2	

Mapping of Lab Course outcomes with Programme outcomes- ECE:

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2
Programming for Problem Solving –I														
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2	2	
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2	2	
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2	2	
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2	2	
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2	2	

Mapping of Lab Course outcomes with Programme outcomes- EEE:

PO's	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
Programming for Problem Solving –I															
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2			
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2			
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2			
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2			
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2			

Mapping of Lab Course outcomes with Programme outcomes- CE:

Pos	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
Programming for Problem Solving –I															
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2			
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2			
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2			
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2			
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2			

Mapping of Lab Course outcomes with Programme outcomes- ME:

Pos	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
Programming for Problem Solving –I															
CO 1: Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool. Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program.	3	2		2	2			2	2			2			
CO 2: Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.	3	2		2	2			2	2			2			
CO 3: Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.	3	2		2	2			2	2			2			
CO 4: Write C programs using 1D and 2D arrays.	3	2		2	2			2	2			2			
CO 5: Write C programs using pointers and also with dynamic memory allocation.	3	2		2	2			2	2			2			

Class Time Table

Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-A	Acad. Year-2020-21 wef: 23-11-2020		
Class Incharge: Dr. J.V.Madhuri				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	M-I	AP	EC		PPS LAB	
Tuesday	M-I	AP	PPS		ENG LAB	
Wednesday	M-I	AP	ENG		EC LAB	
Thursday	PPS	ENG	EC		M-I	—
Friday	PPS	ENG	EC		—	—
Saturday	—	—	—		—	—
S.No	Subject				Faculty Name	
1	English(ENG)			Prof. G. Karuna Kumari		3
2	Applied Physics(AP)			V. Manjula		3
3	Basic Engineering Mathematics(M-I)			S.Lalitha, Dr. N. Subhadra		4
4	Engineering Chemistry(EC)			Dr. J. V. Madhuri		3
5	Programming for Problem Solving(PPS)			Prof. V. Madhusudan Rao		3
6	Programming for Problem Solving Lab(PPS Lab)			Prof. V. Madhusudan Rao		2
7	Engineering Chemistry Lab(EC Lab)			Dr. J. V. Madhuri, Dr. Anurag Gautam		2
8	English Lab(Eng Lab)			Prof. G. Karuna Kumari		2
9	Mentoring			Dr. J.V. Madhuri, S. Lalitha, J. Bhargavi lakshmi		
NOTE: * Represents Tutorial Class						
Date: TT Coord: HOD: Dean Academics: Principal:						

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-B		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Dr. R. Sanjeev				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EC	M-I	PPS		ENG	—
Tuesday	ENG	M-I	PPS		EC LAB	
Wednesday	EC	M-I	AP		PPS LAB	

Thursday	EC	M-I	AP		ENG LAB	
Friday	ENG	PPS	AP			
Saturday	—	—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			Dr. A. Umadevi		3
2	Applied Physics(AP)			V. Manjula		3
3	Basic Engineering Mathematics(M-I)			Dr Triveni, Dr. N. Subhadra		4
4	Engineering Chemistry(EC)			Dr. R. Sanjeev		3
5	Programming for Problem Solving(PPS)			J. Umamahesh		3
6	Programming for Problem Solving Lab(PPS Lab)			J. Umamahesh		2
7	Engineering Chemistry Lab(EC Lab)			Dr. R. Sanjeev, K. Swarupa		2
8	English Lab(Eng Lab)			Dr. A. Umadevi		2
9	Mentoring			Dr. R. Sanjeev, Dr. A. Umadevi, K. Swarupa		

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Wednesday						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-C		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Dr. K.Shashikala			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	PPS	ENG			EC LAB	
Tuesday	EC	M-I	AP		PPS LAB	
Wednesday	EC	ENG	AP		M-I	—
Thursday	PPS	M-I	EC		ENG LAB	
Friday	PPS	ENG	AP		M-I	—
Saturday	—	—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			Dr.B. Nagamani		3
2	Applied Physics(AP)			C. Kalyani		3
3	Basic Engineering Mathematics(M-I)			M.P. Molimol, S. Lalitha		4

4	Engineering Chemistry(EC)	Dr. K. Shashikala	3
5	Programming for Problem Solving(PPS)	J. Umamahesh	3
6	Programming for Problem Solving Lab(PPS Lab)	J. Umamahesh	2
7	Engineering Chemistry Lab(EC Lab)	Dr. K. Shashikala, K. Swarupa	2
8	English Lab(Eng Lab)	Dr.B. Nagamani	2
9	Mentoring	Dr. K. Shashikala, Dr. B. Nagamani, J. Umamahesh	

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-D		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: G. Latha Suhasini			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EC	AP	ENG		PPS LAB	
Tuesday	EC	AP	M-I		ENG LAB	
Wednesday	EC	AP	PPS		M-I	—
Thursday	M-I	ENG	PPS		—	—
Friday	M-I	ENG	PPS		EC LAB	
Saturday	—	—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			G. Latha Suhasini		3
2	Applied Physics(AP)			Dr. J. Shankar		3
3	Basic Engineering Mathematics(M-I)			S. Lalitha, G. Padma		4
4	Engineering Chemistry(EC)			Dr. P. Sreedhar		3
5	Programming for Problem Solving(PPS)			M. Ravinder		3
6	Programming for Problem Solving Lab(PPS Lab)			M. Ravinder		2
7	Engineering Chemistry Lab(EC Lab)			Dr. P. Sreedhar, Dr. K. Shashikala		2
8	English Lab(Eng Lab)			G. Latha Suhasini		2
9	Mentoring			G. Latha Suhasini, M. Ravinder, G. Padma		

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE(DS)		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Y. Anil				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	M-I	EC	AP			
Tuesday	PPS	EC	ENG		M-I	
Wednesday	M-I	EC	ENG		PPS LAB	
Thursday	ENG	PPS	AP		EC LAB	
Friday	M-I	PPS	AP		ENG LAB	
Saturday						
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			Y. Anil		3
2	Applied Physics(AP)			Dr. J. Anjaiah		3
3	Basic Engineering Mathematics(M-I)			Dr. G. Mahesh, N. Nagi Reddy		4
4	Engineering Chemistry(EC)			Dr. J. V. Madhuri		3
5	Programming for Problem Solving(PPS)			Ms. Deepa Panse		3
6	Programming for Problem Solving Lab(PPS Lab)			Ms. Deepa Panse		2
7	Engineering Chemistry Lab(EC Lab)			Dr. J. V. Madhuri, Dr. P. Sreedhar		2
8	English Lab(Eng Lab)			Y. Anil		2
9	Mentoring			Y. Anil, Dr. G. Mahesh, Ms. Deepa Panse		

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE (CS)		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: B. Vanaja Rani				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	PPS	EC	M-I		ENG LAB	
Tuesday	PPS	EC	AP			
Wednesday	M-I	PPS	ENG		EC LAB	
Thursday	M-I	AP	ENG		EC	

Friday	M-I	AP	ENG		PPS LAB
Saturday					
S.No	Subject			Faculty Name	No. Of Hours/Week
1	English(ENG)			B. Vanaja Rani	3
2	Applied Physics(AP)			A. Shiva Kumar	3
3	Basic Engineering Mathematics(M-I)			Dr. K. Venkateswarlu, N. Nagi Reddy	4
4	Engineering Chemistry(EC)			Dr. K. Shashikala	3
5	Programming for Problem Solving(PPS)			M. Ajay	3
6	Programming for Problem Solving Lab(PPS Lab)			M. Ajay	2
7	Engineering Chemistry Lab(EC Lab)			Dr. K. Shashikala, J. Bhargavi Lakshmi	2
8	English Lab(Eng Lab)			B. Vanaja Rani	2
9	Mentoring			B. Vanaja Rani, M. Ajay, A. Shiva Kumar	

NOTE: * Represents Tutorial Class

Date: TT Coord: HOD: Dean Academics: Principal:

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE (AI & ML)		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Dr. P.Sreedhar				Version -I		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	AP	PPS	ENG		EC LAB	
Tuesday	ENG	PPS	EC		M-I	
Wednesday	AP	M-I	EC		ENG LAB	
Thursday	AP	M-I	EC		PPS LAB	
Friday	ENG	M-I	PPS			
Saturday						
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			Dr. M. Upendra		3
2	Applied Physics(AP)			Dr. SK. Mohammad Ali		3
3	Basic Engineering Mathematics(M-I)			Dr. K.Venkateswarlu,M.P. Molimol		4

4	Engineering Chemistry(EC)	Dr. P.Sreedhar	3
5	Programming for Problem Solving(PPS)	Ms.Deepa Panse	3
6	Programming for Problem Solving Lab(PPS Lab)	Ms.Deepa Panse	2
7	Engineering Chemistry Lab(EC Lab)	Dr. P.Sreedhar, J. Bhargavi Lakshmi	2
8	English Lab(Eng Lab)	Dr. M. Upendra	2
9	Mentoring	Dr. P.Sreedhar, Dr. Mohammad Ali, Dr. M. Upendra	

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE(IOT)		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: P. Sailaja			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	ENG	AP	M-I		—	—
Tuesday	EC	PPS	M-I		ENG	—
Wednesday	AP	EC	M-I		PPS LAB	
Thursday	AP	ENG	PPS		EC LAB	
Friday	EC	PPS	M-I		ENG LAB	
Saturday	—	—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			Dr. M. Upendra		3
2	Applied Physics(AP)			Dr. B. Mamatha		3
3	Basic Engineering Mathematics(M-I)			P. Sailaja, S. Lalitha		4
4	Engineering Chemistry(EC)			Dr. Anurag Gautam		3
5	Programming for Problem Solving(PPS)			M. Ajay		3
6	Programming for Problem Solving Lab(PPS Lab)			M. Ajay		2
7	Engineering Chemistry Lab(EC Lab)			Dr. Anurag Gautam , M. Murali		2

8	English Lab(Eng Lab)	Dr. M. Upendra	2
9	Mentoring	P. Sailaja, Dr. Anurag Gautam	

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: IT		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: V. Ramaiah Chary				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EC	M-I	PPS		ENG LAB	
Tuesday	AP	ENG	PPS		EC LAB	
Wednesday	ENG	EWS LAB			M-I	PPS
Thursday	EC	M-I	AP		PPS LAB	
Friday	ENG	EC	AP		M-I	—
Saturday	—	—	—		—	—
S.No	Subject				Faculty Name	
1	English(ENG)			V. Ramaiah Chary		3
2	Applied Physics(AP)			Dr. P. Raju		3
3	Basic Engineering Mathematics(M-I)			Dr. Mahesh, N. NagiReddy		4
4	Engineering Chemistry(EC)			Dr.Anurag Gautam		3
5	Programming for Problem Solving(PPS)			Ms.Fathima		2
6	Programming for Problem Solving Lab(PPS LAB)			Ms.Fathima		2
7	Engineering Chemistry Lab(EC LAB)			Dr.Anurag Gautam, J. Bhargavi Lakshmi		2
8	English Lab(Eng LAB)			V. Ramaiah Chary		2
9	Engineering workshop(EWS LAB)			R.S. Mahipal Reddy, J. Nithin		2
10	Mentoring			V. Ramaiah Charv, Dr. P. Raju, Ms. Fathima		

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-A		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Dr. J.V.Madhuri			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	M-I	AP	EC		PPS LAB	
Tuesday	M-I	AP	PPS		ENG LAB	
Wednesday	M-I	AP	ENG		EC LAB	
Thursday	PPS	ENG	EC		M-I	—
Friday	PPS	ENG	EC		—	—
Saturday	—	—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			Prof. G. Karuna Kumari		3
2	Applied Physics(AP)			V. Manjula		3
3	Basic Engineering Mathematics(M-I)			S.Lalitha, Dr. N. Subhadra		4
4	Engineering Chemistry(EC)			Dr. J. V. Madhuri		3
5	Programming for Problem Solving(PPS)			Prof. V. Madhusudan Rao		3
6	Programming for Problem Solving Lab(PPS Lab)			Prof. V. Madhusudan Rao		2
7	Engineering Chemistry Lab(EC Lab)			Dr. J. V. Madhuri, Dr. Anurag Gautam		2
8	English Lab(Eng Lab)			Prof. G. Karuna Kumari		2
9	Mentoring			Dr. J.V. Madhuri, S. Lalitha, J. Bhargavi lakshmi		

NOTE: * Represents Tutorial Class

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-B		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Dr. R. Sanjeev			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EC	M-I	PPS		ENG	—
Tuesday	ENG	M-I	PPS		EC LAB	
Wednesday	EC	M-I	AP		PPS LAB	
Thursday	EC	M-I	AP		ENG LAB	
Friday	ENG	PPS	AP		—	—

Saturday					
S.No	Subject	Faculty Name	No. Of Hours/Week		
1	English(ENG)	Dr. A. Umadevi	3		
2	Applied Physics(AP)	V. Manjula	3		
3	Basic Engineering Mathematics(M-I)	Dr Triveni, Dr. N. Subhadra	4		
4	Engineering Chemistry(EC)	Dr. R. Sanjeev	3		
5	Programming for Problem Solving(PPS)	J. Umamahesh	3		
6	Programming for Problem Solving Lab(PPS Lab)	J. Umamahesh	2		
7	Engineering Chemistry Lab(EC Lab)	Dr. R. Sanjeev, K. Swarupa	2		
8	English Lab(Eng Lab)	Dr. A. Umadevi	2		
9	Mentoring	Dr. R. Sanjeev, Dr. A. Umadevi, K. Swarupa			
NOTE: * Represents Tutorial Class					
Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____					

Wednesday						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-C		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Dr. K.Shashikala			Version -I			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3		4	5
Monday	PPS	ENG		LUNCH	EC LAB	
Tuesday	EC	M-I	AP		PPS LAB	
Wednesday	EC	ENG	AP		M-I	
Thursday	PPS	M-I	EC		ENG LAB	
Friday	PPS	ENG	AP		M-I	
Saturday						
Sunday						
S.No	Subject	Faculty Name	No. Of Hours/Week			
1	English(ENG)	Dr.B. Nagamani	3			
2	Applied Physics(AP)	C. Kalyani	3			
3	Basic Engineering Mathematics(M-I)	M.P. Molimol, S. Lalitha	4			
4	Engineering Chemistry(EC)	Dr. K. Shashikala	3			
5	Programming for Problem Solving(PPS)	J. Umamahesh	3			
6	Programming for Problem Solving Lab(PPS Lab)	J. Umamahesh	2			

7	Engineering Chemistry Lab(EC Lab)	Dr. K. Shashikala, K. Swarupa	2
8	English Lab(Eng Lab)	Dr.B. Nagamani	2
9	Mentoring	Dr. K.Shashikala,Dr. B. Nagamani, J. Umamahesh	

NOTE: * Represents Tutorial Class

Date: TT Coord: HOD: Dean Academics: Principal:

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE-D		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: G. Latha Suhasini			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EC	AP	ENG		PPS LAB	
Tuesday	EC	AP	M-I		ENG LAB	
Wednesday	EC	AP	PPS		M-I	—
Thursday	M-I	ENG	PPS		—	—
Friday	M-I	ENG	PPS		EC LAB	
Saturday	—	—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			G. Latha Suhasini		3
2	Applied Physics(AP)			Dr. J. Shankar		3
3	Basic Engineering Mathematics(M-I)			S. Lalitha, G. Padma		4
4	Engineering Chemistry(EC)			Dr. P. Sreedhar		3
5	Programming for Problem Solving(PPS)			M. Ravinder		3
6	Programming for Problem Solving Lab(PPS Lab)			M. Ravinder		2
7	Engineering Chemistry Lab(EC Lab)			Dr. P. Sreedhar, Dr. K. Shashikala		2
8	English Lab(Eng Lab)			G. Latha Suhasini		2
9	Mentoring			G. Latha Suhasini, M. Ravinder, G. Padma		

NOTE: * Represents Tutorial Class

Date: TT Coord: HOD: Dean Academics: Principal:

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester				Section: CSE(DS)		
Acad. Year-2020-21				wef: 23-11-2020		
Class Incharge: Y. Anil				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	M-I	EC	AP			
Tuesday	PPS	EC	ENG		M-I	
Wednesday	M-I	EC	ENG		PPS LAB	
Thursday	ENG	PPS	AP		EC LAB	
Friday	M-I	PPS	AP		ENG LAB	
Saturday						
S.No	Subject				Faculty Name	
1	English(ENG)			Y. Anil		3
2	Applied Physics(AP)			Dr. J. Anjaiah		3
3	Basic Engineering Mathematics(M-I)			Dr. G. Mahesh, N. Nagi Reddy		4
4	Engineering Chemistry(EC)			Dr. J. V. Madhuri		3
5	Programming for Problem Solving(PPS)			Ms. Deepa Panse		3
6	Programming for Problem Solving Lab(PPS Lab)			Ms. Deepa Panse		2
7	Engineering Chemistry Lab(EC Lab)			Dr. J. V. Madhuri, Dr.P. Sreedhar		2
8	English Lab(Eng Lab)			Y. Anil		2
9	Mentoring			Y. Anil, Dr. G. Mahesh, Ms. Deepa Panse		
NOTE: * Represents Tutorial Class						
Date: TT Coord: HOD: Dean Academics: Principal:						

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE (CS)		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: B. Vanaja Rani			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	PPS	EC	M-I		ENG LAB	
Tuesday	PPS	EC	AP			
Wednesday	M-I	PPS	ENG		EC LAB	
Thursday	M-I	AP	ENG		EC	

Friday	M-I	AP	ENG		PPS LAB
Saturday					
S.No	Subject			Faculty Name	No. Of Hours/Week
1	English(ENG)			B. Vanaja Rani	3
2	Applied Physics(AP)			A. Shiva Kumar	3
3	Basic Engineering Mathematics(M-I)			Dr. K. Venkateswarlu, N. Nagi Reddy	4
4	Engineering Chemistry(EC)			Dr. K. Shashikala	3
5	Programming for Problem Solving(PPS)			M. Ajay	3
6	Programming for Problem Solving Lab(PPS Lab)			M. Ajay	2
7	Engineering Chemistry Lab(EC Lab)			Dr. K. Shashikala, J. Bhargavi Lakshmi	2
8	English Lab(Eng Lab)			B. Vanaja Rani	2
9	Mentoring			B. Vanaja Rani, M. Ajay, A. Shiva Kumar	

NOTE: * Represents Tutorial Class

Date: TT Coord: HOD: Dean Academics: Principal:

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE (AI & ML)		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: Dr. P.Sreedhar			Version -1			
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	AP	PPS	ENG		EC LAB	
Tuesday	ENG	PPS	EC		M-I	
Wednesday	AP	M-I	EC		ENG LAB	
Thursday	AP	M-I	EC		PPS LAB	
Friday	ENG	M-I	PPS			
Saturday						
S.No	Subject			Faculty Name		No. Of Hours/Week
1	English(ENG)			Dr. M. Upendra		3
2	Applied Physics(AP)			Dr. SK. Mohammad Ali		3
3	Basic Engineering Mathematics(M-I)			Dr. K.Venkateswarlu,M.P. Molimol		4
4	Engineering Chemistry(EC)			Dr. P.Sreedhar		3

5	Programming for Problem Solving(PPS)	Ms.Deepa Panse	3
6	Programming for Problem Solving Lab(PPS Lab)	Ms.Deepa Panse	2
7	Engineering Chemistry Lab(EC Lab)	Dr. P.Sreedhar, J. Bhargavi Lakshmi	2
8	English Lab(Eng Lab)	Dr. M. Upendra	2
9	Mentoring	Dr. P.Sreedhar, Dr. Mohammad Ali, Dr. M. Upendra	
NOTE: * Represents Tutorial Class			
Date: TT Coord:_____ HOD:_____ Dean Academics:_____ Principal:_____			

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CSE(IOT)		Acad. Year-2020-21 wef: 23-11-2020	
Class Incharge: P. Sailaja				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	ENG	AP	M-I			
Tuesday	EC	PPS	M-I		ENG	
Wednesday	AP	EC	M-I		PPS LAB	
Thursday	AP	ENG	PPS		EC LAB	
Friday	EC	PPS	M-I		ENG LAB	
Saturday						
S.No	Subject				Faculty Name	
1	English(ENG)			Dr. M. Upendra		3
2	Applied Physics(AP)			Dr. B. Mamatha		3
3	Basic Engineering Mathematics(M-I)			P. Sailaja, S. Lalitha		4
4	Engineering Chemistry(EC)			Dr. Anurag Gautam		3
5	Programming for Problem Solving(PPS)			M. Ajay		3
6	Programming for Problem Solving Lab(PPS Lab)			M. Ajay		2
7	Engineering Chemistry Lab(EC Lab)			Dr. Anurag Gautam , M. Murali		2
8	English Lab(Eng Lab)			Dr. M. Upendra		2
9	Mentoring			P. Sailaja, Dr. Anurag Gautam		
NOTE: * Represents Tutorial Class						

Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester				Section: IT		
Acad. Year-2020-21				wef: 23-11-2020		
Class Incharge: V. Ramaiah Chary				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EC	M-I	PPS		ENG LAB	
Tuesday	AP	ENG	PPS		EC LAB	
Wednesday	ENG	EWS LAB			M-I	PPS
Thursday	EC	M-I	AP		PPS LAB	
Friday	ENG	EC	AP		M-I	_
Saturday	_	_	_		_	_
S.No	Subject				Faculty Name	
1	English(ENG)			V. Ramaiah Chary		3
2	Applied Physics(AP)			Dr. P. Raju		3
3	Basic Engineering Mathematics(M-I)			Dr. Mahesh, N. NagiReddy		4
4	Engineering Chemistry(EC)			Dr.Anurag Gautam		3
5	Programming for Problem Solving(PPS)			Ms.Fathima		2
6	Programming for Problem Solving Lab(PPS LAB)			Ms.Fathima		2
7	Engineering Chemistry Lab(EC LAB)			Dr.Anurag Gautam, J. Bhargavi Lakshmi		2
8	English Lab(Eng LAB)			V. Ramaiah Chary		2
9	Engineering workshop(EWS LAB)			R.S. Mahipal Reddy, J. Nithin		2
10	Mentoring			V. Ramaiah Chary, Dr. P. Raju, Ms. Fathima		
NOTE: * Represents Tutorial Class						
Date: TT Coord: HOD: Dean Academics: Principal:						

