DSA LAB RECORD

T.Durga Prasad

AP19110010212

CSE-G

11.))Objectives:In this activity we will use different types of data types and increment operators and use loops.

Problem Statement: We aim to understand that by using different types of data types and increment operators and for loops and data types and increment operators of C language by using them we will find solution to the program.

Algorithm:

1. Take n a variable that stores no. Of elements

2. Create an array of size n

3. Iterate via for loop to take array elements as input.

4. For loop to access each element of array to get sum.

5. The avg is caluculated by dividing the overall sum to the no.of elements

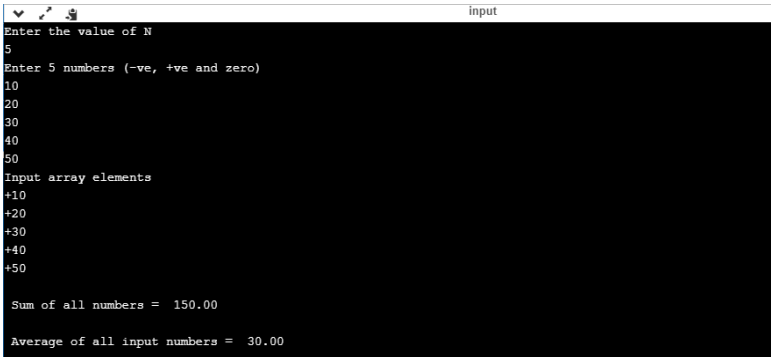
6. Sum andAvg are printed.

INput:

```
#include<stdio.h>

void main()

{
```

```
    int i, num;
    float total = 0.0, average;
    printf ("Enter the value of N \n");
    scanf("%d", &num);
    int array[num];
    printf("Enter %d numbers (-ve, +ve and zero) \n", num);
    for (i = 0; i < num; i++)
    {
        scanf("%d", &array[i]);
    }
    printf("Input array elements \n");
    for (i = 0; i < num; i++)
    {
        printf("%+3d\n", array[i]);
    }
    for (i = 0; i < num; i++)
    {
        total+=array[i];
    }
    average = total / num;
    printf("\n Sum of all numbers =    %.2f\n", total);
    printf("\n Average of all input numbers =    %.2f\n", average);
}
```

OUTPUT:



Conclusion:By above code we will conclude that by using different data types we can find sum and average of elements in an array.

12.))Objectives:We need to write a C program such that program will read a one-dimensional array and find out the largest element present in the array and find out the largest element present in an array.

Problem statement:In this program we need to create an array and enter the elements in it and find the largest number in that array.

Algorithm:

1. Create an array

2. Run the for loop till the user-defined size to insert the element at each location.

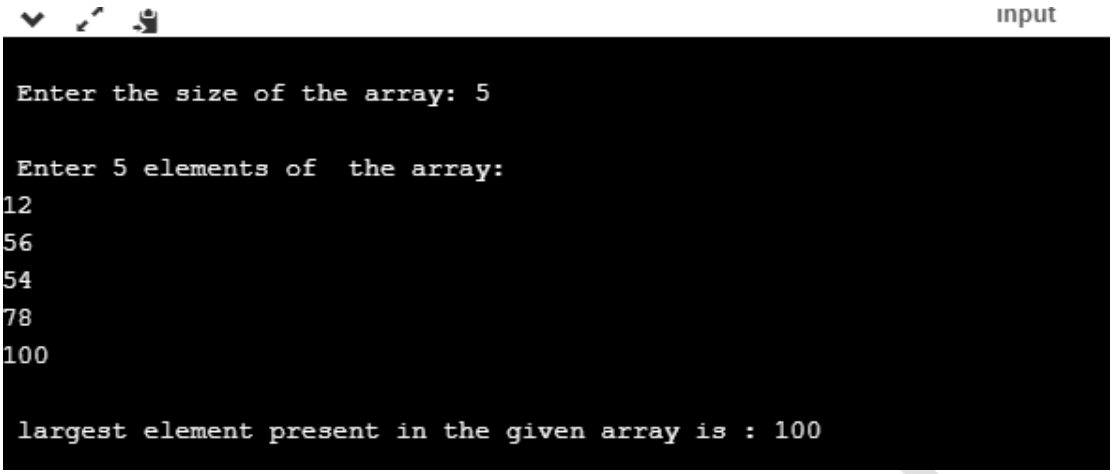
3. Consider the 1st element of array to be largest compare all the remaining elements of array and change the largest value if assumed largest element is smaller than the element being compared.

