SRM VALLIAMMAI ENGINEERINGCOLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203.

DEPARTMENT OF INFORMATION TECHNOLOGY

IT3564- C PROGRAMMING AND DATA STRUCTUIRES LABORATORY

REGULATIONS – 2023

LAB MANUAL

Prepared by

Dr. K. Revathi, Associate Professor/IT
Mr. K. Sivakumar,Assistant Professor /IT

IT3564 C PROGRAMMING AND DATA STRUCTURES LABORATORY

L	T	P	C
0	0	3	1.5

OBJECTIVES:

- To develop applications in C.
- To implement linear and non-linear data structures
- To understand the different operations of search trees.
- To get familiarized to sorting and searching algorithms.
- To understand the different hashing techniques.

LIST OF EXPERIMENTS

1. Practice of C programming using statements, expressions, decision making and iterative Statements.
2. Practice of C programming using Functions and Arrays
3. Implement C programs using Pointers and Structures
4. Implement C programs using Files
5. Development of real time C applications
6. Array implementation of List ADT
7. Array implementation of Stack and Queue ADTs
8. Linked list implementation of List, Stack and Queue ADTs
9. Applications of List, Stack and Queue ADTs
10. Implementation of Binary Trees and operations of Binary Trees
11. Implementation of Binary Search Trees
12. Implementation of searching techniques
13. Implementation of Sorting algorithms: Insertion Sort, Quick Sort, Merge Sort
14. Implementation of Hashing – any two collision techniques

TOTAL: 45 PERIODS

Requirements for a batch of 30 students

Software

- Compiler: A compiler like Dev C++ or GCC
- Operating system: A Windows or UNIX operating system, or Linux with gcc/g++
- Text editor: A text editor like gedit

Hardware

- A computer that can run C/C++ programs

OUTCOMES:

At the end of the course, the students will be able to:

- Use different constructs of C and develop applications
- Write functions to implement linear and non-linear data structure operations
- Suggest and use the appropriate linear / non-linear data structure operations for a given problem
- Apply appropriate hash functions that result in a collision free scenario for data storage and Retrieval
- Implement Sorting and searching algorithms for a given application

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

1. **PEO1:** To afford the necessary background in the field of Information Technology to deal with Engineering problems to excel as engineering professionals in industries.
2. **PEO2:** To improve the qualities like creativity, leadership, team work and skill thus contributing towards the growth and development of society.
3. **PEO3:** To develop ability among students towards innovation and entrepreneurship that caters to the need of Industry and society.
4. **PEO4:** To inculcate an attitude for life- long learning process through the use of information technology sources.
5. **PEO5:** To prepare them to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAM OUTCOMES (POs) ENGINEERING GRADUATES WILL BE ABLE TO:

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

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‘s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OBJECTIVES (PSOs)

PSO1: Design secured database applications involving planning, development and maintenance using state of the art methodologies based on ethical values.

PSO2: Design and develop solutions for modern business environments coherent with the advanced technologies and tools.

PSO3: Design, plan and setting up the network that is helpful for contemporary business environments using latest hardware components.

PSO4: Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.

CO – PO – PSO Mapping

CO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**
CO1	2	3	1	2	2	1	1	-	1	2	1	3	2	1	3
CO2	1	2	1	2	2	-	-	-	1	1	1	2	2	2	2
CO3	2	3	1	2	3	-	-	-	1	1	1	2	2	1	2
CO4	2	1	-	1	1	-	-	-	2	1	1	2	2	3	1
CO5	1	2	1	2	2	1	1	-	1	2	1	3	2	2	3
Avg	2	2	1	2	2	1	1	-	1	1	1	2	2	2	2

EX.NO: 1(A)	PRACTICE OF C PROGRAMMING USING STATEMENTS

AIM:

To check whether the given integer is PALINDROME or NOT.

ALGORITHM :

Steps:

1. [Initialize] Start

2. [Input the original number] read num

3. [Set number num to a variable n] n ← num

4. [Iterate until num is not equal to 0. If num value becomes 0, control comes out of the loop. So num's original value is lost. So, num value is stored in other variable n in step 3. In step 4, reverse of the number is calculated.] while (num != 0) do remainder ← num mod 10 num ← num/10 rev ← rev * 10 +remainder 5. [Print reverse number] print rev

6. [Check if original number & reverse number are same. If it is, number is palindrome. Otherwise, not palindrome] if (rev = n) then print palindrome else print not a palindrome endif

7. [Finished]

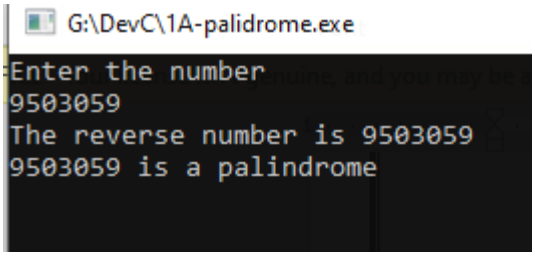
End

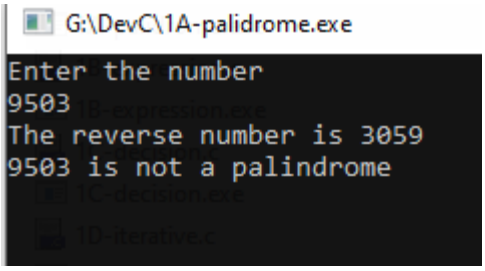
PROGRAM :

```
/* Program to calculate whether a given number is palindrome or not */
#include<stdio.h>
#include<conio.h>
int main()
{
int temp,rev=0,num,remainder ;
clrscr();
printf("Enter the number\n");
scanf("%d",&num);
temp=num;
while(num!=0) //Reversing the number
```

```
{
remainder = num%10;
 num = num/10;
rev = rev*10+ remainder;
}
printf("The reverse number is %d",rev);
if(rev == temp)
printf("\n%d is a palindrome",temp);
else
printf("\n%d is not a palindrome", temp); getch(); }
```

OUTPUT:





RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 1(B)	PRACTICE OF C PROGRAMMING USING EXPRESSIONS

Program:

Write a program to take input of name, rollno and marks obtained by a student in 4 subjects of 100 marks each and display the name, rollno with percentage score secured.

AIM:

To Write a program to take input of name, rollno and marks obtained by a student in 4 subjects of 100 marks each and display the name, rollno with percentage score secured.

Algorithm:

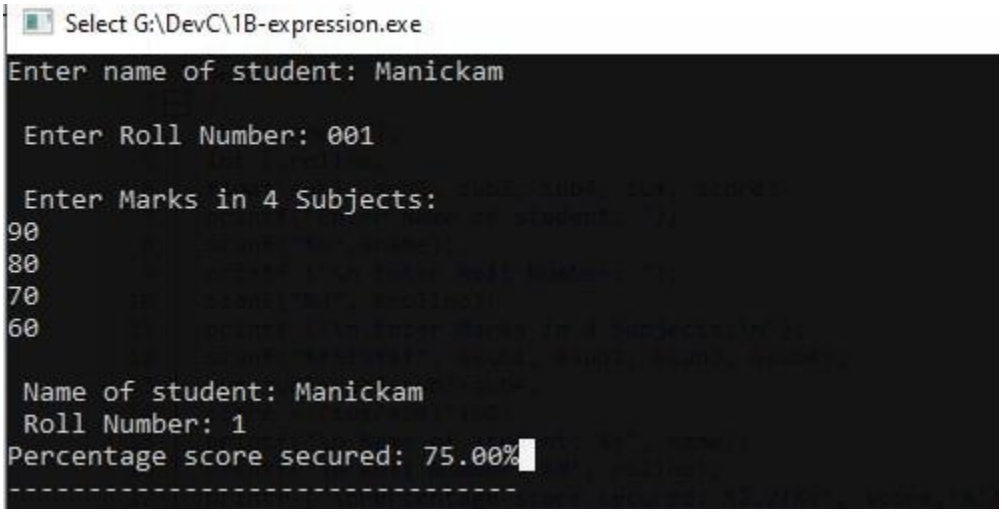
1. Start
2. Define variables: name, rollno, sub1, sub2, sub3, sub4, sum, score
3. Take input from keyboard for all the input variables
4. Calculate the sum of marks of 4 subjects and also calculate the percentage score as: sum = sub1 + sub2 + sub3 + sub4; score = (sum/400) * 100
5. Display the name, roll number and percentage score.
6. Stop

PROGRAM :

```
#include<stdio.h>
#include<conio.h>
void main()
 {
char name[20];
int rollno;
float sub1, sub2, sub3, sub4, , sum, score;
printf("Enter name of student: ");
scanf("%s",&name[]);
printf ("\n Enter Roll Number: ");
scanf("%d", &rollno);
printf ("\n Enter Marks in 4 Subjects:\n");
scanf("%f%f%f%f", &sub1, &sub2, &sub3, &sub4);
```

```
sum=sub1+sub2+sub3+sub4;
score = (sum/500)*100;
printf("\n Name of student: %s", name[]);
printf("\n Roll Number: %d", rollno);
printf ("\nPercentage score secured: %2.2f%c", score,'%');
getch();
}
```

OUTPUT:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 1(C)	Practice of C programming using decision making

AIM:

To find the roots of the quadratic equation (ax2+bx+c=0) with different possible input values for a, b and c. (DECISION MAKING)

ALGORITHM :

Quadratic Equation [This algorithm takes three coefficients as input and computes the roots]

Steps:

1. [Initialize] Start
2. [Input coefficients of quadratic equation] read a ,b, c
3. [Check for valid coefficients] If a =0 and b= 0 then print "Roots cannot be determined"
4. [Check for linear equation] else a=0then root1 ← (-c/b) print "Linear equation",root1 goto step
5. [Compute discriminate value] disc ← b*b-4*a*c
6. [Based on discriminate value, classify and calculate all possible roots and print them]

5.1 [If discriminate value is 0, roots are real & equal.]

if disc=0 then

root1← root2 ← (-b/2*a)

` print "Real & equal roots", root1, root2

5.2 [If discriminate value is >0, roots are real & distinct.]

else if disc>0then

root1← (-b+√disc)/(2*a) root2 ← (-b-√disc)/(2*a)

print "Real & distinct roots", root1, root2

5.3 [If discriminate value is <0 ,roots are imaginary.]

else

real ← -b/(2*a)

imag ← √(fabs(disc))/(2*a)

root1 ← (real) + i (imag)

root2 ← (real) - i (imag)

print "Imaginary roots", root1, root2

endif

endif

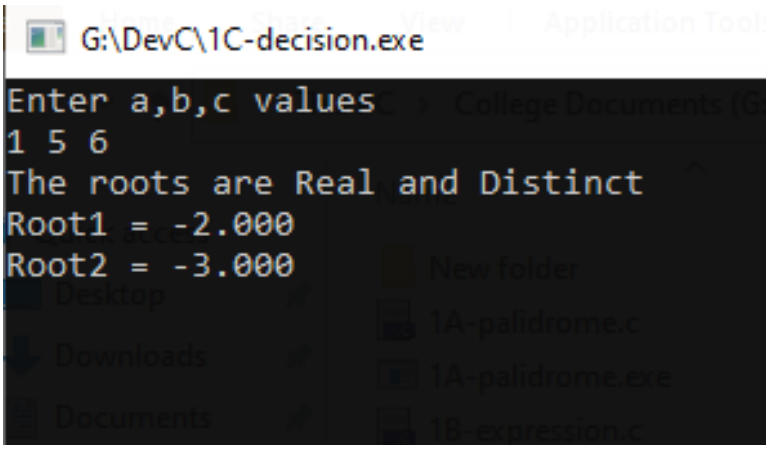
6. [Finished] End

PROGRAM :

```
/* Program to calculate all possible roots of a quadratic equation */
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
float a, b, c, disc;
float root1,root2,real,imag;
clrscr();
printf("Enter a,b,c values\n");
scanf("%f%f%f",&a,&b,&c);
if( (a == 0) && (b == 0) &&(c==0))
{
printf("Invalid coefficients\n");
printf(" Try Again with valid inputs !!!!\n");
getch();
}
disc = b*b - 4*a*c;
if(disc == 0)
{
printf("The roots are real and equal\n");
root1 = root2 = -b/(2*a);
printf("Root1 = %.3f \nRoot2 = %.3f", root1,root2);
}
else if(disc>0)
{
printf("The roots are Real and Distinct\n");
root1 = (-b+sqrt(disc)) / (2*a);
 root2 = (-b-sqrt(disc)) / (2*a);
printf("Root1 = %.3f \nRoot2 = %.3f",root1,root2);
}
else
{
printf("The roots are Real and Imaginary\n");
real = -b / (2*a);
 imag = sqrt(fabs(disc)) / (2*a);//fabs() returns only numberignoring sign
 printf("Root1 = %.3f + i %.3f \n",real,imag);
 printf("Root2 = %.3f - i %.3f",real,imag);
```

```
}
getch();
}
```

OUTPUT:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 1(D)	Practice of C programming using Iterative statements

There are three types of looping statements:

- For Loop(Evaluation of Polynomial)
- While Loop
- Do-while loop

AIM:

To find the value of the polynomial Design and develop an algorithm for evaluating the polynomial f(x) = a4 x4 + a3 x3 + a2 x2+ a1 x + a0, for a given value of x and its coefficients using Horner's method.