Year/Sem/Sec: I-B. Tech-I Semester			Section: ECE-A	Acad. Year-2020-21 wef: 23-11-2020
Class Incharge: C. KALYANI				Version -1

Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	PPS	SSP	M-I		BEE LAB	
Tuesday	PPS	BEE	M-I		EWS LAB	
Wednesday	PPS	SSP	BEE		EG	
Thursday	M-I	SSP	BEE		PPS LAB	
Friday	M-I	EG(10.15-12.15)			SSP (1.00-2.00)	SSP LAB(2.00 - 4.00)
Saturday	—	—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	Solid State Physics(SSP)			C. Kalyani		4
2	Basic Engineering Mathematics(M-I)			Dr. N. Subhadra/Dr. Triveni		4
3	Engineering Graphics(EG)			P. Sudheer Rao		4
4	Programming for Problem Solving(PPS)			Dr. Puja Prasad		3
5	Basic Electrical Engineering(BEE)			M. Prashanth		3
6	Solid State Physics Lab(SSP LAB)			C. Kalyani, V. Manjula		2
7	Programming for Problem Solving Lab(PPS LAB)			Dr. Puja Prasad		2
8	Basic Electrical Engineering Lab(BEE LAB)			M. Prashanth, Mr.Gouse Basha		2
9	Engineering Workshop(EWS LAB)			R. Mahipal Reddy/J. Sumalatha		2
10	Mentors			C. Kalyani, Dr. Puja Prasad, V.Manjula		
Period						
Date: TT Coord: HOD: Dean Academics: Principal:						

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: ECE-B	Acad. Year-2020-21 wef: 23-11-2020		
Class Incharge: Dr. J. Shankar				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	BEE	M-I	SSP		BEE LAB	
Tuesday	SSP	M-I	PPS		EG	
Wednesday	BEE	M-I	SSP		PPS(1.00 -2.00)	PPS LAB(2.00-4.00)

Thursday	PPS	EG(10.15-12.15)			SSP LAB	
Friday	BEE	M-I	SSP		EWS LAB	
Saturday	-	-	-		-	-
S.No	Subject			Faculty Name		No. Of Hours/Week
1	Solid State Physics(SSP)			Dr. J. Shankar		4
2	Basic Engineering Mathematics(M-I)			Dr. N. Subhadra/ K. Nagaraju		4
3	Engineering Graphics(EG)			P. Sudheer Rao		4
4	Programming for Problem Solving(PPS)			Dr. Puja Prasad		3
5	Basic Electrical Engineering(BEE)			K. Ngaraju		3
6	Solid State Physics Lab(SSP LAB)			Dr. J. Shankar, A. Shiva Kumar		2
7	Programming for Problem Solving Lab(PPS LAB)			Dr. Puja Prasad		2
8	Basic Electrical Engineering Lab(BEE LAB)			A. Raghuramachandra, E. Himabindu		2
9	Engineering Workshop(EWS LAB)			R. Mahipal Reddy/J. Sumalatha		2
10	Mentoring			Dr. J. Shankar, K. Nagaraju, S. Rajesham		
Period						
Date: TT Coord: HOD: Dean Academics: Principal:						

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: ECE-C	Acad. Year-2020-21 wef: 23-11-2020		
Class Incharge: Dr.B. Mamatha				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EG		BEE		SSP LAB	
Tuesday	M-I	SSP	PPS		PPS LAB	
Wednesday	M-I	SSP	BEE		PPS(1.00 -2.00)	EWS LAB(2.00-4.00)
Thursday	M-I	SSP	BEE		EG	
Friday	SSP	M-I	PPS		BEE LAB	
Saturday						
S.No	Subject			Faculty Name		No. Of Hours/Week
1	Solid State Physics(SSP)			Dr.B. Mamatha		4

2	Basic Engineering Mathematics(M-I)	M. P.Molimol, N. NagiReddy	4
3	Engineering Graphics(EG)	K. Raju	4
4	Programming for Problem Solving(PPS)	S. Radha	3
5	Basic Electrical Engineering(BEE)	Md. Hafeezuddin	3
6	Solid State Physics Lab(SSP LAB)	Dr.B. Mamatha, Dr. SK. Mohammad Ali, C. Kalyani	2
7	Programming for Problem Solving Lab(PPS LAB)	S. Radha	2
8	Basic Electrical Engineering Lab(BEE LAB)	Md. Hafeezuddin, S. Purnachander Rao	2
9	Engineering Workshop(EWS LAB)	J. Sumalatha/M. Ravikumar	2
10	Mentoring	Dr.B. Mamatha, S. Radha, N. Nagi Reddy	
Period			
Date: TT Coord: _____ HOD: _____ Dean Academics: _____ Principal: _____			
Geethanjali College of Engineering & Technology			
Department of Freshman Engineering (Online Class Time Table)			
Year/Sem/Sec: I-B. Tech-I Semester		Section: ECE- D	Acad. Year-2020-21 wef: 23-11-2020
Class Incharge: M. P.Molimol		Version -1	
Time	09.00-10.00	10.15-11.15	11.30-12.30
Period	1	2	3
Monday	PPS	SSP	M-I
Tuesday	PPS	SSP	M-I
Wednesday	BEE	PPS	M-I
Thursday	BEE	EG	
Friday	BEE	SSP	M-I
Saturday			
S.No	Subject	Faculty Name	No. Of Hours/Week
1	Solid State Physics(SSP)	Dr.J. Anjaiah	4
2	Basic Engineering Mathematics(M-I)	M. P.Molimol/ Dr. Nuslin	4
3	Engineering Graphics(EG)	N. Rajender	4
4	Programming for Problem Solving(PPS)	S. Radha	3
5	Basic Electrical Engineering(BEE)	S. Hareesh reddy	3
6	Solid State Physics Lab(SSP LAB)	Dr.J. Anjaiah, V. Manjula	2
7	Programming for Problem Solving Lab(PPS LAB)	S. Radha	2

8	Basic Electrical Engineering Lab(BEE LAB)	Mr.Gouse Basha, V. Padmaja	2
9	Engineering Workshop(EWS LAB)	J. Sumalatha/M. Ravikumar	2
10	Mentoring	M. P.Molimol, Dr.J. Anjaiah, Dr. Nuslin	
Period			
Date: TT Coord:_____ HOD:_____ Dean Academics:_____ Principal:_____			

Year/Sem/Sec: I-B. Tech-I Semester			Section: EEE	Acad. Year-2020-21 wef: 23-11-2020		
Class Incharge: P. Mercy Kavitha				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	M-I	SSP	PPS		EWS LAB	
Tuesday	M-I	SSP	ENG		EG	
Wednesday	SSP	PPS	ENG		M-I(1.00-2.00)	SSP LAB(2.00-4.00)
Thursday	SSP	PPS	M-I		ENG LAB	
Friday	EG		ENG		PPS LAB	
Saturday	–	–	–		–	–
S.No	Subject				Faculty Name	
1	English(ENG)			P. Mercy Kavitha		3
2	Solid State Physics(SSP)			A. Shiva Kumar		4
3	Basic Engineering Mathematics(M-I)			Dr. K.Venkateswarlu, Dr. V.S. Triveni		4
4	Programming for Problem Solving(PPS)			Sudha Singaraju		3
5	Engineering Graphics(EG)			D.Ramchander		4
6	English Lab(Eng Lab)			P. Mercy Kavitha		2
7	Solid State Physics Lab(SSP LAB)			A. Shiva Kumar, S. Rajesham		2
8	Programming for Problem Solving Lab(PPS LAB)			Sudha Singaraju		2
9	Engineering Workshop(EWS LAB)			R. S. Mahipal Reddy J. Nithin		2
10	Mentors			P. Mercy Kavitha, Sudha Singaraju, Dr. K. Venkateswarlu		
NOTE: * Represents Tutorial Class						
Date: TT Coord: HOD: Dean Academics: Principal: _____						

Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: ME	Acad. Year-2020-21 wef: 23-11-2020		
Class Incharge: P. Sudheer Rao				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EM-1	M-I	EP		PPS LAB	
Tuesday	M-1	EG			EP	PPS
Wednesday	EM-1	EM-I	EP		EWS LAB	
Thursday	PPS	M-I	EP		EG	
Friday	PPS	EP LAB			M-I	EM-1
Saturday						
S.No	Subject				Faculty Name	
1	Engineering Physics(EP)			Dr. SK. Mohammad Ali		4
2	Basic Engineering Mathematics(M-I)			K. Nagaraju, M. P. Molimol		4
3	Programming for Problem Solving(PPS)			Sudha Singaraju		3
4	Engineering Mechanics-I(EM-I)			P. Sudheer Rao		4
5	Engineering Graphics(EG)			B. Anitha		4
6	Engineering Physics Lab(EP LAB)			Dr. SK. Mohammad Ali, Dr. B. Mamatha		2
7	Programming for Problem Solving Lab(PPS LAB)			Sudha Singaraju		2
8	Engineering Workshop(EWS LAB)			R. S. Mahipal Reddy , B.Bhaskar		2
9	Mentoring			P. Sudheer Rao, R. S. Mahipal Reddy , B.Bhaskar		
NOTE: * Represents Tutorial Class						
Date: TT Coord:_____ HOD:_____ Dean Academics:_____ Principal:_____						
Geethanjali College of Engineering & Technology						
Department of Freshman Engineering (Online Class Time Table)						
Year/Sem/Sec: I-B. Tech-I Semester			Section: CE	Acad. Year-2020-21 wef: 23-11-2020		
Class Incharge: G. Raju				Version -1		
Time	09.00-10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Period	1	2	3	LUNCH	4	5
Monday	EM-1	EM-1	M-I		EP LAB	
Tuesday	EM-1	EP	M-I		PPS LAB	
Wednesday	M-1	EP	PPS		EG	

Thursday	PPS	EG			EWS LAB	
Friday	M-1	EP	PPS		EM-1	EP
Saturday		—	—		—	—
S.No	Subject			Faculty Name		No. Of Hours/Week
1	Engineering Physics(EP)			S. Rajesham		4
2	Basic Engineering Mathematics(M-I)			Dr. Nuslin, M.P. Molimol		4
3	Programming for Problem Solving(PPS)			Ms.Fathima		3
4	Engineering Mechanics-I(EM-I)			G. Raju		4
5	Engineering Graphics(EG)			G. Sampath		4
6	Engineering Physics Lab(EP LAB)			S. Rajesham, Ch. Klayani		2
7	Programming for Problem Solving Lab(PPS LAB)			Ms.Fathima		2
8	Engineering Workshop(EWS LAB)			R. S. Mahipal Reddy J. Nithin		2
9	Mentoring			G. Raju, D. Ramchander, P. Supriya		
NOTE: * Represents Tutorial Class						
Date: TT Coord: HOD: Dean Academics: Principal:						

INDIVIDUAL TIME TABLE

Prof. V Madhusudan Rao

CSE-A

CSE A

	9.00 - 10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Mon				LUNCH	CSE-A LAB	
Tue			CSE-A			
Wed						
Thu	CSE-A					
Fri	CSE-A					
Sat						

J. UmaMahesht

CSE-B

CSE-C

	9.00 - 10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Mon	CSE-C		CSE-B	LUNCH		
Tue			CSE-B		CSE-C LAB	
Wed					CSE-B LAB	
Thu	CSE-C					
Fri	CSE-C	CSE-B				
Sat						

M. Ravinder

CSE-D

	9.00 - 10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Mon				LUNCH	CSE-D LAB	
Tue						
Wed			CSE-D			
Thu			CSE-D			
Fri			CSE-D			
Sat						

M. Ajay

CSE-CS

CSE-IOT

	9.00 - 10.00	10.15- 11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Mon	CSE-CS			LUNCH		
Tue	CSE-CS	CSE-IOT				
Wed		CSE-CS			CSE-IOT LAB	
Thu			CSE-IOT			
Fri		CSE-IOT			CSE-CS LAB	
Sat						

Deepa Panse

CSE-AI&ML

CSE-DS

	9.00 - 10.00	10.15- 11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Mon		CSE-AI&ML		LUNCH		
Tue	CSE-DS	CSE-AI&ML				
Wed					CSE-DS LAB	
Thu		CSE-DS			CSE-AI&ML LAB	
Fri		CSE-DS	CSE-AI&ML			
Sat						

Dr. Puja Prasad

ECE-A

ECE-B

	9.00 - 10.00	10.15-11.15	11.30-12.30	12.30-1.30	1.30-2.30	2.30-3.30
Mon	ECE-A			LUNCH		
Tue	ECE-A		ECE-B			
Wed	ECE-A				ECE-B	ECE-B LAB
Thu	ECE-B				ECE-A LAB	
Fri						
Sat	ECE-B LAB					

S. Radha

**ECE-C
ECE-D**

	9.00 - 10.00	10.15- 11.15	11.30- 12.30	12.30-1.30	1.30-2.30	2.30-3.30
Mon	ECE-D			LUNCH		
Tue	ECE-D		ECE-C		ECE-C LAB	
Wed		ECE-D			ECE-C	
Thu						
Fri			ECE-C		ECE-D LAB	
Sat						

**Sudha
Singaraju**

**ME-A
EEE**

	9.00 - 10.00	10.15- 11.15	11.30- 12.30	12.30-1.30	1.30 - 2.30	2.30-3.30
Mon			EEE	LUNCH	ME LAB	
Tue					ME-A	
Wed		EEE				
Thu	ME-A	EEE				
Fri	ME-A				EEE LAB	
Sat						

Fathima

**CE-A
IT-A**

	9.00 - 10.00	10.15- 11.15	11.30- 12.30	12.30-1.30	1.30 - 2.30	2.30-3.30
Mon			IT-A	LUNCH		
Tue			IT-A		CE-A LAB	
Wed			CE-A		IT-A	
Thu	CE-A				IT-A LAB	
Fri			CE-A			
Sat						

LECTURE SCHEDULE WITH METHODOLOGY BEING USED/ ADOPTED

Unit	Number of periods	Topics to be covered	Regular / Additional	Teaching aids used (LCD / OHP / BB)	Remarks
I	2	Basics of Computer Logic Building : Flow Chart, Algorithm	Regular	BB	
	2	Pseudo code. Introduction to Raptor Programming Tool	Regular	LCD	
	1	Introduction to Programming – Computer Languages,	Regular	BB and LCD	
	1	Creating and Running Programs, Program Development	Regular	BB	
	1	Introduction to ‘C’ Language Background	Regular	LCD	
	1	Identifiers, variables, constants	Regular	BB	
	1	Input / Output functions, Data Types	Regular	BB	
	1	Operators	Regular	BB	
	2	Expressions, precedence, associativity	Regular	BB	
	1	Evaluation Evaluation, Type conversions	Regular	BB	
Tot :	13				
II	1	Statements, Sample programs	Regular	BB	
	1	Selection statements: if statements	Regular	LCD/BB	
	1	Switch statement	Regular	LCD/BB	
	2	Loops: while, do-while- examples	Regular	LCD/BB	
	2	For – examples	Regular	LCD/BB	
	1	Break, continue, goto	Regular	BB	
Tot:	8				
III	1	Designing Structured Programs Functions	Regular	BB	
	2	User defined functions	Regular	BB	
	2	Inter function communications	Regular	BB	
	1	Standard functions	Regular	BB	
	2	Scope and storage classes	Regular	BB	

	1	Scope rules and type qualifiers	Regular		
	2	Recursive functions, limitations-examples	Regular	BB	
Tot:	10				
IV	1	Arrays Concepts, using arrays in C	Regular	BB	
	2	Arrays and functions	Regular	BB	
	2	Bubble Sort Linear Search	Regular	BB	
	1	Matrix Addition Matrix Multiplication Two-dimensional arrays	Regular	BB	
	1	Preprocessor Directives	Regular	BB	
Tot:	7				
V	1	Pointers Basic concepts	Regular	BB	
	2	Inter function communications	Regular	BB	
	1	Pointers to pointers, compatibility	Regular	BB	
	1	Void Pointers, NULL pointer	Regular	BB	
	2	Arrays and Pointers, Pointer arithmetic and arrays	Regular	BB	
	1	Passing an array to a function	Regular	BB	
	2	Memory allocation functions, Pointers and functions	Regular	BB	
Tot:	8				

Total Number of periods required: 48

DETAILED NOTES

UNIT – I

Basics of Computers Introduction to components of a computer system: disks, primary and secondary memory, processor, operating system, compilers

Logic Building: Flow chart, Algorithm, Pseudo code. Introduction to Raptor Programming Tool **Introduction to Programming** – Computer Languages, Creating and running programs, Program Development.

Introduction to the C Language – Background, C Programs, Identifiers, Data Types, Variables, Constants, Input/output functions.

Operators - Arithmetic, relational, logical, bitwise, conditional, increment/decrement, assignment etc., C program examples. Expressions, Precedence and Associativity, Expression Evaluation, Type conversions

Basics of Computer

This unit introduces the concepts of computer science, computer systems, and program development. It presents history of computer programming languages to give an understanding of how they have evolved and how C language fits into the course. It gives a review of a program development methodology.

This unit introduces the basics of C language. It explores the power of expressions in C which are closely tied to the operators, precedence, associativity, and statements.

Logic Building:


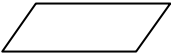


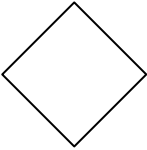
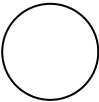

FLOW CHART:

A Flow chart is a Graphical representation of an Algorithm or a portion of an Algorithm. Flow charts are drawn using certain special purpose symbols such as Rectangles, Diamonds, Ovals and small circles. These symbols are connected by arrows called flow lines.

(or)

The diagrammatic representation of way to solve the given problem is called flow chart.

The following are the most common symbols used in Drawing flowcharts:

Oval		Terminal	start/stop/begin/end.
Parallelogram		Input/output	Making data available For processing(input) or recording of the process information(output).
Document		Print Out	show data output in the form of document.
Rectangle		Process	Any processing to be Done .A process changes or moves data. An assignment operation.
Diamond		Decision	Decision or switching type of operations.
Circle		Connector	Used to connect Different parts of flowchart.
Arrow		Flow	Joins two symbols and also represents flow of execution.

ALGORITHM /PSEUDOCODE:

PSEUDOCODE:

Pseudo code is an artificial and informal language that helps programmers to develop algorithms. Pseudo code is similar to everyday English, it is convenient and user friendly although it is not an actual computer programming language. Pseudo code programs are not

actually executed on computer rather they merely help the programmer “think out” a program before attempting to write it in a programming language such as C.

ALGORITHM:

Algorithms was developed by an Arab mathematician. It is chalked out step-by-step approach to solve a given problem. It is represented in an English like language and has some mathematical symbols like \rightarrow , $>$, $<$, $=$ etc. To solve a given problem or to write a program you approach towards solution of the problem in a systematic, disciplined, non-adhoc, step-by-step way is called Algorithmic approach. Algorithm is a penned strategy(to write) to find a solution.

Example: Algorithm/pseudo code to add two numbers

Step 1: Start

Step 2: Read the two numbers in to a,b

Step 3: $c=a+b$

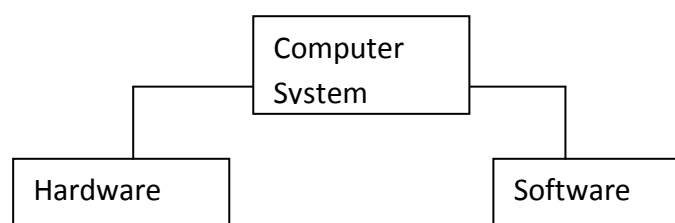
Step 4: write/print c

Step 5: Stop.

Introduction to Programming:

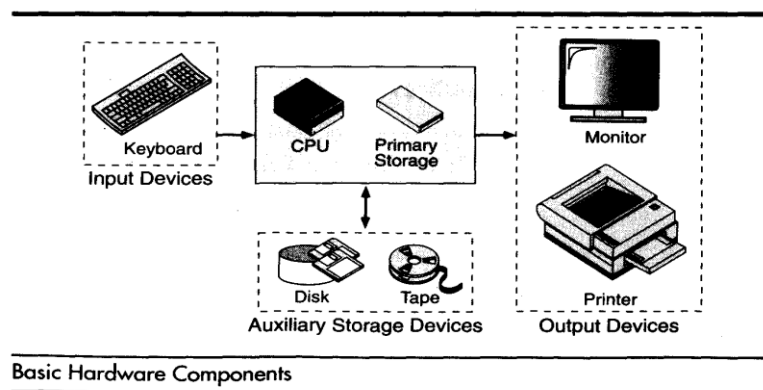
Computer Systems:

A computer is a system made of two major components: hardware and software. The computer hardware is the physical equipment. The software is the collection of programs (instructions) that allow the hardware to do its job.



Computer Hardware

The hardware component of the computer system consists of five parts: input devices, central processing unit (CPU), primary storage, output devices, and auxiliary storage devices.



The **input device** is usually a keyboard where programs and data are entered into the computers. Examples of other input devices include a mouse, a pen or stylus, a touch screen, or an audio input unit.

The **central processing unit (CPU)** is responsible for executing instructions such as arithmetic calculations, comparisons among data, and movement of data inside the system. Today's computers may have one, two, or more CPUs. **Primary storage**, also known as main memory, is a place where the programs and data are stored temporarily during processing. The data in primary storage are erased when we turn off a personal computer or when we log off from a time-sharing system.

The **output device** is usually a monitor or a printer to show output. If the output is shown on the monitor, we say we have a **soft copy**. If it is printed on the printer, we say we have a hard copy.

Auxiliary storage, also known as **secondary storage**, is used for both input and output. It is the place where the programs and data are stored permanently. When we turn off the computer, or programs and data remain in the secondary storage, ready for the next time we need them.

Computer Software

Computer software is divided into two broad categories: system software and application software. System software manages the computer resources. It provides the interface between

the hardware and the users. Application software, on the other hand is directly responsible for helping users solve their problems.

