Estd: 2008

METHODIST

COLLEGE OF ENGINEERING & TECHNOLOGY

UGC AUTONOMOUS Institution Affiliated to Osmania University, Accredited by NBA & Naac with A+

Abids, Hyderabad, Telangana, 500001

DEPARTMENT OF MECHANICAL ENGINEERING

LABORATORY MANUAL

PROGRAMMING FOR PROBLEM SOLVING LABORATORY

BE III Semester

AUTONOMOUS

Name: ....

Roll No: ....

Branch:..SEM:.....................

Academic Year: ....

Empower youth - Architects of Future World

METHODIST

Estd: 2008

METHODIST

COLLEGE OF ENGINEERING & TECHNOLOGY

Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad

Abids, Hyderabad, Telangana, 500001

VISION

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

MISSION

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.

Estd: 2008

METHODIST

COLLEGEOF ENGINEERING & TECHNOLOGY

Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad

Abids, Hyderabad, Telangana, 500001

DEPARTMENT OF MECHANICAL ENGINEERING

LABORATORY MANUAL

PROGRAMMING FOR PROBLEM SOLVING LABORATORY (6ES351CS)

Prepared by

Dr. Md. Fakhruddin.H.N., Professor. Mech. Engg.
Mr. Shaik Shoeb, CAD Lab Assistant. Mech. Engg.

DEPARTMENT OF MECHANICAL ENGINEERING

VISION

To be a reputed centre of excellence in the field of mechanical engineering by synergizing innovative technologies and research for the progress of society.

MISSION

- To impart quality education by means of state-of-the-art infrastructure.
- To involve in trainings and activities on leadership qualities and social responsibilities.
- To inculcate the habit of life-long learning, practice professional ethics and service the society.
- To establish industry-institute interaction for stake holder development.

DEPARTMENT OF MECHANICAL ENGINEERING

After 3-5 years of graduation, the graduates will be able to:

PEO1: Excel as engineers with technical skills, and work with complex engineering systems.

PEO2: Capable to be entrepreneurs, work on global issues, and contribute to industry and society through service activities and/or professional organizations.

PEO3: Lead and engage diverse teams with effective communication and managerial skills.

PEO4: Develop commitment to pursue life-long learning in the chosen profession and/or progress towards an advanced degree

DEPARTMENT OF MECHANICAL ENGINEERING

PROGRAM OUTCOMES

Engineering Graduates will be able to:

Po1. Engineering knowledge: Apply the basic knowledge of mathematics, science and engineering fund a mentals along with the specialized knowledge of mechanical engineering to understand complex engineering problems.

PO2. Problem analysis: Identify, formulate, design and analyse complex mechanical engineering problems using knowledge of science and engineering.

Po3. Design/development of solutions: Develop solutions for complex engineering problems, design and develop system components or processes that meet the specified needs with appropriate consideration of the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Formulate engineering problems, conduct investigations and solve using research-based knowledge.

PO5. Modern tool usage: Use the modern engineering skills, techniques and tools that include IT tools necessary for mechanical engineering practice.

Po6.Theengineerandsociety: Apply the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

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

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities during professional practice.

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

PO10.Communication: Communicate effectively on complex engineering activities to various groups, ability to write effective reports and make effective presentations.

PO11. Project management and finance: Demonstrate and apply the knowledge to understand the management principles and financial aspects in multidisciplinary environments.

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

PROGRAM SPECIFIC OUTCOMES

Mechanical Engineering Graduates will be able to:

PSO1: Apply the knowledge of CAD/CAM/CAE tools to analyse, design and develop the products and processes related to Mechanical Engineering.

PSO 2: Solve problems related to mechanical systems by applying the principles of modern manufacturing technologies.

PSO 3: Exhibit the knowledge and skill relevant to HVAC and IC Engines.

CODE OF CONDUCT

1. Students should report to the concerned labs as per the time table schedule.
2. Students who turn up late to the labs will in no case be permitted to perform the experiment scheduled for the day.
3. After completion of the experiment, certification of the concerned staff in-charge in the observation book is necessary.
4. Staff member in-charge shall award marks based on continuous evaluation for each experiment out of maximum 15 marks and should be entered in the evaluation sheet/attendance register.
5. Students should bring a note book of about 100 pages and should enter the readings/observationsintothenotebookwhileperformingtheexperiment.
6. The record of observations along with the detailed experimental procedure of the experiment performed in the immediate last session should be submitted and certified by the staff member in-charge.
7. Not more than three students in a group are permitted to perform the experiment on a setup for conventional labs and one student in case of computer labs.
8. The components required pertaining to the experiment should be collected from stores in-charge after duly filling in the requisition form.
9. When the experiment is completed, students should disconnect the setup made by them, and should return all the components/instruments taken for the purpose.
10. Any damage of the equipment or burn-out of components will be viewed seriously either by putting penalty or by dismissing the total group of students from the lab for the semester/year.
11. Students should be present in the labs for the total scheduled duration.
12. Students are required to prepare thoroughly to perform the experiment before coming to Laboratory.

DO'S

1. Leave footwear & bag outside the laboratory at their designated place.
2. Enter the system number in the register & use the system alone.
3. Report any broken plugs, exposed electrical wires or any unsafe conditions to your lecturer/laboratory staff immediately.
4. Read and understand the procedure from Lab Manual as how to carry out an activitythoroughly before coming to the laboratory.
5. Always keep anti-virus in active mode
6. Students must carry their Identity Cards & Observation Notes in the Lab.
7. Enter or Leave the lab only with the permission of the lab in charge.
8. Turn off the respective system and arrange the chairs properly before leaving thelaboratory.

DON'TS

1. Do not install, uninstall or alter any software on computer.
2. Do not touch electrical fittings nor connect or disconnect any plug or cable.
3. Do not plug in external drives like pen drive, external hard disk or mobile phone
4. Students are not allowed to work in the Lab without the presence of faculty or instructor.
5. Do not leave your place, misbehave or make noise while in the Lab.
6. Don't scatter around unwanted things while doing an experiment.
7. Do not eat or drink in the laboratory.

COURSE OBJECTIVES

The objectives of this course are

1	Understand the fundamentals of programming in C Language.
2	Write, compile and debug programs in C.
3	Formulate solution to problems and implement in C.
4	Effectively choose programming components to solve computing problems

COURSE OUTCOMES

CO No.	Course Outcomes	PO
CO 1	Choose appropriate data type for implementing programs in C language	1,2,3,4,5,8,9,10
CO 2	Design and implement modular programs involving input output operations, decision making and looping constructs	1,2,3,4,5,8,9,10
CO 3	Apply derived data types and implement programs to store data in structures and files	1,2,3,4,5,8,9,10
CO 4	Develop confidence for self-education and ability towards lifelong learning need of computer languages	1,2,3,4,5,8,9,10

COURSE OUTCOMES VS POs MAPPING

S. NO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
6ES351CS.1	3	1	1	1	2	-	-	2	3	2	-	-	-	1	-
6ES351CS.2	3	1	0	2	0	-	-	3	3	2	-	-	-	1	-
6ES351CS.3	3	3	0	0	3	-	-	3	3	1	-	-	-	1	-
6ES351CS.4	3	2	0	0	3	-	-	3	3	0	-	-	-	1	-
Avg	3.0	1.7	0.25	0.75	2.7	-	-	2.75	3	1.25	-	-	-	1	-

LIST OF EXPERIMENTS

Exp. No.	Experiment Name
1.	Finding maximum and minimum of given set of numbers, finding roots of quadratic equation
2.	Sin x and Cos x values using series expansion.
3.	Generating Pascal triangle, pyramid of numbers.
4.	Factorial, Fibonacci, GCD recursive and non-recursive procedures
5.	Linear search and binary search using recursive and non-recursive procedures.
6.	Bubble sort and selection sort.
7.	Matrix addition and multiplication using arrays,
8.	Programs on pointers: pointer to arrays, pointer to functions.
9.	Programs on structures, union, enum and string manipulations.
10.	File handling programs (Reading, Writing, Copying files)
11.	Program illustrating using Command Line Arguments

INDEX

Experiment No	Experiment Name	Date	Page No	Marks				Remarks/ Signature
				P	R	V	T	

Experiment No	Experiment Name	Date	Page No	Marks				Remarks/ Signature
				P	R	V	T	

Introduction to C Language

CONTENTS

Learning Objectives

1. To understand fundamentals of C language
2. To know structure of C program
3. To develop algorithm/flowcharts
4. To know applications of C language

1. Introduction

Programming in C is one of the most important papers for Electronic Science subject at post graduate level. Every student planning to make career in Electronics Science should have some kind of programming skills. One may ask a question like "***What are the different computer languages an electronics student must learn for his career***?" It is like asking a question "How many cars a driver should practice to learn driving? Practice driving only one car. It is sufficient. Similarly, mastering one programming language is enough. Learn the better implementation strategies in programming.

Overall, in the language learning what is most important is to have deep knowledge about what to tell computer to do and follow the different implementation strategies like how to write algorithm, how to work with computer memory and operating system effectively. Ultimately, you should learn only as much as required to get solution to a problem. Remember that ***language is not the solution but it is a tool to solve the problem***. This course is designed to teach you how to program in C. It assumes no previous background of any programming language. In this module, the focus is on understanding the fundamentals of computer programming and general introduction toC language.

2. Fundamentals of computer programming

Computers are really dumb machines because they do only what they are asked to do. Many computers perform operations on a very basic level like addition/subtraction of numbers etc. These basic operations are decided by the instruction set of the computer. To solve a problem using a computer, it is necessary to express the method to solve the problem using the primitive instructions of the computer. A sequence of instructions given to the computer is known as program. The method or strategy used for solving the problem is known as algorithm. With the algorithm in hand, it is possible to write the instructions necessary to implement the algorithm. Programs are developed to solve some problem. These programs can be written using some computer language like Basic, C, C++, Java etc.

Language levels

When computers were first developed, the only way to program them was in terms of ***binary***

numbers. These binary numbers are corresponds to machine instructions and referred to as ***Machine language*** and are stored in computer's memory. The next step in programming was the development of ***assembly languages*** . This has enabled the programmer to work with the computers on a slightly higher level. Instead of specifying the sequences of binary numbers to carry out particular tasks, the assembly language permits the programmer to use symbolic names to perform various operations. An **assembler** translates the assembly language program into the machine instructions i.e in binary.

Table-1: World of programming languages.

High level languages	**Ada**
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
Middle level languages	Java
	C++
	C
	FORTH
	Macro-assembler
Low level language	Assembler

The machine language and the assembly language is popularly known as **Low level language**. The programmer must learn the instruction set of the particular computer system to write a program in assembly language, and the program is not portable. The program will not run on a different processor. This is because different processors have different instruction sets. As these assembly language programs are written in terms of seperate instruction sets, they are **machine dependent**.

For writing the program once independent of the processor, it was necessary to standardize the syntax of a language. Higher-level language was introduced so that a program could be written in the language to be **machine independent**. That is, a program could run on any machine that supported the language with few or no changes. To execute a higher-level language, a special computer program must be developed that translates the statements of the higher-level language into a form that the computer can understand. This program is known as a **compiler**. Table -1 indicates the position of C in programming languages.

Operating System

Operating system is program that controls the entire operations of computer system. Access to all the resource of the computer e.g. memory and input/output devices is provided through the operating system. This program looks after the file management, memory management and I/O management. It is a program that allows all other program to use the computing capability of the computer and execute them. The most popular operating systems are Windows, Linux, Unix, Mac OS X, MSDOS etc.