4. At last,the largest element will hold the actual largest value in the array.Then print it.

Input:

```
#include <stdio.h>
int main()
{
        int n, i, largest;
        printf("\n Enter the size of the array: ");
        scanf("%d", &n);
        int array[n];
        printf("\n Enter %d elements of  the array: \n", n);
        for (i = 0; i < n; i++)
        {
            scanf("%d", &array[i]);
        }
        largest = array[0];
        for (i = 1; i < n; i++)
        {
            if (largest < array[i])
            largest = array[i];
        }
        printf("\n largest element present in the given array is :
 %d", largest);
        return 0;
}
```

Output:



Conclusion:By above code we can conclude that by using array and data types and arthimetic operators and increment operators and some other variables we can find the largest number in an array.

13.))Objectives:We have to write a C program which finds the position of an element in the array as input from the user and then find the position of that element in array by using linear search algorithm.

Problem statement:To write program for linear search we will define an array of size n to store numbers and we will define a variable and we use data types ,arithmetic operators,increment operators,and we will use for loops.

Algorithm:

1. We will create an array of numbers by taking input from user.We will also read the element to be searched by the user.

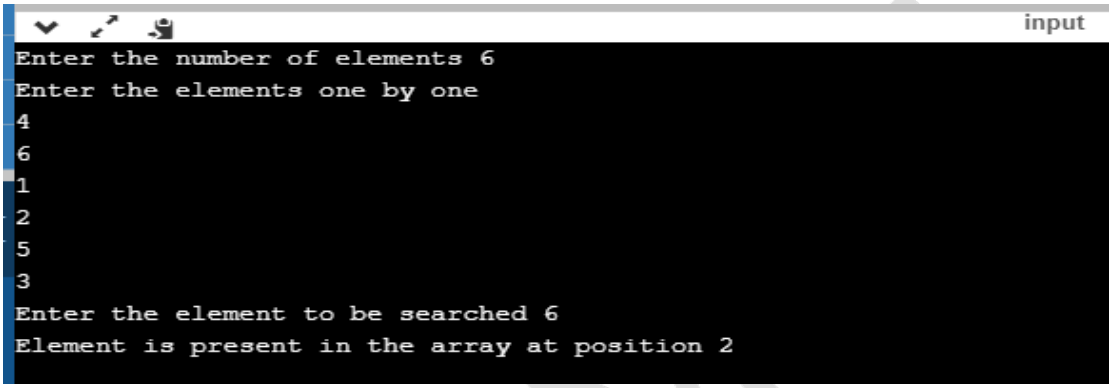
2. In order to look for that element in the array we will sequentially in increasing index values.If we encounter the element requested by the user we will return the position of that element in array,but if it is not there we will return -1 which indicates the absence of element which was searched.

Input:

```
#include <stdio.h>
void main()
{    int n;
      int i,    x, found = 0;
      printf("Enter the number of elements ");
      scanf("%d", &n);
      int array[n];
      printf("Enter the elements one by one \n");
      for (i = 0; i <n; i++)
      {
            scanf("%d", &array[i]);
      }
      printf("Enter the element to be searched ");
      scanf("%d", &x);
      for (i = 0; i < n ; i++)
      {
            if (x == array[i] )
            {
                  found = 1;
                  break;
            }
      }
```

```
    if (found == 1)
        printf("Element is present in the array at position %d",i+1);
    else
        printf("Element is not present in the array\n");
}
```

Output:



Conclusion:By using data types and arithmetic and increment operators and for loops and we can search an element in an array using linear search algorithm.

14.))Objectives:In this program we need to sort the given elements in ascending order I.e low to high using bubble sort technique.

Problem statement:Bubble sort is a simple method that sorts the elements of an array into either increasing or decreasing order.It works by comparing the adjacent elements and swapping them if they are out of order.Multiple passes through the array are necessary.

Algorithm:

1. Create an array of max size.

2. We need to enter elements using for loop.

3. Using two for loops we will write the main part of the bubble

sort.