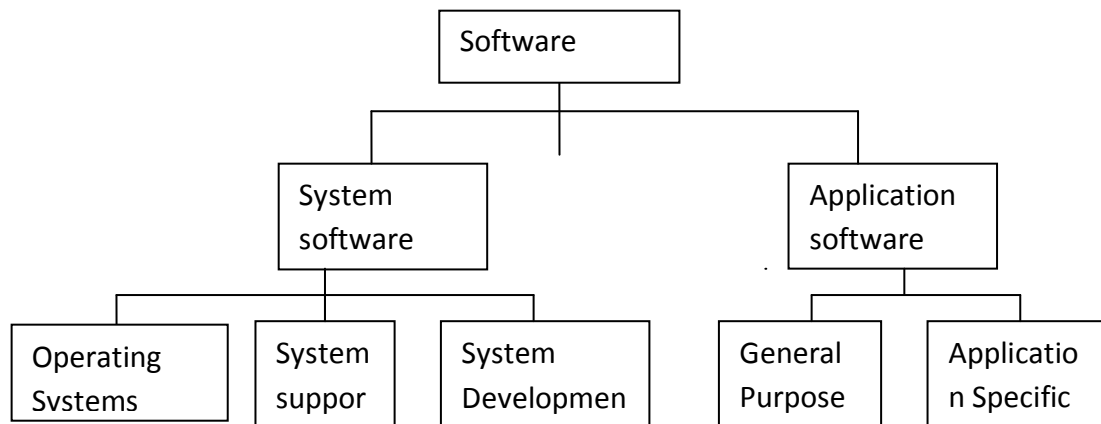


Fig: Types of software

System Software

System software consists of programs that manage the hardware resources of a computer and perform required information processing tasks. These programs are divided into three classes: the operating system, system support, and system development.

The **operating system** provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose of this software is to keep the system operating in an efficient manner while allowing the users access to the system.

System support software provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs. Operating services consists of programs that provide performance statistics for the operational staff and security monitors to protect the system and data.

The last system software category ,**system development software**, includes the language translators that convert programs into machine language for execution ,debugging tools to ensure that the programs are error free and computer –assisted software engineering(CASE) systems.

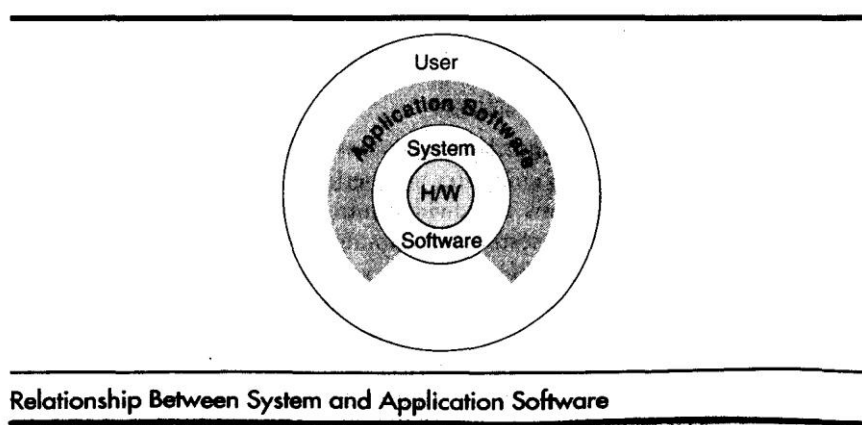
Application software

Application software is broken in to two classes :general-purpose software and application – specific software. **General purpose software** is purchased from a software developer and can be used for more than one application. Examples of general-purpose software include word processors ,database management systems ,and computer aided design systems. They are labelled general purpose because they can solve a variety of user computing problems.

Application –specific software can be used only for its intended purpose.

A general ledger system used by accountants and a material requirements planning system used by a manufacturing organization are examples of application-specific software. They can be used only for the task for which they were designed they cannot be used for other generalized tasks.

The relationship between system and application software is shown in fig-2.In this figure, each circle represents an interface point .The inner core is hard ware. The user is represented by the out layer. To work with the system, +the typical user uses some form of application software. The application software in turn interacts with the operating system ,which is apart of the system software layer. The system software provides the direct interaction with the hard ware. The opening at the bottom of the figure is the path followed by the user who interacts directly with the operating system when necessary.



Computer Languages:

To write a program for a computer, we must use a **computer language**. Over the years computer languages have evolved from machine languages to natural languages.

1940's Machine level Languages

1950's Symbolic Languages

1960's High-Level Languages

Machine Languages

In the earliest days of computers, the only programming languages available were machine languages. Each computer has its own machine language, which is made of streams of 0's and 1's.

Instructions in machine language must be in streams of 0's and 1's because the internal circuits of a computer are made of switches transistors and other electronic devices that can be in one of two states: off or on. The off state is represented by 0 , the on state is represented by 1.

The only language understood by computer hardware is machine language.

Symbolic Languages

In early 1950's Admiral Grace Hopper, A mathematician and naval officer developed the concept of a special computer program that would convert programs into machine language. The early programming languages simply mirror to the machine languages using symbols of mnemonics to represent the various machine language instructions because they used symbols, these languages were known as symbolic languages.

Computer does not understand symbolic language it must be translated to the machine language. A special program called assembler translates symbolic code into machine language. Because symbolic languages had to be assembled into machine language they soon became known as assembly languages.

Symbolic language uses symbols or mnemonics to represent the various ,machine language instructions.

High Level Languages

Symbolic languages greatly improved programming efficiency; they still required programmers to concentrate on the hardware that they were using. Working with symbolic

languages was also very tedious because each machine instruction has to be individually coded. The desire to improve programmer efficiency and to change the focus from the computer to the problem being solved led to the development of high-level language.

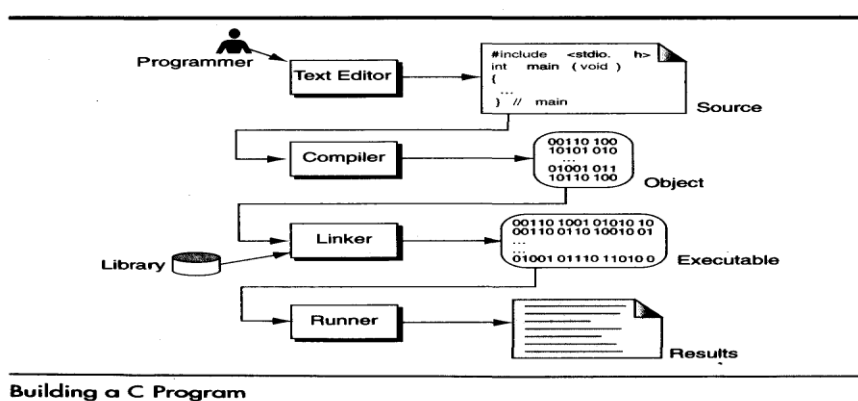
High level languages are portable to many different computers, allowing the programmer to concentrate on the application problem at hand rather than the intricacies of the computer. High-level languages are designed to relieve the programmer from the details of the assembly language. High level languages share one thing with symbolic languages, they must be converted into machine language. The process of converting them is known as compilation.

The first widely used high-level languages, FORTRAN (FORmula TRANslation) was created by John Backus and an IBM team in 1957; it is still widely used today in scientific and engineering applications. After FORTRAN was COBOL (Common Business-Oriented Language). Admiral Hopper played a key role in the development of the COBOL Business language.

C is a high-level language used for system software and new application code.

Creating and Running Programs:

Computer hardware understands a program only if it is coded in its machine language. It is the job of the programmer to write and test the program. There are four steps in this process: 1. Writing and Editing the program 2. Compiling the program 3. Linking the program with the required library modules 4. Executing the program.



1. Writing and Editing Programs

The software used to write programs is known as a **text editor**. A text editor helps us enter, change, and store character data. Depending on the editor on our system, we could use it to

write letters, create reports, or write programs. The main difference between text processing and program writing is that programs are written using lines of code, while most text processing is done with character and lines.

Text editor is a generalized word processor, but it is more often a special editor included with the compiler. Some of the features of the editor are search commands to locate and replace statements, copy and paste commands to copy or move statements from one part of a program to another, and formatting commands that allow us to set tabs to align statements.

After completing a program, we save our file to disk. This file will be input to the compiler; it is known as a **source file**.

2.Compiling Programs

The code in a source file stored on the disk must be translated into machine language, this is the job of the **compiler**. The c compiler is two separate programs. the **preprocessor** and the **translator**.

The pre-processor reads the source code and prepares it for the translator. While preparing the code ,it scans for special instructions known as preprocessor commands. These commands tell the preprocessor to look for special code libraries, make substitutions in the code ,and in other ways prepare the code for translation into machine language. The result of preprocessing is called the translation unit.

After the preprocessor has prepared the code for compilation, the translator does the actual work of converting the program into machine language. The translator reads the translation unit and writes the resulting object module to a file that can then be combined with other precompiled units to form the final program. An object module is the code in machine language. The output of the compiler is machine language code, but it is not ready to run; that is ,it is not executable because it does not have the required C and other functions included.

3.Linking Programs

A C program is made up of many functions. We write some of these functions, and they are a part of our source program. There are other functions, such as input/output processes and,

mathematical library functions, that exist elsewhere and must be attached to our program. The linker assembles all of these functions, ours and systems into our final executable program.

4.Executing Programs

Once program has been linked, it is ready for execution. To execute a program, we use an operating system command, such as run, to load the program into primary memory and execute it. Getting the program into memory is the function of an operating system program known as the loader. It locates the executable program and reads it into memory. When everything is loaded, the program takes control and it begins execution.

In a typical program execution, the reads data for processing ,either from the user or from a file. After the program processes the data, it prepares the output. at output can be to the user's monitor or to a file. When the program has finished its job, it tells the operating system ,which then removes the program from memory.

PROGRAM DEVELOPMENT STEPS:

Program Development is a multistep process that requires that we understand the problem, develop a solution, write the program, and then test it. When we are given the assignment to develop a program, we will be given a program requirements statement and the design of any program interfaces. We should also receive an overview of the complete project so that we will take the inputs we are given and convert them to the outputs that have been specified. This is known as program design.

1.Understand the Problem

The first step in solving any problem is to understand it. By reading the requirements statements carefully, we fully understand it, we review our understanding with the user and the systems analyst to know the exact purpose.

2.Develop the solution

Once we fully understand the problem we need to develop our solution. Three tools will help in this task. 1. Structure chart, 2.Pseudocode &3.Flowcharts. Generally, we will use structure chart and either flowchart or Pseudo code

The structure chart is used to design the whole program .Pseudo code and flowcharts are used to design the individual parts of the program.

3.Structure chart

A structure chart, also known as hierarchy chart, shows the functional flow through our program. The structure chart shows how we are going to break our program into logical steps each step will be a separate module. The structure chart shows the interaction between all the parts (modules) of our program.

We can use flowchart or pseudo code to complete the design of your program will depend on experience and difficulty of the program your designing.

4.Write the program

When we write a program, we start with the top box on the structure chart and work our way to the bottom. This is known as top-down implementation. We will write the programs by using structure chart and flowchart or pseudo code.

5.Test the Program

Program testing can be a very tedious and time- consuming part of program development. As the programmer we are responsible for completely testing our program. In large-development projects test engineers are responsible for testing to make sure all the programs work together.

There are 2 types of testing.

1. **Black box testing:** This is done by the system test engineer and the user. Black box testing is the programs are tested without knowing what is inside it, without knowing how it works. Black box test plans are developed by looking only the requirements statement. The test engineer uses these requirements to develop test plans.
2. **White box testing:** This is the responsibility of the programmer. White box testing assumes that the tester knows everything about the program.

Except for the simplest program, one set of test data will not completely validate a program.

Introduction to the C Language:

C language facilitates a very efficient approach to the development and implementation of computer programs. The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce “an unambiguous and machine independent definition of the language C “ while still retaining its spirit .

C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on. Machine-specific features are isolated in a few modules within the UNIX kernel, which makes it easy for you to modify them when you are porting to a different hardware architecture.

C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was The C Programming Language by Brian Kernighan and Dennis Ritchie, published in 1978.

In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as *ANSI C*, and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard.

It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years; it is currently known as "C9X."

BASIC STRUCTURE OF C LANGUAGE :

The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

- | |
|------------------------------------|
| 1. Documentation section |
| 2. Linking section |
| 3. Definition section |
| 4. Global declaration section |
| 5. Main function section |
| { |
| Declaration section |
| Executable section |
| } |
| 6. Sub program or function section |

1. DOCUMENTATION SECTION:

This section comes first and is used to document the use of logic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with `/*` and `*/`. Whatever is written between these two are called comments.

2. LINKING SECTION :

This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.

e.g. `#include <stdio.h>`

3. DEFINITION SECTION :

It is used to declare some constants and assign them some value.

e.g. `#define MAX 25`

Here `#define` is a compiler directive which tells the compiler whenever `MAX` is found in the program replace it with `25`.

4. GLOBAL DECLARATION SECTION :

Here the variables which are used throughout the program (including main and other functions) are declared so as to make them global(i.e. accessible to all parts of program)

e.g. `int i;` (before `main()`)

5. MAIN FUNCTION SECTION :

It tells the compiler where to start the execution from

```
main()
{
    point from execution starts
}
```

main function has two sections

1. declaration section : In this the variables and their data types are declared.

2. Executable section : This has the part of program which actually performs the task we need.

6. SUB PROGRAM OR FUNCTION SECTION :

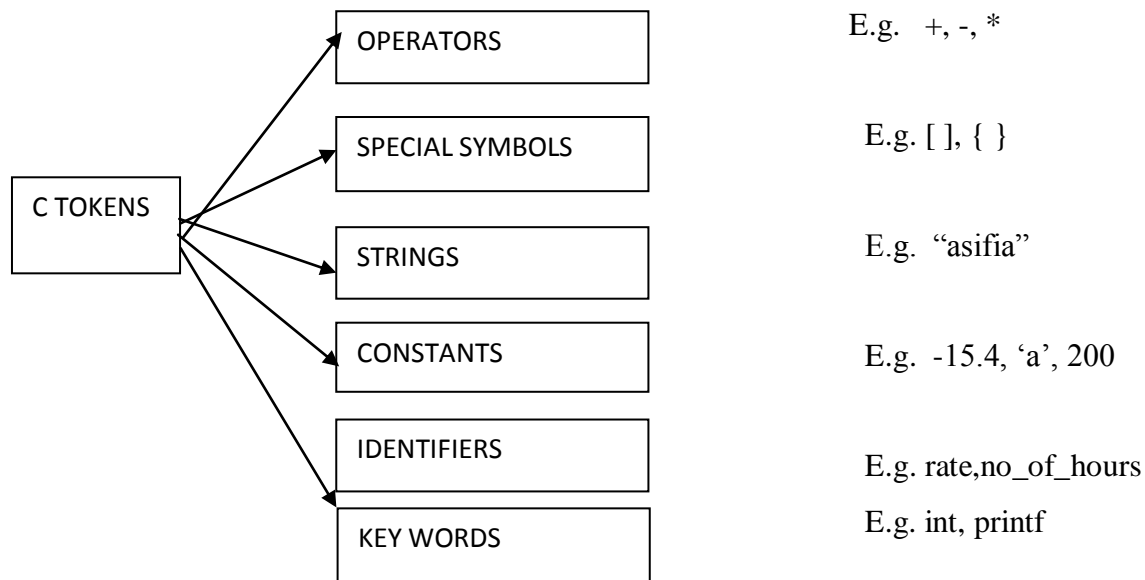
This has all the sub programs or the functions which our program needs.

SIMPLE 'C' PROGRAM:

```
/* simple program in c */
#include<stdio.h>
main()
{
    printf("welcome to c programming");
} /* End of main */
```

C-TOKENS :

Tokens are individual words and punctuations marks in English language sentence. The smallest individual units are known as C tokens.



A C program can be divided into these tokens. A C program contains minimum 3 c tokens no matter what the size of the program is.

IDENTIFIERS

Names of the variables and other program elements such as functions, array, etc, are known as identifiers.

There are few rules that govern the way variable are named(identifiers).

1. Identifiers can be named from the combination of A-Z, a-z, 0-9, _(Underscore).
2. The first alphabet of the identifier should be either an alphabet or an underscore.

digit is not allowed.

3. It should not be a keyword.

Eg: name, ptr, sum

After naming a variable we need to declare it to compiler of what data type it is .

The format of declaring a variable is

Data-type id1, id2,.....idn;

where data type could be float, int, char or any of the data types.

id1, id2, id3 are the names of variable we use. In case of single variable, no commas are required.

eg float a, b, c;
 int e, f, grand total;
 char present_or_absent;

ASSIGNING VALUES

When we name and declare variables we need to assign value to the variable. In some cases we assign value to the variable directly like

 a=10; in our program.

In some cases we need to assign values to variable after the user has given input for that.

eg :we ask user to enter any no. and input it

```
/* Write a program to show assigning of values to variables */
```

```
#include<stdio.h>
```

```
main()
```

```
{  
    int a;  
    float b;  
    printf("Enter any number\n");  
    b=190.5;  
    scanf("%d",&a);  
    printf("user entered %d", a);  
    printf("B's values is %f", b);  
}
```

DATA TYPES :

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

1. Basic Data types or primary data types
2. Derived data types
3. User defined data types

BASIC DATA TYPES:

All arithmetic operations such as Addition , subtraction etc are possible on basic data types.

E.g.: int a,b;

 Char c;

The following table shows the Storage size and Range of basic data types:

TYPE	LENGTH	RANGE
Unsigned char	8 bits	0 to 255
Char	8 bits	-128 to 127
Short int	16 bits	-32768 to 32767
Unsigned int	32 bits	0 to 4,294,967,295
Int	32 bits	-2,147,483,648 to 2,147,483,648
Unsigned long	32 bits	0 to 4,294,967,295
Enum	16 bits	-2,147,483,648 to 2,147,483,648
Long		-2,147,483,648 to 2,147,483,648
Float	32 bits	-2,147,483,648 to 2,147,483,648
Double	32 bits	3.4*10E-38 to 3.4*10E38
Long double	64 bits	1.7*10E-308 to 1.7*10E308
	80 bits	3.4*10E-4932 to 1.1*10E4932

DERIVED DATA TYPES

Derived datatypes are used in 'C' to store a set of data values. Arrays ,functions and pointers are examples for derived data types.

Ex: int a[10];

Char name[20];

USER DEFINED DATATYPES

C Provides a facility for the user to create his own data type depending on the requirement.

The user defined data types include structures, unions, typedef.

Example 1

```
struct student
```

```
{
```

```
int rollno;
```

```
char name[10];
```

```
};
```

Example 2

```
union student
```

```
{
```

```
int rollno;  
char name[10];  
};
```

Example 3

typedef int Integer;

makes the name Integer a synonym of int. Now the type Integer can be used in declarations ,casts,etc,like,

Integer num1,num2;

Which will be treated by the C compiler as the declaration of num1,num2as int variables.

“typedef” is more useful with structures and pointers.

VARIABLES :

C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable. The value of the C variable may get change in the program.C variable might be belonging to any of the data type like int, float, char etc.

Syntax for variable declaration

datatype variablename;

RULES FOR NAMING C VARIABLE:

- 1.Variable name must begin with letter or underscore.
1. Variables are case sensitive
2. They can be constructed with digits, letters.
3. No special symbols are allowed other than underscore.
4. sum, height, _value are some examples for variable name.

CONSTANTS :

A quantity that does not vary during the execution of a program is known as a constant supports two types of constants namely Numeric constants and character constants.

NUMERIC CONSTANTS

1. Example for an integer constant is 786,-127
 2. Long constant is written with a terminal ‘l’or ‘L’,for example 1234567899L is a Long constant.
 3. Unsigned constants are written with a terminal ‘u’ or ‘U’,and the suffix ‘ul’ and ‘UL’ indicates unsigned long. for example 123456789u is an Unsigned constant and 1234567891ul is an unsigned long constant.
 4. The advantage of declaring an unsigned constant is to increase the range of storage.
-

5. Floating point constants contain a decimal point or an exponent or both. For Eg :

123.4, 1e-2, 1.4E-4, etc. The suffixes f or F indicate a float constant while the absence of f or F indicate the double, l or L indicate long double.

CHARACTER CONSTANTS

A character constant is written as one character with in single quotes such as 'a'. The value of a character constant is the numerical value of the character in the machines character set. certain character constants can be represented by escape sequences like '\n'. These sequences look like two characters but represent only one.

The following are the some of the examples of escape sequences:

Escape sequence	Description
\a	Alert
\b	Backspace
\f	Form feed
\n	New Line
\r	Carriage return
\t	Horizontal Tab
\v	Vertical Tab

String constants or string literal is a sequence of zero or more characters surrounded by a double quote. Example , " I am a little boy". quotes are not a part of the string.

To distinguish between a character constant and a string that contains a single character ex: 'a' is not same as "a". 'a' is an integer used to produce the numeric value of letter a in the machine character set, while "a" is an array of characters containing one character and a '\0' as a string in C is an array of characters terminated by NULL.

There is one another kind of constant i.e Enumeration constant , it is a list of constant integer values.

Ex.: enum color { RED, Green, BLUE }

The first name in the enum has the value 0 and the next 1 and so on unless explicit values are specified.

If not all values specified , unspecified values continue the progression from the last specified value. For example

Enum months { JAN=1, FEB,MAR, ..., DEC }

Where the value of FEB is 2 and MAR is 3 and so on.

Enumerations provide a convenient way to associate constant values with names.

KEYWORDS :

There are certain words, called keywords (reserved words) that have a predefined meaning in 'C' language. These keywords are only to be used for their intended purpose and not as identifiers.

The following table shows the standard 'C' keywords

Auto	Break	Case	Char	Const	Continue
Default	Do	Double	Else	Enum	Extern
Float	For	Goto	If	Int	Long
Register	Return	Short	Signed	Sizeof	Static
Struct	Switch	Typedef	Union	Unsigned	void
Volatile	While				

INPUT AND OUTPUT STATEMENTS :

The standard input-output header file, named stdio.h contains the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively.

Syntax of Scanf():

Scanf("control string",arg1,arg2,...,argn);

Example:

Scanf("%d%c",&a,&b);

Syntax of printf():

printf("control string");

printf("control string",arg1,arg2,...,argn);

Example

printf("sum of %d and %d is %d",a,b,c);

OPERATORS :

An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. They form expressions.

C operators can be classified as

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment or Decrement operators
6. Conditional operator
7. Bit wise operators
8. Special operators

1. ARITHMETIC OPERATORS : All basic arithmetic operators are present in C.

operator	meaning
+	add
-	subtract
*	multiplication
/	division
%	modulo division(remainder)

An arithmetic operation involving only real operands(or integer operands) is called real arithmetic(or integer arithmetic). If a combination of arithmetic and real is called mixed mode arithmetic.