Program Execution

Before writing a computer program one must be clear about the steps to be performed by the computer for processing it. To produce an effective computer program it is necessary that every instruction is written in the proper sequence. For this reason, the program must be planned before writing. An algorithm represents the logic of the program. When an algorithm is expressed in the programming language, it becomes theprogram.

A compiler is simply a program developed in a particular computer language usually a higher level language and then translates it into a form that is suitable for execution on a particular computer system. Fig. 1 indicates the different stages right form program coding to execution, Initially, the **text editor** will help the programmer to enter the C program and store it as file with and extension ".c". The program written by the user with the help of text editor is

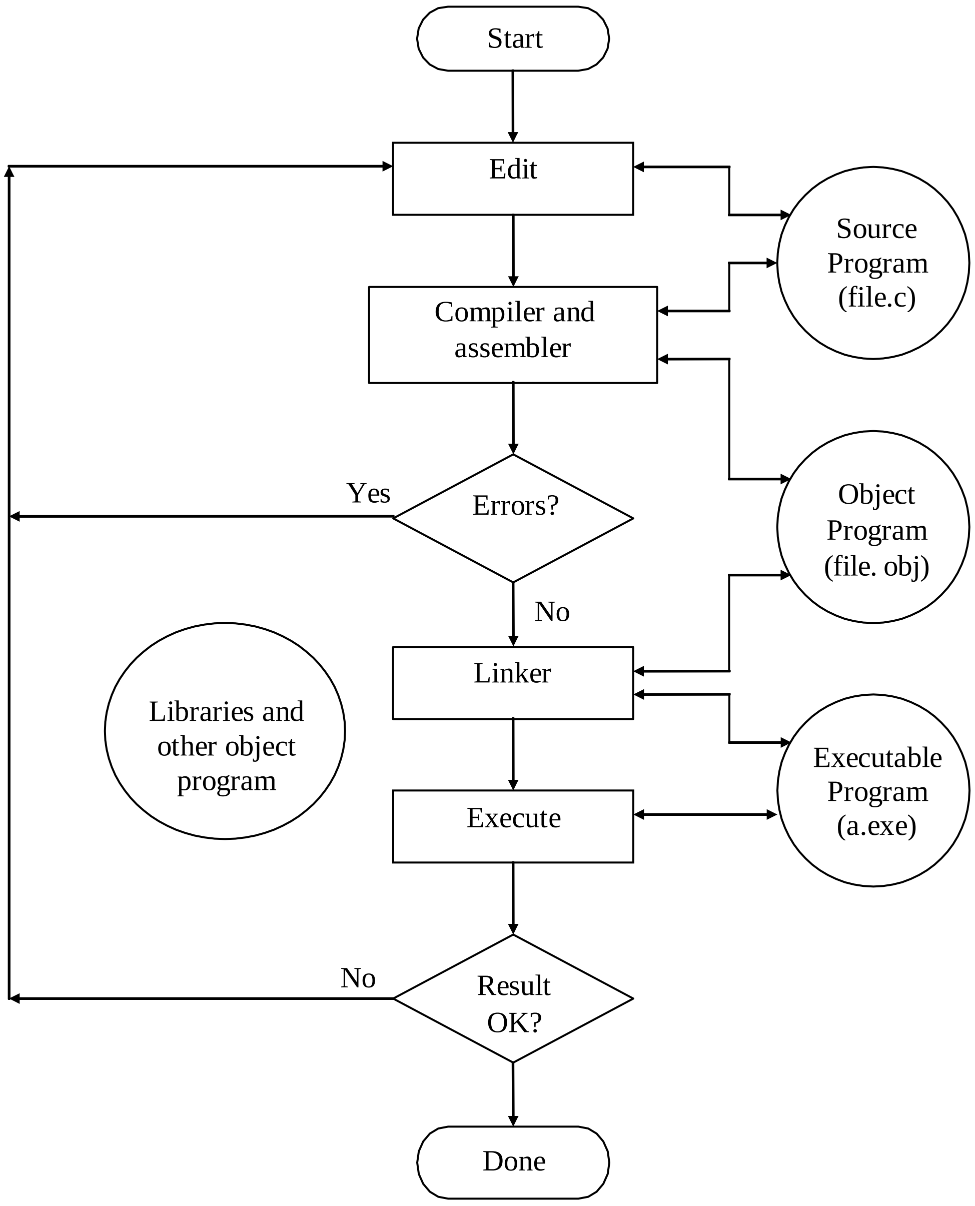


Figure 1: Program development stages

known as the source program. Every source program must have a valid filename e.g. “file.c”. Once the source program is entered into a file then one can proceed to the compilation,

To start the compilation process, the file containing the source C program must be specified. Compiler examines each program statement present in the source programs for correct syntax and semantics of the language. If there are any mistakes or errors detected by the compiler then these are reported to the user and compila tion process isterminated. The errors must be removed from the source program and then compilation process may be reinitiated.

Once all the syntax error (e.g. missing parentheses) or semantic error (e.g. variable not defined) have been removed then compiler restarts it process. Compiler takes the statements of the program and translates it into lower language. After the program has been translated into lower level language like assembly language, the next stage is to translate these assembly instructions into machine language. In many compilers, this stage is automatically executed along with the compilation process. The result of this compilation process is the generation of the object file from the given source file. This object file has an extension ".obj" and contains the object code. The filename is just same as source filename but with different extension name.

After the program is converted into object code, the **linker** automatically starts the process. The purpose of linker phase is to link the present file with any previously compiled file and system's library. At the end of linking the system generates a file with the same source filename but with extension ".exe". Once executable file is generated then the program is ready to execute or run. Computer system execute each program statement sequentially. If any external data is required then user must input it. Once the processing is over the result is displayed as the output of the program.

If the desired results are obtained the complete process is over else go back to the editor and check for the logical error. At this state, the built in debugger can help programmer to remove the bugs from the program. In this case the entire processor of editing, compiling, linking and executing the program. Usually, the process of editing, compilation, executing and debugging is managed under one umbrella by a single Integrated Development Environment (IDE).

3. Introduction to C language

Before learning how to write programs in C it would be important to know a brief history, features, structure and applications of C language. In this section, the attempt is made to present this information in short.

History of C Language

The origin of C is closed related with the development of UNIX operating system for PDP-7 computers. UNIX operating system was originally written by Ritchie and Thompson using **assembly language**. Even for PDP-11, the operating system was developed using assembly language. Developers were planning to rewrite OS using the **B language** using the Thompson’s simplified version of BCPL (Basic Combined Programming Language).

The development of C language started in 1972 on PDP-11 UNIX system by Dennis Ritchie. The name of C language simply continued the alphabetic order. C is the result of a development process that started with an older language called BCPL. It was developed by Martin Richards, and it influenced a language called B, which was implemented by Ken Thompson. B language led to the development of C language in the 1970s.

In the summer of 1983 a committee was established to create an ANSI (American National Standards Institute) standard that would define the C language. This process took much time but the ANSI C standard was finally adopted in December 1989, with the first copies becoming available in early 1990. The ANSI C was also adopted by ISO (International Standards Organization). The 1989 standard for C, along with Amendment 1, became a base document for Standard C++, defining the C subset of C++. The version of C defined by the 1989 standard is commonly referred to as C89.

During the 1990s, the development of the C++ standard consumed most programmers' attention. However, work on C continued quietly along, with a new standard for C being developed. The end result was the 1999 standard for C, usually referred to as C99.

Features of C

C is a general-purpose high level language that was originally developed for the UNIX operating system. The UNIX operating system and virtually all UNIX applications are written in C language. C has now become a widely used professional language for various reasons. Figure 2 indicates feature of c at a glance.

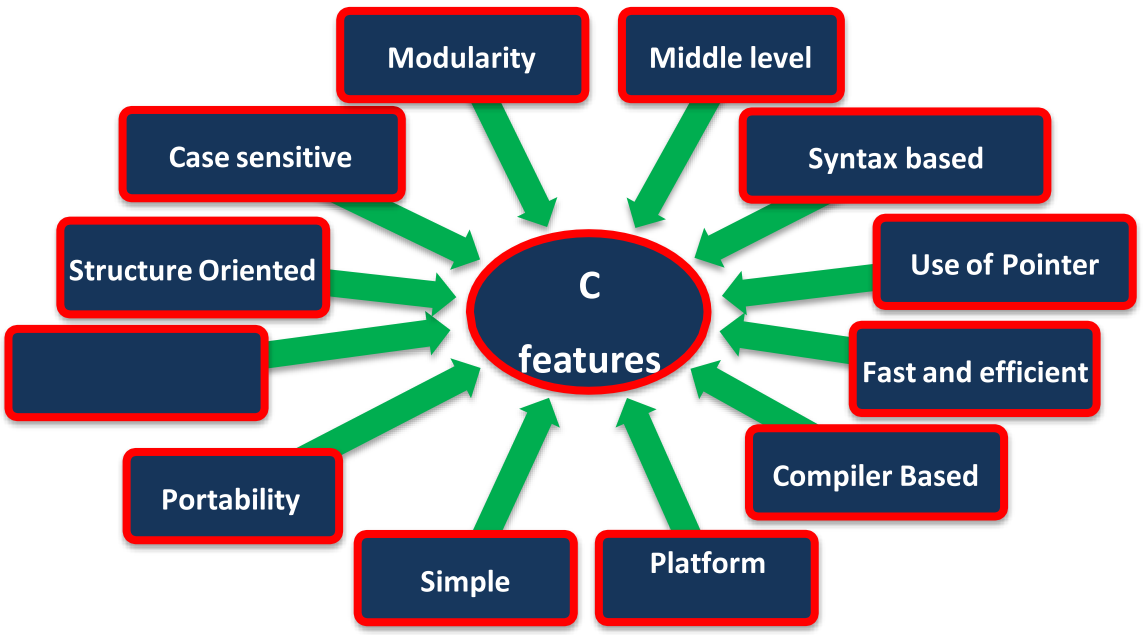


Figure 2: Features of C – at a glance

It is a very simple and easy language. C language is mainly used for develop desktop based application. All other programming languages were derived directly or indirectly from C programming concepts. C language has following features;

1. **Simple**

Every c program can be written in simple English language so that it is very easy to understand and developed by programmer.

2. **Portability**

C programs are portable. It is the concept of carrying the instruction from one system to another system. In C language **.C** file contain source code, we can edit also this code. **.exe** file contain application, only we can execute this file. When we write and compile any C program on window operating system that program easily run on other window based system.

3. **Powerful**

C is a very powerful programming language, it have a wide verity of data types,functions, control statements, decision making statements, etc.

4. **Structure oriented**

C is a Structure oriented programming language.Structure oriented programming language aimed on clarity of program, reduce the complexity of code, using this approach code is divided into sub-program/subroutines. These programming have rich control structure

5. **Case sensitive**

It is a case sensitive programming language. In C programming 'break and BREAK' both are different

6. **Modularity**

It is concept of designing an application in subprogram that is procedure oriented approach. In c programming we can break our code in subprogram.

For example we can write a calculator programs in C language with divide our code in subprograms.

7. **Middle level language**

C programming language can supports two level programming instructions with the combination of low level and high level language that's why it is called middle level programming language.

8. **Syntax based language**

C is a strongly tight syntax based programming language.

9. **Efficient use of pointers**

Pointers is a variable which hold the address of another variable, pointer directly direct access to memory address of any variable due to this performance of application is improve. In C language also concept of pointer are available.

10. **Fast and Efficient**

C programs are fast to compile and run efficiently.

11. **Compiler based**

C is a compiler based programming language that means without compilation no C program can be executed. First we need compiler to compile our program and then execute

12. **Platform independent**

A language is said to be platform independent when the program executes independentof the computer hardware and operating system.

