Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

(Affiliated to Osmania University & Approved by AICTE, New Delhi)

LABORATORY MANUAL

PROGRAMMING FOR PROBLEM SOLVING LABORATORY

BE I &II Semester (AICTE Model Curriculum): 2020-21

NAME: ________________________________

ROLL NO:________________________________

BRANCH:________________ **SEM:**________________________

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERNG

Empower youth- Architects of Future World

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.

METHODIST

Estd:2008 COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL

PROGRAMMING FOR PROBLEM SOLVING LABORATORY

Prepared
By
Dr. M. Sharada Varalakshmi

Professor, Dept. of CSE,

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

Estd:2008

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION & MISSION

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

- To offer flexible programs of study with collaborations to suit industry needs.
- To provide quality education and training through novel pedagogical practices.
- To expedite high performance of excellence in teaching, research and innovations.
- To impart moral, ethical values and education with social responsibility.

METHODIST

Estd:2008 COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES

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

PEO1: Apply technical concepts, Analyze, Synthesize data to Design and create novel products and solutions for the real life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.

METHODIST

Estd:2008 COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

Engineering graduates will be able to:

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

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

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

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

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

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

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

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

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

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

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

PROGRAM SPECIFIC OUTCOMES

At the end of 4 years, Computer Science and Engineering graduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various domains like networking and data mining to manage projects in multidisciplinary environments.

PSO2: Develop software applications with open-ended programming environments**.**

PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

Course Code	Course Title						Core /Elective
ES 155 CS	**Programming For Problem Solving Lab**						**Core**
Prerequisite	Contact Hours per Week				CIE	SEE	Credits
	L	T	D	P			
-	**-**	**-**	**-**	**2**	**25**	**50**	**1**

Course Objectives

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

Course Outcomes

After completing this course, the student will be able to

1. Choose appropriate data type for implementing programs in C language
2. Design and implement modular programs involving input output operations, decision making and looping constructs.
3. Implement search and sort operations on arrays.
4. Apply the concept of pointers for implementing programs on dynamic memory management and string handling.
5. Design and implement programs to store data in structures and files.

List of Experiments to be performed:

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. Conversion of binary to decimal, octal, hexadecimal and vice versa.
4. Generating Pascal triangle, pyramid of numbers.
5. Recursion: factorial, Fibonacci, GCD.
6. Matrix addition and multiplication using arrays, linear search and binary search using recursive and non-recursive procedures.
7. Bubble sort and selection sort.
8. Programs on pointers: pointer to arrays, pointer to functions.
9. Functions for string manipulations.
10. Programs on structures and unions.
11. Finding the number of characters, words and lines of given text file.
12. File handling programs

Suggested Readings:

1. Byron Gottfried, –*Theory and practice of Programming with C”*, Schaum‘s Outline McGraw-Hill, 1996
2. A.K. Sharma, –*Computer Fundamentals and Programming in C,* Universities Press, 2nd Edition, 2018
3. E. Balaguruswamy, –*Programming in ANSI C*‖, Tata McGraw-Hill Education, 2008
4. Brian W. Kernighan and Dennis M. Ritchie,‖*The C Programming Language*‖, Prentice Hall of India,1988.

Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Outcomes (CO's):

SUBJECT NAME : PROGRAMMING FOR PROBLEM SOLVING LAB

CODE : ES 155 CS

SEMESTER : I & II

CO No.	**Course Outcome**	**Taxonomy Level**
ES 155 CS.1	Develop programs using using data types appropriate in C Language	Create
ES 155 CS.2	Apply the concept of input, output operations, loops and constructs	Apply
ES 155 CS.3	Develop programs using arrays	Create
ES 155 CS.4	Implement search and sort algorithms in C	Knowledge
ES 155 CS.5	Apply concept of pointers on dynamic memory management and string handling	Apply
ES 155 CS.6	Develop programs on stored data and file structures	Create

METHODIST

Estd:2008

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the program / experiment details.
3. Student should enter into the laboratory with:
   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
   b. Laboratory Record updated up to the last session experiments.
   c. Formal dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out. If anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CODE OF CONDUCT FOR THE LABORATORY

- All students must observe the dress code while in the laboratory
- Footwear is NOT allowed
- Foods, drinks and smoking are NOT allowed
- All bags must be left at the indicated place
- The lab timetable must be strictly followed
- Be PUNCTUAL for your laboratory session
- All programs must be completed within the given time
- Noise must be kept to a minimum
- Workspace must be kept clean and tidy at all time
- All students are liable for any damage to system due to their own negligence
- Students are strictly PROHIBITED from taking out any items from the laboratory
- Report immediately to the lab programmer if any damages to equipment

BEFORE LEAVING LAB:

- Arrange all the equipment and chairs properly.
- Turn off / shut down the systems before leaving.
- Please check the laboratory notice board regularly for updates.