2. RELATIONAL OPERATORS : We often compare two quantities and depending on their relation take certain decisions for that comparison we use relational operators.

operator	meaning
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or
equal to ==	is equal to
!=	is not equal to

It is the form of

ae-1 relational operator ae-2

3. LOGICAL OPERATORS : C supports 3 types of logical operators.

Operator	meaning
&&	Logical AND
	Logical OR
!	Logical NOT

The logical operators && and || are used when we want to test more than one condition and make decisions.

Op1	Op2	Op1&&op2	Op1 op2
zero	Zero	0	0
zero	Non zero	0	1
Non zero	Zero	0	1
Non zero	Non zero	1	1

5. ASSIGNMENT OPERATORS :

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

6. INCREMENT AND DECREMENT OPERATORS :

++ and == are called increment and decrement operators used to add or subtract.

Both are unary and as follows

++m or m++

--m or m--

The difference between ++m and m++ is

if m=5; y=++m then it is equal to m=5;m++;y=m;

if m=5; y=m++ then it is equal to m=5;y=m;m++;

7. CONDITIONAL OPERATOR :

Syntax:

conditionalExpression ? expression1 : expression2

The conditional operator works as follows:

The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.

If conditionalExpression is true, expression1 is evaluated.

If conditionalExpression is false, expression2 is evaluated.

8. BIT WISE OPERATORS : C supports special operators known as bit wise operators for manipulation of data at bit level. They are not applied to float or double.

operator	meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift
~	one's complement

9. SPECIAL OPERATORS : These operators which do not fit in any of the above classification are ,(comma), sizeof, Pointer operators(& and *) and member selection operators (. and ->). The comma operator is used to link related expressions together.

sizeof operator is used to know the sizeof operand.

/* programs to exhibit the use of operators */

#include<stdio.h>

main()

{

int sum, mul, modu;

float sub, divi;

int i,j;

float l, m;

```

printf("Enter two integers ");
scanf("%d%d",&i,&j);
printf("Enter two real numbers");
scanf("%f%f",&l,&m);
sum=i+j;
mul=i*j;
modu=i%j;
sub=l-m;
divi=l/m;
printf("sum is %d", sum);
printf("mul is %d", mul);
printf("Remainder is %d", modu);
printf("subtraction of float is %f", sub);
printf("division of float is %f", divi);
}
/* program to implement relational and logical */
#include<stdio.h>
main()
{
    int i, j, k;
    printf("Enter any three numbers ");
    scanf("%d%d%d", &i, &j, &k);
    if((i<j)&&(j<k))
        printf("k is largest");
    else if i<j || j>k
    {
        if i<j && j >k
            printf("j is largest");
        else
            printf("j is not largest of all");
    }
}
/* program to implement increment and decrement operators */

```

```

#include<stdio.h>
main()
{
    int i;
    printf("Enter a number");
    scanf("%d", &i);
    i++;
    printf("after incrementing %d ", i);
    i--;
    printf("after decrement %d", i);
}

/* program using ternary operator and assignment */
#include<stdio.h>
main()
{
    int i,j,large;
    printf("Enter two numbers ");
    scanf("%d%d",&i,&j);
    large=(i>j)?i:j;
    printf("largest of two is %d",large);
}

```

EXPRESSIONS :

An expression is a sequence of operands and operators that reduces to a single value. Expression can be simple or complex. An **operator** is a syntactical token that requires an action be taken. An **operand** is an object on which an operation is performed.

A simple expression contains only one operator. E.g: $2 + 3$ is a simple expression whose value is 5. A complex expression contains more than one operator. E.g: $2 + 3 * 2$.

To evaluate a complex expression, we reduce it to a series of simple expressions. In this first we will evaluate the simple expression $3 * 2$ (6) and then the expression $2 + 6$, giving a result of 8.

The order in which the operators in a complex expression are evaluated is determined by a set of priorities known as **precedence**, the higher the precedence, the earlier the expression containing the operator is evaluated. If two operators with the same precedence occur in a complex expression, another attribute of an operator, its associativity, takes control. **Associativity** is the parsing direction used to evaluate the expression. It can be either left-to-right or right-to-left. When two operators with the same precedence occur in an expression

and their associativity is left-to-right ,the left operator is evaluated first. For example ,in the expression $3*4/6$,there are two operators multiplication and division ,with the same precedence and left-to-right associativity .Therefore the multiplication is evaluated before the division .

The following table shows the precedence and associativity of operators:

Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - * (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
	left to right
?:	right to left
= += -= *= /= %= &= ^= = <<= >>=	right to left
,	left to right

EXPRESSION EVALUATION:

ARITHMETIC EXPRESSIONS

It is a combination of variables, constants and operators arranged according to the syntax of C language.

Some examples

$A * B - C$

$(M + N) * (X + Y)$

Evaluation of expressions : Expressions are evaluated using an assignment statement of the form

Variable = expression

Eg $x = x*y + z-3 *(z *y)$

Precedence of arithmetic expressions is used to evaluate an expression to provide unambiguous way of solving an expression. The highest precedence operator is evaluated then next highest precedence operator until no operator is present.

The precedence or priorities are as follows

High * / %

Low + -

An **expression is evaluated** in left to right and value is assigned to variable in left side of assignment operator.

```
/* program to demonstrate evaluation of expressions */
```

```
#include<stdio.h>
```

```
main()
```

```
{ float a,b,c,x,y,z;
```

```
    a=9;b=23;c=3;
```

```
    x=a-b/3+c*2-1;
```

```
    y=a-b/(3+c)*(2-1);
```

```
    z=a-(b/(3+c)*2)-1;
```

```
    printf("values of x,y,z are %d%d%d",x,y,z);
```

```
}
```

TYPE CONVERSION

In an expression that involves two different data types ,such as multiplying an integer and a floating point number to perform these evaluations ,one of the types must be converted.

We have two types of conversions

1.Implicit Type Conversion

2.Explicit Type Conversion

1.IMPLICIT TYPE CONVERSION

When the types of the two operands in a binary expression are different automatically converts one type to another .This is known as implicit type conversion .

2.EXPLICIT TYPE CONVERSION

Explicit type conversion uses the unary cast operator ,to convert data from one type to another. To cast data from one type to another ,we specify the new type in parentheses before the value we want converted.

Syntax:

```
(data_type)expression;
```

For example ,to convert an integer ,a , to a float, we code the expression like

```
(float) a;
```

UNIT-II

Statements- Selection Statements (decision making) – if and switch statements with Raptor Tool, and C program examples.

Repetition statements (loops) - while, for, do-while statements with Raptor Tool, and C Program examples

Statements related to looping – break, continue, goto, Simple C Program examples.

STATEMENTS:

A statement causes an action to be performed by the program. It translates directly in to one or more executable computer instructions.

STATEMENT TYPES:

1. NULL STATEMENT

The null statement is just a semicolon (the terminator).

Eg : ; //null statement

Although they do not arise often, there are syntactical situations where we must have a statement but no action is required .In these situations we use the null statement.

2 . EXPRESSION STATEMENT

An expression is turned in to a statement by placing a semicolon(;)after it.

expression; //expression statement

Eg: a=2;

3. RETURN STATEMENT

A return statement terminates a function. All functions ,including main, must have a return statement. Where there is no return statement at the end of the function ,the system inserts one with a void return value.

return expression; //return statement

The return statement can return a value to the calling function. In case of main ,it returns a value to the operating system rather than to another function. A return value of zero tells the operating system that the program executed successfully.

4. COMPOUND STATEMENTS

A compound statement is a unit of code consisting of zero or more statements .It is also known as a **block**. The compound statement allows a group of statements to become one single entity.

A compound statement consists of an opening brace ,an optional declaration and definition section ,and an optional statement section ,followed by a closing brace.

Eg:

```
{  
    //Local Declarations  
    int x;  
    int y;  
    int z;  
    //Statements  
    x=1;  
    y=2;  
    ...  
}    //End Block
```

Selection Statements:

The selection statements are of 3 types.

- 1.one way selection statement: example if statement
- 2.two way selection statement:example if and else statement
- 3.multiway selection statements:example nested if statement ,else if ladder and switch statement.

if selection statement:

It is a fundamental and basic decision selective statement. It is used to select or skip a statement or block of statements according to the value given by the condition.

The condition is placed in a pair of parentheses with the keyword if called conditional statement.

A statement or a block of statements are placed under the conditional statement.

The statement or block of statements is executed if the condition gives true otherwise skipped from the execution.

Syntax:

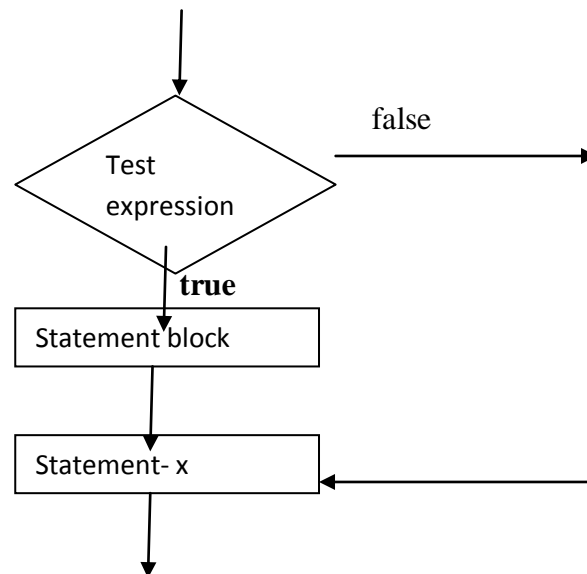
```
if(test expression)  
{
```

Statement block;

}

Statement x;

Flowchart:



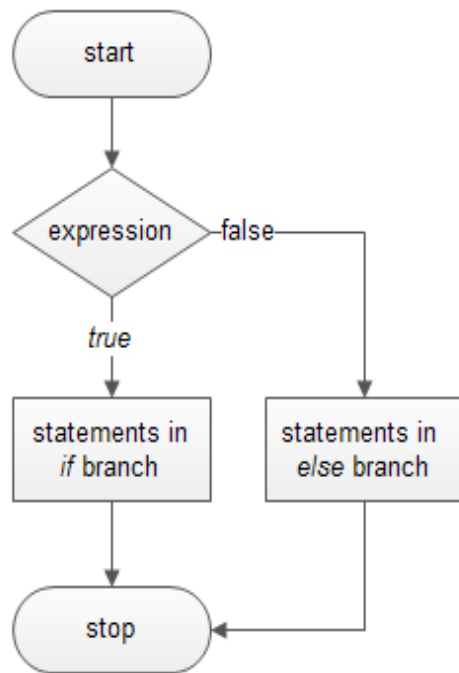
2.If...else statement

Sometimes you want to execute a piece of code in case of the expression in the if statement evaluates to false. You can use the second form of the if statement which is known as if else statement. The following illustrates the syntax of the if else statement:

Syntax:

```
if(expression){  
    /* code block of if statement */  
}else{  
    /* code block of else statement */  
}
```

Flowchart:



Else if ladder statement:

Syntax:

```
if (testExpression1)
{
    // statement(s)
}
else if(testExpression2)
{
    // statement(s)
}
else if (testExpression 3)
{
    // statement(s)
}
.
.
else
```

```
{  
    // statement(s)  
}
```

Nested if statement

Syntax:

If(condition 1)

```
{  
If(condition2)
```

```
{  
Statement1;
```

```
}
```

Else

```
{  
statement 2;
```

```
}
```

Else

```
{  
    Statement 3;
```

```
}
```

Statement x;

```
}
```

Switch statement:

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Syntax

The syntax for a switch statement in C programming language is as follows –

```
switch(expression) {  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
    case constant-expression :  
        statement(s);
```

```

    break; /* optional */
/* you can have any number of case statements */
default : /* Optional */
statement(s);
}

```

The following rules apply to a switch statement –

- The expression used in a switch statement must have an integral or enumerated type or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Repetition statements(Looping):

Sometimes we require a set of statements to be executed repeatedly until a condition is met.

We have two types of looping structures. One in which condition is tested before entering the statement block called entry control.

The other in which condition is checked at exit called exit controlled loop.

WHILE STATEMENT :

Syntax:

```

While(test condition)
{

```

```
        body of the loop
    }
```

It is an entry controlled loop. The condition is evaluated and if it is true then body of loop is executed. After execution of body the condition is once again evaluated and if it is true body is executed once again. This goes on until test condition becomes false.

```
/* program for while */
#include<stdio.h>
main()
{
    int count,n;
    float x,y;
    printf("Enter the values of x and n");
    scanf("%f%d",&x,&n);
    y=1.0;
    count=1;
    while(count<=n)
    {
        y=y*x;
        count++;
    }
    printf("x=%f; n=%d; x to power n = %f",x,n,y);
}
```

DO -WHILE STATEMENT :

The while loop does not allow body to be executed if test condition is false. The do while is an exit controlled loop and its body is executed at least once.

Syntax:

```
do
{
    body
}while(test condition)
```

```
/* printing multiplication table */
```

```

#include<stdio.h>
#define COL 10
#define ROW 12
main()
{
    int row,col,y;
    row=1;
    do
    {
        col=1;
        do
        {
            y=row*col;
            printf("%d",y);
            col=col+1;
        }while(col<=COL);
        printf("\n");
        row=row+1;
    }while(row<=ROW);
}

```

THE FOR LOOP :

It is also an entry control loop that provides a more concise structure

Syntax:

```

for(initialization; test control; increment)
{
    body of loop
}

```

```

/* program of for loop */
#include<stdio.h>
main()
{

```

```

long int p;
int n;
double q;
printf("2 to power n ");
p=1;
for(n=0;n<21;++n)
{
    if(n==0)
        p=1;
    else
        p=p*2;
    q=1.0/(double)p;
    printf("%10ld%10d",p,n);
}
}

```

Statements related to looping:

BREAK STATEMENT:

This is a simple statement. It only makes sense if it occurs in the body of a switch, do, while or for statement. When it is executed the control of flow jumps to the statement immediately following the body of the statement containing the break. Its use is widespread in switch statements, where it is more or less essential to get the control .

The use of the break within loops is of dubious legitimacy. It has its moments but is really only justifiable when exceptional circumstances have happened, and the loop has to be abandoned. It would be nice if more than one loop could be abandoned with a single break but that isn't how it works. Here is an example.

```

#include <stdio.h>
#include <stdlib.h>
main(){
    int i;
    for(i = 0; i < 10000; i++){
        if(getchar() == 's')

```

```

        break;
    printf("%d\n", i);
}
exit(EXIT_SUCCESS);
}

```

It reads a single character from the program's input before printing the next in a sequence of numbers. If an 's' is typed, the break causes an exit from the loop.

If you want to exit from more than one level of loop, the break is the wrong thing to use.

CONTINUE STATEMENT:

This statement has only a limited number of uses. The rules for its use are the same as for break, with the exception that it doesn't apply to switch statements. Executing a continue starts the next iteration of the smallest enclosing do, while or for statement immediately. The use of continue is largely restricted to the top of loops, where a decision has to be made whether or not to execute the rest of the body of the loop. In this example it ensures that division by zero (which gives undefined behaviour) doesn't happen.

```

#include <stdio.h>
#include <stdlib.h>
main(){
    int i;
    for(i = -10; i < 10; i++){
        if(i == 0)
            continue;
        printf("%f\n", 15.0/i);
        /*
        * Lots of other statements .....
        */
    }
    exit(EXIT_SUCCESS);
}

```

The `continue` can be used in other parts of a loop, too, where it may occasionally help to simplify the logic of the code and improve readability. `continue` has no special meaning to a switch statement, where `break` does have. Inside a switch, `continue` is only valid if there is a loop that encloses the switch, in which case the next iteration of the loop will be started.

There is an important difference between loops written with `while` and `for`. In a `while`, a `continue` will go immediately to the test of the controlling expression. The same thing in a `for` will do two things: first the update expression is evaluated, then the controlling expression is evaluated.

GOTO AND LABELS:

In C, it is used to escape from multiple nested loops, or to go to an error handling exit at the end of a function. You will need a *label* when you use a `goto`; this example shows both.

```
goto L1;
/* whatever you like here */
L1: /* anything else */
```

A label is an identifier followed by a colon. Labels have their own ‘name space’ so they can't clash with the names of variables or functions. The name space only exists for the function containing the label, so label names can be re-used in different functions. The label can be used before it is declared, too, simply by mentioning it in a `goto` statement.

Label *must* be part of a full statement, even if it's an empty one. This usually only matters when you're trying to put a label at the end of a compound statement—like this.

```
label_at_end: ; /* empty statement */
}
```

The `goto` works in an obvious way, jumping to the labelled statements. Because the name of the label is only visible inside its own function, you can't jump from one function to another one.

It's hard to give rigid rules about the use of `gotos` but, as with the `do`, `continue` and the `break` (except in switch statements), over-use should be avoided. More than one `goto` every 3–5 functions is a symptom that should be viewed with deep suspicion.

UNIT-III

Functions-Designing Structured Programs, Functions, user defined functions, inter function communication, Standard functions, Scope, Storage classes-auto, register, static, extern, scope rules, type qualifiers, C program examples.

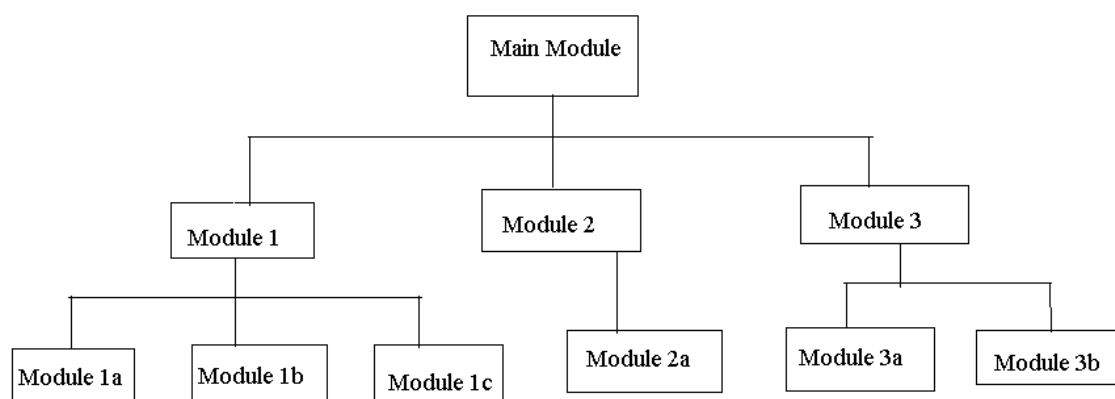
Recursion- recursive functions, Limitations of recursion, example C programs

DESIGNING STRUCTURED PROGRAMS:

The planning for large programs consists of first understanding the problem as a whole, second breaking it into simpler, understandable parts. We call each of these parts of a program a **module** and the process of subdividing a problem into manageable parts **top-down design**.

The principles of top-down design and structured programming dictate that a program should be divided into a main module and its related modules. Each module is in turn divided into sub-modules until the resulting modules are intrinsic; that is, until they are implicitly understood without further division.

Top-down design is usually done using a visual representation of the modules known as a structure chart. The structure chart shows the relation between each module and its sub-modules. The structure chart is read top-down, left-right. First we read Main Module. Main Module represents our entire set of code to solve the problem.



Moving down and left, we then read Module 1. On the same level with Module 1 are Module 2 and Module 3. The Main Module consists of three sub-modules. At this level, however we are dealing only with Module 1. Module 1 is further subdivided into three modules, Module

1a, Module 1b, and Module 1c. To write the code for Module 1, we need to write code for its three sub-modules.

The Main Module is known as a calling module because it has sub-modules. Each of the sub-modules is known as a called module. But because Modules 1, 2, and 3 also have sub-modules, they are also calling modules; they are both called and calling modules.

Communication between modules in a structure chart is allowed only through a calling module. If Module 1 needs to send data to Module 2, the data must be passed through the calling module, Main Module. No communication can take place directly between modules that do not have a calling-called relationship.

How can Module 1a send data to Module 3b?

It first sends data to Module 1, which in turn sends it to the Main Module, which passes it to Module 3, and then on to Module 3b.

The technique used to pass data to a function is known as parameter passing. The parameters are contained in a list that is a definition of the data passed to the function by the caller. The list serves as a formal declaration of the data types and names

FUNCTIONS:

A function is a self-contained program segment that carries out some specific well defined tasks.

Advantages of functions:

1. Write your code as collections of small functions to make your program modular
2. Structured programming
3. Code easier to debug
4. Easier modification
5. Reusable in other programs

Function Definition :

```
type function_name(parameter list )  
{  
Declarations;  
Statements;  
}
```

FUNCTION HEADER :

USER DEFINED FUNCTIONS:

Every program must have a main function to indicate where the program has to begin its execution. While it is possible to code any program utilizing only main function, it leads to a number of problems. The program may become too large and complex and as a result task of debugging, testing and maintaining becomes difficult. If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit, these subprograms called “functions” are much easier to understand debug and test.

There are times when some types of operation for calculation is repeated many times at many points throughout a program. For instance, we might use the factorial of a number at several points in the program. In such situations, we may repeat the program statements whenever they are needed. Another approach is to design a function that can be called and used whenever required.

The advantages of using functions are

1. It facilitates top-down modular programming.
2. The length of a source program can be reduced by using functions at appropriate places.
3. It is easy to locate and isolate a faculty function for further investigations.
4. A function may be used by many other programs.

A function is a self-contained block of code that performs a particular task. Once a function has been designed and packed it can be treated as a “black box” that takes some data from main program and returns a value. The inner details of the program are invisible to the rest of program.

The form of the C functions.

```
function-name ( argument list )  
argument declaration;  
{  
    local variable declarations;  
    executable statement 1;  
    executable    statement 2;  
    - - - - -  
    - - - - -  
    - - - - -
```



```

        return (expression) ;
    }

```

The return statement is the mechanism for returning a value to the calling function. All functions by default returns int type data. we can force a function to return a particular type of data by using a type specifier in the header.

A function can be called by simply using the function name in the statement.

INTER FUNCTION COMMUNICATION:

A function depending on whether arguments are present or not and whether a value is returned or not may belong to.

1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and return values.
4. Functions with no arguments and with return values.