Therefore C is platform independent programming language.

Note: .obj file of C program is platform dependent.

c a Middle-Level Language

C is often called a middle-level computer language. It does not mean that C is less powerful, difficult to use, or less developed than a high-level language such as BASIC or Pascal. Rather, C is thought of as a middle-level language because it combines the best elements of high-level languages with the control and flexibility of assembly language. As a middle-level language, C allows the manipulation of bits, bytes, and addresses, the basic elements with which the computer functions. Despite this fact, C code is also very portable. C language is very often known as a **System Programming Language**, because it is used for writing assemblers, compilers, editors and even operating systems,

Applications of C

C language is used in wide variety of applications. Practical applications of C languageare several, right form writing the operating systems like UNIX, Windows to creating antivirus programs. Some application of C language is given below.

- To design the system software like operating system and compiler.
- To develop application software like database and spread sheets.
- For development of Graphical related application like computer simulators andmobile games.
- C programming language can be used to design Network Devices Drivers
- For writing embedded system software
- Development of Text editors and language interpreters.
- Creating video games, 3 D animations, Multimedia applications, Image processing, modelling and simulations etc.

Structure of C program

Programming structure of C language consists of header file, main function and body offunction as shown in figure 3.

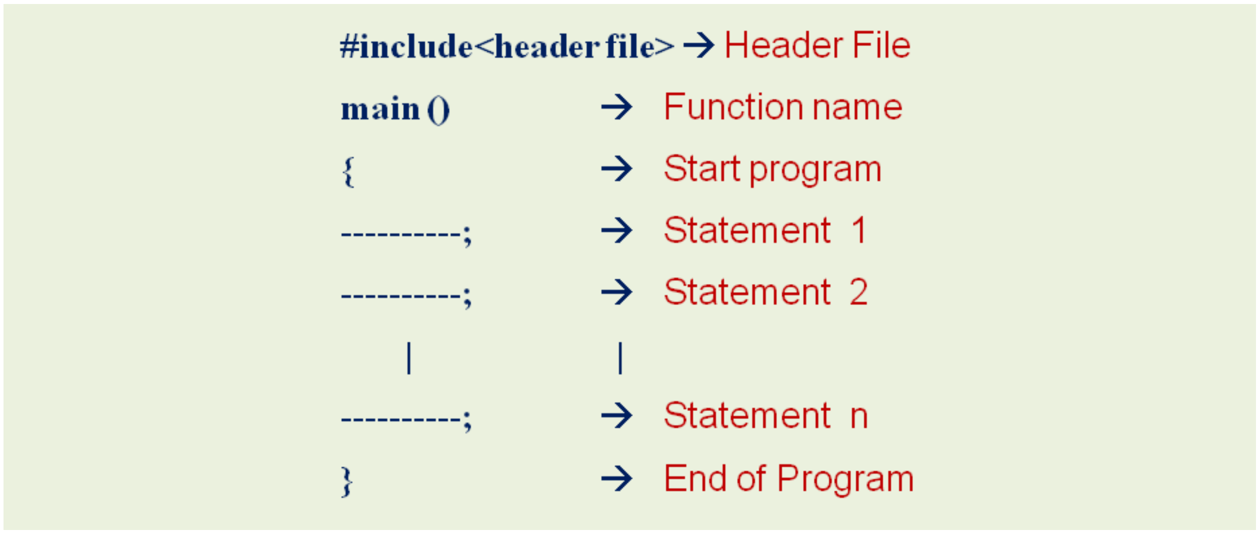

#include<header file> → Header File
main () → Function name
{ → Start program
----------; → Statement 1
----------; → Statement 2
----------; → Statement n
} → End of Program

Figure 3: Structure of c program

The C language program starts form main() function which is **Function name.** The program function start with **{** (open brace) and ends with **}** (close brace). Then valid c statements are to be written between open brace and close brace . The functions written between open and close brace called as **function body. Semicolon (;)** is used to end the valid statement.

Let us consider a very simple program; here programmer wants to display some text message.

```
#include <stdio.h>                //library function for standard input output
main ( )                          //Execution begins
{                                 //Opening brace
printf("Fergusson College");      //Executable statement
}                                  //Closing brace
```

Output of program is

```
        Fergusson College
```

Problem solving process

Any scientific and engineering problem can be solved using some Problem Solvingprocess. Here total six important steps are given to solve any general problem.

1. Understanding a problem
2. Identifying necessary inputs and expected output
3. Design an algorithm/flowchart
4. Writing or coding the program
5. Testing the code
6. Debugging

Everything begins with problem therefore it is necessary to understand the problem clearly. With a given problem, the second step is to know necessary inputs and expected outputs. By knowing the input and output parameter the next step is to design an algorithm to solve the problem or one prefer to draw the design in a graphical way using flowchart. Once the algorithm or flow chart is ready, the obvious step is to start Writing or coding the program in an appropriate programming language which is suitable for given a application Any program is not complete without testing the code with possible inputs. This stage is also referred to as program execution or running the program. If there are any errors in the program then debugging is an important step in problem solving i.e. removing the bugs or error.

4. The algorithm

An algorithm is a finite sequence of instructions which can be carried out to solve aparticular problem in order to obtain the desired results.

- A finite sequence of instructions
- The program must be planned before writing.
- Represents the logic of the program
- Instructions are to be written in the proper sequence.
- Expressed in the programming language, becomes the program

Characteristics of an algorithm

An algorithm must satisfy the following criteria:

1) Input: Zero or more quantities are externally supplied as inputs.

2) Output: At least one quantity as output is produced.

3) Finiteness: An algorithm terminates after a finite number of steps.

4) Definiteness: Each instruction must be clear and unambiguous.

5) Effectiveness: Every instruction must be very basic so that it can b carried out in principle, by a person using just a pencil and paper.

To use Turbo C, here are some common shortcut keys:

1. File Operations:
- New File: Ctrl+N
- Open File: Ctrl+O
- Save File: Ctrl+S
- Save As: Ctrl+Shift+S
- Print: Ctrl+P
- Exit: Alt+F4

2. Editing:
- Cut: Ctrl+X
- Copy: Ctrl+C
- Paste: Ctrl+V
- Undo: Ctrl+Z
- Redo: Ctrl+Y
- Find: Ctrl+F
- Find Next: F3
- Replace: Ctrl+H

3. Compiling and Running:
- Compile: Alt+F9
- Build: Ctrl+F9
- Run: Ctrl+F10

4. Debugging:
- Start Debugging: Ctrl+Shift+F9
- Step Into: F7
- Step Over: F8
- Run to Cursor: Ctrl+F4
- Set/Clear Breakpoint: F5
- View Registers: Alt+R
- View Memory: Alt+M

5. Navigation:
- Go to Line: Ctrl+G
- Next Tab: Ctrl+Tab
- Previous Tab: Ctrl+Shift+Tab

6. Other:
- Help: F1
- Switch between Code and Output: Alt+F5
- Toggle Full Screen: Alt+Enter
- Close: Alt+X

PROGRAM 1.

1. a). Finding the maximum and minimum of given set of numbers?

Aim:
To find the maximum and minimum of given set of numbers. For input & output Statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

If—else statement

if…else statement is used to make a decision based on two choices. It has the following form:
syntax:

```
if(condition)
{
True Stmt block;
}
Else
{
False stmt block;
}
Stmt-x;
```

In this syntax,

If and else are the keywords.

<*condition*> is a relational expression or logical expression or any expression that returns either true or false. It is important to note that the condition should be enclosed within parentheses (and). The *true stmt block* and false stmt block are simple statements or compound statements or null statements. *Stmt-x* is any valid C statement.

Algorithm:

```
Step 1: Start
Step 2: Read ‘n’, which is number of elements
Step 3: If i<n then
Step 4: Accept elements into an array list[i] upto given range
Step 5: min=max=sum=list[0]
Step 6: If i<n then
Step 7: If list[i]>max then
Step 8: max is equal to list[i]
Step 9: If list[i]<min then
Step 10: min is equal to list[i]
Step 11: sum is equal to sum plus list[i]
Step 12: Repeat steps 7 to 11 until i<n
Step 15: Print “max”, “min” and “average”
Step 16: Exit
Step 17: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int a[50], i, n, max, min; clrscr();
        printf(“Enter size of the array: \n”);
        scanf(“%d”, &n);
        printf(“Enter elements in the array: \n”); for(i
        = 0;i < n;i++)
        { scanf(“%d”, &a[i]);
        } max=a[0];
        min=a[0]; for(i =
        1;i < n;i++)
        { if(a[i] > max)
                max=a[i];
                if(a[i] < min)
                        min=a[i];
        } printf(“Maximum element = %d
\n”, max); printf(“Minimum element = %d
\n”, min);
getch();
}
```

Expected Output :

```
Enter size of the array: 6
Enter elements in the array:     78 45 67 23 14 49
Maximum element = 78
Minimum element = 14
```

1. b). Finding the roots of a given Quadratic equation?

Aim:
To find the roots of a given quadratic equation. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Algorithm:

```
Step 1: Start
Step 2: Input the values for a, b, c
Step 3: d=b*b –4*a*c
Step 4: if d greater than zero
        a)  print roots are read and distinct
        b)  r1= -b+ √ d / (2*a)
        c)r2= -b-√ d / (2*a)
Step 5: Print r1 and r2, go to 7
Step 6: if d is equal to zero
a) Print roots are real and equal
b) r1 = - b/2*a
          Print r1, go to 7 else
print roots are imaginary Step
7: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h> void
main()
{ float a, b, c, d, r1, r2, rp, ip; clrscr();
        printf("Enter a b c values: \n");
        scanf("%f%f%f", &a, &b, &c);
        if(a = = 0)
        { printf(" not a QE");
        }
        else
                        { d =
                b*b-4*a*c;
        } if(d >=
        0)
        {
                printf("Roots are real \n");
                r1 = (-b + sqrt(d)) / (2*a);
                r2 = (-b - sqrt(d)) / (2*a);
                printf("Roots are %f and %f \n", r1, r2);
        }
        else
        { printf("Roots are imaginary \n");
                rp = -b / (2*a); ip =
                sqrt(-d) / (2*a);
                printf("Roots are (%f , %f) and (%f , %f) \n", rp, ip, rp, -ip);
        } getch();
}
```

Expected Output 1:

```
Enter a b c values:     3 5 2
Roots are real Roots are -0.666667 and -1.000000
```

Expected Output 2:

```
Enter a b c values:     4 2 8
Roots are imaginary Roots are (-0.250000, 1.391941) and (-0.250000, -1.391941)
```

PROGRAM 2. Sin x and Cos x values using series expansion

2. a). Sin x value using series expansion.

Aim: To find the value of sin x using series expansion. For input & output statements we include the header file <stdio.h>, for arithmetic operations include the <math.h> and for clear the screen include the <conio.h>.

Description:

for statement:
It is one of the looping control statements. It is also called as *entry-controlled looping control statement*. i.e., it tests the condition before entering into the loop body. The syntax for "for" statement is as follows:

Syntax:

```
for(exp1;exp2;exp3)

for-body

next_statement;
```

In this syntax,

for is a keyword. exp1 is the initialization statement. If there is more than one statement, then those should be separated with commas. exp2 is the condition. It is a relational expression or a compound relational expression or any expression that returns either true or false. The exp3 is the updating statement. If there is more than one statement, then those should be separated with commas. exp1, exp2 and exp3 should be separated with two semi-colons. exp1, exp2, exp3, for-body and next_statement are valid 'c' statements. for-body is a simple statement or compound statement or a null statement.