ALGORITHM :

ALGM: EVAL_POLYNOMIAL f(x) = a4x4+ a3x3+ a2x2+ a1x+ a0using Horner's method

Steps:

1. [Initialize]

Start

2. [Input the number of coefficients n] read n
3. Set sum to 0
4. [Read all coefficients a1, a2, a3, a4and constanta0]

```
For each value i in array a(i)do
        read n+1 coefficients
 endfor
```

5.[Input variable x]

```
        read x
```

6. [Iterate from n to 0. Calculate each term a4x4, a3x ,a2x2,a1x, a0 .]

```
For each value iin array a(i) do
        sum ← sum * x + a[i]
        endfor
```

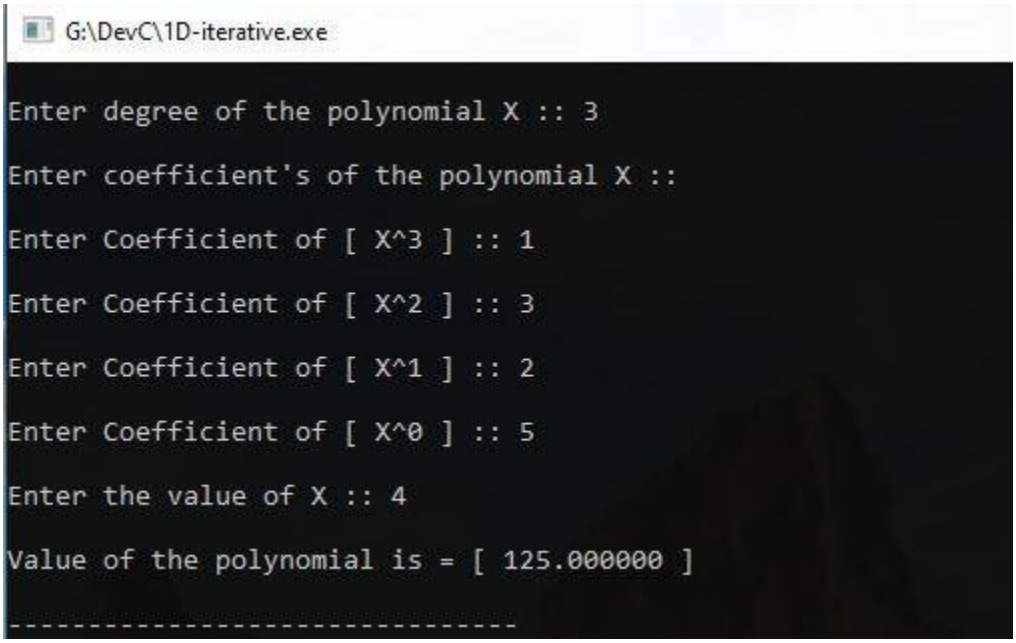
7. Print sum
8. [Finished]

End

PROGRAM :

```
/* Evaluating the polynomial f(x) = a4x4+ a3x3+ a2x2+ a1x+ a0 using Horner's method (n, i,
sum, a[], x) */
#include <stdio.h>

int main()
{
    float a[100],sum=0,x;
    int n,i;
    printf("\nEnter degree of the polynomial X :: ");
    scanf("%d",&n);
    printf("\nEnter coefficient's of the polynomial X :: \n");
    for(i=n;i>=0;i--)
    {
        printf("\nEnter Coefficient of [ X^%d ] :: ",i);
        scanf("%f",&a[i]);
    }

    printf("\nEnter the value of X :: ");
    scanf("%f",&x);

    for(i=n;i>0;i--)
    {
      sum=(sum+a[i])*x;
    }

    sum=sum+a[0];

    printf("\nValue of the polynomial is = [ %f ]\n",sum);

    return 0;
}
```

OUTPUT:



AIM: (USING DO While)

To compute Sin(x) using Taylor series approximation given by

$$\text{Sin(x)} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - .$$

ALGORITHM:

SINE SERIES[this algorithm takes degree as an input to print the sine function value]

Input : Degree of polynomial

Output : Sine function value

Step 1: start Step

2: enter the degree

Step 3: convert degree into radian

X <- degree*(PI/180)

Step 4: check the given number is odd/not

If(it is a odd number)

Then

{

term = nume/deno;

nume = -nume*x*x;

deno = deno*i*(i+1);

}

else

{
{
If(term<0.001)
{
Print the sine value
}
else
{
Check the number
}
}

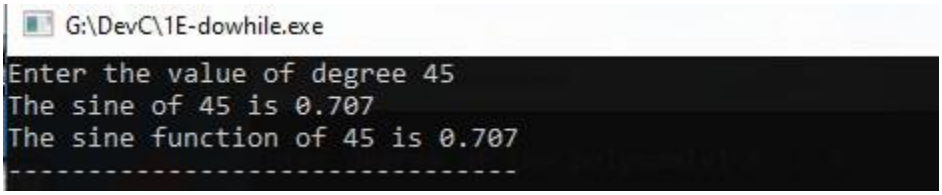
Step 5: stop

PROGRAM:

```
/* Program to calculate sine value of given angle */
#include<stdio.h>
#include<math.h>
#define PI 3.142
int main()
{
int i, degree;
float x, sum=0,term,nume,deno;
clrscr();
printf("Enter the value of degree");
scanf("%d",&degree);
x = degree * (PI/180);                    //converting degree into radian
nume = x;
deno = 1;
i=2;
do
{                                         //calculating the sine value.
term = nume/deno;
nume = -nume*x*x;
deno = deno*i*(i+1);
sum=sum+term;
i=i+2;
}
while (fabs(term) >= 0.00001);            // Accurate to 4 digits
 printf("The sine of %d is %.3f\n", degree, sum);
printf("The sine function of %d is %.3f", degree, sin(x));
 getch();
```

}

OUTPUT:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 2(A)	Practice of C programming using Functions

AIM: (USING Functions)

To compute two integer number addition.

ALGORITHM:

Step 1: start

Step 2: enter the two numbers

Step 3: sum value is the sction of function call

Step 4: addNumbers takes as a part of function definition and their prototype

Step 5: stop

PROGRAM:

```
#include <stdio.h>
int addNumbers(int a, int b);         // function prototype
int main()
{
   int n1,n2,sum;
   printf("Enters two numbers: ");
   scanf("%d %d",&n1,&n2);
   sum = addNumbers(n1, n2);         // function call
   printf("sum = %d",sum);
   return 0;
}
int addNumbers(int a, int b)         // function definition
{
   int result;
   result = a+b;
   return result;                    // return statement
}
```

OUTPUT

RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 2(B)	Practice of C programming using Arrays

AIM: (USING Arrays)

To 5 values from the user and store them in an array.

ALGORITHM:

Step 1: start

Step 2: enter the 5 integer values from user

Step 3: Store the values in Array

Step 4: Then displaying the values in an array

Step 5: stop

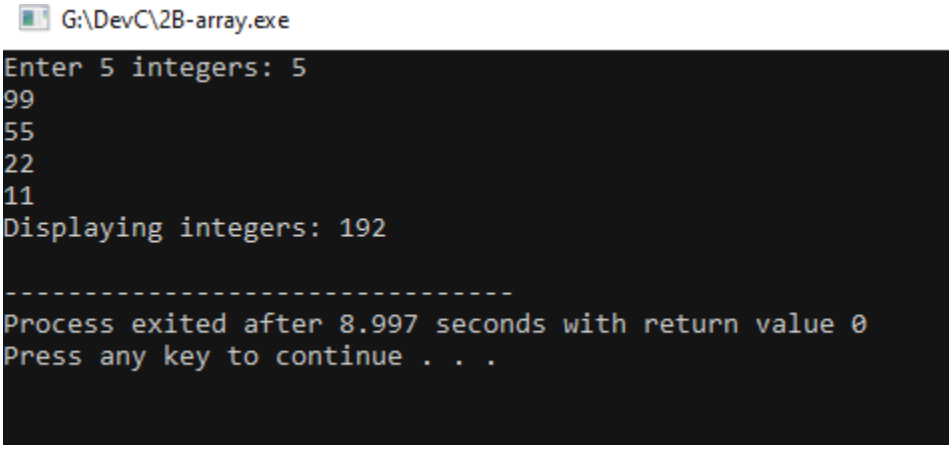
// Program to take 5 values from the user and store them in an array

// Print the elements stored in the array

Program:

```
#include <stdio.h>
int main() {
  int values[5];
  printf("Enter 5 integers: ");
  // taking input and storing it in an array
  for(int i = 0; i < 5; ++i)
{
    scanf("%d", &values[i]);
  }
  printf("Displaying integers: ");
  // printing elements of an array
  for(int i = 0; i < 5; ++i) {
    printf("%d\n", values[i]);
  }
  return 0;
}
```

OUTPUT:

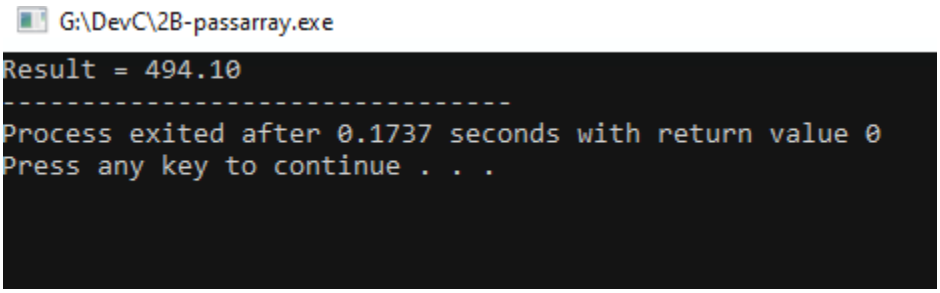


Example 2: Pass Arrays to Functions

```
#include <stdio.h>

float calculateSum(float num[]);

int main()

{

   float result, num[] = {23.4, 55.8, 22.6, 333.8, 40.5, 18};

  // num array is passed to calculateSum()

  result = calculateSum(num);

  printf("Result = %.2f", result);

  return 0;

}

float calculateSum(float num[])

{

        int i;

  float sum = 0.0;

  for (i = 0; i < 6; ++i)

  {
```

```
    sum += num[i];
  }
  return sum;
}
```

OUTPUT



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 3(A)	Implement C programs using Pointers

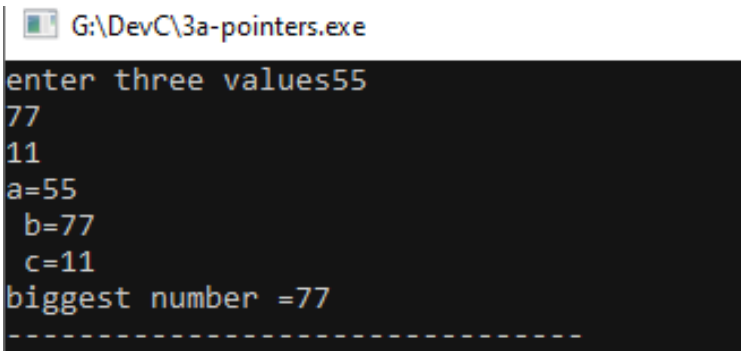
AIM:

To understand programming with Pointer, String and Function call by reference.

Program: Write a program to find biggest among three numbers using pointer.

```
#include <stdio.h>
int main()
{
int a,b,c;
int*ptra=&a,*ptrb=&b,*ptrc=&c;
printf("enter three values");
scanf("%d%d%d",ptra,ptrb,ptrc);
printf("a=%d\n b=%d\n c=%d\n",*ptra,*ptrb,*ptrc);
if((*ptra>*ptrb && *ptra>*ptrc))
printf("biggest number=%d",*ptra);
else if((*ptrb>*ptra && *ptrb>*ptrc))
printf("biggest number =%d",*ptrb);
else
printf("biggest number=%d",*ptrc);
return 0;
}
```

OUTPUT



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 3(B)	Implement C programs using Structures

AIM

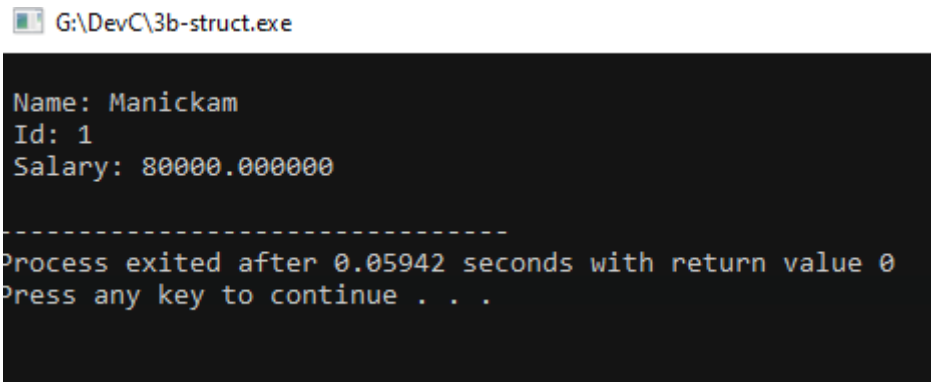
To understand programming with Structure.

Program 1: Write a C program to create, declare and initialize structure.

PROGRAM:

```
#include <stdio.h>
/*structure declaration*/
struct employee
{
char name[30];
int empId;
float salary;
};
int main()
{
/*declare and initialization  of structure variable*/
struct employee emp={"Manickam",001,80000.00};
printf("\n Name: %s" ,emp.name);
printf("\n Id: %d" ,emp.empId);
printf("\n Salary: %f\n",emp.salary);
return 0;
}
```

OUTPUT



STUDENT RECORD USING ARRAY OF STRUCTURES

Write a C program to maintain a record of "n" student details sing an array of structures with four fields (Roll number, Name, marks, and Grade). Each field is of an appropriate data type. Print the marks of the student given name as input.

AIM

To Write a C program to maintain a record of "n" student details.

ALGORITHM:

Input: Enter the number of students

Output: print the marks of the student.