1. Functions with no arguments and no return values :

When a function has no arguments, it does not receive any data from calling function. In effect, there is no data transfer between calling function and called function.

<pre> #include<stdio.h> int Factorial(); int main() { Factorial(); return 0; } int Factorial() { int i,n,fact=1; printf("\nEnter n value"); scanf("%d",&n); for(i=1;i<=n;i++) { fact=fact*i; } Printf("factorial of %d is %d",n,fact); return 0; } </pre>	<p>Output:</p> <p>Enter n value 5</p> <p>Factorial of 5 is 120</p>
--	---

2. Functions with arguments and no return values:

The function takes argument but does not return value.

The actual (sent through main) and formal(declared in header section) should match in number, type and order.

In case actual arguments are more than formal arguments, extra actual arguments are discarded. On other hand unmatched formal arguments are initialized to some garbage values.

<pre>#include<stdio.h> int Factorial(int); int main() { int n; printf("\nEnter n value"); scanf("%d",&n); Factorial(n); return 0; } int Factorial(int n) { int i,fact=1; for(i=1;i<=n;i++) { fact=fact*i; } Printf("factorial of %d is %d",n,fact); return 0; }</pre>	<p>Output:</p> <p>Enter n value 5</p> <p>Factorial of 5 is 120</p>
--	---

3. Functions with arguments and return values.

The called function takes arguments from the calling function and it returns result to the calling function.

<pre>#include<stdio.h> int Factorial(int); int main() { int n,fact; printf("\nEnter n value"); scanf("%d",&n); fact=Factorial(n); Printf("factorial of %d is %d",n,fact); return 0; } int Factorial(int n) {</pre>	<p>Output:</p> <p>Enter n value 5</p> <p>Factorial of 5 is 120</p>
--	---

<pre>int i,fact=1; for(i=1;i<=n;i++) { fact=fact*i; } return fact; }</pre>	
---	--

4. Functions with no arguments and return values:

The called function can't take arguments any from the calling function but it returns result to the calling function

<pre>#include<stdio.h> int Factorial(int); int main() { int fact; fact=Factorial(); Printf("factorial is %d",fact); return 0; } int Factorial() { int i,fact=1,n; printf("\nEnter n value"); scanf("%d",&n); for(i=1;i<=n;i++) { fact=fact*i; } return fact; }</pre>	<p>Output:</p> <p>Enter n value 5</p> <p>Factorial is 120</p>
--	--

STANDARD LIBRARY FUNCTIONS AND HEADER FILES:

C functions can be classified into two categories, namely, library functions and user-defined functions. Main is the example of user-defined functions. Printf and scanf belong to the category of library functions. The main difference between these two categories is that library functions are not required to be written by us where as a user-defined function has to be developed by the user at the time of writing a program. However, a user-defined function can later become a part of the c program library.

ANSI C Standard Library Functions :

The C language is accompanied by a number of library functions that perform various tasks. The ANSI committee has standardized header files which contain these functions.

Some of the Header files are:

<ctype.h>	character testing and conversion functions.
<math.h>	Mathematical functions
<stdio.h>	standard I/O library functions
<stdlib.h>	Utility functions such as string conversion routines memory allocation routines , random number generator,etc.
<string.h>	string Manipulation functions
<time.h>	Time Manipulation fu

MATH.H

The math library contains functions for performing common mathematical operations. Some of the functions are :

abs : returns the absolute value of an integer x

cos : returns the cosine of x, where x is in radians

exp: returns "e" raised to a given power

fabs: returns the absolute value of a float x

log: returns the logarithm to base e

log10: returns the logarithm to base 10

pow: returns a given number raised to another number

sin: returns the sine of x, where x is in radians

sqrt : returns the square root of x

tan : returns the tangent of x, where x is in radians

ceil : The ceiling function rounds a number with a decimal part up to the next highest integer (written mathematically as $\lceil x \rceil$)

floor : The floor function rounds a number with a decimal part down to the next lowest integer (written mathematically as $\lfloor x \rfloor$)

STRING.H

There are many functions for manipulating strings. Some of the more useful are:

strcat : Concatenates (i.e., adds) two strings

strcmp: Compares two strings

strcpy: Copies a string

strlen: Calculates the length of a string (not including the null)

strstr: Finds a string within another string

strtok: Divides a string into tokens (i.e., parts)

STDIO.H

printf: Formatted printing to stdout

scanf: Formatted input from stdin

fprintf: Formatted printing to a file

fscanf: Formatted input from a file

getc: Get a character from a stream (e.g, stdin or a file)

putc: Write a character to a stream (e.g, stdout or a file)

fgets: Get a string from a stream

fputs: Write a string to a stream

fopen: Open a file

fclose: Close a file

STDLIB.H

Atof: Convert a string to a double (not a float)

Atoi: Convert a string to an int

Exit: Terminate a program, return an integer value

free: Release allocated memory

malloc: Allocate memory

rand: Generate a pseudo-random number

system: Execute an external command

TIME.H

This library contains several functions for getting the current date and time.

Time: Get the system time

Ctime: Convert the result from time() to something meaningful

The following table contains some more header files:

<assert.h>	Contains the assert macro, used to assist with detecting logical errors and other types of bug in debugging versions of a program.
<complex.h>	A set of functions for manipulating complex numbers . (New with C99)

< <u>ctype.h</u> >	Contains functions used to classify characters by their types or to convert between upper and lower case in a way that is independent of the used <u>character set</u> (typically <u>ASCII</u> or one of its extensions, although implementations utilizing <u>EBCDIC</u> are also known).
< <u>errno.h</u> >	For testing error codes reported by library functions.
< <u>fenv.h</u> >	For controlling <u>floating-point</u> environment. (New with C99)
< <u>float.h</u> >	Contains defined constants specifying the implementation-specific properties of the <u>floating-point</u> library, such as the minimum difference between two different floating-point numbers (<code>_EPSILON</code>), the maximum number of digits of accuracy (<code>_DIG</code>) and the range of numbers which can be represented (<code>_MIN</code> , <code>_MAX</code>).
< <u>inttypes.h</u> >	For precise conversion between integer types. (New with C99)
< <u>iso646.h</u> >	For programming in <u>ISO 646</u> variant character sets. (New with NA1)
< <u>limits.h</u> >	Contains defined constants specifying the implementation-specific properties of the integer types, such as the range of numbers which can be represented (<code>_MIN</code> , <code>_MAX</code>).
< <u>locale.h</u> >	For <code>setlocale()</code> and related constants. This is used to choose an appropriate <u>locale</u> .
< <u>math.h</u> >	For computing common mathematical functions
< <u>setjmp.h</u> >	Declares the macros <code>setjmp</code> and <code>longjmp</code> , which are used for non-local exits
< <u>signal.h</u> >	For controlling various exceptional conditions
< <u>stdarg.h</u> >	For accessing a varying number of arguments passed to functions.

<u><stdbool.h></u>	For a boolean data type. (New with C99)
<u><stdint.h></u>	For defining various integer types. (New with C99)
<u><stddef.h></u>	For defining several useful types and macros.
<u><stdio.h></u>	Provides the core input and output capabilities of the C language. This file includes the venerable <u>printf</u> function.
<u><stdlib.h></u>	For performing a variety of operations, including conversion, <u>pseudo-random numbers</u> , memory allocation, process control, environment, signalling, searching, and sorting.
<u><string.h></u>	For manipulating several kinds of strings.
<u><tgmath.h></u>	For type-generic mathematical functions. (New with C99)
<u><time.h></u>	For converting between various time and date formats.
<u><wchar.h></u>	For manipulating wide streams and several kinds of strings using wide characters - key to supporting a range of languages. (New with NA1)
<u><wctype.h></u>	For classifying wide characters. (New with NA1)

SCOPE RULES FOR FUNCTIONS :

Variables defined within a function (including main) are local to this function and no other function has direct access to them. The only way to pass variables to function is as parameters. The only way to pass (a single) variable back to the calling function is via the return statement

Ex:

```
int func (int n)
{
printf("%d\n",b);
```

```

return n;
} //b not defined locally!
int main (void)
{
int a = 2, b = 1, c;
c = func(a);
return 0;
}

```

STORAGE CLASSES :

In 'C' a variable can have any one of four Storage Classes.

1. Automatic Variables
2. External Variables
3. Static Variables
4. Register Variables

SCOPE :

The Scope of variable determines over what parts of the program a variable is actually available for use.

LONGEVITY :

Longevity refers to period during which a variable retains a given value during execution of a program (alive). So Longevity has a direct effect on utility of a given variable.

The variables may also be broadly categorized depending on place of their declaration as internal(local) or external(global). Internal variables are those which are declared within a particular function, while external variables are declared outside of any function.

AUTOMATIC VARIABLES :

They are declared inside a function in which they are to be utilized. They are created when function is called and destroyed automatically when the function is exited, hence the name automatic. Automatic Variables are therefore private (or local) to the function in which they are declared. Because of this property, automatic variables are also referred to as local or internal variables.

By default declaration is automatic. One important feature of automatic variables is that their value cannot be changed accidentally by what happens in some other function in the program.

<pre>#include<stdio.h> main() { int m=1000; func2(); printf("%d\n",m); } func1() { int m=10; printf("%d\n",m); } func2() { int m=100; func1(); printf("%d",m); }</pre>	<pre>Output: 10 100 1000</pre>
--	--------------------------------

First, any variable local to main will normally live throughout the whole program, although it is active only in main.

Secondly, during recursion, nested variables are unique auto variables, a situation similar to function nested auto variables with identical names.

EXTERNAL VARIABLES :

Variables that are both alive and active throughout entire program are known as external variables. They are also known as Global Variables. In case a local and global have same name local variable will have precedence over global one in function where it is declared.

<pre>#include<stdio.h> int x; main() { x=10; printf("%d",x); printf("x=%d",fun1()); printf("x=%d",fun2()); printf("x=%d",fun3()); }</pre>	<pre>fun1() { x=x+10; return(x); } fun2() { int x; x=1; return(x); }</pre>	<pre>Output;</pre>
---	--	--------------------

}	} fun3() { x=x+10; return(x); }	
---	--	--

An extern within a function provides the type information to just that one function.

STATIC VARIABLES :

The value of Static Variable persists until the end of program. A variable can be declared Static using Keyword Static like Internal & External Static Variables are differentiated depending whether they are declared inside or outside of auto variables, except that they remain alive throughout the remainder of program.

<pre>#include<stdio.h> main() { int I; for (I=1;I<=3;I++) stat(); } stat() { static int x=0; x=x+1; printf("x=%d\n",x); }</pre>	Output: X=1 X=2 X=3
--	------------------------------

REGISTER VARIABLES :

We can tell the Compiler that a variable should be kept in one of the machines registers, instead of keeping in the memory. Since a register access is much faster than a memory access, keeping frequently accessed variables in register will lead to faster execution

Syntax: register int Count.

RECURSIVE FUNCTIONS :

Powerful concept in the development of programs is the ability for a function to refer to itself to solve a problem.

This control technique, called recursion is an important concept in computer science and is convenient for a variety of problems that would be difficult to solve using iterative constructs such as for, while and do-while loops. Recursion is used extensively in many of the data structures and algorithms in the present text. Recursive functions can be directly implemented in C language.

In mathematics and computer science, recursion means self reference. A recursion function therefore, is a function whose definition is based upon itself. In other words, a function containing either a call statement to itself or a call statement to another function that may eventually result in a call statement back to the original function, then that function is called a recursive function.

In order that the program should not continue indefinitely, a recursive function must have the following properties:-

1. There must be certain criteria, for which the function does not call itself.
2. Each time the function does call itself (directly or indirectly), it must be closer to the base criteria, i.e., it uses an argument(s) closer than the one it was given.

A recursive function with these two properties is said to be well-defined. Suppose P is a recursion function. During the execution of a program, which contains P, we associate level number with each execution of P is executed because of recursion call, its level is one more than the level of execution that had made the recursive call. The depth of a recursive function P with given set of arguments of arguments refers to the maximum level number of P during its execution.

Advantages of Recursion:-

The various advantages of recursion are:-

1. It is simple, easily understandable, concise, compact and transparent to view a program.
 2. It is very useful in solving mathematical(factorial, GCD, Fibonacci series, etc.), trigonometric), logical games(tower of Hanoic, generating random number in system simulation, puzzles) and algebraic problems(to solve Ackermann's function).
 3. Lesser number of programming statements (coding) required with the use of recursion.
 4. It is very useful if the solution to problem is in recursive terms.
 5. It is very useful where user-defined functions or sub-program or procedure required,
-

6. It is very useful in the multiprocessing and multitasking environment.
7. It is very useful in solving the data structure problems like linked list, queues, stack, tree, quick sort and merge sort etc.
8. Recursion saves the memory. In other words it utilizes the memory well.

Limitations of the recursion:

1. It requires more memory to store the automatic variable to solve variable to solve the memory space i.e., to consume more storage space.
2. If properly recursion procedure is not checked, then it will create a problem for the processing and procedure and procedure run out of memory.
3. In some problems, it is a time consuming process and is not efficient.
4. It will create indefinite looping process(non – terminating iteration), if condition not be applied at the proper.

Example:

Factorial of a given number using recursion

<pre>#include<stdio.h> int rFactorial(int); int main() { int n,fact; printf("\nEnter n value"); scanf("%d",&n); fact=rFactorial(n); Printf("factorial of %d is %d",n,fact); return 0; } int Factorial(int n) { int i,fact=1; if(n<2) return 1; else { for(i=n;i>=1;i--) { fact=return (i*(rFactorial(i-1))) } } }</pre>	<p>Output:</p> <p>Enter n value 5</p> <p>Factorial of 5 is 120</p>
--	---

UNIT -IV

Arrays – Concepts, using arrays in C, arrays and functions, array applications, two – dimensional arrays, multidimensional arrays, C program examples.

ARRAYS :

An array is a group of related data items that share a common name.

Ex:- Students

The complete set of students are represented using an array name students. A particular value is indicated by writing a number called index number or subscript in brackets after array name. The complete set of value is referred to as an array, the individual values are called elements.

- For example, suppose we have a temperature reading for each day of the year and need to manipulate their values several times within a program
- With simple variables,
- We would need to declare 365 variables.
- We would not be able to loop over these variables
- The above problem can be solved by using an array: Temp [365]
- The elements in an array share the same name and are distinguished from one another by their numbers or subscripts: 0,1,2..... Size-1
- An array must be declared before it can be used.
- Array declaration and definition provides information to the compiler - the name of the array, type of array and the size.

CHARACTERISTICS OF AN ARRAY

- An array represents a group of related data.
- An array has two distinct characteristics:
- An array is ordered: data is grouped sequentially: element 0, element 1,... n-1
- An array is homogenous: every value within the array must share the same data type.

In other words, an int array can only hold integers, but not other data types.

HOW TO CREATE AN ARRAY

- Before we create an array, we need to decide on two properties:

- **Element type:** what kind of data will the array hold? int, double, char? Remember, we can only choose one type.
- **Array size:** how many elements will the array contain?

TYPES OF ARRAYS

- Arrays can be classified based on how the data items will be arranged.
- Arrays are broadly classified into three categories.
 1. One dimensional arrays
 2. Two dimensional arrays
 3. Multi dimensional arrays

ONE DIMENSIONAL ARRAY:

- One dimensional array is a linear list consisting of related and similar data items.
- In memory, all the data items are stored in contiguous memory locations.

Declaring a one dimensional array

- To declare ordinary variables, we use the following notation:

int number;

- To declare an array, the following syntax can be used:

Syntax: type arrayname[size];

For example:

int number[5];

- Creates an array of 5 integer elements.

Initializing

- There are four options to initialize one dimensional arrays.

Option A: Basic Initialization

- If the values of all the data elements are known, the values must be specified within the braces.

int temp [5] = {75, 79, 82, 70, 68};

Memory Layout

temp[0] temp[1] temp[2] temp[3] temp [4]

75	79	82	70	68
----	----	----	----	----

2. Inputting values

- Another way to fill the array is to read the values from the keyboard or a file.
- This could be done by using a loop.

Ex:

```
for(i=0; i<9; i++)
    scanf("%d",&scores[i]);
```

3. Accessing elements of an array

- Index is used to access individual elements of an array.

Ex: scores[0];

4. Printing/Output values

- Another common application is printing the contents of an array.
- This can be easily done by using a loop.

Ex.

```
for(i=0; i<9; i++)
    printf("%d",scores[i]);
```

ARRAYS AND FUNCTIONS:

- To process arrays in a large program, arrays are required to be passed to functions.
- Arrays can be passed to functions in two ways:

Passing Individual Elements:

- Individual elements can be passed to a function either by passing the data values or by passing the addresses

Passing the whole Array

- By using this, an Array name can be passed to the called function

C program to pass a single element of an array to function

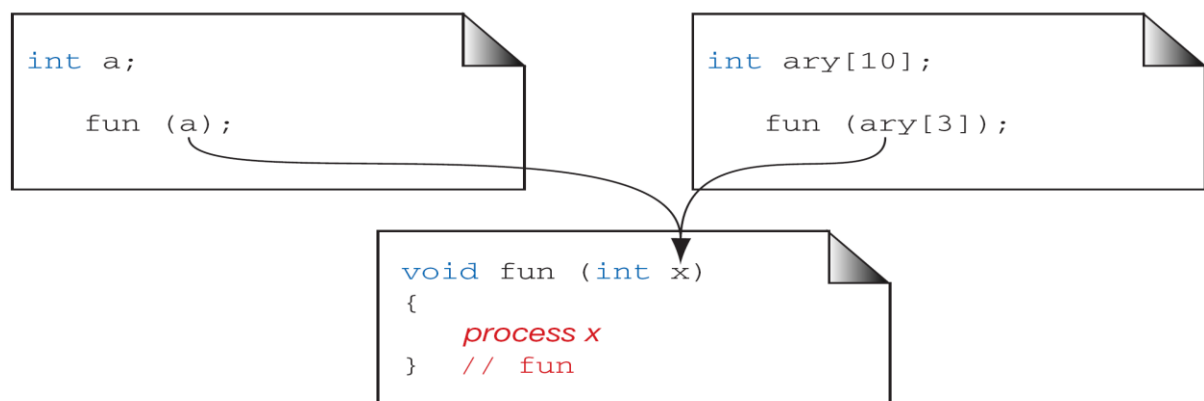
```
#include<stdio.h>
float average(float a[]);
int main(){
    float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
    avg=average(c); /* Only name of array is passed as argument. */
    printf("Average age=%.2f",avg);
```

```

return 0;
}
float average(float a[]){
int i;
float avg, sum=0.0;
for(i=0;i<6;++i){
    sum+=a[i];
}
avg =(sum/6);
return avg;
}

```

Passing Array Elements



EXAMPLE PROGRAMS ON ONE DIMENSIONAL ARRAY:

- 1) Write a c program to find largest and smallest values in a list of array

<pre> #include<stdio.h> #include<conio.h> int main() { int i,n,x,y; clrscr(); printf("enter no.of elements "); scanf("%d",&n); </pre>	<p>Output:</p> <pre> enter no.of elements 5 enter 0 value :10 enter 1 value :20 enter 2 value :30 enter 3 value :40 </pre>
---	---

<pre> int a[n]; for(i=0;i<n;i++) { printf("\n enter %d value:"); scanf("%d",&a[i]); } x=max(a,n); printf("the largest value is %d",x); y=small(a,n); printf("\nthe smallet value is %d",y); } int max(int arr[],int n) { int largest,i; largest=arr[0]; for(i=0;i<n;i++) { if(arr[i]>largest) { largest=arr[i]; } } return(largest); } int small(int arr[],int n) { int smallest,i; smallest=arr[0]; for(i=0;i<n;i++) { if(arr[i]<smallest) { smallest=arr[i]; } } return(smallest); } </pre>	<p>enter 4 value :50</p> <p>the largest value is 50 the smallet value is10</p>
--	--

2) Write a c program to sort the array elements in the ascending order

<pre> #include<stdio.h> #include<conio.h> void main() { int i,j,temp,n,a[20]; clrscr(); printf("Enter the number of elements:"); scanf("%d",&n); </pre>	<p>Output:</p> <p>Enter the number of elements:5</p> <p>Enter the elements:1 3 6 5 4</p> <p>Elements after sorting:</p> <p>1</p> <p>3</p> <p>4</p> <p>5</p> <p>6</p>
---	--

```

printf("\nEnter the elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n-1;i++)
for(j=i+1;j<n;j++)
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
printf("\nElements after sorting:");
for(i=0;i<n;i++)
printf("\n%d",a[i]);
getch();
}

```

3) Write a search an element in a list of array elements

```

#include<stdio.h>
#include<conio.h>
int main()
{
int i,n,a[20],pos;
clrscr();
printf("Enter the number of elements:");
scanf("%d",&n);
printf("\nEnter the elements:");
for(i=0;i<n;i++)
{
printf("enter element %d",i);
scanf("%d",&a[i]);
}
printf("enter the target element to be searched");
scanf("%d",&target);
pos=linearsearch(a[],n,target);
if(pos== -1)
printf("\n the target is not found in the list");
else
printf("\n the target %d is found at position
%d",target,pos);
return 0;
}
int linearsearch(int a[],int n, int target)
{
int i,j,temp;
for(i=0;i<n-1;i++)
{
if(a[i]==target)

```

Output:
Enter the number of elements:5
Enter the elements:1 3 6 5 4
Enter key
4
Key found at 3 position

<pre> return (i); } return -1; } </pre>	
--	--

APPLICATIONS OF ARRAYS :

1. Frequency arrays with their graphical representations.
2. Histograms
3. Random number permutations.

FREQUENCY ARRAYS :

Two common statistical applications that use arrays are frequency distributions and histograms. A frequency array shows the number of elements with an identical value found in a series of numbers.

For example ,suppose we have taken a sample of 100 values between 0 and 19.We want to know how many of the values are 0,how many are 1,how many are 2,and so forth up through 19.

We can read these numbers into an array called numbers. Then we create an array of 20 elements that will show the frequency of each number in the series.