Algorithm:

```
    Step 1: Start the program.
    Step 2: Declare all the required variables.
    Step 3: Read the Input for Number of terms (n) and Angle in degres (x).
    Step 4: Compute x = (x * 3.14) / 180.
    Step 5: Set sum = x and term = x.
    Step 6: if(i < = n) then goto step 7.
    Step 7: Compute and set term = (term * (-1) * x * x) / ((2 * i) * (2 * i + 1)).
    Step 8: Compute and set sum+ = term.
    Step 9: Compute i = i + 1 then goto step 6.
```

Step 10: Display the sinx value.

Step 11: Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h> void
main()
{
          float  x,  sum,
          term; int i, n;
          clrscr();
          printf("Enter the no of terms: \n");
          scanf("%d", &n);
          printf("Enter the angle in degrees x: \n");
          scanf("%f", &x);
          x = (x*3.14) /
          180;
          sum = x;
          term = x;
          for(i = 1;i <= n;i++)
          { term = (term*(-1)*x*x) / ((2*i)*(2*i+1));
                    sum+ = term;
          } printf("Sin valve of given angle is
%f", sum); getch();
}
```

Expected Output:

```
Enter the no of terms: 3
Enter the angle in degrees x: 30
Sin valve of given angle is 0.499770
```

2. b). Cos x value using series expansion?

Aim:
To find the value of cos x using series expansion. For input & output statements we include the header file <stdio.h>, for arithmetic operations include the <math.h> and for clear the screen include the <conio.h>.

Algorithm:

Step 1: Start the program.

Step 2: Declare all the required variables.

Step 3: Read the Input for Number of terms (n) and Angle in degres (x).

```
Step 4: Compute x = (x * 3.14) / 180.
Step 5: Set sum = 1 and term = 1.
Step 6: if(i < = n) then goto step 7.
Step 7: Compute and set term = (term * (-1) * x * x) / ((2 * i) * (2 * i - 1)).
Step 8: Compute and set sum+ = term.
Step 9: Compute i = i + 1 then goto step 6.
Step 10: Display the cosx value.
Step 11: Stop the program.
```

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h> void
main()
{ float x, sum, term; int i,
        n; clrscr();
        printf("Enter the no of terms: \n");
        scanf("%d", &n);
        printf("Enter the angle in degrees x: \n");
        scanf("%f", &x); x
        = (x*3.14) / 180;
        sum = 1; term = 1;
        for(i = 1;i <=
        n;i++)
        { term = (term*(-1)*x*x) / ((2*i)*(2*i-1));
                sum+ = term;
        } printf("Cos valve of given angle is
%f", sum); getch();
}
```

Expected Output:

```
Enter the no of terms: 3
Enter the angle in degrees x: 30
Cos valve of given angle is 0.866158
```

PROGRAM 3. Generating a Pascal Triangle and Pyramid of Numbers.

3. a). Generating a Pascal Triangle?

Aim:
To print pascal triangle. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

All values outside the triangle are considered zero (0). The first row is 0 1 0 whereas only 1 acquire a space in pascal's triangle, 0s are invisible. Second row is acquired by adding (0+1) and (1+0). The output is sandwiched between two zeroes. The process continues till the required level is achieved. Pascal's triangle can be derived using binomial theorem. We can use combinations and factorials to achieve this.

Algorithm:

```
Step1:  Start.
Step2: Declare variables x, y, n, a, z, s.
Step3:  Enter the limit.
Step4:  Initialize the value of variables, s=n , x=0, y=0 , z=s.
Step5: Do the following operations in loop. x = 0 to n. a= 1, x++ z=s to 0.
Step6: print space. z—- y = o to x. print a. a = a*(x-y)/(y+1) y= y+1. go to next line.
Step7:  Repeat the process to n.
Step8: Print the final required triangle.
Step9:  Stop.
```

Program:

```
#include<stdio.h>
#include<conio.h>
long fact(int); void
main()
{
        int n, i, j;
        clrscr();
        printf("Enter the no. of lines: \n");
        scanf("%d", &n);
        for(i = 0;i < n;i++)
        { for(j = 0;j < n-i-1;j++) printf(" "); for(j = 0;j <=
                i;j++) printf("%ld ", fact(i) / (fact(j)*fact(i-j)));
                printf("\n");
        } getch();
}
long fact(int num)
{ long f = 1;
```

```
                int i = 1;
                while(i <= num)
                {
                        f = f*i; i++;
                } return
                f;
        }
```

Expected Output:

```
Enter the no. of lines:8
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
```

3. b). Generating a Pyramid of Numbers (Floyd's Triangle)?

Aim:
To print Pyramid of Numbers (Floyd's Triangle). For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

A **pyramid of numbers** is a graphical representation that shows the **number** of organisms at each trophic level. It is an upright **pyramid** in light of the fact that in an ecosystem, the producers are always more in **number** than other trophic levels
.
Algprithm:

```
Step 1 - Take number of rows to be printed, n.
Step 2 - Make outer iteration I for n times to print rows
Step 3 - Make inner iteration for J to I
Step 3 - Print n
Step 4 - Print NEWLINE character after each inner iteration Step
5 - Return
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int i, j, r, k = 1; clrscr(); printf("Enter the
        range: \n"); scanf("%d", &r);
        printf("FLOYD'S TRIANGLE: \n \n");
        for(i = 1;i <= r;i++)
        { for(j = 1;j <= i;j++,k++)
                printf(" %d", k);
                printf("\n");
        } getch();
}
```

Expected Output:

```
Enter the range: 10
FLOYD'S TRIANGLE
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 7 48 49 50 51 52 53 54 55
```

PROGRAM 4. Recursion and Non-Recursive: Factorial, Fibonacci, GCD.

4. a). Recursion and Non-Recursive Factorial?

Aim:
To find factorial of a given number using recursion. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

Recursion is the process of repeating items in a self-similar way…...The **C** programming language supports **recursion**, i.e., a function to call itself. But while using **recursion**, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Algorithm:

```
Step 1: Start
Step 2: Read 'n'
Step 3: a=recfactorial(n)
Step 4: if 'x' is equal to zero then return 1 else
Step 5: f = x*recfactorial(x-1) and return value of 'f'
Step 6: Print "recfactorial"
Step 7: b=nonrecfactorial(n)
Step 8: Initialize 'f' to one
Step 9: If i<=x then f=f*I until i<=n
Step 10: End for and return 'f'
Step 11: Print "nonrecfactorial"
Step 12: Exit
Step 13: Stop
```

Program:

```c
#include  <stdio.h>
#include <conio.h>
void main()
{
  int n, a, b;
  clrscr();
  printf("Enter any number\n");
  scanf("%d", &n);
  a = recfactorial(n);
  printf("The factorial of a given number using recursion is %d \n", a);
  b = nonrecfactorial(n);
  printf("The factorial of a given number using nonrecursion is %d ", b);
  getch();
}
int recfactorial(int x)
{
  int f;
  if(x == 0)
  {
   return(1);
  }
  else
  {
   f = x * recfactorial(x - 1);
   return(f);
  }
}
int nonrecfactorial(int x)
{
  int i, f = 1;
  for(i = 1;i <= x; i++)
  {
    f = f * i;
  }
  return(f);
}
```

Expected Output:

```
Enter any number
5
The factorial of a given number using recursion is 120
The factorial of a given number using nonrecursion is 120
```

4. b). Recursive and Non-Recursive: Fibonacci

Aim:
To print fibonacci series using recursion. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:
The Fibonacci Sequence is the series of numbers:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
The next number is found by adding up the two numbers before it:

- the 2 is found by adding the two numbers before it (1+1),
- the 3 is found by adding the two numbers before it (1+2), • the 5 is (2+3),
- and so on!

Algorithm:

```
Step 1: Start

Step 2: Read n

Step 3: a=0, b=1

Step 4: write a, b

Step 5: c= a + b

Step 6: write c

Step 7: a=b

Step 8: b=c

Step 9: n--

Step 10: Repeat Steps 5 to 9 until n>0

Step 11: Stop
```

Recursive Program:

```
#include<stdio.h> #include<conio.h>
void fibonacci(int);

void main()
{ int k, n;
        long int i = 0, j = 1, f;
        clrscr();
        printf("Enter the range of the Fibonacci series: \n");
        scanf("%d", &n); printf("Fibonacci Series: \n");
        printf("%2d %2d ", i, j); fibonacci(n); getch();
}

void fibonacci(int n)
{ static long int first = 0, second = 1, sum; if(n
        > 0)
        { sum = first + second; first
                = second; second =
                sum; printf("%ld ",
                sum); fibonacci(n-);
        }
}
```

Expected Output:

```
Enter the range of the Fibonacci series:10
Fibonacci Series:0 1 1 2 3 5 8 13 21 34 55 89
```

Non-Recursive Program:

```
#include <stdio.h>

int main()
{
   int i, n, firstTerm=0, secondTerm=1, sum=0;
   printf("\nEnter number of terms required in Fibonacci Series: ");
   scanf("%d",&n);

    // Showing the first two term of the Fibonacci Series
   printf("\nFibonacci Series is:\n\n\n %d %d ", firstTerm, secondTerm);
   //start i =2
   i=2;
   //i starts from 2, as the first two terms of the series have already been shown
   while (i<n)
   {
      sum=firstTerm+secondTerm;
      firstTerm=secondTerm;
      secondTerm=sum;
      ++i;
      printf("%d ",sum);
   }
   return 0;
}
```

Expected Output:

```
Enter number of terms required in Fibonacci Series: 8

Fibonacci Series is:

 0 1 1 2 3 5 8 13
```

4. c). Recursive and Non-Recursive GCD

Aim:
To find gcd of a number using recursion. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

The GCD of two numbers in C, i.e., integer 8 and 12 is 4 because, both 8 and 12 are divisible by 1, 2, and 4 (the remainder is 0) and the largest positive integer among the factors 1, 2, and 4 is 4. GCD of two numbers in the C program allows the user to enter two positive integer values. Then, we are going to calculate the Highest Common Factor of those two values.