4. If arr [1]>arr [2] we will swap.

5. We will define a temp variable.

6. After doing this process repeatedly.

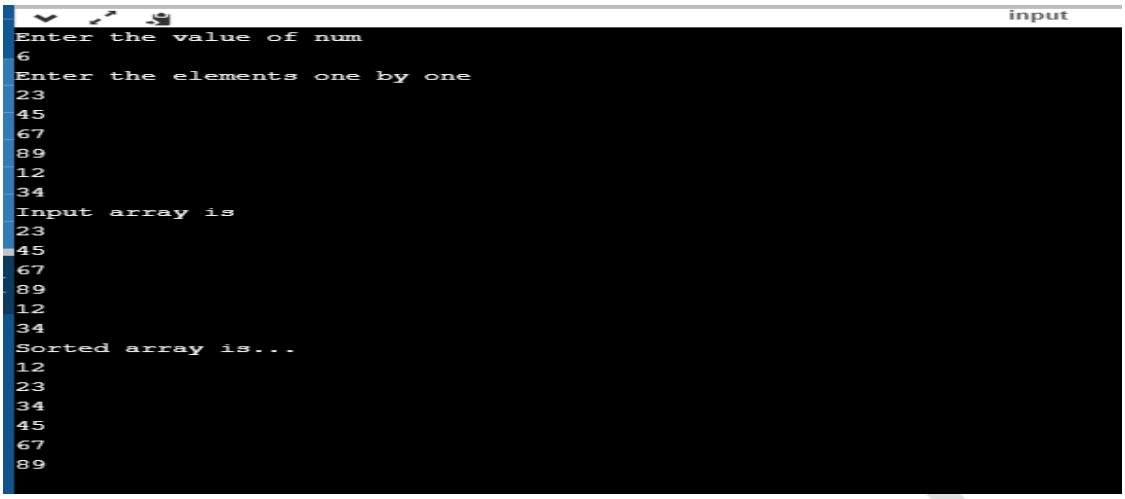
7. Finally we will print the sorted array.

Input:

```
#include <stdio.h>
#define MAXSIZE 10
void main()
{
	int array[MAXSIZE];
	int i, j, num, temp;
	printf("Enter the value of num \n");
	scanf("%d", &num);
	printf("Enter the elements one by one \n");
	for (i = 0; i < num; i++)
	{
		scanf("%d", &array[i]);
	}
	printf("Input array is \n");
	for (i = 0; i < num; i++)
	{
		printf("%d\n", array[i]);
```

```
    }
    for (i = 0; i < num; i++)
    {
        for (j = 0; j < (num - i - 1); j++)
        {
            if (array[j] > array[j + 1])
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
    printf("Sorted array is...\n");
    for (i = 0; i < num; i++)
    {
        printf("%d\n", array[i]);
    }
}
```

Output:-



Conclusion:By above code we can print all the elements in ascending order using bubble sort.

15.))Objectives:In this program we will sort elements in ascending order using Insertion sort.

Problem Statement:Using for loop we are reading n elements from input next we are comparing elements of the array so that we can insert them in the proper position at the end we display it.

Algorithm:

1. WE will create an array.

2. We will traverse this array and insert each element to its correct position where it should actually be by shifting the other.

3. 1st element in array is considered as sorted even if it is unsorted array.The array is divided into two parts the 1st part holds the 1st element which is sorted and 2nd part contains remaining elements.

4. With each iteration one element from the second part is picked and inserted into 1st part of the array.

5. This goes until the last element in the 2nd part of array is placed in correct position.

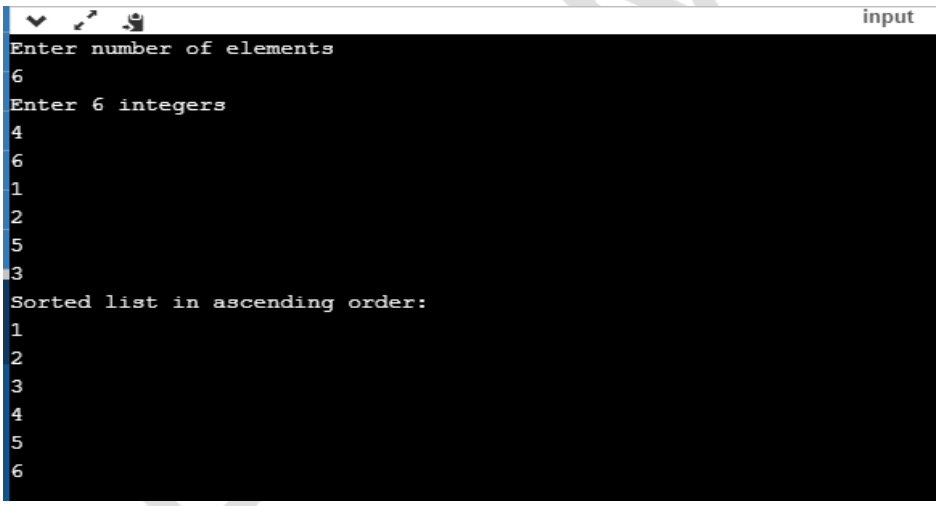
6. Now,we have array in sorted order.