One way to do it is to assign the value from the data array to an index and then use the index to access the frequency array.

```

F=numbers[i];
Frequency[F]++;

```

Since an index is an expression ,however ,we can simply use the value from our data array to index us in to the frequency array .The value of numbers[i] is determined first ,and then that value is used to index in to frequency.

```

Frequency[numbers[i]]++;

```

HISTOGRAMS:

A histogram is a pictorial representation of a frequency array .Instead of printing the values of the elements to show the frequency of each number, we print a histogram in the form of a bar chart. For example ,the following figure is a histogram for a set of numbers in the range 0 ...19.In this example, asterisks (*) are used to build the bar. Each

asterisk represents one occurrence of the data value.

0	0	
1	4 * * * *	—————(four 1s)
2	7 * * * * * *	
3	7 * * * * * *	—————(seven 3s)
.		
.		
.		
18	2 * *	
19	0	—————(zero 19s)

RANDOM NUMBER PERMUTATIONS :

A random number permutation is a set of random numbers in which no numbers are repeated. For example, given a random number permutation of 10 numbers, the values from 0 to 9 would all be included with no duplicates.

Generating random numbers:

To generate a random integral in a range x to y, we must first scale the number and then, if x is greater than 0, shift the number within the range. We scale the number using the modulus operator.

Ex: To produce a random number in the range 0 ...50, we simply scale the random number and scaling factor must be one greater than the highest number needed.

$\text{rand}() \% 51$

modulus works well when our range starts at 0. for example, suppose we want a random number between 10 and 20. for this we will use one formula

$\text{range} = (20 - 10) + 1;$

$\text{randno} = \text{rand}() \% \text{range} + 10;$

To generate a permutation, we need to eliminate the duplicates. Using histogram concept we can solve the problem most efficiently by using two arrays. The first array contains the random numbers. The second array contains a logical value that indicates whether or not the number represented by its index has been placed in the random number array.

Only the first five random numbers have been placed in the permutation. For each random number in the random number array, its corresponding location in the have-random array is

set to 1. Those locations representing numbers that have not yet been generated are still set to 0.

TWO – DIMENSIONAL ARRAYS:

- An array of arrays is called multi-dimensional array.
- A multidimensional array can have two dimensions, three dimensions, four dimensions, etc.
- A Multi-dimensional array which has only two dimensions is called as a two dimensional array.
- Two-dimensional array can be regarded as a table with rows and columns:

Example:

'J' 'o' 'h' 'n'	is a 3 × 4 array: 3 rows, 4 columns	'J' 'o' 'h' 'n'
'M' 'a' 'r' 'y'		'M' 'a' 'r' 'y'
'I' 'v' 'a' 'n'		'I' 'v' 'a' 'n'

- This is a two dimensional array having 3 rows and 4 columns

DECLARATION

Syntax:

- `Type arrayName [rows][cols];`

Example

- `char names[3][4];`

In the above declaration

- **Char (elementType)** → specifies type of element in each slot
- **Names (arrayName)** → specifies name of the array
- **[3] (rows)** → specifies number of rows
- **[4] (cols)** → specifies number of columns

INITIALIZATION:

- A two dimensional array can be initialized at the time of declaration:

```
char names [3][4] = {    {'J', 'o', 'h', 'n'},  
                        {'M', 'a', 'r', 'y'},  
                        {'I', 'v', 'a', 'n'}  
};
```

- An integer array can be initialized to all zeros by using the following statement:
int nums[3][4] = {0};
- The declaration of a two dimensional array should have the column size so that ,the elements can be in the form of rows and columns.

To access an element of a 2D array, requires both the row and the column.

int m[3][5] = {{0},{0},{0}};

Examples :

- 1) Write a c program to perform the following operations.
 - a) Addition of 2 matrices
 - b) Multiplication of 2 matrices.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[100][100],b[100][100],c[100][100];
int i,j,n,m;
clrscr();
printf("enter row size of the matrix(<100)");
scanf("%d",&n);
printf("enter column size of the matrix(<100)");
scanf("%d",&m);
printf("enter the matrix a");
for(i=0;i<n;i++)
{
For(j=0;j<m;j++)
{
scanf("%d",&a[i][j]);
}
}
Printf("enter the matrix b");
For(i=0;i<n;i++)
{
For(j=0;j<m;j++)
{
Scanf("%d",&b[i][j]);
}
}
Printf("MATRIX a+MATRIX b");
For(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
}
```

```

Printf("addition of two matrices is\n");
For(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
Printf("%d\t",c[i][j]);
}
}
return 0;
}

```

b) Multiplication of two matrices:

```

#include<stdio.h>
#include<stdio.h>
int main()
{
int a[100][100],b[100][100],c[100][100];
int i,j,n,m;
clrscr();
printf("enter the row size,column size(<100)");
scanf("%d%d",&n,&m);
printf("enter the matrix a");
for(i=0;i<n;i++)
{
For(j=0;j<m;j++)
{
Scanf("%d",&a[i][j]);
}
}
Printf("enter matrix b");
For(i=0;i<n;i++)
{
For(j=0;j<m;j++)
{
Scanf("%d",&a[i][j]);
}
}
Printf("MATRIX a*MATRIX b");
For(i=0;i<n;i++)
{
For(j=0;j<m;j++)
{
C[i][j]=0;
For(k=0;k<n;k++)
{
C[i][j]=c[i][j]+(a[i][k]*b[k][j]);
}
}
}
}

```

```

    }
    Printf("Resultant matrix is");
    For(i=0;i<n;i++)
    {
        For(j=0;j<m;j++)
        {
            Printf("%d\t",c[i][j]);
        }
    }
    return 0;
}

```

MULTI-DIMENSIONAL ARRAYS:

C allows arrays of three or more dimensions. The exact limit is determined by Compiler.

type array-names[s1][s2][s3] - - - - [s_n];

where s_i is size of dimension.

Ex:- int Survey[3][5][2];

```

#include <stdio.h>
void printArray( const int a[][ 3 ] );
int main()
{
    int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
    int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
    int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
    printf( "Values in array1 by row are:\n" );
    printArray( array1 );
    printf( "Values in array2 by row are:\n" );
    printArray( array2 );
    printf( "Values in array3 by row are:\n" );
    printArray( array3 );
    return 0;
}
void printArray( const int a[][ 3 ] )
{
    int i;
    int j;
    for ( i = 0; i <= 1; i++ ) {
        for ( j = 0; j <= 2; j++ ) {
            printf( "%d ", a[ i ][ j ] );
        }
        printf( "\n" );
    }
}

```


UNIT - V

Pointers – Introduction (Basic Concepts), Pointers for inter function communication, pointers to pointers, compatibility, void pointer, null pointer.

Pointer Applications - Arrays and Pointers, Pointer Arithmetic and arrays, passing an array to a function.

Memory allocation functions – malloc(), calloc(), realloc(), free().

Array of pointers, pointers to functions, C program examples.

POINTERS :

One of the powerful features of C is ability to access the memory variables by their memory address. This can be done by using Pointers. The real power of C lies in the proper use of Pointers.

A pointer is a variable that can store an address of a variable (i.e., 112300). We say that a pointer points to a variable that is stored at that address. A pointer itself usually occupies 4 bytes of memory (then it can address cells from 0 to 232-1).

Advantages of Pointers :

1. A pointer enables us to access a variable that is defined outside the function.
2. Pointers are more efficient in handling the data tables.
3. Pointers reduce the length and complexity of a program.
4. They increase the execution speed.

Definition :

A variable that holds a physical memory address is called a pointer variable or Pointer.

Declaration and initialization of pointers :

Datatype * Variable-name;

Eg:-int *ad; /* pointer to int */

char *s; /* pointer to char */

float *fp; /* pointer to float */

char **s; /* pointer to variable that is a pointer to char */

A pointer is a variable that contains an address which is a location of another variable in memory.

Consider the Statement

p=&i;

Here '&' is called address of a variable. 'p' contains the address of a variable i.

The operator & returns the memory address of variable on which it is operated, this is called Referencing.

The * operator is called an indirection operator or dereferencing operator which is used to display the contents of the Pointer Variable.

Consider the following Statements :

Int x;

int *p; //declaration of pointer

x =5;

p= &x; //initialization of pointer

Assume that x is stored at the memory address 2000. Then the output for the following printf statements is :

	Output
Printf("The Value of x is %d",x);	5
Printf("The Address of x is %u",&x);	2000
Printf("The Address of x is %u",p);	2000
Printf("The Value of x is %d",*p);	5
Printf("The Value of x is %d",&x);	5

Accessing a Variable through its pointer:

Once a pointer has been assigned address of a variable then access the value of the variable can be done by using *(indirection operator).

Ex:

int x,*p,n;

x=25;

p=&x;

n=*p;

- 1) first line declares x and n integer variables and p as pointer variable
- 2) 2nd line declares value assigned to x.
- 3) 3rd line assigns the address of x to the p.
- 4) 4th line indicates indirection operator(*) before placed pointer variable then the pointer returns the value of variable.

Example:

```
main()
{
int x,*p,n;
  x=25;
  p=&x;
  n=*p;
  printf("value of x is %d",x);
  printf("%d is stored at address %u",x,&x);
  printf("%d is stored at address %u",&x,&x);
  printf("%d is stored at address %u",*p,p);
  printf("%d is stored at address %u",y,&x);
  getch();
}
```

Output;

value of x is 10

10 is stored at address 1000

10 is stored at address 1000

10 is stored at address 1000

10 is stored at address 1000

POINTERS TO POINTERS :

So far ,all pointers have been pointing directly to data.It is possible and with advanced data structures often necessary to use pointers to that point to other pointers.

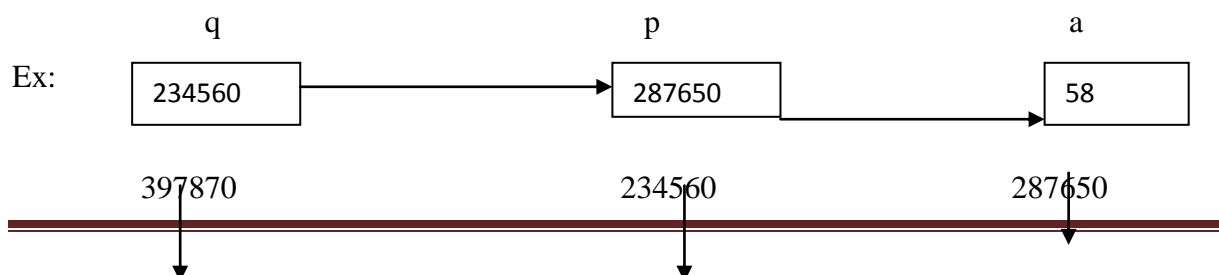
For example,we can have a pointer pointing to a pointer to an integer.This two level indirection is seen as below:

//Local declarations

int a;

int* p;

int **q;



pointer to pointer to integer

pointer to integer

integer variable

//statements

a=58;

p=&a;

q=&p;

printf(“%3d”,a);

printf(“%3d”,*p);

printf(“%3d”,**q);

There is no limit as to how many level of indirection we can use but practically we seldom use more than two. Each level of pointer indirection requires a separate indirection operator when it is dereferenced .

In the above figure to refer to ‘a’ using the pointer ‘p’, we have to dereference it as shown .

*p

To refer to the variable ‘a’ using the pointer ‘q’ ,we have to dereference it twice to get to the integer ‘a’ because there are two levels of indirection(pointers) involved.If we dereference it only once we are referring ‘p’ which is a pointer to an integer .Another way to say this is that ‘q’ is a pointer to a pointer to an integer.The double dereference is shown below:

**q

In above example all the three references in the printf statement refer to the variable ‘a’.

The first printf statement prints the value of the variable ‘a’ directly,second uses the pointer ‘p’,third uses the pointer ‘q’.The result is the value 58 printed 3 times as below

58 58 58

Pointers for inter-function communication

POINTERS AND ARRAYS :

Integer Scale Factor=2

Character Scale Factor=1

Float Scale Factor=4

Double Scale Factor=8

When an array is declared, elements of array are stored in contiguous locations. The address of the first element of an array is called its base address.

Consider the array

2000	2002	2004	2006	2008
a[0]	a[1]	a[2]	a[3]	a[4]

The name of the array is called its base address.

i.e., a and &a[0] are equal.

Now both a and a[0] points to location 2000. If we declare p as an integer pointer, then we can make the pointer P to point to the array a by following assignment

P = a;

We can access every value of array a by moving P from one element to another.

i.e., P points to 0th element
P+1 points to 1st element
P+2 points to 2nd element
P+3 points to 3rd element
P +4 points to 4th element

Reading and Printing an array using Pointers:

```
main()
{
    int *a,i;
    printf("Enter five elements:");
    for (i=0;i<5;i++)
        scanf("%d",&a[i]);
    printf("The array elements are:");
    for (i=0;i<5;i++)
        printf("%d", *(a+i));
}
```

In one dimensional array, a[i] element is referred by

(a+i) is the address of ith element.

* (a+i) is the value at the ith element.

In two-dimensional array, a[i][j] element is represented as

((a+i)+j)

POINTERS AND FUNCTIONS :

Parameter passing mechanism in 'C' is of two types.

1.Call by Value

2.Call by Reference.

The process of passing the actual value of variables is known as Call by Value. The process of calling a function using pointers to pass the addresses of variables is known as Call by Reference. The function which is called by reference can change the value of the variable used in the call.

Example of Call by Value:

```
#include <stdio.h>

void swap(int,int);

main()
{
    int a,b;
    printf("Enter the Values of a and b:");
    scanf("%d%d",&a,&b);
    printf("Before Swapping \n");
    printf("a = %d \t b = %d", a,b);
    swap(a,b);
    printf("After Swapping \n");
    printf("a = %d \t b = %d", a,b);
}

void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

Example of Call by Reference:

```
#include<stdio.h>

main()
```

```

    {
        int a, b;
        a = 10;
        b = 20;
        swap (&a, &b);
        printf("After Swapping \n");
        printf("a = %d \t b = %d", a, b);
    }
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

```

NULL POINTER :

'Zero' is a special value we can assign to a pointer which does not point to anything most frequently, a symbolic constant NULL is used. It is guaranteed, that no valid address is equal to 0. The bit pattern of the NULL pointer does not have to contain all zeros usually it does or it depends on the processor architecture. On many machines, dereferencing a NULL pointer causes a segmentation violation.

NULL ptr is not the same as an EMPTY string.

```

const char* psz1 = 0;
const char* psz2 = "";
assert(psz1 != psz2);

```

Always check for NULL before dereferencing a pointer.

```

if (psz1)
    /* use psz1 */
sizeof(psz1)    //doesn't give you the number of elements in psz1. Need
additional size variable.

```

VOID POINTER :

In C ,an additional type void *(void pointer) is defined as a proper type for generic pointer. Any pointer to an object may be converted to type void * without loss of information. If the result is converted back to the original type ,the original pointer is recovered .

Ex:

```
main()
{
    void *a;
    int n=2,*m;
    double d=2.3,*c;
    a=&n;
    m=a;
    printf("\n%d %d %d",a,*m,m);
    a=&d;
    c=a;
    printf("\n%d %3.1f %d",a,*c,c);
}
```

In the above program a is declared as a pointer to void which is used to carry the address of an int(a=&n)and to carry the address of a double(a=&d) and the original pointers are recovered with out any loss of information.

DYNAMIC MEMORY ALLOCATION :

Dynamic memory allocation uses predefined functions to allocate and release memory for data while the program is running. It effectively postpones the data definition ,but not the declaration to run time.

To use dynamic memory allocation ,we use either standard data types or derived types .To access data in dynamic memory we need pointers.

MEMORY ALLOCATION FUNCTIONS:

Four memory management functions are used with dynamic memory. Three of them, malloc, calloc, and realloc,are used for memory allocation. The fourth , free is used to return memory

when it is no longer needed. All the memory management functions are found in standard library file(stdlib.h).

BLOCK MEMORY ALLOCATION(MALLOC) :

The malloc function allocates a block of memory that contains the number of bytes specified in its parameter. It returns a void pointer to the first byte of the allocated memory. The allocated memory is not initialized.

Declaration:

```
void *malloc (size_t size);
```

The type size_t is defined in several header files including Stdio.h. The type is usually an unsigned integer and by the standard it is guaranteed to be large enough to hold the maximum address of the computer. To provide portability the size specification in malloc's actual parameter is generally computed using the sizeof operator. For example if we want to allocate an integer in the heap we will write like this:

```
Pint=malloc(sizeof(int));
```

Malloc returns the address of the first byte in the memory space allocated. If it is not successful malloc returns null pointer. An attempt to allocate memory from heap when memory is insufficient is known as overflow.

The malloc function has one or more potential error if we call malloc with a zero size, the results are unpredictable. It may return a null pointer or it may return someother implementation dependant value.

Ex:

```
If(!(Pint=malloc(sizeof(int))))  
    // no memory available  
    exit(100);  
    //memory available
```

...

In this example we are allocating one integer object. If the memory is allocated successfully,ptr contains a value. If does not there is no memory and we exit the program with error code 100.

Example:

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using malloc() function.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int n,i,*ptr,sum=0;
printf("Enter number of elements: ");
scanf("%d",&n);
ptr=(int*)malloc(n*sizeof(int)); //memory allocated using malloc
if(ptr==NULL)
{
printf("Error! memory not allocated.");
exit(0);
}
printf("Enter elements of array: ");
for(i=0;i<n;++i)
{
scanf("%d",ptr+i);
sum+=*(ptr+i);
}
printf("Sum=%d",sum);
free(ptr);
return0;
}
```

CONTIGIOUS MEMORY ALLOCATION(calloc) :

Calloc is primarily used to allocate memory for arrays. It differs from malloc only in that it sets memory to null characters. The calloc function declaration:

Void *calloc(size_t element_count, size_t element_size);

The result is the same for both malloc and calloc.

calloc returns the address of the first byte in the memory space allocated. If it is not successful calloc returns null pointer.

Example:

```
If(!(ptr=(int*)calloc(200,sizeof(int))))  
    //no memory available  
    exit(100);  
    //memory available  
...
```

In this example we allocate memory for an array of 200 integers.

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using calloc() function.

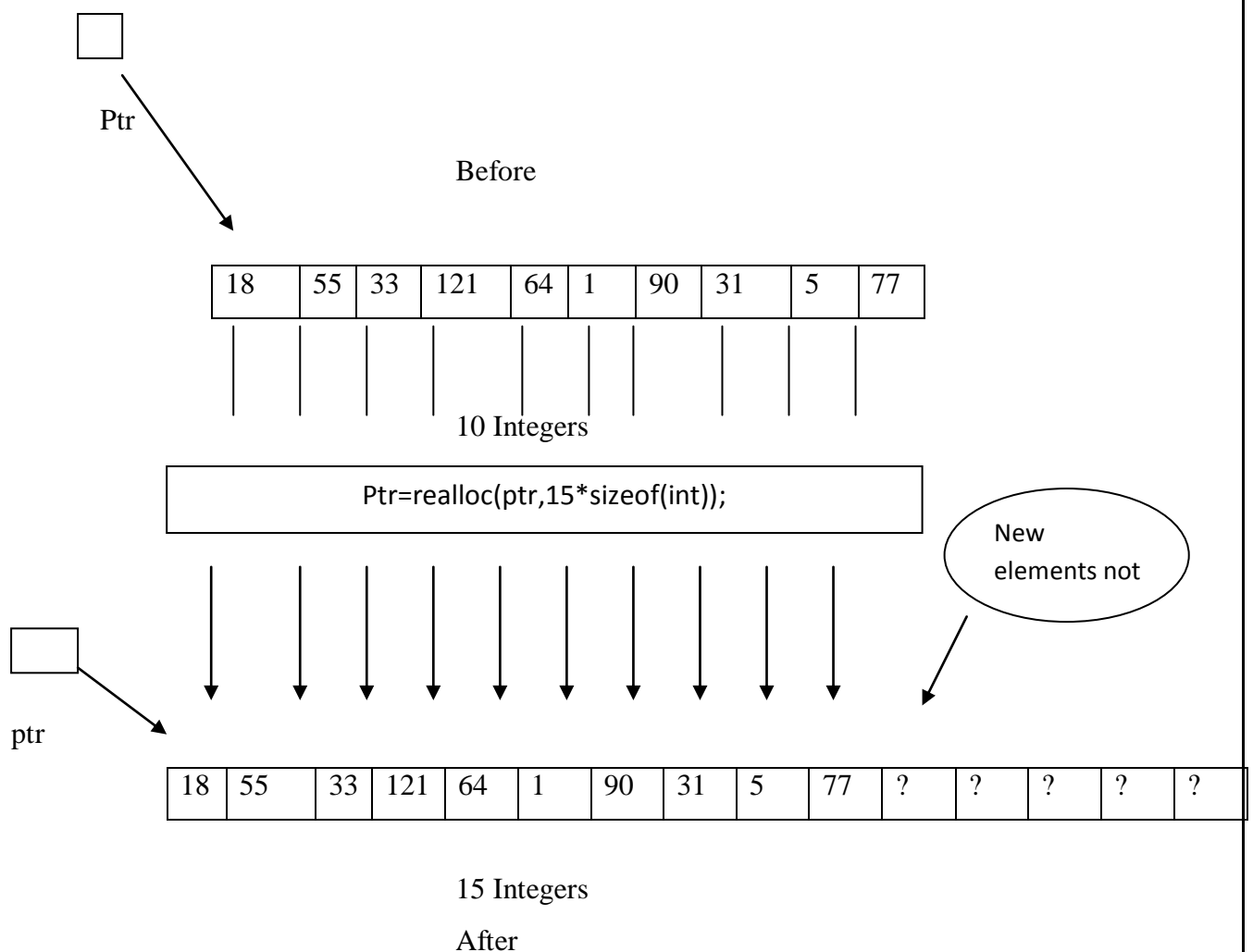
```
#include<stdio.h>  
#include<stdlib.h>  
int main(){  
    int n,i,*ptr,sum=0;  
    printf("Enter number of elements: ");  
    scanf("%d",&n);  
    ptr=(int*)calloc(n,sizeof(int));  
    if(ptr==NULL)  
    {  
        printf("Error! memory not allocated.");  
        exit(0);  
    }  
    printf("Enter elements of array: ");  
    for(i=0;i<n;++i)  
    {  
        scanf("%d",ptr+i);  
        sum+=*(ptr+i);  
    }  
    printf("Sum=%d",sum);  
    free(ptr);  
}
```

```
return 0;
}
```

REALLOCATION OF MEMORY(realloc):

The realloc function can be highly inefficient and therefore should be used advisedly. When given a pointer to a previously allocated block of memory realloc changes the size of the block by deleting or extending the memory at the end of the block. If the memory can not be extended because of other allocations realloc allocates completely new block, copies the existing memory allocation to the new location, and deletes the old allocation.

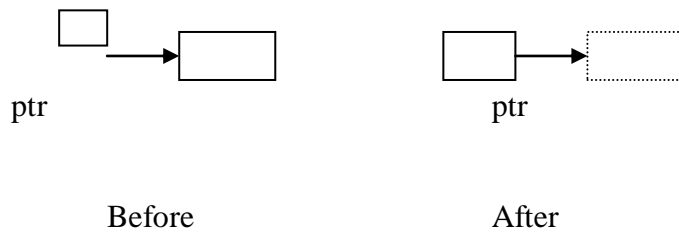
```
Void *realloc(void*ptr,size_tnewsize);
```



Releasing Memory(free): When memory locations allocated by malloc, calloc or realloc are no longer needed, they should be freed using the predefined function free. It is an error to free memory with a null pointer, a pointer to other than the first element of an allocated block, a

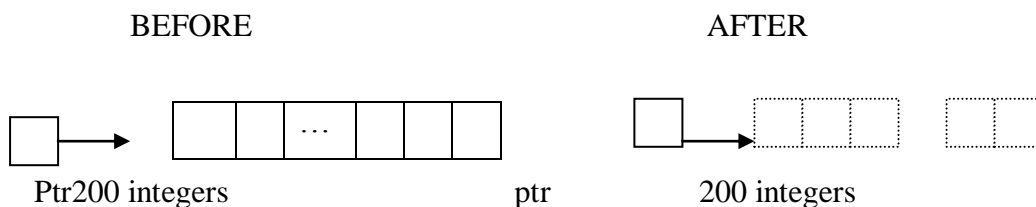
pointer that is a different type then the pointer that allocated the memory, it is also a potential error to refer to memory after it has been released

Void free(void *ptr);



Free(ptr);

In above example it releases a single element to allocated with a malloc,back to heap.