<u>Algorithm :</u>

```
Step 1: Start
Step 2: Read ‘a’ and ‘b’
Step 3: Call gcdrecursive(n,m)
Step 4: If n>m then return gcdrecursive(n,m)
Step 5: If n is equal to zero return value of ‘m’ else
Step 6: Return the value of gcdrecursive(n,m%n)
Step 7:  Print ‘a’, ‘b’ and gcdrecursive(a,b)
Step 8: Call gcdnonrecursive(p,q)
Step 9: remainder=p-(p/q*q)
Step 10: If remainder is zero then return value of ‘q’ else
Step 11: Call gcdrecursive(q,remainder)
Step 12: Print ‘a’, ‘b’ and gcdnonrecursive(a,b)
Step 13: Exit
Step 14: Stop
```

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a, b, c, d;
  clrscr();
  printf("Enter two numbers a,
        b\n");
  scanf("%d%d", &a, &b);
  c = recgcd(a, b);
  printf("The gcd of two
        numbers using
        recursion is %d\n", c);
  d = nonrecgcd(a, b);
```

```
   printf("The gcd of two
            numbers using
            nonrecursion is %d",
            d);
   getch();
   }
int recgcd(int x, int y)
{
   if(y == 0)
   {
      return(x);
   }
   else
   {
     return(recgcd(y, x % y));
   }
}
int nonrecgcd(int x, int y)
{
   int z;
   while(x % y != 0)
   {
    z = x % y;
    x = y;
    y = z;
   }
   return(y);
}
```

Expected Output:

```
Enter two numbers a, b
3 6
The gcd of two numbers using recursion is 3
The gcd of two numbers using nonrecursion is 3
```

PROGRAM 5. Matrix Addition and Multiplication using Arrays

5. a). Matrix Addition using Arrays

Aim:
To perform addition of two matrices. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

Array:- An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of same data type and each item can be accessed using the same name.

Declaration of an array:- We know that all the variables are declared before the are used in the program. Similarly, an array must be declared before it is used. During declaration, the size of the array has to be specified. The size used during declaration of the array informs the compiler to allocate and reserve the specified memory locations.

Syntax:- data_type array_name[n]; where, n is the number of

data items (or) index(or) dimension.

0 to (n-1) is the range of

array. **Ex:** int a[5]; float

x[10];

Algorithm:

```
Step 1: Start the process.
Step 2: Read the values for m,n,p,q.
Step 3: If(m!=p || n!=q) then goto step 4 else goto step 6.
Step 4: Print ‘addition is not possible’. Step
5: Initialize I=0,j=0.
Step 6: If(I<m) then goto step 7.
Step 7: If(j<n)then goto step 8.
Step 8: Read a[I][j] goto step 9.
Step 9: Set I=I+1,j=j+1.
Step 10: If(I<p) then goto step 11.
Step 11: If(j<q)then goto step 12.
Step 12: Read b[I][j] goto step 9.
Step 13: Compute & set c[i][j] = a[i][j] + b[i][j].
Step 14: Display the value of c[I][j] goto step 9.
```

Step 15: Stop the process.

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int a[5][5], b[5][5], c[5][5], i, j, n, m, p, q; clrscr(); printf("Enter
        first matrix size: \n"); scanf("%d%d", &m, &n);
        printf("Enter second matrix size: \n"); scanf("%d%d", &p,
        &q); if(m != p || n != q) printf("Size mismatch, Addition is
        not possible \n");
        else
        { printf("Enter first matrix elements: \n"); for(i
                = 0;i < m;i++) for(j = 0;j < n;j++)
                scanf("%d", &a[i][j]);
                printf("Enter second matrix elements: \n");
                for(i = 0;i < p;i++) for(j = 0;j <
                        q;j++) scanf("%d", &b[i][j]);
                printf("Addition of two Matrices is \n");
                for(i = 0; < m;i++)
                { for(j = 0;j < n;j++) c[i][j] = a[i][j] +
                        b[i][j];
                }
                for(i = 0;i < m;i++)
                { for(j = 0;j < n;j++) printf("%4d",
                        c[i][j]);
                        printf("\n");
                }
        }
        getch();
}
```

Expected Output 1:

```
Enter first matrix size:        2
                                 3

Enter second matrix size:       2
                                 3
Enter first matrix elements:    1
                                 2
                                 3
                                 4
                                 5
                                 6
Enter second matrix elements: 6
                                 5
```

```
                                    4
                                    3
                                    2
                                    1

    Addition of two Matrices is:     7 7 7
                                     7 7 7
```

Expected Output 2:

```
    Enter first matrix size:         2
                                     3

    Enter second matrix size:        4
                                     3

    Size mismatch, Addition is not possible
```

5. b). Matrix Multiplication using Arrays

Aim:
To multiply two matrices using arrays. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Algorithm:

```
Step 1: Start the process.
Step 2: Read the values for m,n,p,q.
Step 3: If(n!=p) then goto step 4 else goto step 6.
Step 4: Print 'multiplication is not possible'. Step
5: Initialize I=0,j=0.
Step 6: If(I<m) then goto step 7.
Step 7: If(j<n)then goto step 8.
Step 8: Read a[I][j] goto step 9.
Step 9: Set I=I+1,j=j+1.
Step 10: If(I<p) then goto step 11.
Step 11: If(j<q)then goto step 12.
Step 12: Read b[I][j] goto step 9.
Step 13: If((I<m) &&(j<q) )then set c[I][j]=0 else goto step 9.
Step 14: Compute & set c[i][j] = c[I][j]+ a[i][k] * b[k][j].
Step 15: Diplay the value of c[I][j] goto step 9.
```

Step 16: Stop the process.

Program :

```
#include<stdio.h>
#include<conio.h> void
main()
{ int a[5][5], b[5][5], c[5][5], i, j, n, m, p, q;
        clrscr();
        printf("Enter first matrix size: \n"); scanf("%d%d", &m, &n);
        printf("Enter second matrix size: \n"); scanf("%d%d", &p, &q);
        if(n != p) printf("Not compatible, Multiplication is not possible
        \n");
        else
        { printf("Enter first matrix elements: \n"); for(i
                = 0;i < m; i++) for(j = 0;j < n; j++)
                scanf("%d", &a[i][j]);

                printf("Enter second matrix elements: \n");
                for(i = 0;i < p;i++) for(j = 0;j <
                     q; j++) scanf("%d",
                     &b[i][j]);

                printf("Product of two Matrices is \n");
                for(i = 0;i < m; i++)
                { for(j = 0;j < q;j++)
                     { c[i][j] = 0; for(k = 0;k < n;k++)
                           c[i][j] += a[i][k] * b[k][j];
                     }
                }
                for(i = 0;i < m;i++)
                { for(j = 0;j < q;j++)
                     printf("%4d", c[i][j]);
                     printf("\n");
                }
          }
        getch(); }
```

Expected Output 1:

```
Enter first matrix size:        2
                                3
Enter second matrix size:       4
                                3
Not compatible, Multiplication is not possible
```

Expected Output 2:

```
Enter first matrix size:          2
                                  3
Enter second matrix size:         3
                                  1
Enter first matrix elements:      1
                                  2
                                  3
                                  4
                                  5
                                  6
Enter second matrix elements: 1
 2
                                   3
Product of two Matrices is:       14
                                  32
```

Program on Linear Search and Binary Search using Recursive and Non – Recursive Procedures

6. a). Program on Linear Search using Recursive Procedure?

Aim:
To implement linear search using recursive procedure. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Algorithm:

```
Step 1: Start the process.
Step 2: Read n.
Step 3: Set i=0.
Step 4: If(i<n) then goto step 5 else goto 7.
Step 5: Read a[i].
Step 6: Set i=i+1 and goto step 4.
Step 7: Read ele.
Step 8: If(ele==a[i]) then set check=1 goto step 9 else goto step 10.
Step 9: If(check) then write element is found else write element not found.
Step 10: Stop the process.
```

Program:

```
#include<stdio.h> #include<conio.h>
int linear(int[], int, int);
void main()
{ int a[100], n, i, x;
        clrscr();
        printf("Enter Size: \n");
        scanf("%d", &n);
        printf("Enter %d elements: \n", n);
        for(i = 0;i < n;i++)
                scanf("%d", &a[i]);
        printf("Enter element to search: \n");
        scanf("%d", &x);
        i=linear(a, n-1, x);
        if(i != -1) printf("The element %d found at %d location", x,
        i); else printf("The element %d is not found", x);
        getch();
}
int linear(int a[100], int n, int x)
```

```
{ if(n<= 0) return -1; if(a[n] = = x)
        return n; else return linear(a,
        n-1, x);
}
```

Expected Output:

```
Enter Size:      5
Enter 5 elements:        25 30 12 54 60
Enter element to search:         60
The element 60 found at 4 location
```

6. b). Program on Linear Search using Non - Recursive Procedure?

Aim:
To implement linear search using non – recursive procedure. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

Linear search technique is also known as sequential search technique. The linear search is a method of searching an element in a list in sequence. In this method, the array is searched for the required element from the beginning of the list/array or from the last element to first element of array and continues until the item is found or the entire list/array has been searched. **Algorithm:**

X is an array with n elements. This algorithm search for an item and finds the location of item in array X.

```
Step 1: Start
Step 2: Read n,loc=-1
Step 3: Read array elements
Step 4: Enter element to be searched
Step 5: Call function
        1: Repeat Step 4, for i=0 to n-1
        2: [Search for an item in the array X]
                If (ele=a[i])
        3: Begin
            loc=i
            break
            endif end
            for
        4: if (loc>0)
                    Display message “Search is successful and element is found at location:
        loc” and  Exit
        5: else
                Display the message “Search Unsuccessful” and exit.
    Step 6: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int a[20], i, n, x, xloc = -1;
    clrscr();
    printf("Enter array size: \n");
    scanf("%d", &n);
    printf("Enter any %d elements: \n", n);
    for(i=0;i<n;i++) scanf("%d", &a[i]);
    printf("Enter the element you want to search for \n");
    scanf("%d", &x); for(i = 0;i < n;i++)
    { if(a[i] = = x)
        { xloc = i;
            break;
        }
    }
    if(xloc = = -1) printf("%d is not found in the array \n",
    x); else printf("%d is found at location %d \n", x,
    xloc); getch();
}
```

Expected Output 1:

```
Enter array size          6
Enter any 6 elements 78 45 67 23 14 49
Enter the element you want to search for       23
23 is found at location 3
```

Expected Output 2:

```
Enter array size          5
Enter any 5 elements 1 2 34 5 6
Enter the element you want to search for       89
89 is not found in the array
```

6. c). Program on Binary Search using Recursive Procedure?

Aim:
To implement binary search using recursive procedure. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

Binary search is quicker than the linear search. However, it cannot be applied on unsorted data structure. The binary search is based on the approach **divide-and-conquer**. The binary search starts by testing the data in the middle element of the array. This determines target is whether in the first half or second half. If target is in first half, we do not need to check the second half and if it is in second half no need to check in first half. Similarly we repeat this process until we find target in the list or not found from the list. Here we need 3 variables to identify first, last and middle elements.

To implement binary search method, the elements must be in sorted order. Search is performed as follows:

- The key is compared with item in the middle position of an array
- If the key matches with item, return it and stop
- If the key is less than mid positioned item, then the item to be found must be in first half of array, otherwise it must be in second half of array.
- Repeat the procedure for lower (or upper half) of array until the element is found.

<u>Algorithm:</u>

```
Step 1: Start the process.
Step 2: Read n.
Step 3: Set i=0.
Step 4: If(i<n) then goto step 5 else goto 7.
Step 5: Read a[i].
Step 6: Set i=i+1 and goto step 4.
Step 7: Read ele.
Step 8: Set low=0 high=n-1 and check=0.
Step 9: If(low<=high) then goto step 10 else goto step 14.
Step 10: Set mid=(low+high)/2.
Step 11: If(ele==a[mid-1]) then set check=1 goto step else goto STEP 12.
Step 12: If(ele<a[mid-1]) then  set high=mid-1 else goto step 13.
Step 13: If(ele>a[mid-1]) then set low=mid+1.
Step 14: If(check) then write element is found else write element not found.
```

Step 15: Stop the process.

Program:

```
        #include<stdio.h>
        #include<stdlib.h>
        #include<conio.h> #define
        size 10
        int binary(int[], int, int, int);
        void main()
        { int n, i, key, position;
                int low, high, list[size];
                clrscr();
                printf("\nEnter the total number of elements: \n");
                scanf("%d", &n);
                printf("Enter the elements of list: \n"); for(i
                = 0; i < num; i++)
                { scanf("%d", &list[i]);
                } low = 0; high
                = num - 1;
                printf("Enter element to be searched: \n");
                scanf("%d", &key);
                position = binary(list, key, low, high); if(position
                != -1)
                { printf("Number present at %d \n", position);
                } else printf("The number is not present in the list \n");
                getch();
        }
int binary(int a[], int x, int low, int high)
        { int mid; if (low > high) return -1;
                mid = (low + high) / 2; if (x =
                = a[mid])
                        { return mid;
                        }
                        else if (x < a[mid])
                        { binary(a, x, low, mid - 1);
                        }
                else
                { binary(a, x, mid + 1, high); }
        }
```

Expected Output:

```
Enter the total number of elements: 5
Enter the elements of list:      11 22 33 44 55
Enter element to be searched: 33
Number present at 2
```

6. d). Program on Binary Search using Non - Recursive Procedure?