Input:-

```
#include <stdio.h>
int main()
{
	int n, i, j, temp;
	int arr[64];
	printf("Enter number of elements\n");
	scanf("%d", &n);
	printf("Enter %d integers\n", n);
	for (i = 0; i < n; i++)
	{
		scanf("%d", &arr[i]);
	}
	for (i = 1 ; i <= n - 1; i++)
	{
	  j = i;
		while ( j > 0 && arr[j-1] > arr[j])
		{
			temp	= arr[j];
			arr[j]	= arr[j-1];
			arr[j-1] = temp;
			j--;
```

```
                }
        }
        printf("Sorted list in ascending order:\n");
        for (i = 0; i <= n - 1; i++)
        {
                printf("%d\n", arr[i]);
        }
        return 0;
}
```

Output:-



Conclusion:-By above program we can say that we print elements in ascending order using insertion sort and operators and for loops.

18.))Objectives:In this program we need to perform binary search I.e to search an element in the array.

Problem statement:-Search a sorted array by repeatedly dividing the search interval covering the whole array.If the the value of search key is less than the item in middle narrow the interval to the lower half.Otherwise narrow it to upper half.

Algorithm:-

1. Create an array of size n and enter elements into it.

2. Compare x with middle element.

3. If x matches with middle element we return mid index.

4. Else if x is grater than the middle element x will lie in the right half sub-array after mid element we recur for right half.

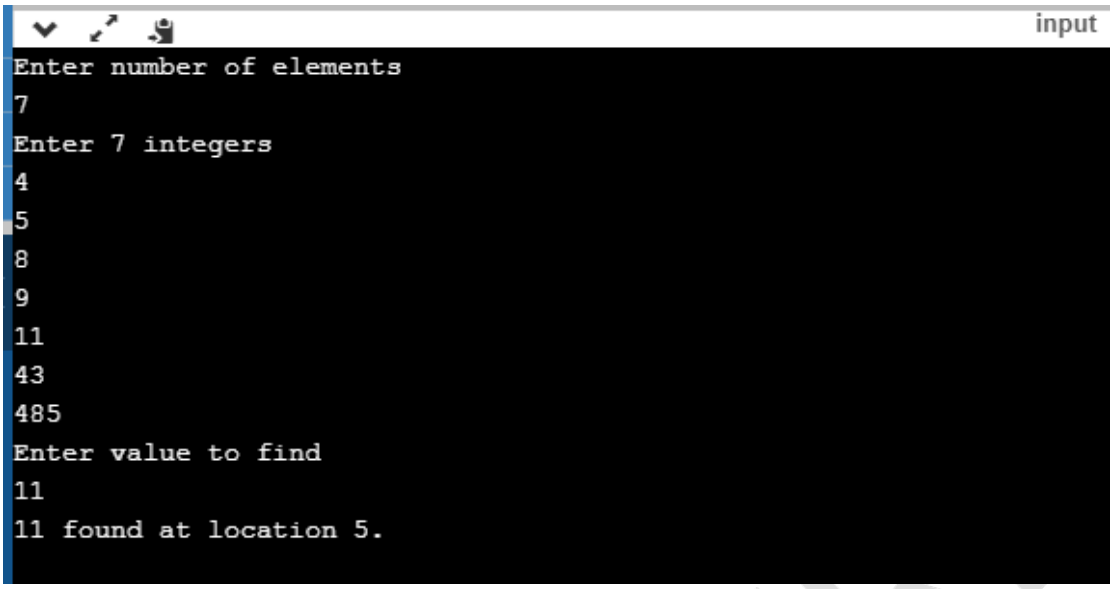
5. Else recur for left half.

Input:-

```
#include <stdio.h>

int main()

{

   int c, f, l, mid, n, x, a[100];

   printf("Enter number of elements\n");

   scanf("%d", &n);

   printf("Enter %d integers\n", n);

   for (c = 0; c < n; c++)

      scanf("%d", &a[c]);

   printf("Enter value to find\n");

   scanf("%d", &x);

   f = 0;

   l = n - 1;
```

```
    mid = (f+l)/2;
    while (f <= l) {
        if (a[mid] < x)
            f = mid + 1;
        else if (a[mid] == x) {
            printf("%d found at location %d.\n", x, mid+1);
            break;
        }
        else
            l = mid - 1;

        mid = (f + l)/2;
    }
    if (f > l)
        printf("Not found! %d isn't present in the list.\n", x);
    return 0;
}
```

Output:-



Conclusion:-In this program we will use different types of arithmetic operators and data types and increment operators and we will find the x element in the given elements using binary search.

22.))Objectives:In this program we need to find the sum of natural numbers using functions.

Problem statement:-We will write this program for finding sum of natural numbers using functions and data types and arithmetic operators and increment operators and finally we will print the sum of numbers.