```
#include<stdio.h>
#include<stdlib.h>
int main(){
int *ptr,i,n1,n2;
printf("Enter size of array: ");
scanf("%d",&n1);
ptr=(int*)malloc(n1*sizeof(int));
printf("Address of previously allocated memory: ");
for(i=0;i<n1;++i)
printf("%u\t",ptr+i);
printf("\nEnter new size of array: ");
scanf("%d",&n2);
ptr=realloc(ptr,n2);
for(i=0;i<n2;++i)
printf("%u\t",ptr+i);
return0;
```

```
}
```

Free(ptr);

In the above example the 200 elements were allocated with calloc. When we free the pointer in this case, all 200 elements are return to heap. First, it is not the pointers that are being released but rather what they point to. Second , To release an array of memory that was allocated by calloc() , we need only release the pointer once. It is an error to attempt to release each element individually.

Releasing memory does not change the value in a pointer. Still contains the address in the heap. It is a logic error to use the pointer after memory has been released.

Note: Memory space should be freed whenever it is not required. The operating system will release all memory when your program terminates

15. ADDITIONAL TOPICS

Basics of UNIX.

An operating system (OS) is a set of programs that act as an interface between the user and the computer. The OS manages computer hardware and system resources. Some popular operating systems are: Linux, UNIX, MS-DOS, MS Windows, Apple Mac. Ken Thompson and Dennis Ritchie developed a small, general purpose operating system UNIX in 1969. In 1973, they rewrote OS in C. In 1974, UNIX was licenced to universities for educational purposes and made commercially available later. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops. There are many different versions of UNIX and they share similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.

Some essential unix commands:

Command	Action
login: <username>	To login to the user account – enter password at the prompt.
<ctrl>d or logout	To log out from the system.
Ls	To have list of files.

cd <path>	To change the directory
mkdir <path>	To create a directory
rm <filename>	To remove a file
cp <source><dest>	To copy source file to destination.
mv <source><dest>	To move source file to destination.

File Redirections

Linux / Ubuntu operating systems allows an user to redirect the contents of one file to another by using a simple CAT Command

Syntax of cat command

User1@localhost \$: cat source_filename1 >> destination_filename

Note: User1@localhost \$: is a prompt of the user logged in.

Example

\$: cat even_odd.c >> week1

To check the contents of the file

\$: vi week1

Output Redirection

If the user wants to print the program with output of the program to the line printer (lpr) the following procedure has to be adopted.

Step 1: Compile the already written C Program.

Eg: cc even_odd.c

Step 2: Once the file has successfully compiled without any errors then redirect the output to the same file to which the program has been redirected

Eg: cat ./a.out >> week1

Note: Once the user redirects the output the terminal will wait for a valid input from the user same as given for the program

Step 3: Redirect the file to the Line Printer using Line printing command

File Printing Commands

\$: lpr -u username -p filename

(or)

\$: lpr -u username -p filename.c

Note: If the above command is giving the output as “**Not Allowed to Print**” then the printer specified for your username is not been active.

b) Introduction to Visual Editors in Linux with basic commands.

To create files (documents and programs), we use text editors. A text editor is a program that enables us create, modify, store, and retrieve text files. The *visual editor*, popularly known as **vi editor**, is one of the most widely used UNIX-based text editors. Generally, the programs are written using vi editor. It links with the syntax of most of the computer languages. To invoke vi editor, we give the following command at the shell prompt:

```
vi filename <enter>
```

For example,

```
vi hello.c <enter>
```

If the file “hello.c” exists, its contents will be displayed on the screen. If does not exist, a file by that name will be created and the following screen is displayed.

```
|
```

```
~
```

```
~
```

```
"hello.c" [New File]
```

The vi editor works in two modes: *edit* and *command*. Edit mode allows us to add text to the file. Command mode allows us to perform administrative tasks. The editor is initially in the command mode.

Every character is a command. Give an insert command to start editing the file. The following commands enable to edit.

Insert and Replace commands:

Comman	Action
D	
A	Enables us to append text after the current character.
A	Enables us to append text at the end of the line.
I	Enables us to insert text before the current character.
I	Enables us to insert text at the beginning of the line.
O	Insert a blank line below the current line, and allows us to insert text.
O	Insert a blank line above the current line, and allows us to insert text.

Navigation commands within the text:

Command	Action
H	Moves the cursor to the previous character.
L	Moves cursor to the next character.
K	Moves cursor to the line above.
J	Moves cursor to the line below.
<ctrl>d or D	Scrolls down half screen.
<ctrl>u or U	Scrolls up half screen.
<ctrl>F	Moves a page forward.
<ctrl>B	Moves a page backward.
0 (zero)	Moves cursor to the beginning of the line.
\$	Moves cursor to the end of the line.
W	Moves cursor to the next word.
B	Moves cursor to the previous word
E	Moves cursor to the end of the word.

Modification Commands:

Command	Action
Dw	Delete a word.
Dd	Delete a line.
Ndd	Delete N lines.
Cw	Changes a word.
Cc	Changes a line.
J	Joins lines.
U	Undo the last change.
. (dot)	Redo the last change.
Y or yy	Yanks the current line to be copied.
Nyy or NY	Yanks N line to be copied.
P	Places the yanked text after the current line.
P	Places the yanked text before the current line.

Searching commands:

Command	Action
/pattern <enter>	Finds the next line containing the pattern.
?pattern <enter>	Finds the previous containing the pattern.

File related commands: (: stands for file related commands. A sequence of letters follow after :)

:w <enter>	Saves the file.
:q <enter>	Quits the file.
:! <enter>	Forcible execution of a command.
:e <enter>	Opens specified file.
:w <filename><enter>	Writes to the specified file.
:r <filename><enter>	Reads and inserts the contents of the file after the current line.

After the colon, more letters are allowed to execute more command at a time. For example, “:wq” saves the file and quits from the editor, where as “:q!” quits forcefully without saving and “:w! <filename>” forcefully writes to the specified file.

16. UNIVERSITY QUESTION PAPERS OF PREVIOUS YEARS

No: 18CS1101
Geethanjali College of Engineering and Technology (Autonomous), Hyderabad
1 B.Tech (CE/CSE/EEE/ECE/ME) 1 Semester (Regular) End Examinations, Dec 2018
3 hours
AR18
Max. Marks: 70

Programming for Problem Solving

Answer all the questions.

Part-A (10 x 2 = 20)

- How do you differentiate Pseudo code with flow chart for a program. - 601
- List the different Data types supported by C-Language. - 602
- Write a for loop statement to print numbers from 25 to 1. - 603
- Write the syntax for "IF-ELSE" statement. - 604
- Illustrate floor and ceil functions in c language with examples. - 605
- Compare recursion and iteration. - 606
- How do you declare and initialize 2-D array in C language? - 607
- Write a c program to find the length of a string in C language? - 608
- What is a pointer to a function? Give the general syntax for the same. - 609
- Represent a 2-D array of integers using pointer to pointer in c language. - 610

Part-B (5 x 10 = 50)

- Mention the different symbols used in drawing the flowchart for a program. Draw the flowchart to find the largest of three numbers. - 601 [5M]
- Draw the Basic Structure of a general C program and Explain. - 602 [5M]

(OR)

- Explain how an Expression is evaluated in C with an example. - 602 [5M]
- Write the differences between Implicit and Explicit Type conversion in C with example. [5M]

- Summarize the Syntactic rules "Switch-Case" statement. - 603 [3M]
- Write a C program to find the number of days in a Month for the month given as numerical value (eg. Jan=1, Feb=2 ...) using "Switch-Case" & "Break" statement. - 603 [7M]

(OR)

- Write the difference between "Do-While" & "While" statements in C. - 603 [5M]
- Write a C program to print Multiplication table of given number using "While" statement. - 603 [5M]

- Write a function to print fibonacci series upto a given number. - 603 [5M]
- Discuss about Call by Value and Call by Reference with illustrations. - 604 [5M]

(OR)

- Illustrate the different storage classes in C with example. - 604 [3M]
- Write a C program to compute factorial of a given number using recursion. - 604 [7M]

- Write a c program to find sum of elements of two 3-D arrays. - 605 [4M]
- Write a C program to print the product of two matrices. - 606 [6M]

(OR)

- Write a C program to check whether given string is palindrome or not. - 607 [5M]
- Write a C program to search for substring in a given string without using library functions. - 608 [5M]

- What is a pointer? Explain how to pass a pointer and a pointer to pointer to a function with examples. - 609 [6M]
- Write a function which accepts an array of integers as argument and returns the sum of elements of that array. - 610 [4M]

(OR)

- Write the usage of realloc() and free() functions in c language with a code snippet each. - 611 [5M]
- Write two c functions to find the sum of even and odd numbers (one function for each) of a give array when the address of the array is passed to the function as an argument. - 612 [5M]

Code No:131AD

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD
B.Tech I Year I Semester Examinations, December - 2018
COMPUTER PROGRAMMING IN C
(Common to CE, ME, MCT, MMT, AE, MIE, PTM, CEE, MSNT)

Time: 3 hours

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A.
Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub-questions.

Max. Marks: 75

PART - A

- 1.a) Write about Computer Systems in detail.
- b) Explain flowchart symbols with an example.
- c) Explain about scope rules.
- d) Write the array applications in brief.
- e) Write about Pointer Arithmetic with an example.
- f) Explain arrays of strings in C.
- g) Explain self referential structures in brief.
- h) Write short note on Enumerations.
- i) Write about File streams.
- j) Write a program for Opening and Closing files.

(25 Marks)

[2]
[3]
[2]
[3]
[2]
[3]
[2]
[3]
[2]
[3]

PART-B

- 2.a) Explain in detail about Selection Statements in C.
- b) Discuss about string integers and real numbers.
- 3.a) Explain in detail about Computer Languages.
- b) Discuss about Expressions, operator precedence and associativity.
- 4.a) Explain types of Functions.
- b) Explain multidimensional arrays in detail.
- 5.a) Write a program to implement bubble sort.
- b) Explain about recursive functions in detail.
- 6.a) Discuss about Pointers for inter function communication.
- b) Explain about String manipulation functions.
- 7.a) Explain in detail about void pointer with examples.
- b) Explain String Concepts in detail.

(50 Marks)

[5+5]

[5+5]

[5+5]

[5+5]

[5+5]

[5+5]

- 8.a) Define Structure and explain initialization of structures with an example.
b) Discuss about Pre-processor commands.

- 9.a) Write a C program to implement unions.
b) Explain structures and functions with an example program.

- 10.a) Explain Binary File modes in brief.
b) Write a C program to implement feof() and ferror() functions .

OR

- 11.a) Explain about fseek(), ftell() and rewind() file functions with examples.
b) Write a C program reverse the contents of a given file.

---ooOoo---

Code No: 16CS1102

AR16

Geethanjali College of Engineering and Technology (Autonomous), Hyderabad

I B.Tech I Semester Supplementary Examinations, May 2017

Introduction to Computer Programming

(CE / ME)

Time: 3 hours

Max. Marks: 70

Answer all Questions

Part-A (10 x 2 = 20)

- Mention the steps to compile a program in "C".
- Explain bitwise XOR operator with an appropriate example.
- Discuss the limitations of "nested if" compared with "switch statement".
- Explain the need for "continue" statement in "C".
- Categorize the functions in "C" according to their parameters and return types.
- Explain the significance of "stdio.h" in "C".
- How is a two dimensional array represented in memory?
- State the key characteristics of a Single Dimensional Array in "C".
- Define pointer? How are pointers assigned?
- Write the syntax for "realloc" function.

Part-B (5 x 10 = 50)

- Define flow chart and state its characteristics. [5]
 - Discuss all bitwise operators with examples. [5]
- (OR)
- Express the significance of operator precedence? Discuss its role in expression evaluation. [5]
 - Write an Algorithm to compute the area and perimeter of a circle. (Read the radius as input from the user). [5]
- Explain various kinds of control statements used in "C" Programming. [5]
 - Write a program in "C" to find whether a given integer is palindrome or not. [5]
- (OR)
- Illustrate various looping constructs used in "C" with appropriate examples. [6]
 - Write a "C" program that adds 'n' integers as input and then display result using functions. [4]
- Explain in detail 'call-by-value' mechanism in "C". [5]
 - Discuss the uses of inter function communication in "C". [5]
- (OR)
- Explain in detail the scope of variables with a suitable example. [5]
 - Write a "C" program to find the factorial of a given number using recursive function. [5]
- Illustrate the process of passing a single dimensional array to a function in "C". [5]
 - Write a "C" program that prints one dimensional array in its reverse order. [5]
- (OR)
- Explain how elements of a two dimensional array are accessed with an appropriate example. [4]
 - Write a "C" program that fills diagonal of a square matrix with 1s and remaining with 0s (identity matrix). Read the order of the square matrix from user. [6]
- State the differences between static and dynamic memory management in "C". [5]
- Describe how pointers can be returned from a function in "C". [5]
- (OR)
- Explain various memory management techniques used for Dynamic memory management in "C". [7]
- Explain the significance of "void" pointer with an example. [3]

Time: 3 hours

Answer all Questions
 Part-A (10 x 2 = 20)

- Define algorithm. Discuss about its characteristics.
- List any five relational operators in C.
- Mention the limitations of a "goto" statement.
- Describe usage of "continue" statement.
- State the significance of "main ()" function in "C".
- Mention various scope rules in C.
- Explain how an array can be passed as argument to a function.
- Describe characteristics of multi dimensional arrays.
- Write the syntax of free ().
- What is a NULL pointer?

Part-B (10 x 5 = 50)

- Use the Raptor programming tool to draw a flow-chart for finding a given number is positive or not? [6]
[4]
- What are bitwise operators? Provide suitable examples. [5]
(OR)
- Explain the process of creating and running a C program with an example. [5]
- Write an algorithm to compute the area of a Circle (Read the necessary inputs from the user) [5]

Draw a flow chart and write a C program to perform one of the given arithmetic operations through switch statement.

- (OR) [7]
[3]
- Illustrate the Fibonacci series of given "n" number using loops.
- State the applications of break statement.

Define a recursive function. Write a C program to demonstrate working of recursive mechanism. [10]
(OR)

- Mention and Explain various storage classes. [5]
[5]
- Explain how inter function communication takes place in modular programming.
- Explain how elements of a multi-dimensional array is accessed by the user. [5]
[5]
- Illustrate how you pass the single dimensional array to a function in "C".

- (OR) [3]
- Write applications of arrays.
- Write a "C" program that performs the sum of the two given matrixes. Read the order of the matrixes and its elements from user. [7]

- Write a C program to implement call by reference mechanism. [5]
[5]
- How dynamic memory allocation can be performed in C.

- (OR)
- Write a C program to add the contents of an array (with numbers) using an integer pointer. [7]
[3]
- Specify the syntax for realloc ().

17. ASSIGNMENT QUESTIONS

UNIT-1

1. Design a flow chart to find whether a given number is prime or not.
2. Write an algorithm to find the sum of individual digits of a given number.
3. Design a flow chart to find whether a given number is palindrome or not.
4. Design a flow chart to check the given number is Armstrong number or not. Define Constants?
5.
 - a. Illustrate different types of data types in C
 - b. Describe different control statements in C language with an example? .
6. Explain various Data types in C-Language with an example.
7. Define an operator? Explain different operators in Language.
8. Illustrate all the possible ways of solving a problem?
9. Write a C program to implement a conditional operator?
10. State the rules that could be applicable while evaluating expression in automatic type conversion

UNIT-II

1. The Fibonacci series is defined as $n_0 = 1, n_1 = 1, n_{i+2} = n_i + n_{i+1}$ for $i=0,1,2,\dots$ thus the First few Fibonacci numbers are 1,1,2,3,5,8,13... write a complete C-program to compute and print the first m Fibonacci numbers, where m is the input to the program.
2. Write a C program to find all the prime numbers between 1 and n(where n value is supplied by the user).
3. Summarize the need of goto statement?
4. Write a C program to print the following output.