Step1: start

Step2: enter the num of students

Step3: if(strcmp(s[i].name, sname==0)

Then

{

Print the marks of students name & USN

}

Else

{

Print the given date is invaid

}

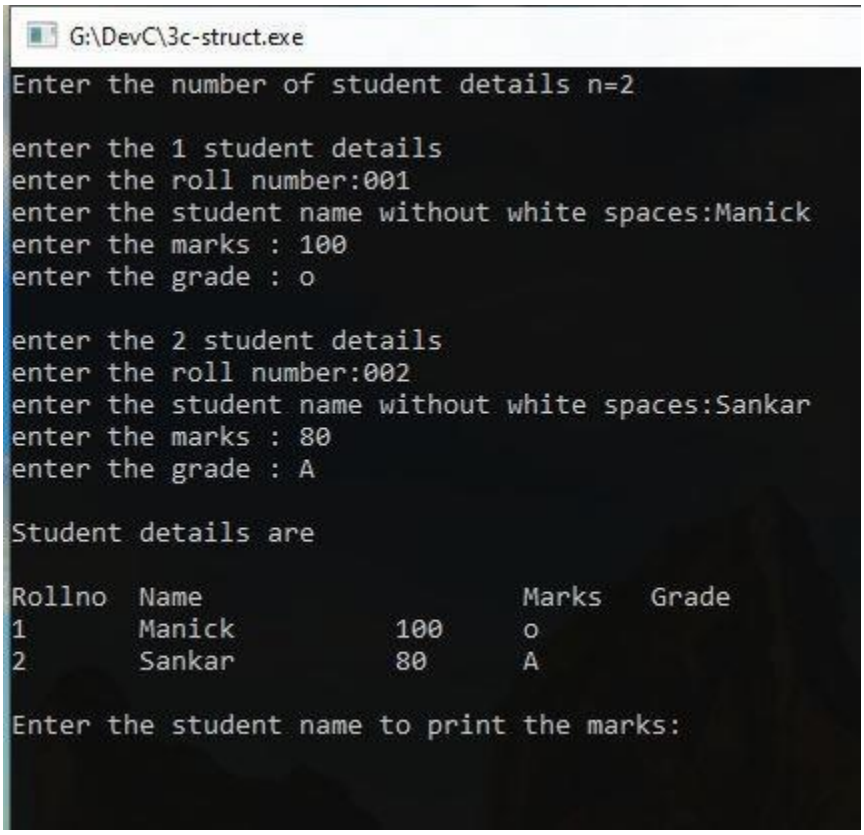
Step4: stop

Program:

```
/* program to maintain a record of student using structrue */
#include <stdio.h>
#include <conio.h>
struct student
{
int rollno, marks;
char name[20], grade;
};
void main()
{
int i,n,found=0;
struct student s[10];
char sname[20];
```

```
clrscr();
 printf("Enter the number of student details n=");
scanf("%d",&n);
 for(i=0;i<n;i++)
{
printf("\nenter the %d student details \n",i+1);
printf("enter the roll number:");
scanf("%d",&s[i].rollno);
printf("enter the student name without white spaces:");
scanf("%s",  s[i].name);  printf("enter the marks : ");
scanf("%d", &s[i].marks);
printf("enter the grade : ");
fflush(stdin); scanf("%c",&s[i].grade);
 }
printf("\nStudent details are \n");
printf("\nRollno\tName\t\t\tMarks\tGrade\n");
for(i=0;i<n;i++)
printf("%d\t%s\t\t%d\t%c\n", s[i].rollno, s[i].name, s[i].marks, s[i].grade); printf("\nEnter the
student name to print  the marks:");
scanf("%s", sname);
for(i=0;i<n;i++)
{
if(strcmp(s[i].name,sname)==0)
{
Printf("\Marks of the student is : %d",s[i].marks);
found=1;
}
}
if(found ==0)
printf(" Given student name not found\n");
getch();
}
```

Output:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 4	Implement C programs using Files

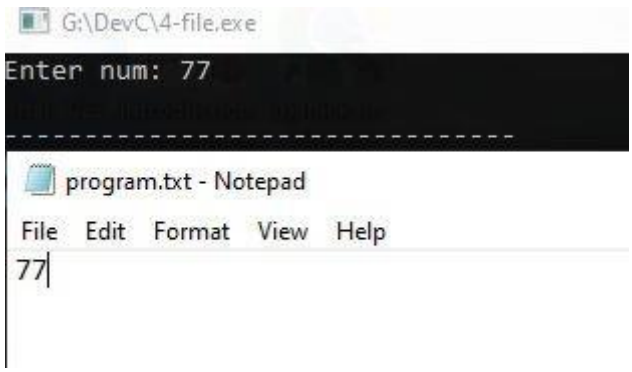
Aim:

To implement C programs using Files.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int num;
   FILE *fptr;

   // use appropriate location if you are using MacOS or Linux
   fptr = fopen("F:\\program.txt","w");

   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }

   printf("Enter num: ");
   scanf("%d",&num);

   fprintf(fptr,"%d",num);
   fclose(fptr);

   return 0;
}
```

OUTPUT:



Program 2

COPY THE CONTENTS OF FILE

Given two university information files "studentname.txt" and "usn.txt" that contains students names and USN respectively. Write a C program to a new file called "output.txt" and copy the content of files "studentname.txt" and "usn.txt" into output file in the sequence shows below. Display the content of output file "output.txt" on to the screen.

Note : Students are required to create two files "Studname.txt" and "Studusn.txt" with the Contents

ALGORITHM:

Input : Create the file fp1,fp2,fp3

Output : Print whether the files are found or not

step1:start

Step2: create the files (fp1,fp2, & fp3)

Step3: while(!feof((fp1)&&!feof(fp2))

Step4:Both files the not created the print the file not found

a). if only one file is created then print files not found.

b).if both files are created the print files are found.

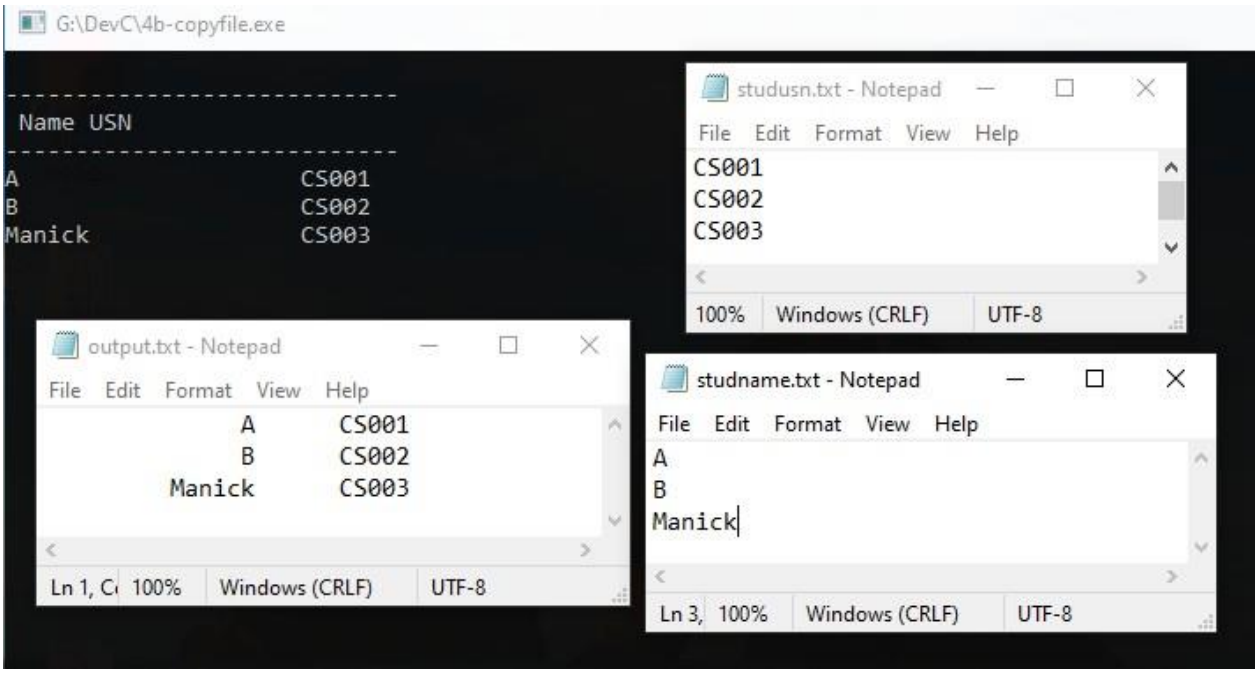
Step5: stop

Program

```
/* Program on merge two files */
#include<stdio.h>
int main()
{
FILE *fp1,*fp2,*fp3;
char usn[20], name[20];
fp1=fopen("studname.txt", "r");
if(fp1 == NULL)
printf(" File not found");
fp2=fopen("studusn.txt", "r");
if(fp2 == NULL)
printf(" File not found");
```

```
fp3=fopen("output.txt","w");
while( !feof(fp1) && !feof(fp2) )
{
fscanf(fp1,"%s",name);
fscanf(fp2,"%s",usn);
fprintf(fp3,"%15s %10s\n", name, usn);
}
fclose(fp1);
fclose(fp2);
fclose(fp3);
fp3=fopen("output.txt","r");
printf("\n----------------\n");
printf(" Name USN \n");
printf("-------------------------- \n");
while(!feof(fp3))
{
fscanf(fp3,"%s",name);
fscanf(fp3,"%s \n",usn);
 printf("%-15s %10s \n", name,usn);
}
fclose(fp3);
getch();
}
```

Output:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 5	Development of real time C applications

Aim:

To develop an calculator app with the help of C programming.

Algorithm of Calculator Program

Step 1: Declare local variables n1, n2, res, opt. For example, where n1 and n2 take two numeric values, res will store results and opt variable define the operator symbols.

Step 2: Print the Choice (Addition, Subtraction, multiplication, division, etc.

Step 3: Enter the Choice

Step 4: Takes two numbers, n1 and n2

Step 5: Switch case jump to an operator selected by the user

Step 6: Store result into res variable.

Step 7: Display the operation result

Step 8: Exit from the program.

Different ways to create a Calculator Program in C

Following are the different ways to write a Calculator Program in the C language.

1. Calculator Program in C using the switch statement
2. Calculator Program in C using if else if statement
3. Calculator Program in C using do-while loop and switch statement
4. Calculator Program in C using function and switch statement

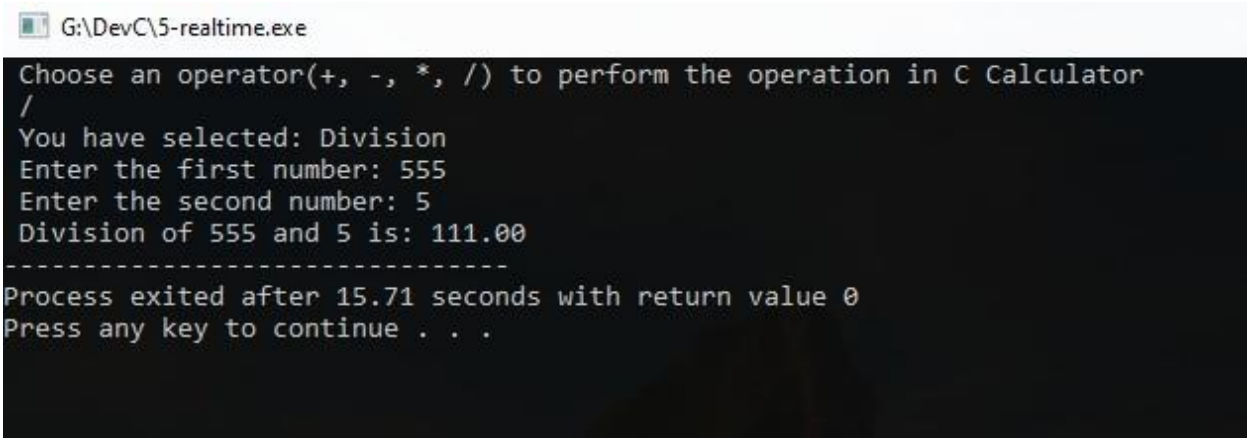
Example 1: Calculator Program in C using the switch statement

Let's write a program to create a Calculator program using switch statement

```
#include <stdio.h>
int main()
{
// declare local variables
char opt;
int n1, n2;
float res;
printf (" Choose an operator(+, -, *, /) to perform the operation in C Calculator \n ");
scanf ("%c", &opt); // take an operator
if (opt == '/' )
{
printf (" You have selected: Division");
13.     }
else if (opt == '*')
{
printf (" You have selected: Multiplication");
17.      }
18.
else if (opt == '-')
{
printf (" You have selected: Subtraction");
          }
             else if (opt == '+')
          {
             printf (" You have selected: Addition");
          }
printf (" \n Enter the first number: ");
scanf(" %d", &n1); // take fist number
printf (" Enter the second number: ");
scanf (" %d", &n2); // take second number 31.
switch(opt)
{
case '+':
res = n1 + n2; // add two numbers
printf (" Addition of %d and %d is: %.2f", n1, n2, res);
break;

case '-':
```

```
        res = n1 - n2; // subtract two numbers
        printf (" Subtraction of %d and %d is: %.2f", n1, n2, res);
        break;

    case '*':
        res = n1 * n2; // multiply two numbers
        printf (" Multiplication of %d and %d is: %.2f", n1, n2, res);
        break;