Lab In – charge

Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF EXPERIMENTS

SI. No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1.	a) Finding maximum and minimum of given set of numbers b) Finding roots of quadratic equation.			1 3	
2.	Sin x and Cos x values using series expansion.			5 6	
3.	Conversion of binary to decimal, octal, hexadecimal and vice versa.			8	
4.	Generating Pascal triangle, pyramid of numbers.			15	
5.	Recursion: Factorial, Fibonacci, GCD.			18	
6.	Matrix addition and multiplication using arrays. Linear search and binary search using recursive and non-recursive procedures.			22 27	
7.	Bubble sort and selection sort.			35	
8.	Programs on pointers: pointer to arrays, pointer to functions.			39	
9.	Functions for string manipulations.			42	
10.	Programs on structures and unions.			47	
11.	Finding the No. of Characters, Words and Lines of given Text File			50	
12.	File handling programs			53	

ADDITIONAL EXPERIMENTS

SI. No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1	Write a C program to read arguments at the command line and display it.			59	
2.	Write a C program to compute the volume for spheres of radius 5, 10 and 15 meters.			60	
3	Write a C program to convert a Roman numeral to its decimal equivalent.			61	

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. Conversion of Binary to Decimal, Octal, Hexa and Vice – Versa.

3. a). Conversion of Binary to Decimal?

Aim:
To convert binary to decimal. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

While:

The simplest of all the looping structures in c is the while statement. The basic format of the while statement is:

Syntax:
```
While(condition)
{
Statements;
}
```

The while is an entry –controlled loop statement The condition is evaluated and if the condition is true then the statements will be executed.After execution of the statements the condition will be evaluated and if it is true the statements will be executed once again. This process is repeated until the condition becomes false and the control is transferred out of the loop .On exit the program continues with the statement immediately after the body of the loop.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        long int binary, decimal = 0, j = 1, rem;
        clrscr();
        printf("Enter any number any binary number: \n");
        scanf("%ld", &binary);
        while(binary != 0)
        {
                rem = binary % 10;
                decimal = decimal+rem*j;
                j = j*2;
                binary = binary / 10;
        }
printf("Equivalent decimal value: %ld", decimal);
getch();
}
```

Expected Output:

```
Enter any number any binary number:1101
Equivalent decimal value: 13
```

3. b). Conversion of Decimal to Binary?

Aim:
To convert decimal to binary. 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 decimalnum, rem, temp, binarynum
Step 3: Read decimalnum
Step 4: Set binarynum to 0 and temp to 1
Step 5: rem equals decimalnum mod 2
Step 6: decimalnum equals decimalnum by 2
Step 7: binarynumequlasbinarynum plus rem times temp
Step 8: temp equals temp times 10
Step 9: Repeat step 5 to step 8 until decimalnum!=0
Step 10: End while
Step 11: print binary equivalent of decimal number
Step 12: Exit
Step 13: Stop
```

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        long int decimal, rem, q;
        int binary[100], i = 1, j;
        clrscr();
        printf("Enter any decimal number: \n");
        scanf("%ld", &decimal);
        q = decimal;
        while(q != 0)
        {
                binary[i++] = q % 2;
                q = q / 2;
        }
        printf("Equivalent binary value of decimal number %d: ", decimal);
        for(j = i -1 ;j > 0;j--)
                printf("%d", binary[j]);
getch();
}
```

Expected Output:

```
Enter any decimal number:50
Equivalent binary value of decimal number 50: 110010
```

3. c). Conversion of Binary to Octal?

Aim:
To convert binary to octal. 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()
{
        long int binary, octal = 0, j = 1, rem;
        clrscr();
        printf("Enter any number any binary number: \n");
        scanf("%ld", &binary);
        while(binary != 0)
        {
                rem = binary % 10;
                octal = octal+rem*j;
                j = j*2;
                binary = binary / 10;
        }
printf("Equivalent octal value: %lo", octal);
getch();
}
```

Expected Output:

```
Enter any number any binary number:1101
Equivalent octal value: 15
```

3. d). Conversion of Octal to Binary?