Algorithm:-

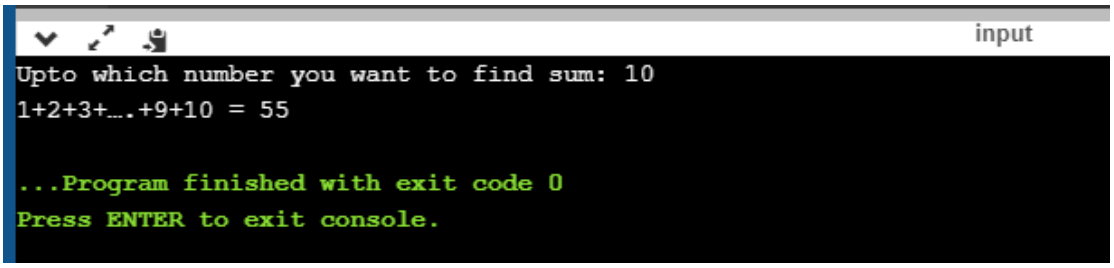
1. First we will give function name using int.
2. We define a variable n and add and initialize it as 0.
3. We will use for loop and add numbers.

4. Using return statement we will get output.

Input:-

```
#include<stdio.h>
int sum(int n)
{
    int add = 0;
    for(int i=1; i<=n; i++)
    {
       add += i;
    }
    return add;
}
int main()
{
    int range, result;
    printf("Upto which number you want to find sum: ");
    scanf("%d", &range);
    result = sum(range);
    printf("1+2+3+….+%d+%d = %d",range-1, range, result);
}
```

Output:-



Conclusion:-In this program we defined a function sum() which takes one argument.Using for loop,the function sum() finds the sum of series later this value is returned back to the caller function.

23.))Objectives:-In this program we will find factorial of a number using recursion method.

Problem statement:-A factorial is a product of all the number from 1 to the user specified number.Recursion method means if a function is called by itself the technique can be called as recursion.

Algorithm:-

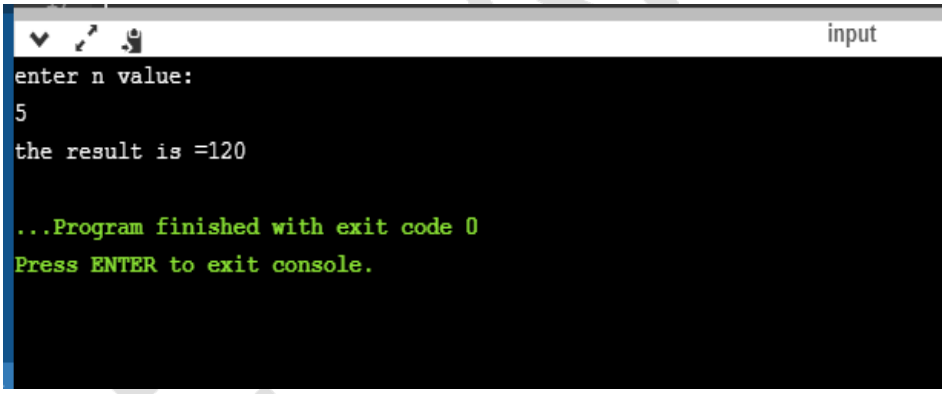
1. We will define variables n,k
2. We will define function name as factorial.
3. We will give if statement that if(n==1) using return we will return 1.
4. Using else statement we will n*factorial(n-1)
5. We will print the output.

Input:-

```
#include<stdio.h>
int main()
{
   int n,k;
   int factorial(int);
```

```
    printf("enter n value:\n");
    scanf("%d",&n);
    k=factorial(n);
    printf("the result is =%d",k);
}
int factorial(int n)
{
      if(n==1)
         return 1;
      else
         return(n*factorial(n-1));
}
```

Output:-



Conclusion:-In this program we will use return statement and recursion method and arithmetic operators and data types and we will print the output of the program.

24.))Objectives:In this program we will print print he fibonacci series using return statement I.e reursion method

Problem statement:It adds previous two numbers value to

compute the next number value. In this program fibonacci series is calculated using recursion, with seed as 0 and 1. Recursion means a function calling itself, in the below code fibonacci function calls itself with a lesser value several times. An termination condition is very important to recursion function, i.e n == 0 and n == 1 or the recursive call would be infinite leading to stack overflow error.

Algorithm:-

1. We will ask till which number we need to print fibonacci series.
2. WE will writee for loop.
3. We will if(n==0) then return 0
4. Next we will write else if(n==1) then return 1.
5. Next we will write else then return fib(n-1)+fib(n-2).