    case '/':
        if (n2 == 0)   // if n2 == 0, take another number
        {
            printf (" \n Divisor cannot be zero. Please enter another value ");
            scanf ("%d", &n2);
            }
        res = n1 / n2; // divide two numbers
        printf (" Division of %d and %d is: %.2f", n1, n2, res);
        break;
    default:  /* use default to print default message if any condition is not satisfied */
        printf (" Something is wrong!! Please check the options ");
  }
  return 0;
}
```

Output



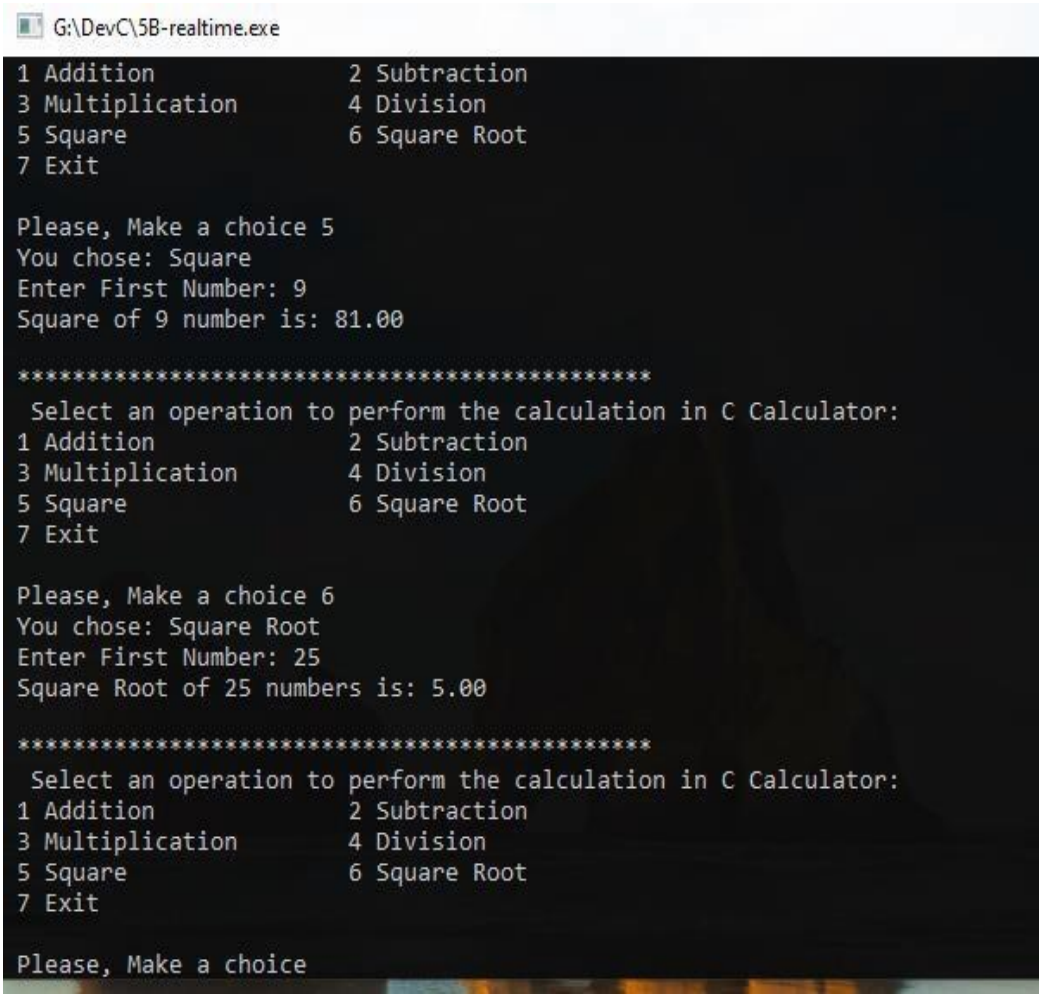
Example 2: Calculator Program in C using do while loop and switch statement

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

int main()
{
   // declaration of local variable op;
   int op, n1, n2;
   float res;
   char  ch;
   do
   {
      // displays the multiple operations of the C Calculator
      printf (" Select an operation to perform the calculation in C Calculator: ");
      printf (" \n 1 Addition  \t \t 2 Subtraction \n 3 Multiplication \t 4 Division  \n 5 Square \t \t 6
Square Root \n 7 Exit \n \n Please, Make a choice ");

      scanf ("%d", &op); // accepts a numeric input to choose the operation


   // use switch statement to call an operation
   switch (op)
   {
      case 1:
         // Add two numbers
         printf (" You chose: Addition");
         printf ("\n Enter First Number: ");
         scanf (" %d", &n1);
         printf (" Enter Second Number: ");
         scanf (" %d", &n2);
         res = n1 + n2; // Add two numbers
         printf (" Addition of two numbers is: %.2f", res);
         break; // break the function

      case 2:
         // Subtract two numbers
         printf (" You chose: Subtraction");
         printf ("\n Enter First Number: ");
         scanf (" %d", &n1);
         printf (" Enter Second Number: ");
         scanf (" %d", &n2);
         res = n1 - n2; // subtract two numbers
         printf (" Subtraction of two numbers is: %.2f", res);
         break; // break the function
```

```
case 3:
   // Multiplication  of the numbers
   printf (" You chose: Multiplication");
   printf ("\n Enter First Number: ");
   scanf (" %d", &n1);
   printf (" Enter Second Number: ");
   scanf (" %d", &n2);
   res = n1 * n2; // multiply two numbers
   printf (" Multiplication of two numbers is: %.2f", res);
   break; // break the function

case 4:
   // Division of the numbers
   printf (" You chose: Division");
   printf ("\n Enter First Number: ");
   scanf (" %d", &n1);
   printf (" Enter Second Number: ");
   scanf (" %d", &n2);
   if (n2 == 0)
      {
         printf (" \n Divisor cannot be zero. Please enter another value ");
         scanf ("%d", &n2);
      }
   res = n1 / n2; // divide two numbers
   printf (" Division of two numbers is: %.2f", res);
   break; // break the function

case 5:
   // getting square of a number
   printf (" You chose: Square");
   printf ("\n Enter First Number: ");
   scanf (" %d", &n1);

   res = n1 * n1; // get square of a number
   printf (" Square of %d number is: %.2f", n1, res);
   break; // break the function

case 6:
   // getting the square root of the number
   printf (" You chose: Square Root");
   printf ("\n Enter First Number: ");
   scanf (" %d", &n1);

   res = sqrt(n1); // use sqrt() function to find the Square Root
   printf (" Square Root of %d numbers is: %.2f", n1, res);
   break; // break the function
```

```
        case 7:
            printf (" You chose: Exit");
            exit(0);
            break; // break the function

        default:
            printf(" Something is wrong!! ");
            break;
    }
    printf (" \n \n ********************************************** \n ");
    } while (op != 7);

    return 0;
}
```

OUTPUT



RESULT: Thus the C programs can be executed and the output was verified successfully.

EX.NO: 6	Array implementation of List ADT

AIM:

To implement array implementation of List ADT.

Algorithm:

structure could be used to create a struct

```
struct Node

{
        int node ;

        struct Node *next;
};
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include  <math.h>
#include <stdlib.h>
#define MAX 10

void create();
void insert();
void deletion();
void search();
void display();
int a,b[20], n, p, e, f, i, pos;

void main()
{
//clrscr();
int ch;
char g='y';
```

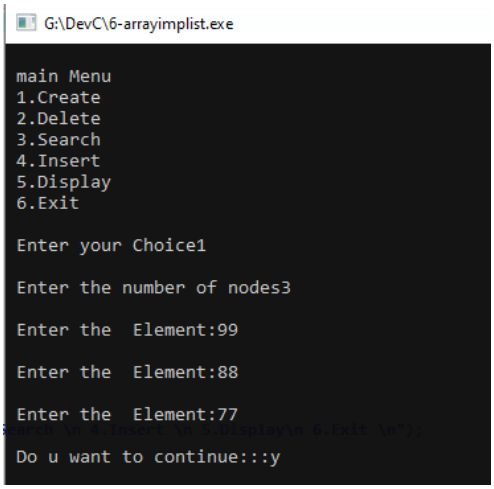
```
do
{
printf("\n main Menu");
printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
printf("\n Enter your Choice");
scanf("%d", &ch);

switch(ch)
{
case 1:
create();
break;

case 2:
deletion();
break;

case 3:
search();
break;

case 4:
insert();
break;

case 5:
display();
break;
default:
printf("\n Enter the correct choice:");
}
printf("\n Do u want to continue:::");
scanf("\n%c", &g);
}
while(g=='y'||g=='Y');
```

```
getch();
}


void create()
{
printf("\n Enter the number of nodes");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n Enter the Element:",i+1);
scanf("%d", &b[i]);
}

}

void deletion()
{
printf("\n Enter the position u want to delete::");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location::");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];
}
n--;
}
printf("\n The Elements after deletion");
for(i=0;i<n;i++)
{
```

```
printf("\t%d", b[i]);
}
}


void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);

for(i=0;i<n;i++)
{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}
else
{
printf("Value %d is not in the list::", e);
continue;
}
}
}


void insert()
{
printf("\n Enter the position u need to insert::");
scanf("%d", &pos);

if(pos>=n)
 {
 printf("\n invalid Location::");
 }
 else
```

```
 {
 for(i=MAX-1;i>=pos-1;i--)
 {
 b[i+1]=b[i];
 }
 printf("\n Enter the element to insert::\n");
 scanf("%d",&p);
 b[pos]=p;
 n++;
 }
 printf("\n The list after insertion::\n");
display();
}


void display()
{
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}
}
```

OUTPUT



main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice1

Enter the number of nodes3

Enter the Element:99

Enter the Element:88

Enter the Element:77

Do u want to continue:::y

main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice2

Enter the position u want to delete::0

The Elements after deletion 88 77
Do u want to continue:::y

main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice4

Enter the position u need to insert::1

Enter the element to insert::
11

The list after insertion::

The Elements of The list ADT are:

88

11

77
Do u want to continue:::y

main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exit

Enter your Choice3

Enter the Element to be searched:77
Value 77 is not in the list::Value 77 is not in the list::Value is in the 2 Position
Do u want to continue:::

RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 7(A)	Array implementation of Stack ADTs

Aim:

To write a C program for Implementation of stack using array.

Algorithm:

Adding an element onto the stack (push operation)

Adding an element into the top of the stack is referred to as push operation. Push operation involves following two steps.

1. Increment the variable Top so that it can now refere to the next memory location.
2. Add element at the position of incremented top. This is referred to as adding new element at the top of the stack.

Stack is overflown when we try to insert an element into a completely filled stack therefore, our main function must always avoid stack overflow condition.

```
begin
  if top = n then stack full
    top = top + 1
    stack (top) : = item;
end
```

Deletion of an element from a stack (Pop operation)

Deletion of an element from the top of the stack is called pop operation. The value of the variable top will be incremented by 1 whenever an item is deleted from the stack. The top most element of the stack is stored in an another variable and then the top is decremented by 1. the operation returns the deleted value that was stored in another variable as the result.

The underflow condition occurs when we try to delete an element from an already empty stack.

```
begin
   if top = 0 then stack empty;
     item := stack(top);
     top = top - 1;
end;
```

Program:

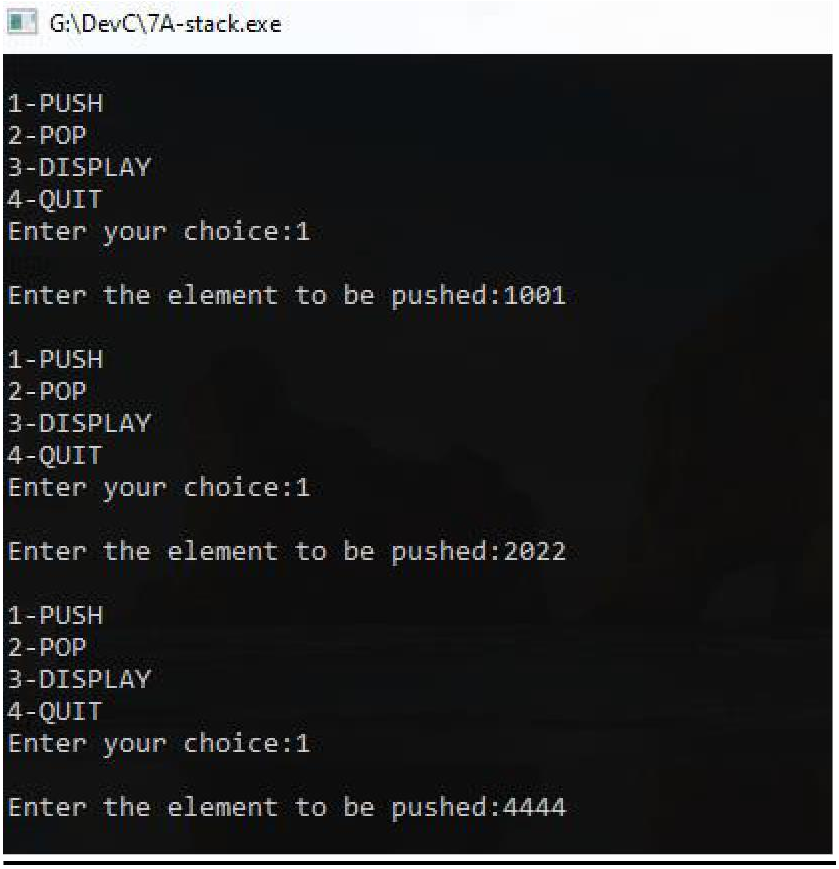
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAXSTK 100
int top=-1;
```

```
int items[MAXSTK];
int Isempty();
int Isfull();
void Push(int);
int Pop();
void Display();
void main()
{
int x;
char ch='1';
clrscr();
while(ch!='4')
{
printf("\n 1-PUSH");
printf("\n 2-POP");
printf("\n 3-DISPLAY");
printf("\n 4-QUIT");
printf("\n Enter your choice:");
fflush(stdin);
ch=getchar();
switch(ch)
{
case '1':
printf("\n Enter the element to be pushed:");
scanf("%d",&x);
Push(x);
break;
case '2':
 x=Pop();
printf("\n Pop element is %d\n:",x);
break;
case '3':
```

```
Display();
break;
case '4':
break;
default:
printf("\n Wrong choice!Try again:");
}
}
}
 int Isempty()
{
if(top==-1)
return 1;
else return 0;
}
int Isfull()
{
if(top==MAXSTK-1)
return 1;
else
return 0;
}
 void Push(int x)
{
 if(Isfull())
{
printf("\n Stack full");
return;
}
top++;
items[top]=x;
 }
```