Aim:

To implement binary search using non – recursive procedure. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Algorithm:

X is sorted array. The lb and ub are lower bounds and upper bounds of segment. This algorithm search for an item and finds the location of item in array X.

```
Step 1: Start

Step 2: [Initialize variables]

            Set beg=0, end=n-1, flag=0

Step 3: Repeat Steps 3 and 4 while beg<=end Step

4: [Compute location of middle element]

            mid=(beg+end)/2

Step 5:  [compare the element]

            if x[mid] > item, then

                set end=mid-1

            else if x[mid] < item, then
                et beg= mid+1

            else

                set flag=1

            display the message “Search is successful” and exit.

Step 6:  If flag=0 then

            display the message “Search Unsuccessful” and exit.

Step 7: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int a[20], i, n, x, xloc = -1, low, high, mid;
        clrscr();
        printf("Enter array size: \n");
        scanf("%d", &n);
        printf("Enter any %d elements in the sorted order: \n",n);
        for(i = 0;i < n;i++) scanf("%d", &a[i]);
        printf("Enter the element you want to search for \n");
        scanf("%d", &x);
        low = 0; high = n-
        1; while(low <=
        high)
        { mid = (low+high) / 2;
                if(a[mid] = = x)
                { xloc = mid;
                        break;
                } else if(x < a[mid])
                high = mid - 1;
                else low = mid + 1;
        } if(xloc = = -1) printf("%d is not found in the
        array \n", x);
        else printf("%d is found at location %d \n", x, xloc);
        getch();
}
```

Expected Output 1:

```
Enter array size:        6
Enter any 6 elements: 78 45 67 23 14 49
Enter the element you want to search for      23
23 is found at location 3
```

Expected Output 2:

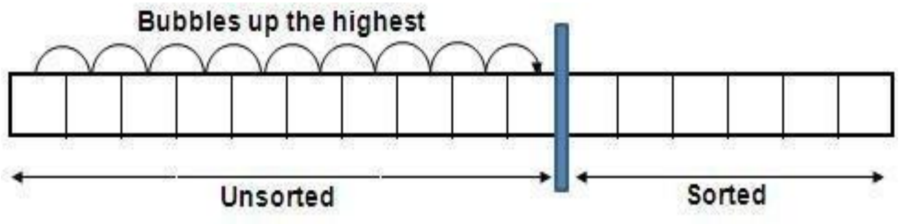
```
Enter array size:        5
Enter any 5 elements: 1 2 34 5 6
Enter the element you want to search for      89
89 is not found in the array
```

PROGRAM 7. Bubble Sort, Selection Sort

7. a). Bubble Sort

Aim:
To implement bubble sort. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:
In bubble sort method the list is divided into two sub-lists sorted and unsorted. The smallest element is bubbled from unsorted sub-list. After moving the smallest element the imaginary wall moves one element ahead. The bubble sort was originally written to bubble up the highest element in the list. But there is no difference whether highest / lowest element is bubbled. This method is easy to understand but time consuming. In this type, two successive elements are compared and swapping is done. Thus, step-by-step entire array elements are checked. Given a list of 'n' elements the bubble sort requires up to n-1 passes to sort the data**.**



Algorithm:

```
Step 1: Start the process.
Step 2: Read n.
Step 3: Set i=0.
Step 4: If(i<n) then goto step 5 else goto step 7.
Step 5: Read a[i].
Step 6: Set i=i+1 and goto step 4.
Step 7: Set i=0.
Step 8: If(i<n-1) then goto step 9.
Step 9: Set j=i+1 then goto step 10 else go to step 14.
Step 10: If(j<n) then goto step 11else goto step 14.
Step 11: If(a[i]>a[j]) then goto step 12.
Step 12: Set temp=a[i].
    a[i]=a[j].      a[j]=temp.
Step 13: Set j=j+1 and goto step 10.
Step 14: Set i=i+1 and goto step 8.
Step 15: Set i=0.
Step 16: If(i<n) then goto step 17 else goto step 19.
Step 17: Write a[i].
Step 18: Set i=i+1 and goto step 16.
Step 19: Stop the process.
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int a[20], i, j, n, temp; clrscr(); printf("Enter
        array size: \n"); scanf("%d", &n);
        printf("Enter any %d elements: \n", n);
        for(i = 0;i < n;i++)
                scanf("%d", &a[i]);
        for(i = 1;i <= n-1;i++)
        { for(j = 0;j < n-i;j++)
                { if(a[j] > a[j+1])
                        { temp = a[j]; a[j] =
                                a[j+1]; a[j+1]
                                = temp;
                        }
                }
        }
        printf("After sorting, the element are
        \n"); for(i = 0;i < n;i++) printf("%4d",
        a[i]); getch();
}
```

Expected Output:

```
 Enter array size:        6
Enter any 6 elements: 78 95 62 30 12 39
After sorting, the element are 12 30 39 62 78 95
```

7. b). Selection Sort?

Aim:

To implement selection sort. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

In selection sort the list is divided into two sub-lists sorted and unsorted. These two lists are divided by imaginary wall. We find a smallest element from unsorted sub-list and swap it to the beginning. And the wall moves one element ahead, as the sorted list is increases and unsorted list is decreases.

Assume that we have a list on n elements. By applying selection sort, the first element is compared with all remaining (n-1) elements. The smallest element is placed at the first location. Again, the second element is compared with remaining (n-1) elements. At the time of comparison, the smaller element is swapped with larger element. Similarly, entire array is checked for smallest element and then swapping is done accordingly. Here we need n-1 passes or iterations to completely rearrange the data

Algorithm:

```
Step 1: Start
Step 2: Read size of array 'n'
Step 3: Read number of elements of an array 'a[i]'
Step 4: Call 'selection_sort()' function
Step 5: for(i=0;i<n;i++)
Step 6: Equate 'min' to 'i'
Step 7: for(j=i+1;j<n;j++)
Step 8: If 'a[j]' is greater than 'a[min]' then
Step 9: Equate 'min' to 'j'
Step 10: End for
Step 11: 'temp = a[i]'
Step 12: 'a[i]=a[min]'
Step 13: 'a[min]=temp'
Step 14: End for
Step 15: Print all the elements in the array 'after sorting'
Step 16: Exit
Step 17: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int a[20], i, j, n, temp, maxloc, k; clrscr();
        printf("Enter array size: \n");
        scanf("%d", &n); printf("Enter any
        %d elements: \n", n);
        for(i = 0;i < n;i++)
                scanf("%d", &a[i]);
        for(i = 0;i < n-1;i++)
        { maxloc = 0; for(j = 0;j <= (n-
                1-i);j++)
```

```
                    { if(a[j] > a[maxloc])
                            maxloc = j;
                    } temp = a[n-1-i];
                    a[n-1-i] =
                    a[maxloc];
                    a[maxloc] = temp;
            }
            printf("After sorting, the element are
            \n"); for(i = 0;i < n;i++) printf("%4d",
            a[i]);
            getch();
    }
```

Expected Output:

```
Enter array size:         6
Enter any 6 elements: 78 95 62 30 12 39
After sorting, the element are 12 30 39 62 78 95
```

PROGRAM 8. Programs on pointers: pointer to arrays, pointer to functions

8.a). Pointer to Array

Aim:

To print address of each individual element of an array. For input & output statements we include the header file <stdio.h>, for clear the screen include the <conio.h> **Description:**

Pointer is a user defined data type that creates special types of variables which can hold the address of primitive data type like char, int, float, double or user defined data type like function, pointer etc. or derived data type like array, structure, union, enum.

Examples:

int *ptr; **int** (*ptr)();

int (*ptr)[2];

Benefits of using pointers are:-

1) Pointers are more efficient in handling arrays and data tables.
2) Pointers can be used to return multiple values from a function via function arguments.
3) The use of pointer arrays to character strings results in saving of data storage space in memory.
4) Pointers allow C to support dynamic memory management.
5) Pointers provide an efficient tool for manipulating dynamic data structures such as structures , linked lists , queues , stacks and trees.
6) Pointers reduce length and complexity of programs.
7) They increase the execution speed and thus reduce the program execution time.

Algorithm:

```
Step 1: Start
Step 2: Declare an array
Step 3: enter the number of elements
Step 4: Print the address of each element in the array Step
5: Stop
```

Program :

```
#include<stdio.h>
int main()
{
        int
        x[4];
        int i;

        for( i = 0; i < 4; ++i )

        {     printf("&x[%d] = %u\n",I,

        &x[i]);

        }

        printf("Address of array x: %u", x);

          return 0;

    }
```

Expected Output:

```
    &x[0] = 1450734448
      &x[1] = 1450734452

     &x[2] = 1450734456

      &x[3] = 1450734460

   Address of array x: 1450734448
```

8. b). Pointer to Functions:

Aim:

To implement pointer to Functions. For input & output statements we include the header file <stdio.h>, for clear the screen include the <conio.h>

Description:

Pointers can be used to pass addresses of variables to called functions, thus allowing the called function to alter the values stored there.

Passing only the copy of values to the called function is known as **"call by value".** Instead of passing the values of the variables to the called function, we pass their addresses, so that the called function can change the values stored in the calling routine. This is known as **"call by reference",** since we are referencing the variables.

Here the addresses of actual arguments in the calling function are copied into formal arguments of the called function. Here The formal parameters should be declared as pointer variables to store the address.

Algorithm:

Step 1: Start

Step 2: Declare a function to swap two numbers

Step 3: Enter two integer numbers to be swapped

Step 4: return to main function Step

5: End

Program:

```
#include <stdio.h> void
swap(int *n1, int *n2); int
main()
{ int num1 = 5, num2 = 10;
        swap( &num1, &num2);
        printf("num1 = %d\n", num1);
        printf("num2 = %d", num2);
        return 0;
}
void swap(int* n1, int* n2)
{

    int temp;
    temp =*n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Expected Output:

```
num1 = 10
num2 = 5
```

PROGRAM 9. String Manipulations, Structures and Unions

9.a). Find the Length of a given String using Library Function?

Aim:
To find the length of a given string using library function. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

C Strings:- In C language a string is group of characters (or) array of characters, which is terminated by delimiter \0 (null). Thus, C uses variable-length delimited strings in programs.

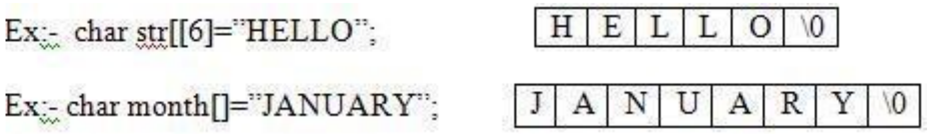
Declaring Strings:- Since strings are group of characters, generally we use the structure to store these characters is a character array.

Syntax:- char string_name[size];
The size determines the number of characters in the string name. Ex:-
char city[10];

Char name[30];

<u>Initializing strings</u>:-

There are several methods to initialize values for string variables.



Ex:- char str[[6]="HELLO";

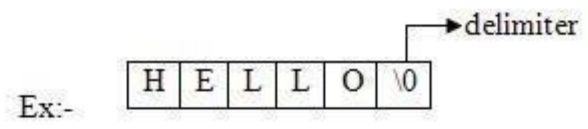
Ex:- char month[]="JANUARY";

Ex:- char city[8]=”NEWYORK”;

Char city[8]={‘N’,’E’,’W’,’Y’,’O’,’R’,’K’,’\0’};

The string city size is 8 but it contains 7 characters and one character space is for NULL terminator.

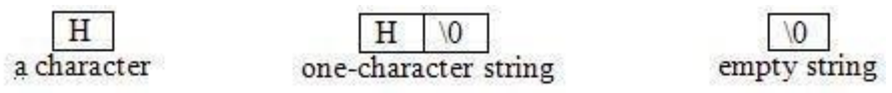
<u>Storing strings in memory</u>:-
In C a string is stored in an array of characters and terminated by \0 (null).



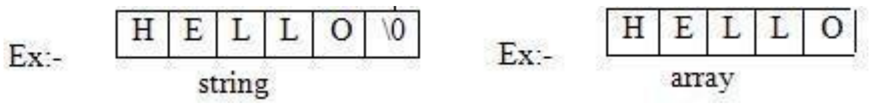
A string is stored in array, the name of the string is a pointer to the beginning of the string.

The character requires only one memory location.

If we use one-character string it requires two locations. The difference shown below,



The difference between array & string shown below,



Because strings are variable-length structure, we must provide enough room for maximum-length string to store and one byte for delimiter.

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ char arr[ ] = "Methodist College" ;
        int len1, len2 ; clrscr(); len1 =
        strlen (arr) ; len2 = strlen
        ("Engineerimg") ;
        printf ("String = %s length = %d \n", arr, len1) ; printf
        (“String = %s length = %d \n", "Engineerimg", len2) ;
        getch();
}
```

Expected Output:

```
string = Methodist College length = 17 string
= Engineerimg length = 11
```

9. b). Copy the given String to another String using Library Function?