Input:-

```
#include<stdio.h>
int fibonacci(int);
int main()
{
int n, m= 0, i;
printf("Enter Total terms:");
scanf("%d", &n);
printf("Fibonacci series terms are:\n");
for(i = 1; i <= n; i++)
{
printf("%d\n", fibonacci(m));
m++;
```

```
}
return 0;
}
int fibonacci(int n)
{
if(n == 0 || n == 1)
return n;
else
return(fibonacci(n-1) + fibonacci(n-2));
}
```

Output:-

Conclusion:-In this program, we take the end term from the user. We must display a Fibonacci series up to that number. This is done by using a while loop. We take input from the user which is the last term. Then print the first and second terms. After this, add first and second and store it in sum. Then, there is a while loop. It runs till the value of the sum is less than that of the number entered by the user. Inside the while loop, Print out the sum first.

25.))Objectives:-In this program we will write program using structures.A structure is a user defined data type,unlike arrays a structure variable can hold similar types of data elements,whereas structure can hold different types of elements.

Problem Statement:-Then, we created an array of structures s having 5 elements to store information of 5 students.Using a for loop, the program takes the information of 5 students from the user and stores it in the array of structure. Then using another for loop, the information entered by the user is displayed on the screen.

Algorithm:-

1. First we will define a structure named student.

2. And we will enter three members in to the structure:roll no,marks,name.

3. And then we will ask user for information we will write for loop to enter information.

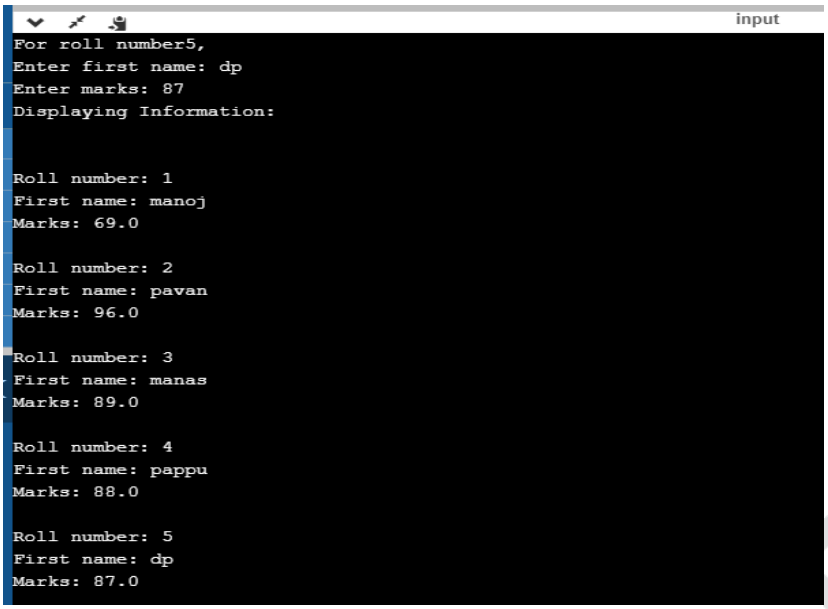
4. And then to display information we will write another for loop to display the entered information 5 students.

Input:-

```
#include <stdio.h>

struct student {

    char firstName[50];

    int roll;

    float marks;

} s[10];

int main() {
```

```
int i;
printf("Enter information of students:\n");
for (i = 0; i < 5; ++i) {
    s[i].roll = i + 1;
    printf("\nFor roll number%d,\n", s[i].roll);
    printf("Enter first name: ");
    scanf("%s", s[i].firstName);
    printf("Enter marks: ");
    scanf("%f", &s[i].marks);
}
printf("Displaying Information:\n\n");
for (i = 0; i < 5; ++i) {
    printf("\nRoll number: %d\n", i + 1);
    printf("First name: ");
    puts(s[i].firstName);
    printf("Marks: %.1f", s[i].marks);
    printf("\n");
}
return 0;
}
```

Output:-



Conclusion:-In this program we will display or print the information 5 students using structures,data types,arithmetic operators,increment operators,for loops.

26.))Objectives:In this program we will print the length of the string using pointers.A pointer is also a variable which is used to store the address of another variable.Pointers are very much helpful in accessing the data indirectly.

Problem Statement:In this program first we will enter string and then to find the length of the string we will define a variable count and and then we will write while loop to count the length of the string.

Algorithm:-

1. gets() is used to accept string with spaces.