```
int Pop()
{
 int x;
if(Isempty())
{
printf("\n Stack empty");
exit(0);
}
x=items[top];
top--;
 return x;
 }
void Display()
{
int i;
if(Isempty())
{
 printf("\n Stack empty");
return;
}
printf("\n Elements in the Stack are :\n");
 for(i=top;i>=0;i--)
printf("%d\n",items[i]);
 }
```

Output:



1-PUSH
2-POP
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the element to be pushed:1001
1-PUSH
2-POP
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the element to be pushed:2022

1-PUSH
2-POP
3-DISPLAY
4-QUIT
Enter your choice:1

Enter the element to be pushed:4444

1-PUSH
2-POP
3-DISPLAY
4-QUIT
Enter your choice:2

Pop element is 4444
:
1-PUSH
2-POP
3-DISPLAY
4-QUIT
Enter your choice:3
Elements in the Stack are :
2022
1001
1-PUSH
2-POP
3-DISPLAY
4-QUIT
Enter your choice: 4

RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 7(B)	Array implementation of Queue ADTs

Aim:

To write a C program for Implementation of queue using array.

Algorithm:

Step 1: IF REAR = MAX – 1

Write OVERFLOW
Go to step
[END OF IF]

Step 2: IF FRONT = -1 and REAR = -1

SET FRONT = REAR = 0
ELSE
SET REAR = REAR + 1
[END OF IF]

Step 3: Set QUEUE[REAR] = NUM

Step 4: EXIT

Program:

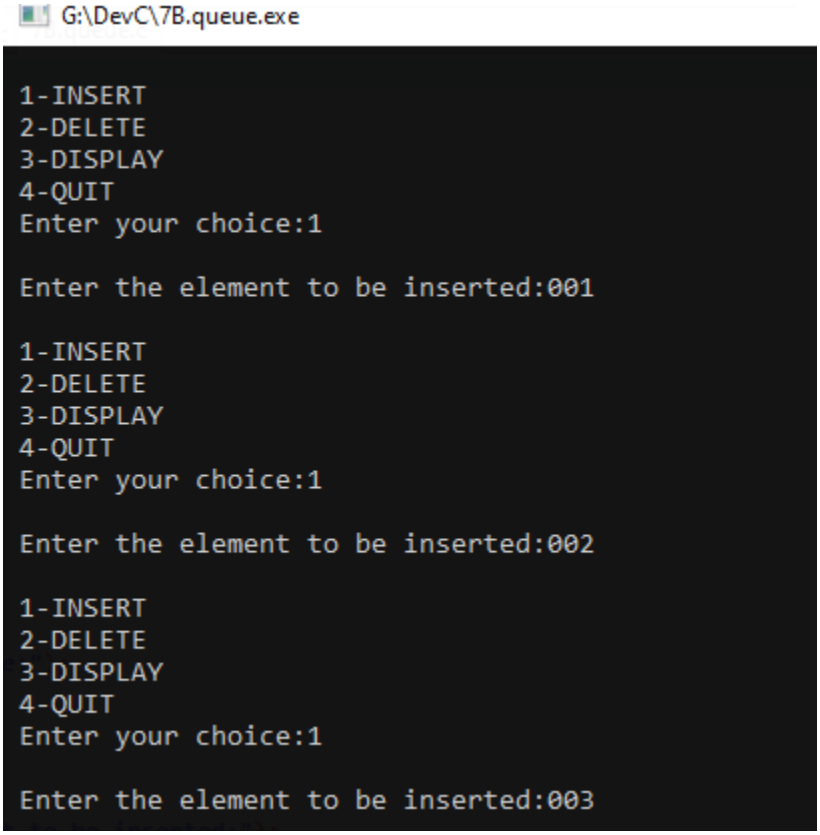
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>#define MAXQ 100
int front=0,rear=-1;
int items[MAXQ];
int Isempty();
int Isfull();
void Insert(int);
int Delete();
void Display();
void main()
{
 int x;
char ch='1';
clrscr();
while(ch!='4')
```

```
{
printf("\n 1-INSERT");
printf("\n 2-DELETE");
printf("\n 3-DISPLAY");
printf("\n 4-QUIT");
printf("\n Enter your choice:");
fflush(stdin);
ch=getchar();
switch(ch)
{
case '1':
printf("\n Enter the element to be inserted:");
scanf("%d",&x);
Insert(x);
break;
case '2':
x=Delete();
printf("\n Delete element is %d\n:",x);
break;
case '3':
Display();
break;
case '4':
break;
default:
printf("\n Wrong choice!Try again:");
}
}
getch();
}
 int Isempty()
{
```

```
if(rear<front)
return 1;
else return 0;
}
int Isfull()
{
if(rear==MAXQ-1)
return 1;
else
return 0;
}
void Insert(int x)
{
if(Isfull())
{
printf("\n Queue full");
return;
}
 rear++;
items[rear]=x;
}
int Delete()
{
 int x;
 if(Isempty())
{
printf("\n Queue is empty");
exit(0);
}
x=items[front];
front++;
return x;
```

```
}
void Display()
{
 int i;
if(Isempty())
{
printf("\n Queue is empty");
return;
}
printf("\n Elements in the Queue are :\n");
for(i=front;i<=rear;i++)
printf("%d\n",items[i]);
}
```

Output:



1-INSERT
2-DELETE

3-DISPLAY
4-QUIT
Enter your choice:1

Enter the element to be inserted:001
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1
Enter the element to be inserted:002
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:1
Enter the element to be inserted:003
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:2

Delete element is 1
:
1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:3

Elements in the Queue are :
2
3

1-INSERT
2-DELETE
3-DISPLAY
4-QUIT
Enter your choice:

RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 8(A)	Linked list implementation of List ADTs

Aim:

Write a C program that uses functions to perform the following on Singly Linked List: i) Creation ii) Insertion iii) Deletion iv) Traversal

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int count=0;
struct node
{
int data;
struct node *next;
}*head,*newn,*trav;
//-----------------------------------
void create_list()
{
int value;
struct node *temp;
temp=head;
newn=(struct node *)malloc(sizeof (struct node));
printf("\nenter the value to be inserted");
scanf("%d",&value);
newn->data=value;
if(head==NULL)
{
head=newn;
head->next=NULL;
count++;
}
Else
{
while(temp->next!=NULL)
{
temp=temp->next;
}
temp->next=newn;
newn->next=NULL;
count++;
```

```
}
}
//----------------------------------------
void insert_at_begning(int value)
{
newn=(struct node *)malloc(sizeof (struct node));
newn->data=value;
if(head==NULL)
{
head=newn;
head->next=NULL;
count++;
}
Else
{
newn->next=head;
head=newn;
count++;
}
}
//----------------------------------------
void insert_at_end(int value)
{
struct node *temp;
temp=head;
newn=(struct node *)malloc(sizeof (struct node));
newn->data=value;
if(head==NULL)
{
head=newn;
head->next=NULL;
count++;
}
Else
{
while(temp->next!=NULL)
{
temp=temp->next;
}
temp->next=newn;
newn->next=NULL;
count++;
}
}
//----------------------------------------
int insert_at_middle()
```

```
{
if(count>=2)
{
struct node *var1,*temp;
int loc,value;
printf("\n after which value you want to insert : ");
scanf("%d",&loc);
printf("\nenter the value to be inserted");
scanf("%d",&value);
newn=(struct node *)malloc(sizeof (struct node));
newn->data=value;
temp=head;
/* if(head==NULL)
{
head=newn;
head->next=NULL;
count++;
return 0;
}
Else
{*/
while(temp->data!=loc)
{
temp=temp->next;
if(temp==NULL)
{
printf("\nSORRY...there is no %d element",loc);
return 0;
}
}
//var1=temp->next;
newn->next=temp->next;//var1;
temp->next=newn;
count++;
//}
}
Else
{
printf("\nthe no of nodes must be >=2");
}
}
//----------------------------------------
int delete_from_middle()
{
if(count==0)
printf("\n List is Empty!!!! you can't delete elements\n");
```

```
else if(count>2)
{
struct node *temp,*var;
int value;
temp=head;
printf("\nenter the data that you want to delete from the list shown above");
scanf("%d",&value);
while(temp->data!=value)
{
var=temp;
temp=temp->next;
if(temp==NULL)
{
printf("\nSORRY...there is no %d element",value);
return 0;
}
}
if(temp==head)
{
head=temp->next;
}
Else
{
var->next=temp->next;
temp->next=NULL;
}
count--;
if(temp==NULL)
printf("Element is not avilable in the list \n**enter only middle elements..**");
else
printf("\ndata deleted from list is %d",value);
free(temp);
}
Else
{
printf("\nthere no middle elemts..only %d elemts is avilable\n",count);
}
}
//----------------------------------------
int delete_from_front()
{
struct node *temp;
temp=head;
if(head==NULL)
{
printf("\nno elements for deletion in the list\n");
```

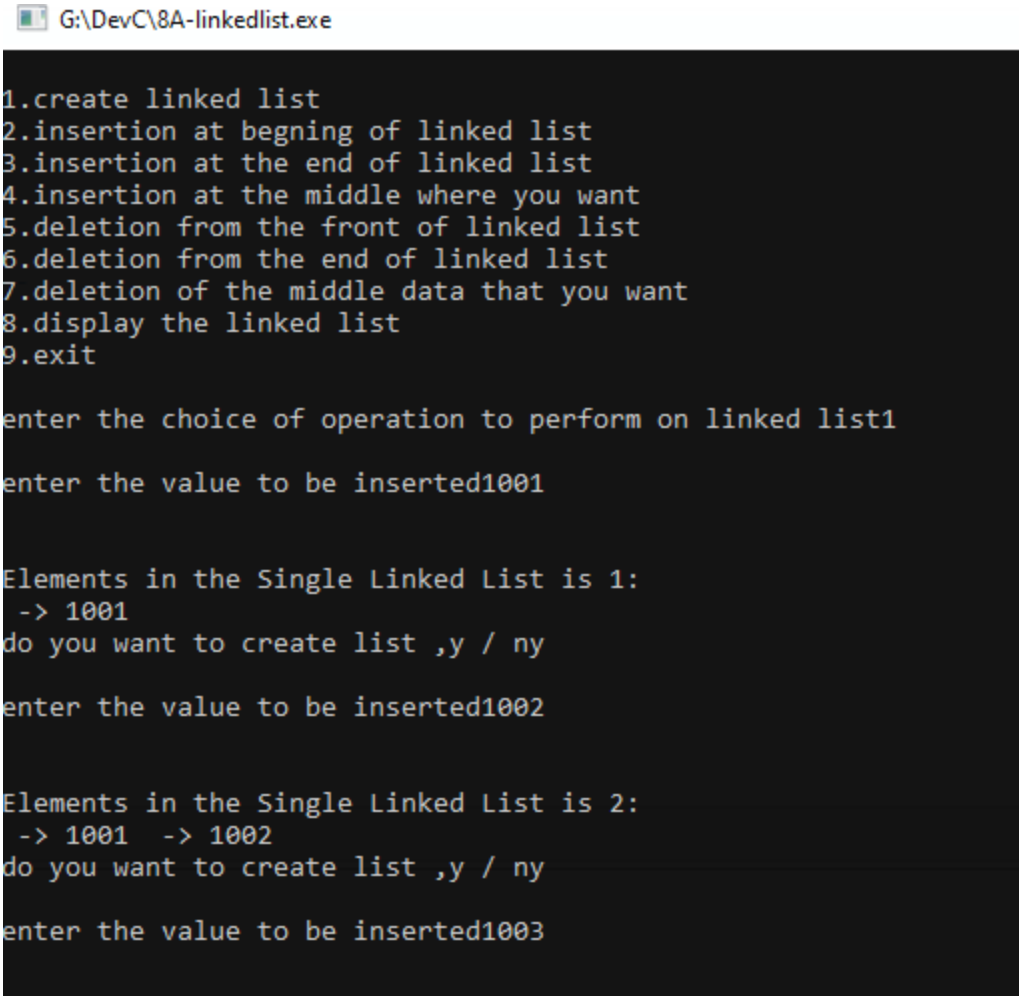
```
return 0;
}
Else
{
printf("\ndeleted element is :%d",head->data);
if(temp->next==NULL)
{
head=NULL;
}
else{
head=temp->next;
temp->next=NULL;
}
count--;
free(temp);
}}
//-----------------------------------
int delete_from_end(){
struct node *temp,*var;
temp=head;
if(head==NULL)
{
printf("\nno elemts in the list");
return 0;
}
else{
if(temp->next==NULL )
{
head=NULL;//temp->next;
}
else{
while(temp->next != NULL)
{
var=temp;
temp=temp->next;
}
var->next=NULL;
}
printf("\ndata deleted from list is %d",temp->data);
free(temp);
count--;
}
return 0;}
//-----------------------------------
int display()
{
```

```
trav=head;
if(trav==NULL)
{
printf("\nList is Empty\n");
return 0;}
else
{
printf("\n\nElements in the Single Linked List is %d:\n",count);
while(trav!=NULL)
{
printf(" -> %d ",trav->data);
trav=trav->next;
}
printf("\n");
}
}
//----------------------------------------
int main()
{
int ch=0;
char ch1;
head=NULL;
while(1)
{
printf("\n1.create linked list");
printf("\n2.insertion at begning of linked list");
printf("\n3.insertion  at the end of linked list");
printf("\n4.insertion  at the middle where you want");
printf("\n5.deletion from the front of linked list");
printf("\n6.deletion from the end of linked list ");
printf("\n7.deletion of the middle data that you want");
printf("\n8.display the linked list");
printf("\n9.exit\n");
printf("\nenter the choice of operation to perform on linked list");
scanf("%d",&ch);
switch(ch)
{
case 1:
{
do{
create_list();
display();
printf("do you want to create list ,y / n");
getchar();
scanf("%c",&ch1);
}while(ch1=='y');
```