Aim:
To copy the given string to another string using library function. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
```

```
{ char source[ ] = "Methodist College of Engineering & Technology" ; char
          target[20] ;
          cltscr();
          strcpy(target, source) ; printf("Source
          String = %s \n", source ) ; printf("Target
          String = %s \n", target ) ; getch();
}
```

Expected Output:

```
Source String = Methodist College of Engineering & Technology Target
String = Methodist College of Engineering & Technology
```

9. c). Appending one String at the end of another String using Library Function?

Aim:

To appends one string at the end of another string using library function. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ char source[ ] = "Methodist College";
          char target[30] = “Welcome to”;
          cltscr();
          strcat ( target, source) ; printf("Source
          String = %s \n", source); printf("Target
          String = %2s \n", target); getch();
}
```

Expected Output:

```
Source String = Methodist College
Target String = Welcome to Methodist College
```

9. d). Reverse the Contents of the given String using Library Function?

Aim:
To reverse the contents of the given string using library function. For input & output statements we include the header file <stdio.h>, for clear the screen include the <conio.h> and for string functions include <string.h>.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h> void
main()
{ char a[20]; cltscr(); printf("Enter
        any string: \n"); gets(a);
        strrev(a); printf("Reverse
        string: \n", a); getch();
}
```

Expected Output:

```
Enter any string: methodist
Reverse string: tsidohtem
```

9. e). Compare the Contents of given two Strings using Library Function?

Aim:
To compare the contents of the given two strings using library function. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ char string1[ ] = "Methodist";
        char string2[ ] = "MCET";
        int i, j, k;
        cltscr();
        i = strcmp(string1, "Methodist"); j
        = strcmp(string1, string2);
        k = strcmp(string1, "Methodist College");
        printf("i = %d \n", i);
        printf("j = %d \n", j);
        printf("k = %d \n", k);
        getch();
}
```

Expected Output:

```
 i = 0 j = 34 k
        = -32
```

9. f). Abbreviate the Contents of given Strings using Library Function?

Aim:

To abbreviate the contents of the given string using library function. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Algorithm:

```
Step 1: Start
Step 2: Declare a file pointer fp, ch
Step 2: Attach filename using fp=fopen(“filename”,”r+”)
Step 3: Read n (no.of characters from given file until end of the file)
Step 4: Print the characters
Step5: Close file pointer
Step 6: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ char s[100];
        int i;
        clrscr();
        printf("Enter a String:
        \n"); gets(s); i = 0;
        while(s[i] = = ' ')
        i++; putchar(s[i]);
        putchar(' ');
        while(s[i] != '\0')
        { if(s[i] = = ' ' && s[i+1] != ' ')
                { putchar(s[i+1]);
                        putchar(' ');
                }
                i++;
        } getch();
}
```

Expected Output:

```
Enter a String: Methodist College Engineering Technology
M C E T
```

9. g). Print employees name and gross salary using structures.

Aim:

To print employees name and gross salary using structures.

Description:

A structure is a collection of one or more variables of different data types, grouped together under a single name. By using structures variables, arrays, pointers etc can be grouped together.

Structures can be declared using two methods as follows:

(i) Tagged Structure:

The structure definition associated with the structure name is referred as tagged structure. It does not create an instance of a structure and does not allocate any memory.

(ii) Type-defined structures:-

The structure definition associated with the keyword typedef is called type-defined structure. General Syntax:

```
struct tag_name {
type member1;
type member2;
  /* declare as many members as desired, but the entire structure size must be known to the
compiler. */
};
```

Algorithm:

```
Step 1: Start
Step 2: Declare structure for employee details
Step 3: Read 100 employee details
Step 4: Repeat loop 100 employees and calculate gross salary of each employee
Step 5: Print each employee name and gross salary
Step 6: Stop
```

Program:

```
#include<stdio.h> struct
employee
{ char name[20];
        float basic;
        float da; float
        gross;
```

```
    }e[5];

    void main()
    {
        int i;
        for(i=0;i<5;i++)
          scanf("%s%f",e[i].name,&e[i].basic);
        for(i=0;i<5;i++)
        { e[i].da=52.0/100*e[i].basic;
            e[i].gross=e[i].da+e[i].basic
            ;
            printf("\n name=%s    gross=%f",e[i].name,e[i].gross); }
     }
```

Expected Output:

```
Enter name and basic
A 2300
B 32000
C 15000
D12000
F 22000

Name=A gross 3496.0000
Name =B gross = 48640.0000
Name = C gross = 22800.0000
Name = D gross = 18240.0000
Name = E gross = 334400.0000
```

9. h). Display your present address using union

Aim:
To display your present address using union

Description:
To define a union, you must use the union statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows

Syntax of Union union

```
    [union tag] {
    member definition;
    member definition;
      ...
      member definition;
} [one or more union variables];
```

Algorithm:

```
Step 1: Start
Step 2: Declare union for address details
Step 3: Assign address to union
Step 4: Print union details
 Step 5: Stop
```

Program:

```
#include<stdio.h>
#include<string.h>
union details
{ char name[20]; char
home_add[30]; char
hostel_add[30]; char
city[10]; char state[10];
int pincode; }a; void
main()
{ strcpy(a.name,"rama"); printf("\n
        %s",a.name); strcpy(a.home_add,"1-
        83/10,jublie hills");  printf("\n
        %s",a.home_add);
        strcpy(a.hostel_add,"mcet college");
        printf("\n %s",a.hostel_add);
        strcpy(a.city,"hyderabad"); printf("\n
        %s",a.city); strcpy(a.state,"telangana");
        printf("\n %s",a.state);
        a.pincode=500043;  printf("\n
        %d",a.pincode);  getch();
}
```

Expected Output:

```
rama
1-83/10, jublie hills
mcet college
Hyderabad
Telangana
500043
```

PROGRAM 10. File Handling Programs.

10. a). Program to create a file?

Aim:
To create a file. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

In order to perform the file operations in C we must use the high level I/O operation functions which are in C standard I/O library.

i) fopen():
The function **fopen** is one of the Standard Library functions and returns a file pointer which you use to refer to the file you have opened e.g.

fp = fopen(“prog.c”, “r”) ;

The above statement **opens** a file called prog.c for **reading** and associates the file pointer fp with the file.When we wish to access this file for I/O, we use the file pointer variable fp to refer to it. You can have up to about 20 files open in your program - you need one file pointer for each file you intend to use.

ii) fclose():
Closing a file ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.

- Another instance where we have to close a file is to reopen the same file in a different mode.
- The I/O library supports the following function to do this:

fclose (file_pointer);

- Where fp is the file pointer returned by the call to fopen ().
- fclose () returns 0 on success (or) -1 on error.

Once a file is closed, its file pointer can be reused for another file.

iii) getc():
getc() is used to read a character from a file that has been opened in a read mode. For example the statement c=getc(fp);
would read a character from the file whose file pointer is fp. The file pointer moves by one character for every operation of getc(). The getc() will return an end-of –marker EOF, when an end of file has been reached.

iv) putc(): putc(c,fp);

IT writes the character contained in the character variable c to the file associated with the FILE pointer fp. similarly like getc() put c() also will return an end-of –marker EOF, when an end of file has been reached**.**

v) fprintf() & fscanf():

In order to handle a group of mixed data simultaneously there are two functions that are fprintf() and fscanf().These two functions are identical to printf and scanf fuctions,except they work on files. The first argument of these functions is a file pointer which specifies the file to be used . the general form of fprintf is

fprintf(fp,"control string",list);

where fp is a file pointer associated with a file that has been opened for writing . the control string contains output specifications for the items in the list. .

fprintf(fp,"%s %f %d",name,6.6,age);

Like fprintf fscanf() also contains the same syntax which is used to read a no of values from a file.

fscantf(fp,"control string",list);

like scanf , fscanf also returns the number number of items that are successfully read.when the end of file is it returns the value EOF.

vi) getw() &putw():

The getw() and putw() are integer oriented functions .They are similar to the getc() and putc() functions and are used to read and write integer values . These functions would be useful when we deal with only integer data. The general form of

putw(integer,fp); getw(fp);

vii) ftell:-

ftell takes a file pointer and returns a number of type **long,** that corresponds to the current position. This function is useful in saving the current position of the file,which can be later used in the program.

Syntax: **N=ftell(fp);**

N would give the Relative offset (In bytes) of the current position. This means that already **n** bytes have a been read or written.

viii) rewind:-

It takes a file pointer and resets the position of to the start of the file.

Syntax: rewind(fp);
n=ftell(fp);
would assign 0 to **n** because the file position has been set to start of the file by rewind. The first byte in the file is numbered 0,second as 1, so on. This function helps in reading the file more than once, without having to close and open the file.

xi) fseek:-

fseek function is used to move the file pointer to a desired location in the file.

Syntax: fseek(file ptr,offset,position);

is a pointer to the file concerned, offset is a number or variable of type file pointers.**long**,and position is an integer number. The offset specifies the number of

positions(**Bytes**) to be moved le from the location specified by the position.

Algorithm:

```
Step 1: Start
Step 2: Declare a file pointer fp, ch
Step 2: Attach filename using fp=fopen("filename","r+")
Step 3: Read n (no.of characters from given file until end of the file)
Step 4: Print the characters
Step5: Close file pointer
Step 6: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{
        FILE *fp;
        char ch;
        clrscr();
        fp = fopen("text.dat", "w"); printf("Enter
        text, to stop press Ctrl – Z: \n");
        while((ch = getchar()) != EOF)
        { fputc(ch,fp);
        }
        fclose(fp);
        getch();
}
```

Expected Output:

```
Enter text, to stop press Ctrl-Z: This
is my first program in C language
to create a file.
Bye.
To see the contents of this file
open text.dat in any editor
program.
^Z
File Saved on disk text.dat:
This is my first program in C language
to create a file.
Bye.
To see the contents of this file
open text.dat in any editor
program.
```

10. b). Program to print the contents of the file on the monitor?