2. We are passed accepted string to the function.

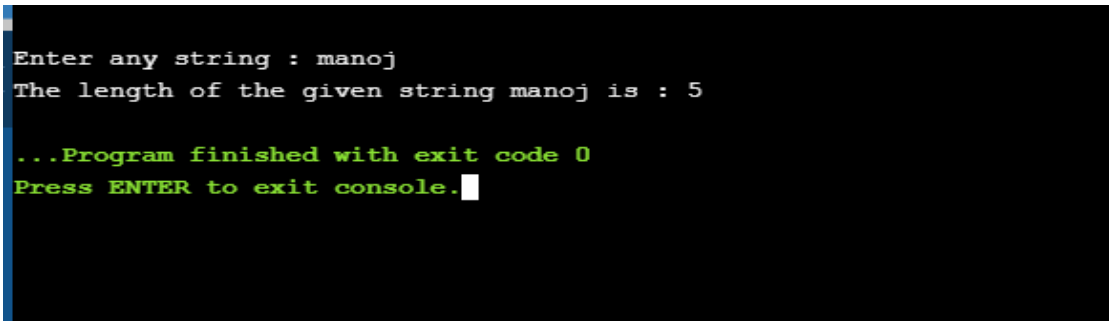
3. Inside function we have stored string in this pointer I.e string is stored in base of pointer variable.

4. Inside while loop we are going to count the single letter and continue this until we get null character.

Input:-

```
#include<stdio.h>
#include<conio.h>
int string_ln(char*);
int main() {
    char str[20];
    int length;
    printf("\nEnter any string : ");
    gets(str);
    length = string_ln(str);
    printf("The length of the given string %s is : %d", str, length);
    getch();
}
int string_ln(char*p) /* p=&str[0] */
{
    int count = 0;
    while (*p != '\0') {
        count++;
        p++;
    }
    return count;
}
```

Output:-



Conclusion:In this program we use pointers,while loop,Arithmetic operators,etc.to print the length of the string using pointers.

27.))Objectives:-In this program we will enter one string and we need to copy that string and print that string we need to do all these process using pointers.

Problem Statement:-The program implements the Copy string to another string. The inner function “copy_string” takes 2 string pointers as arguments. “copystr” pointer type function declared and pointer type char variable declare.A pointer is also a variable which is used to store the address of another variable.Pointers are very much helpful in accessing the data indirectly.

Algorithm:-

1. We will define function name Copy-string.
2. Next we will define two arrays str1,str2 to enter strings
3. Gets to store the string.
4. We will write while loop to copy things in string 1 to string 2.

Input:-

```
#include<stdio.h>

void copy_string(char*, char*);

int main()
```

```
{
    char str1[100], str2[100];
    printf("Enter string1\n");
    gets(str1);
    copy_string(str2, str1);
    printf(" string2 is \"%s\"\n", str2);
    return 0;
}
void copy_string(char *str2, char *str1)
{
    while(*str1)
    {
        *str2 = *str1;
        str1++;
        str2++;
    }
    *str2 = '\0';
}
```

Output:-

```
Enter string1
Pappu dp
 string2 is "Pappu dp"


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:-In this program we will use arithmetic operators and while loop to copy the things from string1 to string2 using pointers and finally we will print the copied string.

28.))Objectives:-In this program we will compare two strings usings pointers,arithmetic operators,while loop,functions to compare two strings and to say that they are lexographically equal or not.

Problem Statement:-In this program we will do while and if in a single condition.We know in C program Null is represented using 0.Hence null condition in program should be stripped out.

Algorithm:-

1.Input two strings from user. Store it in some variable say str1 and str2.

2.Compare two strings character by character till an unmatched character is found or end of any string is reached.

3.If an unmatched character is found then strings are not equal.

4.Else if both strings reached their end then both strings are equal.

Input:-

```
#include <stdio.h>

#define MAX_SIZE 100

int compare(char * str1, char * str2);

int main()

{
```

```
    char str1[MAX_SIZE], str2[MAX_SIZE];
    int res;
    printf("Enter first string: ");
    gets(str1);
    printf("Enter second string: ");
    gets(str2);
    res = compare(str1, str2);
    if(res == 0)
    {
        printf("Both strings are equal.");
    }
    else if(res < 0)
    {
        printf("First string is lexicographically smaller than
second.");
    }
    else
    {
        printf("First string is lexicographically greater than
second.");
    }
    return 0;
}
int compare(char * str1, char * str2)
```

```
{
    while((*str1 && *str2) && (*str1 == *str2)) { str1++; str2++; }
    return *str1 - *str2;
}
```

Output:-

```
Enter first string: pappu
Enter second string: dp
First string is lexicographically greater than second.

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:-In this program we learned how to compare two strings using pointers and arithmetic operators,while loop and functions.

29.))Objectives:-In this program we will reverse a string both recursively and non-recursively.

Problem statement:non-recursively means using count and begin and while and for loops we will reverse a string.Recursion means If a function is called by itself the technique can be called recursion.To apply the recursion the user has to know when the recursion should be stopped.

Algorithm:-

1.We find the length of the string without using strlen function and then copy its characters in reverse order (from end to beginning) to a new string using a for loop.

2.In the recursive method, we swap characters at the beginning and the end of the string and move towards the middle of the string. This method is inefficient due to repeated function calls.