```
break;
}
case 2:
{
int value;
printf("\nenter the value to be inserted");
scanf("%d",&value);
insert_at_begning(value);
display();
break;
}
case 3:
{
int value;
printf("\nenter value to be inserted");
scanf("%d",&value);
insert_at_end(value);
display();
break;
}
case 4:
{
insert_at_middle();
display();
break;
}
case 5:
{
delete_from_front();
display();
}break;
case 6:
{
delete_from_end();
display();
break;
}
case 7:
{
display();
delete_from_middle();
display();
break;
}
case 8:
{
```

```
display();
break;
}
case 9:
{
exit(1);
}
default:printf("\n****Please enter correct choice****\n");
}
}
getch();
}
```

OUTPUT



1. create linked list
2. insertion at begning of linked list
3. insertion at the end of linked list

```
4. insertion at the middle where you want
5. deletion from the front of linked list
6. deletion from the end of linked list
7. deletion of the middle data that you want
8. display the linked list
9. exit

enter the choice of operation to perform on linked list1

enter the value to be inserted1001


Elements in the Single Linked List is 1:
 -> 1001
do you want to create list ,y / ny

enter the value to be inserted1002


Elements in the Single Linked List is 2:
 -> 1001  -> 1002
do you want to create list ,y / ny

enter the value to be inserted1003


Elements in the Single Linked List is 3:
 -> 1001  -> 1002  -> 1003
do you want to create list ,y / nn

1. create linked list
2. insertion at begning of linked list
3. insertion at the end of linked list
4. insertion at the middle where you want
5. deletion from the front of linked list
6. deletion from the end of linked list
7. deletion of the middle data that you want
8. display the linked list
9. exit

enter the choice of operation to perform on linked list7


Elements in the Single Linked List is 3:
 -> 1001  -> 1002  -> 1003
```

```
enter the data that you want to delete from the list shown above1002

data deleted from list is 1002

Elements in the Single Linked List is 2:
 -> 1001  -> 1003

1. create linked list
2. insertion at begning of linked list
3. insertion at the end of linked list
4. insertion at the middle where you want
5. deletion from the front of linked list
6. deletion from the end of linked list
7. deletion of the middle data that you want
8. display the linked list
9. exit

enter the choice of operation to perform on linked list 9
```

RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 8(B)	Linked list implementation of Stack ADTs

Aim:

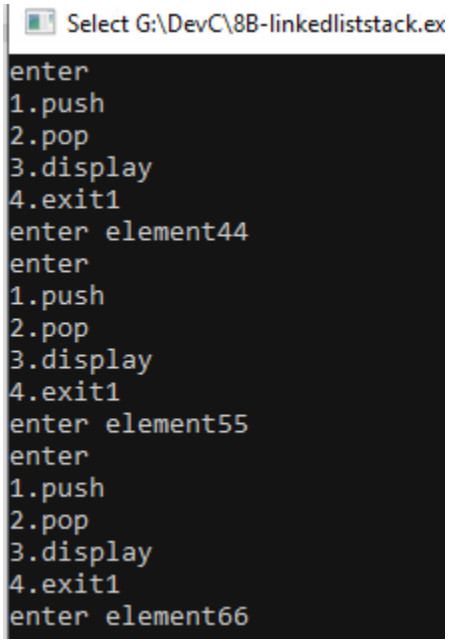
Write a program that implement Stack (its operations) using Arrays

Program:

```
#include<stdio.h>
#include<conio.h>
#define max 5
int st[max],top=-1;
void push()
{
int x;
if(top==max-1)
{
printf("stack is full");
return;
}
printf("enter element");
scanf("%d",&x);
top++;
st[top]=x;
}
void pop()
{
int x;
if(top==-1)
{
printf("stack is empty");
return;
}
printf("enter element");
scanf("%d",&x);
top--;
printf("enter deleted element=%d",x);
}
void display()
{
int i;
if(top==-1)
{
printf("stack is empty");
return;}
for(i=top;i>=0;i--)
```

```
{
printf("%d",st[i]);}}
int main()
{
int ch;
while(1)
{
printf("enter \n1.push\n2.pop\n3.display\n4.exit");
scanf("%d",&ch);
switch(ch)
{
case 1:push();break;
case 2:pop();break;
case 3:display();break;
case 4:exit(1);break;
}}
return 1;
}
```

OUTPUT:



enter
1.push
2.pop
3. display
4. exit1
enter element44
enter
1.push

```
2.pop
3. display
4. exit1
enter element55
enter
1.push
2.pop
3. display
4. exit1
enter element66
enter
1.push
2.pop
3. display
4. exit2
enter element55
enter deleted element= 55enter
1.push
2.pop
3. display
4. exit3
5544enter
1.push
2.pop
3. display
4. exit
```

RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 8(C)	Linked list implementation of Queue ADTs

Aim:

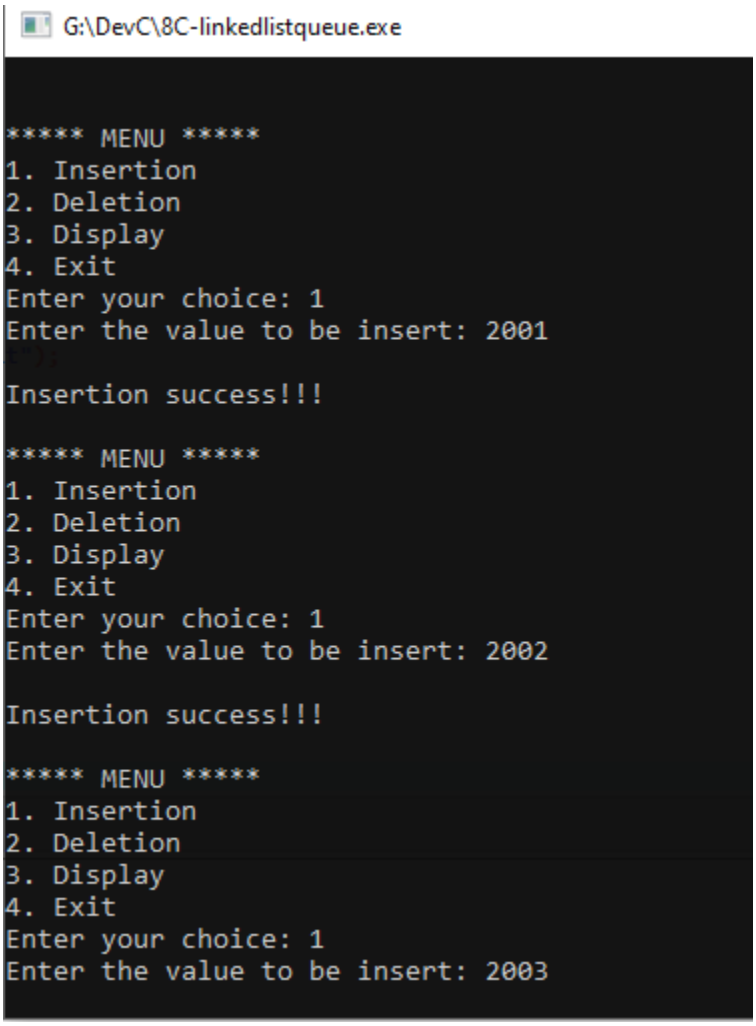
Write a program that implement Queue (its operations) using Arrays

Program:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 5
int queue[SIZE], front = -1, rear = -1;
void enQueue(int value){
if(rear == SIZE-1)
printf("\nQueue is Full!!! Insertion is not possible!!!");
else{
if(front == -1)
front = 0;
rear++;
queue[rear] = value;
printf("\nInsertion success!!!");
}
}
void deQueue(){
if(front == rear)
printf("\nQueue is Empty!!! Deletion is not possible!!!");
else{
printf("\nDeleted : %d", queue[front]);
front++;
if(front == rear)
front = rear = -1;
}
}
void display(){
if(rear == -1)
printf("\nQueue is Empty!!!");
```

```
else{
int i;
printf("\nQueue elements are:\n");
for(i=front; i<=rear;  i++)
printf("%d\t",queue[i]);
}
}
void main()
{
int value, choice;
while(1){
printf("\n\n***** MENU *****\n");
printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice){
case 1: printf("Enter the value to be insert: ");
scanf("%d",&value);
enQueue(value);
break;
case 2: deQueue();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong selection!!! Try again!!!");
} } }
```

OUTPUT:



******* MENU *******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 2001

Insertion success!!!

******* MENU *******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1

Enter the value to be insert: 2002

Insertion success!!!

******* MENU *******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 2003

Insertion success!!!

******* MENU *******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 2001

******* MENU *******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
2002 2003

******* MENU *******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice:
RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 9(A)	Applications of Stack ADTs

Aim:

Write a program that implement Applications of Stack ADTs

Applications of stacks

Stacks are used in a variety of applications. While some of these applications are "natural", most other are essentially "pedantic". Here is a list anyway.

For processing nested structures, like checking for balanced parentheses, evaluation of postfix expressions.

For handling function calls and, in particular, recursion.

For searching in special data structures (depth-first search in graphs and trees), for example, for implementing *backtracking*.

Implementations of the stack ADT

A stack is specified by the ordered collection representing the content of the stack together with the choice of the end of the collection to be treated as the top. The top should be so chosen that pushing and popping can be made as far efficient as possible

Program:

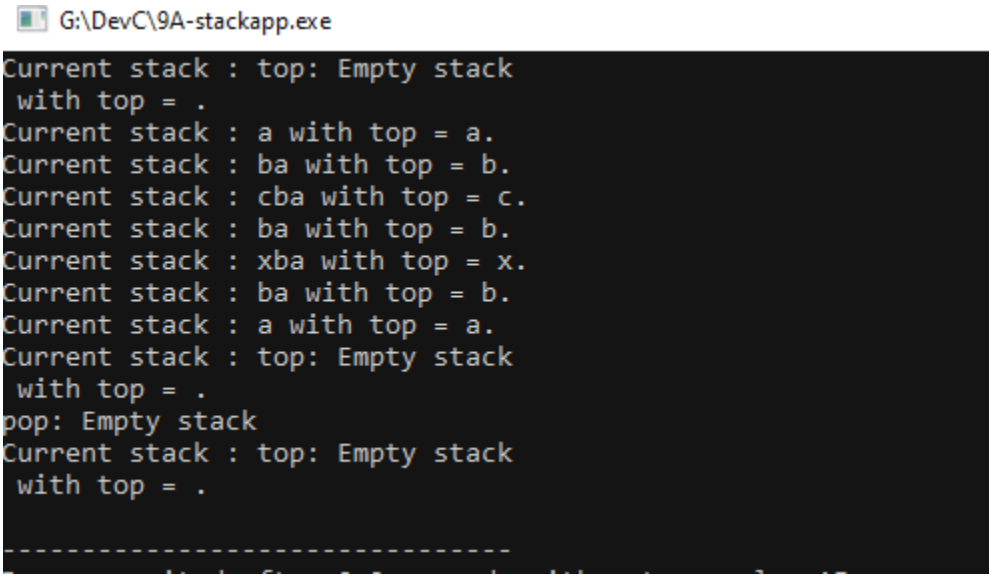
```
#define MAXLEN 100
typedef struct {
     char element[MAXLEN];
     int top;
  } stack;
 stack init ()
  {
     stack S;
     S.top = -1;
     return S;
  }
  int isEmpty ( stack S )
  {
     return (S.top == -1);
  }
  int isFull ( stack S )
  {
     return (S.top == MAXLEN - 1);
  }
  char top ( stack S )
  {
     if (isEmpty(S)) {
```

```
    fprintf(stderr, "top: Empty stack\n");
    return '\0';
  }
  return S.element[S.top];
}
stack push ( stack S , char ch )
{
  if (isFull(S)) {
    fprintf(stderr, "push: Full stack\n");
    return S;
  }
  ++S.top;
  S.element[S.top] = ch;
  return S;
}
stack pop ( stack S )
{
  if (isEmpty(S)) {
    fprintf(stderr, "pop: Empty stack\n");
    return S;
  }
  --S.top;
  return S;
}
void print ( stack S )
{
  int i;
  for (i = S.top; i >= 0; --i) printf("%c",S.element[i]);
}
```

Here is a possible main() function calling these routines:

```
int main ()
{
  stack S;
  S = init(); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = push(S,'a'); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = push(S,'b'); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = push(S,'c'); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = pop(S); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = push(S,'x'); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = pop(S); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = pop(S); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = pop(S); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
  S = pop(S); printf("Current stack : "); print(S); printf(" with top = %c.\n", top(S));
}
```

Output:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 9(B)	Applications of Queue ADTs

Aim:

Write a program that implement Applications of Queue ADTs.

Implementations of the queue ADT

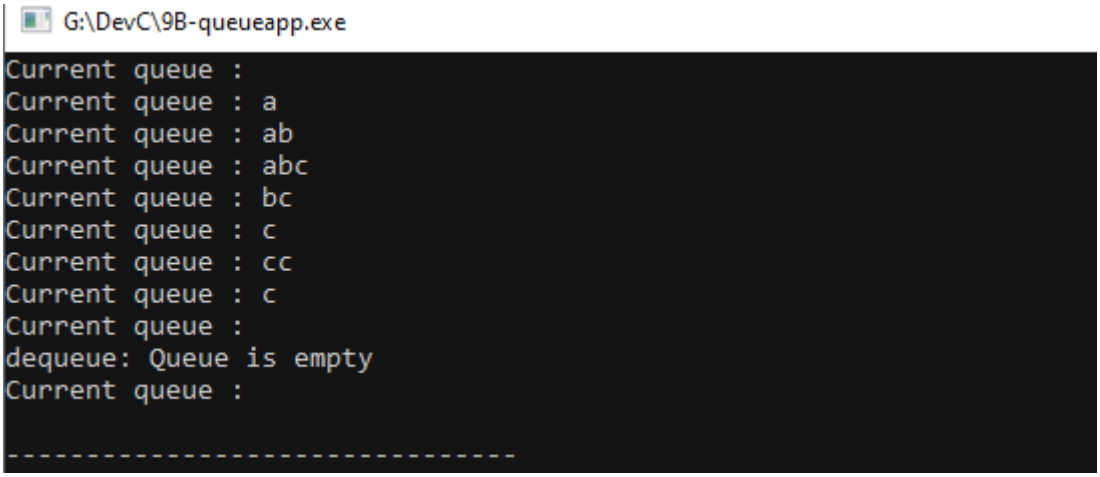
Continuing with our standard practice followed so far, we are going to provide two implementations of the queue ADT, the first using static memory, the second using dynamic memory. The implementations aim at optimizing both the insertion and deletion operations.