```
      1
    2   2
  3   3   3
4   4   4   4
```

5. An electric power distribution company charges its domestic consumers as follows:
Consumption Units

	Rate of Charge
0-200	Rs. 0.50 per unit
201-400	Rs.100 plus
401-600	Rs.0.65 per unit
	excess of 200

Rs.230 plus
Rs.0.80 per unit

excess of 400.

6. Write a C program that reads the customer number and power consumed and prints the amount to be paid by the customer

UNIT-III

1. Explain Inter function communication in C language with an example.
2. Interpret in your own words on why Fibonacci series is the best example for static variable
3. Write different ways of declaring global variables?
4. What are Recursive functions? Explain with an example the limitations of recursion
5. Illustrate different storage classes in C language

UNIT-IV

1. What are the different ways to initialize an array? Explain them with an example program?
2. Write a C program to find the largest and smallest elements in a list of array elements. (Array size should be declared at the time of execution)
3. Write a C program to find sum of even elements and odd elements in a list of array elements by passing the complete array to a function.
4. Write a C program to add and multiply two matrices by taking one matrix as identity matrix?
5. Illustrate applications of an array?

UNIT-V

1. Compare malloc() and calloc() functions with an example?
2. State the rules for pointer arithmetic?
3. Write a C program to find the sum of even and odd elements using pointer, array and functions.
4. State and Explain void pointer with an example.
5. A. State the need of call- by- reference with an example?
b. Write a C program to implement array of pointers?

18. QUESTION BANK

UNIT 1

Demonstrate problem solving skills by developing algorithms to solve problems using Raptor tool.

1. Draw a flowchart to find the maximum Of three numbers.
2. Draw a flowchart to find the sum of “n” natural numbers..
3. Write an algorithm to find the quadratic roots of an equation.
4. Differentiate between pseudo code and algorithm?

Incorporate the concept of variables, constants, basic data types and input and output statement in a C language program

1. Explain the formatted input and output functions with an example
2. State and Explain Type conversions in C with appropriate examples
3. Explain the concept of expression evaluation with an example?
4. State the rules of identifiers with few examples?
5. Interpret in your own words the difference between bitwise-OR and bitwise-AND?
6. Illustrate with an examples the following operators
 - a. Ternary operator
 - b. Special operators
7. State and explain Precedence and Associativity with example?
8. What is data type? Explain different types of data types in c?

UNIT II

Incorporate the use of sequential, selection and repetition control statements into the algorithms implemented as computer programs using C language.

1. Write the syntax and flow chart of else-if ladder?
2. Write the syntax and flow chart of nested if?
3. Explain the need of nested for loop? Explain with an example.
4. Write a C program to implement the given number is multiple of 7 and 3.
5. Differentiate between entry-controlled loop and exit-controlled loop
6. Write the syntax and flow chart of else-if ladder?
7. Write the syntax and flow chart of nested if?
8. Explain the need of nested for loop? Explain with an example.
9. Write a C program to implement the given number is multiple of 7 and 3.
10. Differentiate between entry-controlled loop and exit-controlled loop
11. Write the syntax for the selection statements with examples?
12. Differentiate between one way, two way and multi-way selection statements with examples?
13. Discuss about various repetitive statements with examples?
14. Write a c program to find Individual digits of a positive integer?
15. Write a c program to find the factorial of a given number?
16. Write a c program to print the sequence of n terms using Fibonacci series?
17. Write a c program to find sum of even numbers below 100?
18. Write a c program to determine whether the given number is Armstrong number or not?
19. Write a c program to solve the following sum
$$\text{Sum} = 1 + x + x^2 + x^3 + x^4 + \dots + x^n$$
 (where x and n values are supplied by the user)
20. Write a c program to find all the prime numbers between 1 and n (where n value is supplied by the user).

UNIT III

Demonstrate an understanding of structured design by implementing programs with functions and passing of parameters to solve more complex problems.

1. What is function?
 2. How is a function defined in C?
 3. What are the steps in writing a function in a program?
 4. What is the purpose of the function main()?
 5. Is it better to use a macro or a function?
 6. What is built in function?
 7. What is use of return statement?
 8. What is recursive function?
 9. What do you mean by call by value?
 10. What do you mean by call by reference?
 11. State features of pre-processor?
 12. What are command line arguments?
 13. Explain the functions with no arguments and no return values with an example?
 14. State the purpose of Extern variables with syntax?
 15. Explain the need of automatic variables with syntax? 4) Illustrate the need of Recursive functions in C
 16. State the differences between automatic and static variable?
 17. Write a program to find largest between two numbers using functions?
 18. What is recursion explain with suitable example.
 19. Explain Automatic storage class specifier with suitable example.
 20. Explain Static storage class with suitable example. How are the data elements initialized in the case of static type variable?
 21. Explain Register storage class with suitable example.
 22. Explain Extern storage class with suitable example.
 23. What is function? How is a function defined?
 24. Explain the difference between calling function and called function with examples?
 25. Explain void function?
 26. Explain function with argument and return type .Illustrate with an example.
 27. List out the advantages of function.
 28. What is the purpose of return statement? List out the rules used in return statement.
 29. What is mean by register variable and what is the scope of it?
 30. What is the storage class used in recursive function
 31. What is a recursive function? List out their merits and demerits.
 32. What is a function and list out advantages and disadvantages of functions .
 33. What is mean by function argument, function call and return value ?
 34. What is the automatic variable and what is the use of it? How can data be initialized in the automatic variable?
 35. How is the #include directive is used?
 36. How can #define directive be continued to a new line ?
 37. Write a c program using recursion to find GCD of two given numbers?
 38. Categorize functions with relevant examples?
-

39. State and explain storage classes with examples?
40. Write a c program using functions to construct a pyramid?
41. Write a c program using functions to print Pascal triangle?
42. Write a c program to find factorial of a given number using recursive and non recursive?

UNIT IV

Write C programs using arrays, and pointers and also with dynamic memory allocation.

1. Write a c program to find largest and smallest number in a list of array?
2. Write a c program to calculate i) addition of 2 matrices ii) Multiplication of 2 matrices?
3. Write a c program to sort the elements of an array in ascending order?
4. Write a c program to search an element in a list of array?
5. Write a c program to read 10 integer elements and calculate their sum using arrays?
6. Explain the use of functions strcpy () and strcmp ()
7. Explain how strings are declared and initialized in 'C'
8. Write a C function to find the length of a string passed as an argument?
9. Explain the process of passing an array to a function with an example?
10. Explain various ways of initializing an array in C?
11. What are the application areas of an array?
12. Write a C program to find sum of even numbers using an array?
13. State the differences between fixed-length and variable-length array?

UNIT V

Write C programs using pointers and also with dynamic memory allocation.

1. State the rules of pointer operations?
2. Discuss pointers and arrays with an example?
3. Illustrate pointers to functions with an example?
4. Mention pointer arithmetic with an appropriate example?
5. Explain pointer to pointer with an example program?
6. Compare static memory allocation and heap memory allocation?
7. a. What is void pointer? Explain with an example?
b. What is NULL pointer? Explain with an example?
8. Explain call by value and call by reference with examples?
9. Write a c program to find the largest number in a list of elements using calloc () function?
10. Write a C program to find sum of even numbers and odd numbers using pointers and Arrays?
11. Explain the concept of array of pointers with an example program?

19. UNIT WISE QUIZ QUESTIONS

UNIT-I

- 1) Pseudo code is used for
 - (a) Denoting the program flow
 - (b) To make structure chart
 - (c) For coding the program
 - (d) To write program steps
 - 2) What symbol is used to represent output in a flowchart
 - (a) Square
 - (b) Circle
 - (c) parallelogram
 - (d) Triangle
 - 3) Method which uses a list of well defined instructions to complete a task starting from a given initial state from a given initial state to end state is calls as
 - (a) Program
 - (b)Flowchart
 - (c)Algorithm
 - (d)A & B
 - 4) The chart that contains only function flow and no code is called as
 - (a)flowchart
 - (b)Structure chart
 - (c)Both A and B
 - (d)None
 - 5.Which of the following is a program planning tool?
 - (a)Sequential
 - (b)decision
 - (c)Pseudo code
 - (d)Both B and C
 6. Flowcharts and Algorithms are used for
 - (a) Better Programming
 - (b)Efficient Coding
 - (c)Easy testing and Debugging
 - (d)All
 7. An Algorithm represented in the form of programming languages is ____
 - (a)Flowchart
 - (b)Pseudo code
 - (c)Program
 - (d)None
 8. Which of the following is a pictorial representation of an algorithm?
 - (a)Pseudo code
 - (b)Program
 - (c)Flowchart
 - (d)Algorithm
 9. Which of the following symbol in a flowchart are used to indicate all arithmetic processes of adding, subtracting, multiplying and dividing ?
 - (a) Input/output
 - (b)terminal
 - (c)Processing
 - (d)Decision
 10. A flowchart that outlines the main segments of program is called as
 - (a)Micro flowchart
 - (b)Macro flowchart
 - (c)Flowchart
 - (d)Algorithm
 - 11.Pseudo code is also known as
 - (a)Program Design Language
 - (b)Software Language
 - (c)Hardware Language
 - (d)Algorithm
 12. Pseudo code emphasizes on
 - a. Development
 - (b)Coding
 - (c)Design
 - (d)Debugging
-

13. What is the standard terminal symbol for a flowchart?
a) circle (b) rectangle (c) diamond (d) square
14. Which of the following is not a characteristic of good algorithm?
(a) Precise (b) Finite number of steps (c) Ambiguous (d) Logical flow of control
15. The following pseudo code is an example of _____ structure:
a) Get number b) get another number c) multiply numbers d) print result
(a) Sequence (b) Decision (c) Loop (d) Nested
16. Which of the following is not a principle of structured- programming?
a. Design the program in top-down manner
b. Write each program module as a series of control structure
c. Code the program so that it runs correctly without testing
d. Use good programming
17. Which of the following keyword is followed by an integer or character constant?
(a) switch (b) case (c) for (d) void
18. Which of the following enhances the versatility of the computer to perform a set of instructions repeatedly?
(a) Function (b) Loop (c) header files (d) statement
19. The programming language that closely resembles the machine language is
(a) High-level languages (b) C language (c) FORTRAN (d) Assembly language
20. The tool used to convert a „C“ program to machine language is called as
(a) Linker (b) Language translator (c) Compiler (d) Pre-processor
21. The programmer original program code is called as
(a) Object file (b) Source file (c) Executable file (d) Application file
22. Which of the following is not a principle of structured- programming?
a) Design the program in top-down manner
b) Write each program module as a series of control structure
c) Code the program so that it runs correctly without testing
d) Use good programming
23. Which of the following keyword is followed by an integer or character constant?
(a) switch (b) case (c) for (d) void
24. Which of the following enhances the versatility of the computer to perform a set of instructions repeatedly?
-

- (a)Function (b)Loop (c)header files (d)statement
24. The programming language that closely resembles the machine language is
(a)High-level languages (b)C language (c)FORTRAN (d)Assembly language
25. The tool used to convert a „C“ program to machine language is called as
(a)Linker (b)Language translator (c)Compiler (d)Pre-processor
26. The programmer original program code is called as
(a)Object file (b)Source file (c)Executable file (d)Application file
27. Which of the following statements is wrong?
a. `int=123;` (b)`value="" +5` (c)`lime=20*"T"` (d)`count+5=result`
28. Which of the following statement is incorrect?
(a) `rem=3%2;` (b)`rem=3.14%2.1;` (c)`rem="a" % „c“` (d)None of above
29. Which will be the output of following program?

```
#include<stdio.h>
void main()
{
    int a;
    printf("%d\n" a)
}
```


a. Error (b)0 (c)-1 (d)Garbage value
30. What is the result of `16<<2`?
(a) 4 (b) 8 (c) 2 (d) 5
31. What is the result of the expression `(10/3)*3+5%3`?
(a)11 (b) 10 (c) 8 (d) 1
32. What will the value of c after execution of the program?

```
#include<stdio.h>
void main()
{
    int a,b,c;
    a=9,b=10;
    c=(b<a || b>a);
    printf("\nc=%d",c);
}
```


a. c=1 (b) c=0 (c) c=-1 (d) none of the above
33. What is the ASCII range for ATOZ letters?
a. 65 to 90 (b) 48 to 57 (c) 97 to 122 (d) none of the above
34. The escape sequence `"\t"` is a
(a) tab (b) newline (c) backspace (d) none of the above
-

35. What will be the output of the following program?

```
main() {  
    System("");  
}
```

- a. control goes to the dos prompt (b) syntax error
(c) bad command or file name (d) none of the above

UNIT-II

1) What will be the value of „x" after the execution of following program? #include<stdio.h>

```
void main()  
{  
    int k; float x=0;  
    clrscr();  
    for(k=0;k<10;k++)  
        x+=1;  
    printf("\nx=%g",x);  
}
```

- (a) x=1 (b) x=0 (c) x=1.1 (d) none of the above
- 2) How many while statements are possible in do...while loop?
- (a) 2 (b) 1 (c) 3 (d) none of the above
- 3) Why this program runs infinite times?

```
#include<stdio.h>  
void main()  
{  
    int i;  
    for(i=32200;i<=32768;i++) {  
        printf(" The Value I %d",i);  
    }  
}
```

- (a) With in the range of Integer (b) It will not infinite
(c) Error (d) None of above
- 4) Looping in a program means
- (a) Branching to be specified branch or label in the program
(b) repeating a given set of instruction

- (c) Both of above
 - (d) None of above
- 5) The difference between while and do-while statements is
- a. In the while statement the control first enters into the loop then condition is tested at the end of first iteration
 - b. In do while the condition is tested in first iteration and if the condition is true ,it enters into of first iteration
 - c. The do-while statement's condition is used to decide whether to enter the loop or not whereas the while statement's condition is used to decide whether to exit the loop (or) not
 - d. The while statement's condition is used to decide whether to enter the loop or not whereas the do-while statement's condition is used decide whether to exit the loop or not
- 6) Observe the following block of code and determine what happens when x=2?
- ```
switch(x)
{
 Case 1: printf("x is 1"); Break;
 Case 2:
 Case 3: printf("x is 3"); break;
 default: printf("X is not within the range");
 break;
}
```
- a. Program jumps to the end of switch statement since there is nothing to do for x=2
  - b. The code inside default will run since there is no task for x=2,so
  - c. Will display x is 3,and then come outside the switch statement
  - d. None of above
- 7) Which of the following is false for a "switch" statement in C?
- a. break statement is false is compulsory after each case
  - b. default statement is compulsory
  - c. There is a limit on the maximum number of cases
  - d. None of the above
- 8) Find the output of the following program
-

```
#include<stdio.h>
void main()
{
 int x=4;
 float y=4;
 if(x==y)
 printf("x and y are equal");
 else
 printf("x and y are not equal");
}
```

- a. x and y are equal
  - b. x and y are not
  - c. Unpredictable
  - d. No output
- 9) To repeat a set of the statements for 25 times , which kind of statement will be required?
- (a)Iterative (b)Selective (c)Either (a) or (b) can be used(d)None of the above
- 10) The minimum number of update/increment/decrement allowed in a “for” loop are\_\_\_\_\_
- (a)1 (b)2 (c)3 (d)None of above
- 11) The” while” loop can be replaced by “for” loop in all the cases
- (a) True (b)False (c)Depends on the condition (d)None of the above
- 12) **”break ”** statement when executed the control is transferred ( )
- a. Outside the loop ,to the next statement after the loop
  - b. beginning of the loop i.e. to the first statement in the loop
  - c. outside the function, to the next function in the program
  - d. beginning of the function i.e. to the first statement in the function

### UNIT-III

- 1) The main() is a
  - (a)User –defined (b)Library functions (c) Keyword (d) None of the above
- 2) The function name itself is
  - a) an address (b) value (c) definition (d) None of the above
- 3) What is the data type of variable m

Void main()

```

{
int x=2;
sqr(x);
}
sqr(m)
{
Return (m*m);
}

```

(a) int                      (b) float                      (c) char                      (d) void

4) By default the function returns

(a) integer value                      (b) float value                      (c) char value                      (d) None of the above

5) A static variable is one that

(a) retains its value through the life of the program

(b) cannot be initialized

(c) is initialized once at the commencement the execution and cannot be changed at the run time

(d) is same as an automatic variable but is placed at the end of the program

6) If a storage class is not mentioned in the declaration then default storage class is

(a) automatic                      (b) static                      (c) external                      (d) register

7) If CPU fails to keep the variables in CPU registers, in that case the variables are assumed

(a) Automatic                      (b) static                      (c) external                      (d) None of the above

8) Recursion is a process in which a function calls

(a) itself                      (b) another function                      (c) main() function                      (d) None of the above

9) The meaning of keyword void before the function name means

(a) Function should not return any value

(b) Function should return any value

(c) No arguments are passed

(d) None of the above

10) What will be the values of x and s on execution?

```

int x,s;
void main(int);
void main(x)
{
Printf("\nx=%d s=%d",x,s); }

```

- (a) x=1 s=0    (b) x=0,s=0    (c) x=1 s=1    (d) none of the above

11) An external variable is one

- (a) Which is globally accessible by all functions
- (b) Which declared outside the body of any function
- (c) Which resides in the memory till the end of the program
- (d) None of the above

#### UNIT-IV

1.What is right way to Initialize array?

- A. ☐ int num[6] = { 2, 4, 12, 5, 45, 5 };
- B. ☐ int n{ } = { 2, 4, 12, 5, 45, 5 };
- C. ☐ int n{6} = { 2, 4, 12 };
- D. ☐ int n(6) = { 2, 4, 12, 5, 45, 5 };

2.What will be the output of the program ?

```
#include<stdio.h>
void main()
{
 int a[5] = {5, 1, 15, 20, 25};
 int i, j, m;
 i = ++a[1];
 j = a[1]++;
 m = a[i++];
 printf("%d, %d, %d", i, j, m);
}
```

- A. ☐ 3, 2, 15
- B. ☐ 2, 3, 20
- C. ☐ 2, 1, 15
- D. ☐ 1, 2, 5

3.What will be the output of following program code?

```
#include <stdio.h>
int main(void)
{
 char p;
 char buf[10] = {1, 2, 3, 4, 5, 6, 9, 8};
 p = (buf + 1)[5];
 printf("%d", p);
 return 0;
}
```

- A. ☐ 5
- B. ☐ 6

- C. ☐ 9
- D. ☐ Error
- E. ☐ None of the above

4. An array elements are always stored in \_\_\_\_\_ memory locations.

- A. ☐ Sequential
- B. ☐ Random
- C. ☐ Sequential and Random
- D. ☐ None of the above

5. Let x be an array. Which of the following operations are illegal?

- I. ++x
- II. x+1
- III. x++
- IV. x\*2

- A. ☐ I and II
- B. ☐ I, II and III
- C. ☐ II and III
- D. ☐ I, III and IV
- E. ☒ III and IV

6. What is the maximum number of dimensions an array in C may have?

- A. ☐ 2
- B. ☐ 8
- C. ☐ 20
- D. ☐ 50
- E. ☐ Theoretically no limit. The only practical limits are memory size and compilers.

7. Size of the array need not be specified, when

- A. ☐ Initialization is a part of definition
- B. ☐ It is a declaration
- C. ☐ It is a formal parameter
- D. ☐ All of these

8. Consider the following type definition.

```
typedef char x[10];
```

```
x myArray[5];
```

What will sizeof(myArray) be ? (Assume one character occupies 1 byte)

- A. ☐ 15
- B. ☐ 10
- C. ☐ 50
- D. ☐ 30
- E. ☐ None of these

9. What will be printed after execution of the following code?

```
void main()
{
 int arr[10] = {1,2,3,4,5};
 printf("%d", arr[5]);
}
```

- A. ☐ Garbage Value
- B. ☐ 5
- C. ☐ 6
- D. ☐ 0
- E. ☐ None of these

10. What will be the output of the following program?

```
void main()
{
 char str1[] = "abcd";
 char str2[] = "abcd";
 if(str1==str2)
 printf("Equal");
 else
 printf("Unequal");
}
```

- A. ☐ Equal
- B. ☐ Unequal
- C. ☐ Error
- D. ☐ None of these.

11. What will be the output of the following code?

```
void main()
{
 int a[10];
 printf("%d %d", a[-1], a[12]);
}
```

- A. ☐ 0 0
- B. ☐ Garbage value 0

- C. ☐ 0 Garbage Value
- D. ☐ Garbage vlaue Garbage Value
- E. ☐ Code will not compile

12.What does the following declaration mean?

`int (*ptr)[10];`

- A. ☐ ptr is array of pointers to 10 integers
- B. ☐ ptr is a pointer to an array of 10 integers
- C. ☐ ptr is an array of 10 integers
- D. ☐ ptr is an pointer to array

13.Array passed as an argument to a function is interpreted as

- A. ☐ Address of the array.
- B. ☐ Values of the first elements of the array.
- C. ☐ Address of the first element of the array.
- D. ☐ Number of element of the array.

14.What will be the output of the program if the array begins at 65472 and each integer occupies 2 bytes?

```
#include
void main()
{
 int a[3][4] = {1, 2, 3, 4, 4, 3, 2, 1, 7, 8, 9, 0};
 printf("%u, %u", a+1, &a+1);
}
```

- A. ☐ 65474, 65488
- B. ☐ 65480, 65488
- C. ☐ 65480, 65496
- D. ☐ 65474, 65476
- E. ☐ None of these

15.What will be the output of the program ?

```
#include<stdio.h>
int main()
{
 int arr[1] = {10};
 printf("%d", 0[arr]);
 return 0;
}
```

- A. ☐ 1
- B. ☐ 0

- C. ☐ 10
- D. ☐ 6
- E. ☐ None of these

16. What will be the output of the program if the array begins at address 65486?

```
#include
void main()
{
 int arr[] = {12, 14, 15, 23, 45};
 printf("%u, %u", arr, &arr);
}
```

- A. ☐ 65486, 65488
- B. ☐ 65486, 65490
- C. ☐ 65486, 65487
- D. ☐ 65486, 65486
- E. ☐ None of these

17. What will be the output of the program ?

```
#include
void main()
{
 float arr[] = {12.4, 2.3, 4.5, 6.7};
 printf("%d", sizeof(arr)/sizeof(arr[0]));
}
```

- A. ☐ 5
- B. ☐ 4
- C. ☐ 6
- D. ☐ 7
- E. ☐ None of these

18. Which of the following is correct way to define the function fun() in the below program?

```
#include<stdio.h>
void main()
{
 int a[3][4];
 fun(a);
}
```

- A. ☐ void fun(int p[][4]){}  
B. ☐ void fun(int \*p[4]){}  
C. ☐ void fun(int \*p[][4]){}  
D. ☐ void fun(int \*p[3][4]){}



E. ☐ None of these

19. Which of the following statements are correct about the program below?

```
#include<stdio.h>
void main()
{
 int size, i;
 scanf("%d", &size);
 int arr[size];
 for(i=1; i<=size; i++)
 {
 scanf("%d", arr[i]);
 printf("%d", arr[i]);
 }
}
```

- A. ☐ The code is erroneous since the statement declaring array is invalid.
- B. ☐ The code is erroneous since the subscript for array used in for loop is in the range 1 to size.
- C. ☐ The code is correct and runs successfully.
- D. ☐ The code is erroneous since the values of array are getting scanned through the loop.
- E. ☐ None of these

20. Which of the following statements are correct about an array?

1. The array `int num[26]`; can store 26 elements.
2. The expression `num[1]` designates the very first element in the array.
3. It is necessary to initialize the array at the time of declaration.
4. The declaration `num[SIZE]` is allowed if `SIZE` is a macro.

- A. ☐ 1
- B. ☐ 1, 4
- C. ☐ 2, 3
- D. ☐ 2, 4
- E. ☐ None of these

21. What is the value of an array element which is not initialized.?

- A) By default Zero 0
- B) 1
- C) Depends on Storage Class
- D) None of the above.

22. What happens when you try to access an Array variable outside its Size.?

- A) Compiler error is thrown
- B) 0 value will be returned

- C) 1 value will be returned
- D) Some garbage value will be returned.

23. What is the size of an array in the below C program statement.?

```
int main()
{
 int ary[9];
 return 0;
}
```

- A) 8
- B) 9
- C) 10
- D) None of the above

24. Can we change the starting index of an array from 0 to 1 in any way.?

- A) Yes. Through pointers.
- B) Yes. Through Call by Value.
- C) Yes. Through Call by Reference.
- D) None of the above.

25. What is the need for C arrays.?

- A) You need not create so many separate variables and get confused while using.
- B) Using a single Array variable, you can access all elements of the array easily.
- C) Code maintainability is easy for programmers and maintainers.
- D) All the above.

## 20. TUTORIAL QUESTIONS

### Unit-I

- 1) Write a C program which accepts integer values from the user and calculates the division and display the results in floating point type.
- 2) Evaluate the following expression by using precedence and associativity of operators

$$(((a+b)*(c-d))*(e-f)) / (g+h)$$

### Unit -II

- 1) Explain with an example the usage of Do-While Loop.
- 2) Write a C program to display whether the given number is multiple of 2,4 and 8.

### Unit -III

- 1) Write a C program to find the factorial of a given number with different categories of functions available in C.

- 2) Write a C program to solve the Towers of Hanoi using Recursion.
- 3) Write a C program to depict the scope of various Storage Classes.

#### **Unit-IV**

- 1) Write a C program to find the second largest and second smallest elements in an array of integers.
- 2) Write a C program to insert an element in a specified position in a given array.

#### **Unit- V**

- 1) Write a C program to find sum of two matrices by allocating memory dynamically using malloc(), calloc().
- 2) Write a C program to reallocate memory for an array and find the sum of elements in the array before and after allocation of memory.

#### **21. KNOWN GAPS ,IF ANY**

None

#### **22. DISCUSSION TOPICS , IF ANY**

None

#### **23. REFERENCES, JOURNALS, WEBSITES AND E-LINKS**

##### **Websites**

1. [www.cs.wustl.edu](http://www.cs.wustl.edu)
  2. [www.csd.uwo.ca](http://www.csd.uwo.ca)
  3. [www.digitalmars.com](http://www.digitalmars.com)
  4. [Cminusminus.org](http://Cminusminus.org)
  5. [www.c-faq.com](http://www.c-faq.com)
  6. [www.osborne.com](http://www.osborne.com)
  7. [raptor.martincarlisle.com](http://raptor.martincarlisle.com)
  8. [www.alice.org](http://www.alice.org)
  9. <https://www.khanacademy.org>
-

## **24.QUALITY MEASUREMENT SHEETS**

- a. Course End Survey
- b. Teaching Evaluation

## **25. STUDENT LIST**

## **26.GROUP-WISE STUDENTS LIST FOR DISCUSSION TOPICS**