Aim:
To convert octal to binary. 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>
#define MAX 1000
void main()
{
        char octal[MAX];
        long int i = 0;
        clrscr();
        printf("Enter any octal number: \n");
        scanf("%s", octal);
        printf("Equivalent binary value: \n");
        while(octal[i])
        {
```

```
                        switch(octal[i])
                        {
                                case '0': printf("000");
                                        break;
                                case '1': printf("001");
                                        break;
                                case '2': printf("010");
                                        break;
                                case '3': printf("011");
                                        break;
                                case '4': printf("100");
                                        break;
                                case '5': printf("101");
                                        break;
                                case '6': printf("110");
                                        break;
                                case '7': printf("111");
                                        break;
                                default: printf("Invalid octal digit %c ", octal[i]);
                        }
                        i++;
                }
        getch();
        }
```

Expected Output:

```
        Enter any octal number:123
        Equivalent binary value:001010011
```

3. e). Conversion of Binary to Hexadecimal?

Aim:
To convert binary to hexadecimal. 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()
        {
                long int binary, hexadecimal = 0, j = 1, rem;
                clrscr();
                printf("Enter any number any binary number: \n");
                scanf("%ld", &binary);
                while(binary != 0)
                {
                        rem = binary % 10;
                        hexadecimal = hexadecimal+rem*j;
                        j = j*2;
                        binary = binary / 10;
```

```
            }
            printf("Equivalent hexadecimal value: %lX", hexadecimal);
    getch();
    }
```

Expected Output:

```
    Enter any number any binary number:1101
    Equivalent hexadecimal value: D
```

3. f). Conversion of Hexadecimal to Binary?

Aim:
To convert hexadecimal to binary. For input & output statements we include the header file <stdio.h> and for clear the screen include the <conio.h>.

Description:

switch statement:

switch statement is one of decision-making control-flow statements. Just like else if ladder, it is also used to make a decision among multiple choices. switch statement has the following form:

```
switch(<exp>)
{
case <exp-val-1>: statements block-1

break;

case <exp-val-2>: statements block-2

break;

case <exp-val-3>: statements block-3

break;
:
:
case <exp-val-N>: statements block-N

break;

default: default statements block
}
-------------------
Next-statement;
```

In this syntax,

- switch, case, default and break are keywords.

- *<exp>* is any expression that should give an integer value or character value. In other words, it should never return any floating-point value. It should always be enclosed with in parentheses (and). It should also be placed after the keyword switch.
- *<exp-val-1>, <exp-val-2>, <exp-val-3>.... <exp-val-N>* should always be integer constants or character constants or constant expressions. In other words, variables can never be used as *<exp-val>*. There should be a space between the keyword case and *<exp-val>*. The keyword case along with its *<exp-val>* is called as a case label. *<exp-val>* should always be unique; no duplications are allowed.
- *Statements block-1, statements-block-2, statements block-3… statements block-N* and *default statements block* are simple statements, compound statements or null statements. It is important to note that the statements blocks along with their own case labels should be separated with a colon (:).
- The break statement at the end of each statements block is an optional one. It is recommended that break statement always be placed at the end of each statements block. With its absence, all the statements blocks below the matched case label along with statements block of matched case get executed. Usually, the result is unwanted.
- The statement block and break statement can be enclosed with in a pair of curly braces { and }.
- The default along with its statements block is an optional one. The break statement can be placed at the end of default statements block. The default statements block can be placed at anywhere in the switch statement. If they placed at any other places other than at end, it is compulsory to include a break statement at the end of default statements block.
- *Next-statement* is a valid C statement.

Program:

```
#include<stdio.h>
#include<conio.h>
#define MAX 1000
void main()
{
        char binary[MAX], hexadecimal[MAX];
        long int i = 0;
        clrscr();
        printf("Enter any hexadecimal number: \n");
        scanf("%s", hexadecimal);
        printf("Equivalent binary value: \n");
        while(hexadecimal[i])
        {
                switch(hexadecimal[i])
                {
                        case '0': printf("0000");
                                break;
                        case '1': printf("0001");
                                break;
                        case '2': printf("0010");
                                break;
                        case '3': printf("0011");
                                break;
                        case '4': printf("0100");
                                break;
```

```
                                        case '5': printf("0101");
                                                  break;
                                        case '6': printf("0110");
                                                  break;
                                        case '7': printf("0111");
                                                  break;
                                        case '8': printf("1000");
                                                  break;
                                        case '9': printf("1001");
                                                  break;
                                        case 'A': printf("1010");
                                                  break;
                                        case 'B': printf("1011");
                                                  break;
                                        case 'C': printf("1100");
                                                  break;
                                        case 'D': printf("1101");
                                                  break;
                                        case 'E': printf("1110");
                                                  break;
                                        case 'F': printf("1111");
                                                  break;
                                        case 'a': printf("1010");
                                                  break;
                                        case 'b': printf("1011");
                                                  break;
                                        case 'c': printf("1100");
                                                  break;
                                        case 'd': printf("1101");
                                                  break;
                                        case 'e': printf("1110");
                                                  break;
                                        case 'f': printf("1111");
                                                  break;
                                        default: printf("Invalid hexadecimal digit %c ", hexadecimal[i]);
                              }
                    i++;
                    }
          getch();
          }
```

Expected Output:

```
Enter any hexadecimal number:2AD5
Equivalent binary value:0010101011010101
```

PROGRAM 4. Generating a Pascal Triangle and Pyramid of Numbers.