Program:

```
#define MAXLEN 100
  typedef struct {
    char element[MAXLEN];
    int front;
    int back;
  } queue;
  queue init ()
  {
    queue Q;
    Q.front = 0;
    Q.back = MAXLEN - 1;
    return Q;
  }
  int isEmpty ( queue Q )
  {
    return (Q.front == (Q.back + 1) % MAXLEN);
  }
  int isFull ( queue Q )
  {
    return (Q.front == (Q.back + 2) % MAXLEN);
  }
  char front ( queue Q )
  {
    if (isEmpty(Q)) {
      fprintf(stderr,"front: Queue is empty\n");
      return '\0';
    }
    return Q.element[Q.front];
  }
  queue enqueue ( queue Q , char ch )
  {
    if (isFull(Q)) {
      fprintf(stderr,"enqueue: Queue is full\n");
```

```
    return Q;
  }
  ++Q.back;
  if (Q.back == MAXLEN) Q.back = 0;
  Q.element[Q.back] = ch;
  return Q;
}
queue dequeue ( queue Q )
{
  if (isEmpty(Q)) {
    fprintf(stderr,"dequeue: Queue is empty\n");
    return Q;
  }
  ++Q.front;
  if (Q.front == MAXLEN) Q.front = 0;
  return Q;
}
void print ( queue Q )
{
  int i;
  if (isEmpty(Q)) return;
  i = Q.front;
  while (1) {
    printf("%c", Q.element[i]);
    if (i == Q.back) break;
    if (++i == MAXLEN) i = 0;
  }
}
int main ()
{
  queue Q;
  Q = init(); printf("Current queue : "); print(Q); printf("\n");
  Q = enqueue(Q,'a'); printf("Current queue : "); print(Q); printf("\n");
  Q = enqueue(Q,'b'); printf("Current queue : "); print(Q); printf("\n");
  Q = enqueue(Q,'c'); printf("Current queue : "); print(Q); printf("\n");
  Q = dequeue(Q); printf("Current queue : "); print(Q); printf("\n");
  Q = dequeue(Q); printf("Current queue : "); print(Q); printf("\n");
  Q = enqueue(Q,'c'); printf("Current queue : "); print(Q); printf("\n");
  Q = dequeue(Q); printf("Current queue : "); print(Q); printf("\n");
  Q = dequeue(Q); printf("Current queue : "); print(Q); printf("\n");
  Q = dequeue(Q); printf("Current queue : "); print(Q); printf("\n");
}
```

Output:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 10	Implementation of Binary Trees and operations of Binary Trees

Aim:

To Write a program that Implementation of Binary Trees and operations of Binary Trees.

Binary Tree: A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

Binary Tree Representation: A tree is represented by a pointer to the topmost node of the tree. If the tree is empty, then the value of the root is NULL.

A Tree node contains the following parts.

1. Data
2. Pointer to the left child
3. Pointer to the right child

```
// Structure of each node of the tree

struct node
{
    int data;
    struct node* left;
    struct node* right;
};
```

Basic Operation On Binary Tree:

- Inserting an element.
- Removing an element.
- Searching for an element.
- Traversing an element. There are three types of traversals in a binary tree which will be discussed ahead.

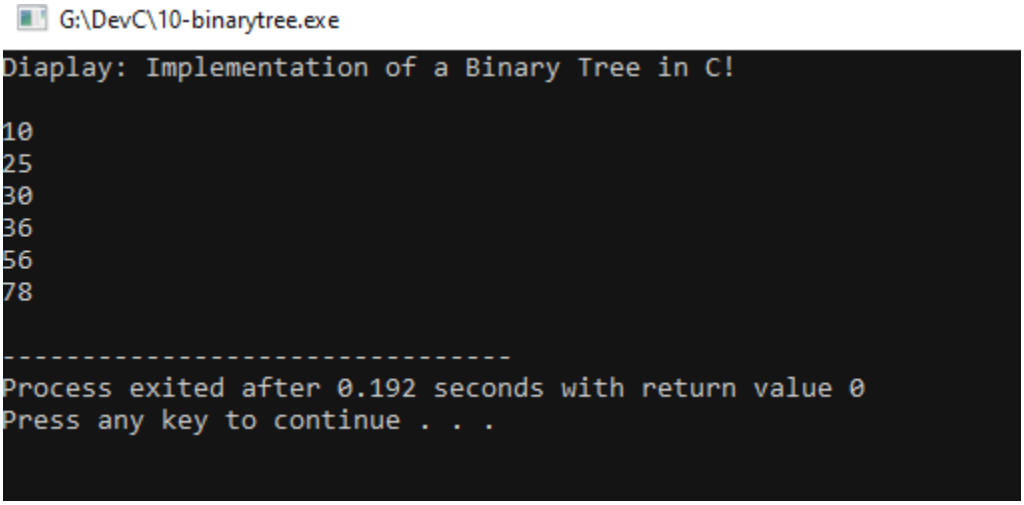
Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int value;
struct node *left_child, *right_child;
};
struct node *new_node(int value)
```

```
{
struct node *tmp = (struct node *)malloc(sizeof(struct node));
tmp->value = value;
tmp->left_child = tmp->right_child = NULL;
return tmp;
}
void print(struct node *root_node) // displaying the nodes!
{
if (root_node != NULL)
{
print(root_node->left_child);
printf("%d \n", root_node->value);
print(root_node->right_child);
}
}
struct node* insert_node(struct node* node, int value) // inserting nodes!
{
if (node == NULL) return new_node(value);
if (value < node->value)
{
node->left_child = insert_node(node->left_child, value);
}
else if (value > node->value)
{
node->right_child = insert_node(node->right_child, value);
}
return node;
}
int main()
{
printf("Diaplay: Implementation of a Binary Tree in C!\n\n");
struct node *root_node = NULL;
root_node = insert_node(root_node, 10);
insert_node(root_node, 10);
insert_node(root_node, 30);
insert_node(root_node, 25);
insert_node(root_node, 36);
insert_node(root_node, 56);
insert_node(root_node, 78);
print(root_node);
```

```
return 0;
}
```

Output:-



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 11	Implementation of Binary Search Trees

Aim:

To Write a program that Implementation of Binary Search Trees.

Algorithm:

If root==Null
return Null;
If number==root->data
return root->data
If number<root->data
return search(root->left)
If number>root->data
return search(root->right)

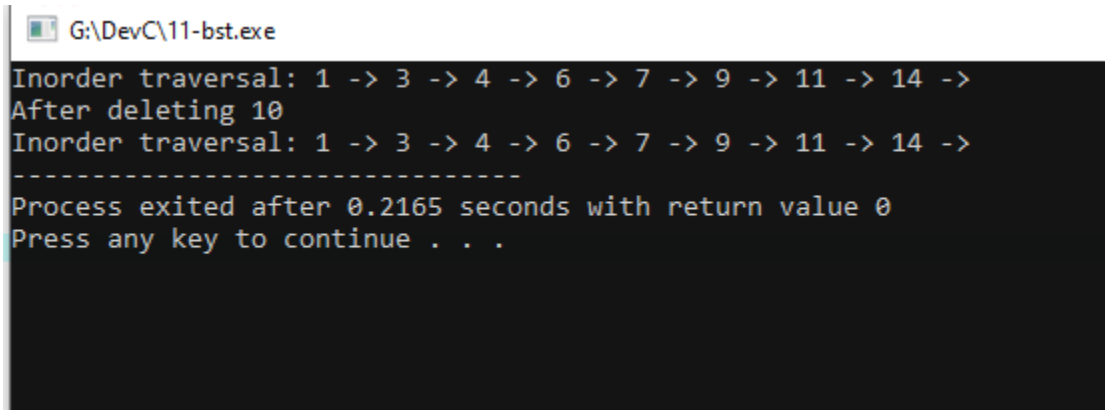
Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int key;
  struct node *left, *right;
};
// Create a node
struct node *newNode(int item) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}
// Inorder Traversal
void inorder(struct node *root) {
  if (root != NULL) {
    // Traverse left
    inorder(root->left);
    // Traverse root
    printf("%d -> ", root->key);
    // Traverse right
    inorder(root->right);
  }
}
```

```
// Insert a node
struct node *insert(struct node *node, int key) {
  // Return a new node if the tree is empty
  if (node == NULL) return newNode(key);
  // Traverse to the right place and insert the node
  if (key < node->key)
    node->left = insert(node->left, key);
  else
    node->right = insert(node->right, key);
  return node;
}
// Find the inorder successor
struct node *minValueNode(struct node *node) {
  struct node *current = node;
  // Find the leftmost leaf
  while (current && current->left != NULL)
    current = current->left;
  return current;
}
// Deleting a node
struct node *deleteNode(struct node *root, int key) {
  // Return if the tree is empty
  if (root == NULL) return root;
  // Find the node to be deleted
  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);
  else {
    // If the node is with only one child or no child
    if (root->left == NULL) {
      struct node *temp = root->right;
      free(root);
      return temp;
    } else if (root->right == NULL) {
      struct node *temp = root->left;
      free(root);
      return temp;
    }
    // If the node has two children
    struct node *temp = minValueNode(root->right);
    // Place the inorder successor in position of the node to be deleted
    root->key = temp->key;
    // Delete the inorder successor
    root->right = deleteNode(root->right, temp->key);
  }
```

```
  return root;
}
//Driver code
int main() {
  struct node *root = NULL;
  root = insert(root, 9);
  root = insert(root, 3);
  root = insert(root, 1);
  root = insert(root, 6);
  root = insert(root, 7);
  root = insert(root, 11);
  root = insert(root, 14);
   root = insert(root, 4);
  printf("Inorder traversal: ");
  inorder(root);
  printf("\nAfter deleting 10\n");
  root = deleteNode(root, 10);
  printf("Inorder traversal: ");
  inorder(root);
}
```

Output:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 12 DATE:	Implementation of searching techniques

Aim:

To Write a program that Implementation of searching techniques.

Linear Search Algorithm

Algorithm:

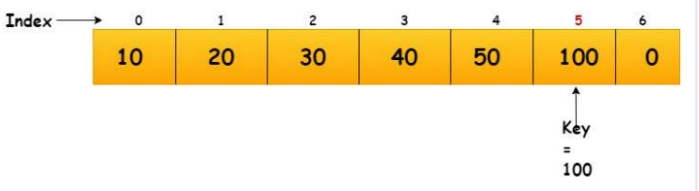
Linear_Search (Array X, Value i)
Set j to 1
If j > n, jump to step 7
If X[j] == i, jump to step 6
Then, increment j by 1 i.e. j = j+1
Go back to step 2
Display the element i which is found at particular index i, then jump to step 8
Display element not found in the set of input elements.
Exit/End

Procedure:

```
procedure LINEAR_SEARCH (array, key)

  for each item in the array
    if match element == key
      return element's index
    end if
  end for

end procedure
```

Implementation of Linear Search in C

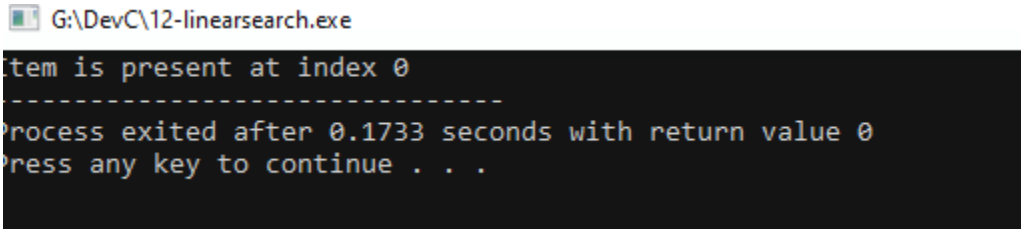
- Initially, we need to mention or accept the element to be searched from the user.
- Then, we create a for loop and start searching for the element in a sequential fashion.
- As soon as the compiler encounters a match i.e. array[element] == key value, return the element along with its position in the array.
- If no values are found that match the input, it returns -1.



Program:

```
#include <stdio.h>
int LINEAR_SEARCH(int inp_arr[], int size, int val)
{
       int i;
      for (i = 0; i < size; i++)
            if (inp_arr[i] == val)
                  return i;
      return -1;
}
int main(void)
{
      int arr[] = { 1000, 2000, 3000, 4000, 5000, 100, 0 };
      int key = 1000;
      int size = 10;
      int res = LINEAR_SEARCH(arr, size, key);
      if (res == -1)
      printf("ELEMENT NOT FOUND!!");
   else
   printf("Item is present at index %d", res);

      return 0;
}
```

Output:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 13	Implementation of Sorting algorithms : Insertion Sort

Write a program to explain insertion sort .

Aim:

To Write a program that Implementation of Sorting algorithms : Insertion Sort.