Aim:

To print the contents of the file on the monitor. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Algorithm:

```
Step 1: Start
Step 2: Declare a file pointer fp, ch
Step 3: Attach filename using fp=fopen("filename","r+")
Step 4: Read n (no.of characters from given file until end of the file)
Step 5: Print the characters
Step 6: Close file pointer
Step 7: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{
        FILE *fp;
        char ch;
        clrscr();
        fp = fopen("text.dat", "r");
        if(fp==NULL) printf("No such
        file \n"); else
        { printf("File contents are: \n");
                while(!feof(fp))
                {
                        ch = fgetc(fp);
                        putchar(ch);
                }
                fclose(fp);
        }
        getch();
}
```

Expected Output 1:

```
Given text.dat File contents are:
This is my first program in C to
create a file with some text.
Bye
To see the contents of the file open text.dat in
any editor.
```

Expected Output 2: Given

```
example.dat
No such file
```

10. c). Program to Copy one File in to another?

Aim:

To copy one file into another. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Algorithm:

```
Step 1: Start
Step 2: create two file pointers
Step 3: open first file in read mode
Step 4: open second file in write mode using another file pointer.
Step 5:  If either of the two file pointers are NULL, then print unable to open file
```

Step 6: read the character from first file and convert it to uppercase and write that character into second file

Step 7: repeat Step 6 till end of first file

Step 8: close both the files Step

9: Stop.

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{
          FILE *fs, *ft ; char ch;
          clrscr(); fs = fopen
          ("text.dat", "r"); if( fs ==
          NULL )
          { puts ("Cannot open source file");
          }
          ft = fopen ("example.dat", "w");
          if(ft == NULL)
          { puts("Cannot open target file");
                    fclose(fs);
          } while
          (1)
          { ch = fgetc (fs); if(ch
                    = = EOF)
                    break;
                    else
                    { fputc (ch, ft); puts("File
                              copied");
                    }
          } fclose
          (fs);
          fclose (ft);
          getch();
}
```

Expected Output:

```
File copied
```

10 d) Finding the No. of Characters, Words and Lines of given Text File?

Aim:

To find the no. of characters, words and lines of given text file. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

Naming and opening a file: A name is given to the file used to store data. The file name is a string of characters that make up a valid file name for operating system. It contains two parts. An optional name and an optional period with extension.

Examples: Student.dat, file1.txt, marks.doc, palin.c.

The general formal of declaring and opening a file is

```
FILE *fp;          //declaration
fp=fopen
("filename","mode");             //ststement to open file.
```

Here FILE is data structures defined for files.

fp is a pointer to data type file. It acts as link between systems and program. Filename is the name of the file.

Mode tells in which mode the file should be opened.

For reading data from file, Input functions used are (Input and output operations on files)

a) **getc();** It is used to read characters from file that has been opened for read operation.

   Syntax: C=gets (file pointer)

This statement reads a character from file pointed to by file pointer and assign to c. It returns an end-of-file marker EOF, when end of file as been reached

b) **fscanf();** This function is similar to that of scanf function except that it works on files.

   Syntax: fscanf (fp, "control string", list);

   Example : fscanf(f1,"%s%d",str,&num);
   The above statement reads string type data and integer types data from file.

c) **getw();** This function reads an integer from file.

Syntax: getw (file pointer);

d) **fgets():** This function reads a string from a file pointed by a file pointer. It also copies the string to a memory location referred by an array.
   Syntax: fgets(string,no of bytes,filepointer);
e) **fread():** This function is used for reading an entire structure block from a given file.

   Syntax: fread(&struct_name,sizeof(struct_name),1,filepointer);

<u>Writing data to a file:</u>

To write into a file, following C functions are used

a) **putc():** This function writer a character to a file that has been opened in write mode.

   Syntax: putc(variable,fp);

This statement writes the character contained in character variable c into a file whose pointer is fp.

b) **fprintf():** This function performs function, similar to that of printf.

   Syntax: fprintf(f1,"%s,%d",str,num);

c) **putw():** It writes an integer to a file.

   Syntax: putw (variable, fp);

d) **fputs():** This function writes a string into a file pointed by a file pointer.

   Syntax: fputs(string,filepointer);

e) **fwrite():** This function is used for writing an entire block structure to a given file. Syntax: fwrite(&struct_name,sizeof(struct_name),1,filepointer);

<u>Closing a file:</u>

A file is closed as soon as all operations on it have been completed. Library function for closing a file is fclose(file pointer);

Syntax: fclose(fp);

This statement closes a file pointed by file pointer fp. Or **fcloseall(),** to close all opened files at a time.

<u>Algorithm:</u>

Step 1: Start

```
Step 2: Initialize str[100], i=0, l=0 and f=1
Step 3:  Read string ‘str’
Step 4: for(i=0;str[i]!=’\0’;i++)
Step 5: Equate ‘l’ is equal to ‘l’ plus 1 until str[i]!=’\0’
Step 6: End for
Step 7: Print number of characters in the string
Step 8: for(i=0;i<=l-1;i++)
Step 9: if ‘str[i]’ is equal to space then increment ‘f’ by 1
Step 10: End for
Step 11: Print number of words in the string
Step 12: Exit
Step 13: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{ int noc = 0, now = 0, nol = 0;
        FILE *fw,*fr; char
        fname[20], ch; clrscr();
        printf("Enter the source file name: \n");
        gets(fname); fr =
        fopen(fname, "r");
        if(fr = = NULL)
        { printf("\n error \n");
        exit(0); } ch = fgetc(fr);
        while(ch != EOF)
        {
                noc++;
                if(ch = = ' ');
                        now++;
                if(ch = = '\n')
                { nol++;
                        now++;
                } ch =
                fgetc(fr);
        }
        fclose(fr);
        printf("Total no. of character = %d \n", noc);
        printf("Total no of words = %d \n", now);
        printf("Total no of lines = %d \n", nol);
        getch();
}
```

Expected Output:

```
Enter the source file name:     namecount.c
Total no. of character = 488
Total no of words = 522
Total no of lines = 34
```

ADDITIONAL PROGRAMS

1. Aim: To read arguments at the command line and display it

Algorithm:

1. Start
2. Pass a to arguments argc and argv in main function
3. Check the condition argc equal to 2 then printf the argv[1] value
4. Else if argc is greter than 2 the print Too many arguments supplied
5. Else One argument expected
6. Stop

Program :

```
#<include stdio.h> #include<string.h>
void main(int argc,char *argv[])
{
        int i;
        printf("\n total no of
        arguments=%d",argc);
        for(i=0;i<argc;i++) printf("\n
        args[%d]=%s",i,argv[i]);
}
```

Expected Output:

```
total no of arguments = 4
args[0]   =   c:\pathname
args[1]  =  ram  args[2]  =
ravi
args[3] = raj
```

2. Aim: To compute the volume for spheres of radius 5, 10 and 15 meters

Algorithm:

1. Start
2. Define PI constant value 3.14
3. Define macro function to pass a radius value as a argument
4. Call the macro function
5. Print the result
6. Stop

Program:

```
#include<stdio.h>
#include<math.h>
#define PI 3.142
#define volume(r) ((4/3.0)*PI*pow(r,3)) void
main()
{ int r; float v;
        scanf("%d",&r);
        v=volume(r);
        printf("\n volume of the sphere v=%.3f",v);
}
```

Expected Output:

```
Volume of sphere 5 = 314.15
Volume of sphere 10 =1256.6000
Volume of sphere 15 = 2827.35000
```

3. Aim: To convert a Roman numeral to its decimal equivalent

Algorithm:

1. Start
2. Read roman numbers
3. Repeat loop for all given roman letters and convert each letter into digit and add to variable
4. Print the result
5. Stop

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{ char a[10]; int
     total[10],sum=0,i,l;
     printf("\n enter a roman number: ");
     gets(a); l=strlen(a);  for(i=0;i<l;i++)
     { switch(a[i])
          { case 'M': total[i]=1000; break;
               case 'D': total[i]=500; break;
               case 'C': total[i]=100; break;
               case 'L': total[i]=50; break;
               case 'X': total[i]=10; break;
               case 'I': total[i]=1;  break;
          }
          sum=sum+total[i];
     }
     for(i=0;i<l-1;i++) if(total[i]<total[i+1])
  sum=sum-2*total[i];      printf("\n the decimal
  equivalent is %d",sum);
  }
```

Expected Output:

```
enter a roman number: XVII
the decimal equivalent is 17
```

VIVA VOCE QUESTIONS

1. INTRODUCTION TO C

1) What is an algorithm?
2) What is a flowchart?
3) What are the characteristics of an algorithm?
4) What is a procedural oriented language?
5) How can u say C is a middle-level language?
6) What are the steps required to develop a C program?
7) What are the characteristics of C language?
8) What is difference between compiler and interpreter?
9) What is preprocessor directive?
10) When preprocessing is done?
11) Name some of the activities done at the time of preprocessing?

2. DATA TYPES AND OPERATORS

1) What is a C token?
2) What is a key word? List different keywords in C language?
3) What is a data type? Give different data types in C.
4) What is the use of typedef?
5) How to declare a constant in C?
6) What is the difference between a constant and a variable?
7) Give some examples of header files in C?
8) Why we include header files in a C program?
9) What are the different types of binary operators in C?
10) List different arithmetic operators and what is their precedence?
11) List different relational operators in C.
12) List different logical operators in C and give their precedence.
13) What is conditional operator? Write the syntax.

14) List different unary operators in C.
15) What is the associativity of assignment operator?
16) What is the use of sizeof operator?
20) What are shorthand assignment operators?
21) What is type casting?
22) List different bit-wise operators. What is the use of bit-wise operators?

3. CONTROL STRUCTURES

1) What are the different branch control statements in C language?
2) Is C a block structured language?
3) What is the meaning of block structured language?
4) What are the different loop control statements in C language?
5) What is the difference between while loop and do-while loop?
6) Draw the flowchart of while, do-while and for loop? 7) How many iterations occurs in the following while loop:

    while(1)   { … }

8) How to swap two variables?
9) How to swap two variables without using a temporary variable?
10) What is a compound statement? Give an example.
11) Write the syntax of for loop.
12) How a for loop executes?
13) How many iterations occurs in the following for loop:

    for(; ;)   { … }

14) How a nested for loop executes?
15) Differentiate between a for loop and a while loop?
16) What is the purpose of break statement in a loop?
17) What is the purpose of continue statement in a loop?
18) What does exit() do?

19) Write the syntax of switch statement.

20) Draw the flowchart of switch statement.

21) In switch control structure when default block is executed?

22) In switch control structure if there is no break statement in particular case block then what happen?

23) Why the usage of goto control structure is not recommended in C?

24) Write the recurrence relation to find nth Fibonacci number.

25) Write the recurrence relation to find the factorial of a given number.

4. SORTING TECHNIQUES

1) How many number of comparisons required to sort n numbers using bubble sort?

2) When quick sort gives worst performance?

3) What are the properties of heap?

4) What is the time complexity of quick sort?

5) What is the worst case time complexity of quick sort?

6) What is the time complexity of bubble sort?

7) What is the time complexity of insertion sort?

5. SEARCHING TECHNIQUES

1) What are the different searching mechanisms?

2) If the file contains large number of records in sorted order based on a key, then which searching technique is used to find a particular record efficiently?

3) Explain linear searching method.

4) Explain binary searching method.

5) What is the time complexity of linear search?

6) What is the time complexity of binary search?

OPEN ENDED PROGRAM

1. Write a C program for implementing Merge Sort
2. Write a C program for implementing Insertion sort

Estd: 2008

METHODIST

COLLEGE OF ENGINEERING & TECHNOLOGY

Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad

Abids, Hyderabad, Telangana, 500001