4. 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
```

4. 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 5. Recursion: Factorial, Fibonacci, GCD.

5. a). Recursion: 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:

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main()
{
        int num,f;
        clrscr();
        printf("Enter a number: \n");
        scanf("%d", &num);
        f = fact(num);
        printf("Factorial of %d is: %d", num, f);
        getch();
}
int fact(int n)
```

```
{
        if(n = = 1)
                return 1;
        else
                return(n*fact(n-1));
}
```

Expected Output:

```
Enter a number:5
Factorial of 5 is: 120
```

5. b). Recursion: 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
```

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-1);
        }
}
```

Expected Output:

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

5. c). Recursion: 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.

Algorithm :

```
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>
int findgcd(int, int)
void main()
{
        int n1, n2, gcd;
        clrscr();
        printf("Enter two numbers: \n");
        scanf("%d %d", &n1, &n2);
        gcd = findgcd(n1, n2);
        printf("GCD of %d and %d is: %d \n", n1, n2, gcd);
        getch();
}
int findgcd(int x, int y)
{
        while(x != y)
        {
                if(x > y)
                        return findgcd(x-y, y);
                else
                        return findgcd(x, y-x);
        }
        return x;
}
```

Expected Output:

```
Enter two numbers: 366
                   60
GCD of 366 and 60 is: 6
```

PROGRAM 6. Matrix Addition and Multiplication using Arrays?

6. 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
```

6. 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. c). 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. d). 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. e). 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.

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: 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. f). 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
```

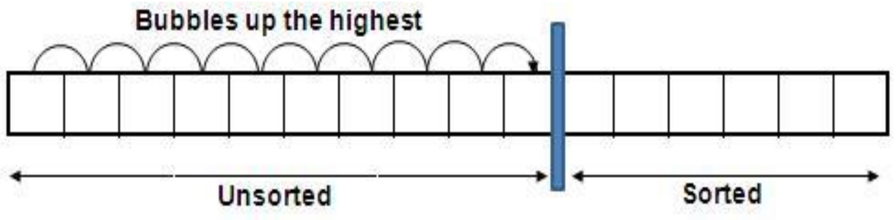
PROGRAM 7. Bubble Sort, Selection Sort?

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.

<u>Algorithm:</u>

```
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("Addressofarrayx:%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. Functions (Library Functions) of String Manipulations?

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.

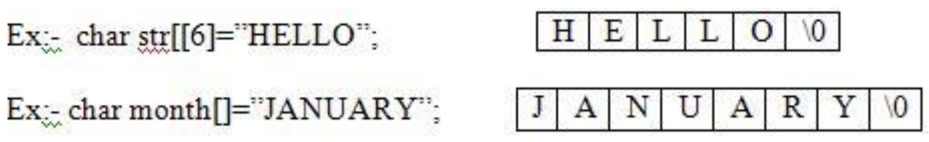
Declaring Strings:- Since strings are group of characters, generally we use the structure to store these characters is a character array.

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";

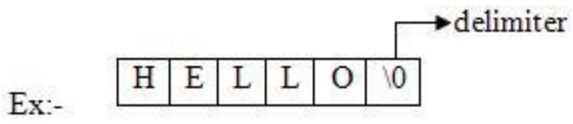
Ex:- char city[8]="NEWYORK";

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>:-

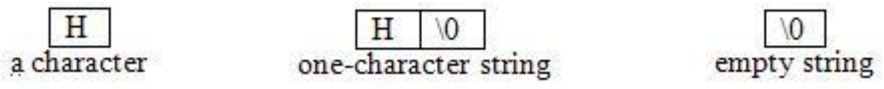
In C a string is stored in an array of characters and terminated by \0 (null).



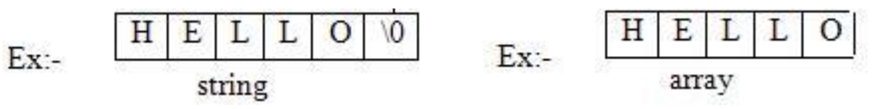
A string is stored in array, the name of the string is a pointer to the beginning of the string.

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
```

PROGRAM 10. Programs on structures and unions.

10. a). 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
```

10. b). 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 11. 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.

<u>For reading data from file</u>, 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);

Writing data to a file:

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);

Closing a file:

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.

Algorithm:

```
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
```

PROGRAM 12. File Handling Programs.

12. 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.
```

12. 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:

```c
#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
```

12. 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
```

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