Algorithm:

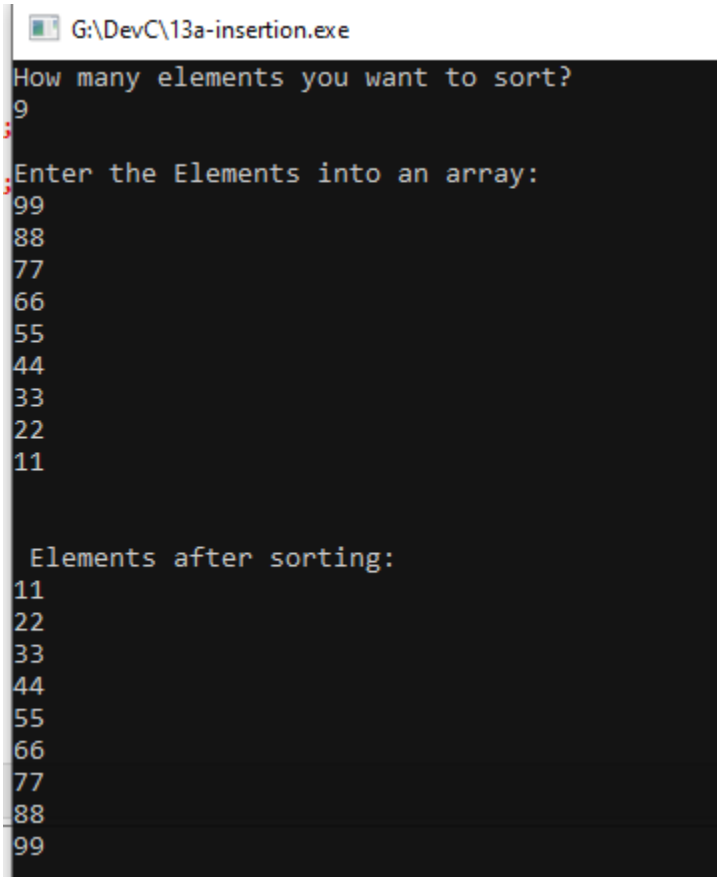
Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted

Program:

```
/*Program to sort elements of an array using insertion sort method*/
#include<stdio.h>
#include<stdlib.h>
void main( )
{
int a[10],i,j,k,n;
clrscr( );
printf("How many elements you want to sort?\n");
scanf("%d",&n);
printf("\nEnter the Elements into an array:\n");
for (i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=1;i<n;i++)
{
 k=a[i];
for(j= i-1; j>=0 && k<a[j];j--)
a[j+1]=a[j];
a[j+1]=k;
}
printf("\n\n Elements after sorting: \n");
for(i=0;i<n;i++)
```

```
printf("%d\n", a[i]);
getch( );
}
```

OUTPUT:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 13(B)	Implementation of Sorting algorithms : Quick Sort

Aim:

To Write a program that Implementation of Sorting algorithms : Quick Sort.

Algorithm:

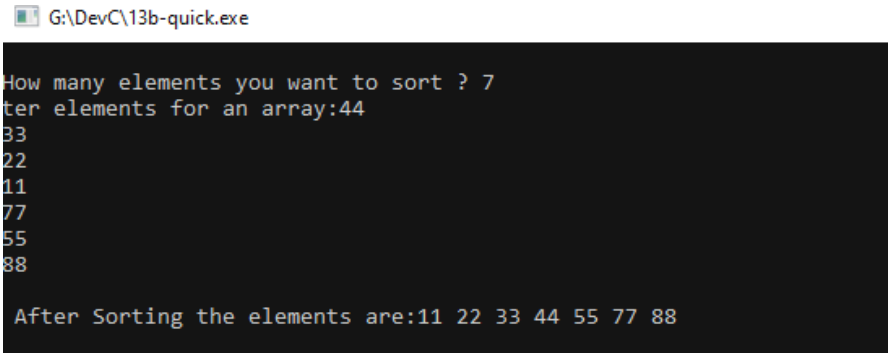
```
/* low –> Starting index, high –> Ending index */
quickSort(arr[], low, high) {
   if (low < high) {
      /* pi is partitioning index, arr[pi] is now at right place */
      pi = partition(arr, low, high);
      quickSort(arr, low, pi – 1); // Before pi
      quickSort(arr, pi + 1, high); // After pi
   }}
```

Program:

```
/* program to sort elements of an array using Quick Sort */
#include<stdio.h>
void quicksort(int[ ],int,int);
void main( )
{
int low, high, pivot, t, n, i, j, a[10];
clrscr( );
printf("\nHow many elements you want to sort ? ");
 scanf("%d",&n);
printf("\Enter elements for an array:");
for(i=0; i<n;i++)
scanf("%d",&a[i]);
low=0;
high=n-1;
quicksort(a,low,high);
printf("\After Sorting the elements are:");
for(i=0;i<n;i++)
printf("%d ",a[i]);
getch( );
}
 void quicksort(int a[ ],int low,int high)
{
```

```
int pivot,t,i,j;
if(low<high)
{
pivot=a[low];
i=low+1;
j=high;
while(1)
{
while(pivot>a[i]&&i<=high)
i++;
 while(pivot<a[j]&&j>=low)
j--;
if(i<j)
{
 t=a[i];
a[i]=a[j];
a[j]=t;
}
else
break;
 }
a[low]=a[j];
a[j]=pivot;
quicksort(a,low,j-1);
quicksort(a,j+1,high);
}
}
```

OUTPUT:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 13(C)	Implementation of Sorting algorithms : Merge Sort

Aim:

To Write a program that Implementation of Sorting algorithms : Merge Sort.

Algorithm:

step 1: start

step 2: declare array and left, right, mid variable

step 3: perform merge function.

```
if left > right
   return
mid= (left+right)/2
mergesort(array, left, mid)
mergesort(array, mid+1, right)
merge(array, left, mid, right)
```

step 4: Stop

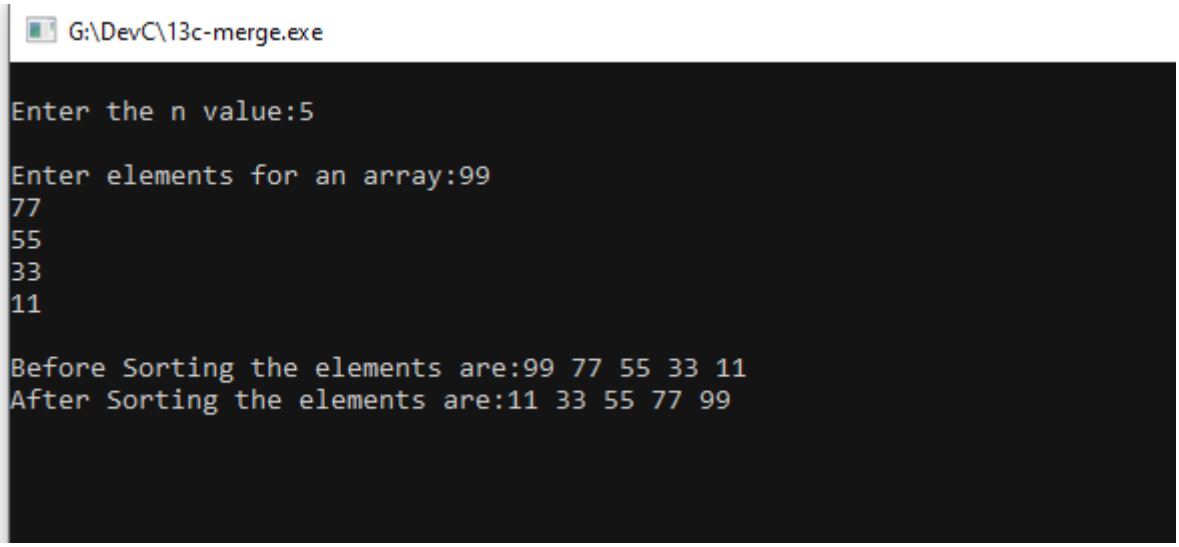
Program:

```
/* program to sort elements of an array using Merge Sort */
#include <stdio.h>
void disp( );
void mergesort(int,int,int);
void msortdiv(int,int);
int a[50],n;
void main( )
{
int i;
clrscr( );
printf("\nEnter the n value:");
scanf("%d",&n);
printf("\nEnter elements for an array:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\nBefore Sorting the elements are:");
```

```
disp( );
msortdiv(0,n-1);
printf("\nAfter Sorting the elements are:");
disp( );
getch( );
}
void disp( )
{
int i;
for(i=0;i<n;i++)
printf("%d ",a[i]);
}
void mergesort(int low,int mid,int high)
{
 int t[50],i,j,k;
i=low;
j=mid+1;
k=low;
while((i<=mid) && (j<=high))
{
 if(a[i]>=a[j])
t[k++]=a[j++];
else
t[k++]=a[i++];
}
while(i<=mid)
t[k++]=a[i++];
while(j<=high)
t[k++]=a[j++];
for(i=low;i<=high;i++)
a[i]=t[i];
}
void msortdiv(int low,int high)
{
int mid;
```

```
if(low!=high)
{
mid=((low+high)/2);
msortdiv(low,mid);
msortdiv(mid+1,high);
mergesort(low,mid,high);
 }
 }
```

OUTPUT:



RESULT:

Thus the C programs can be executed and the output was verified successfully.

EX.NO: 14	Implementation of Hashing – any two collision techniques

Aim:

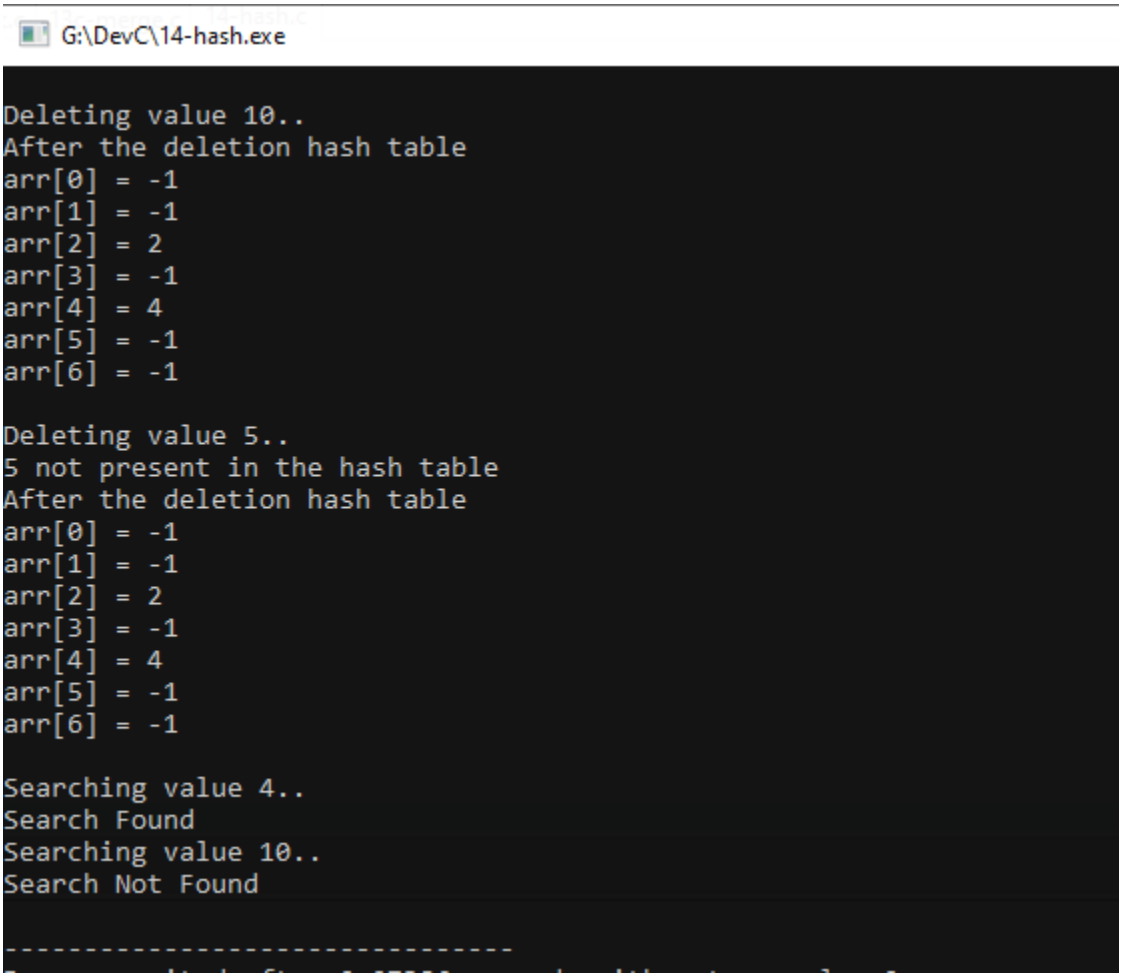
To Write a program that Implementation of Hashing – any two collision techniques.

Program:

```
#include<stdio.h>
#define size 7
int arr[size];
void init()
{
   int i;
   for(i = 0; i < size; i++)
      arr[i] = -1;
}

void insert(int value)
{
   int key = value % size;

   if(arr[key] == -1)
   {
      arr[key] = value;
      printf("%d inserted at arr[%d]\n", value,key);
   }
   else
   {
      printf("Collision : arr[%d] has element %d already!\n",key,arr[key]);
      printf("Unable to insert %d\n",value);
   }
}

void del(int value)
{
   int key = value % size;
   if(arr[key] == value)
      arr[key] = -1;
   else
      printf("%d not present in the hash table\n",value);
}
```

```
void search(int value)
{
   int key = value % size;
   if(arr[key] == value)
      printf("Search Found\n");
   else
      printf("Search Not Found\n");
}

void print()
{
   int i;
   for(i = 0; i < size; i++)
      printf("arr[%d] = %d\n",i,arr[i]);
}

int main()
{
   init();
   insert(10); //key = 10 % 7 ==> 3
   insert(4); //key = 4 % 7  ==> 4
   insert(2); //key = 2 % 7  ==> 2
   insert(3);  //key = 3 % 7  ==> 3 (collision)

   printf("Hash table\n");
   print();
   printf("\n");

   printf("Deleting value 10..\n");
   del(10);
   printf("After the deletion hash table\n");
   print();
   printf("\n");

   printf("Deleting value 5..\n");
   del(5);
   printf("After the deletion hash table\n");
   print();
   printf("\n");

   printf("Searching value 4..\n");
   search(4);
   printf("Searching value 10..\n");
   search(10);
   return 0;
}
```

Output:



RESULT:

Thus the C programs can be executed and the output was verified successfully.