Input:-

C program to reverse a string a string non-recursively:-

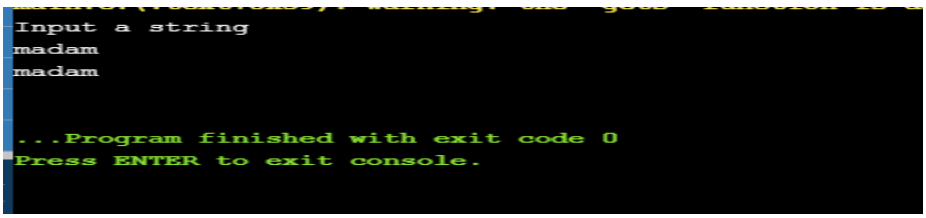
```
#include <stdio.h>
int main()
{
    char s[1000], r[1000];
    int begin, end, count = 0;
    printf("Input a string\n");
    gets(s);
    while (s[count] != '\0')
        count++;
    end = count - 1;
    for (begin = 0; begin < count; begin++) {
        r[begin] = s[end];
        end--;
    }
    r[begin] = '\0';
    printf("%s\n", r);
    return 0;
}
```

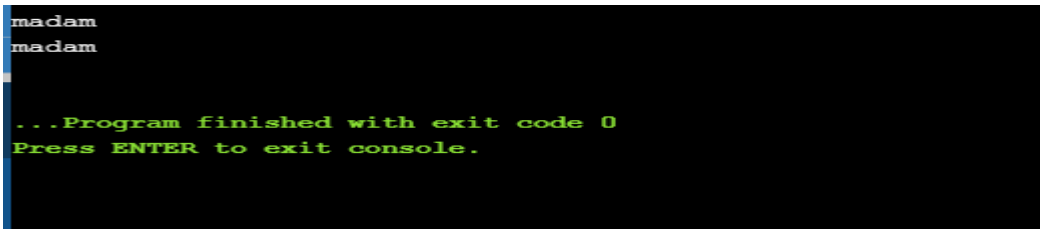
C program to reverse string recursively

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

```
void reverse(char*, int, int);
int main()
{
    char a[100];
    gets(a);
    reverse(a, 0, strlen(a)-1);
    printf("%s\n", a);
    return 0;
}
void reverse(char *x, int begin, int end)
{
    char c;
    if (begin >= end)
        return;
    c               = *(x+begin);
    *(x+begin) = *(x+end);
    *(x+end)     = c;
    reverse(x, ++begin, --end);
}
```

Output:-





Conclusion:-The above programs lets us know how to successfully reverse a string by recursively and non-recursively.

19.))Objective:-

At the end of the activity, we shall be able to

- Use for loops and branching statements.
- Use arrays in C - Program.

Problem Statement:-

In this program we aim to understand and use arrays and for loops to return an array with unique elements. It is required to get the input from the admin such as :

Number of elements :

Elements that you want to search for the unique elements. Once, these are collected, we print unique elements.

Algorithm:-

START

DEFINE VARIABLES: n, count, sum=0

INPUT: Reads the input from the user.

COMPUTATION: loops the code until the condition is satisfied.

DISPLAY: Prints the unique elements.

STOP.

Input:-

```
#include<stdio.h>
Void main ()
{
Int n, sum = 0; count;
printf(“Enter number of elements in an array :”);
scanf (“%d”,&n);
Int a [n],f[n];
printf(“Enter the elements in the array :|);
for (int i=0; i<n; i++)
{
Scan f (“%d”,&a[i]);
f[i]= -1;
}
for (int i =0; i<n; i ++)
{
count = 1;
for (int j=j+1; j<n;j++)
{
if (a[i[= a [j])
```

```
{
Count ++;
f[j]=0;
        }
}
if (f[i]=0)
{
f[i]= count;
      }
}
print f (“Unique elements are :|n”);
For (int i = 0 ; i<n; i++)
{
if (f[i] == 1)
{
printf (“%d”,a[i] );
          }
        }
}
```

Conclusion:-

Simulation of this program, helped me to understand the use of loops and branching statements in C programs.

20. Objective:-

At the end of this activity, we shall be able to

- Use for loops
- Use 2- dimensional arrays in c - program.

Problem statement:-

In this problem we aim to understand and we use 2 D arrays and for loops to get the addition of 2 matrices. It is required to get the input from the admin such as:-

Elements in the first matrix:

Elements in the second matrix:

Once these are collected, we print the output of an additional matrix.

Algorithm:-

START

DEFINE VARIABLES : A=5,B=5,C=5,D=5

INPUT: Reads inputs from the user.

COMPUTATION: Adds 2 matrices elements.

DISPLAY: Prints the resultant matrix.

STOP

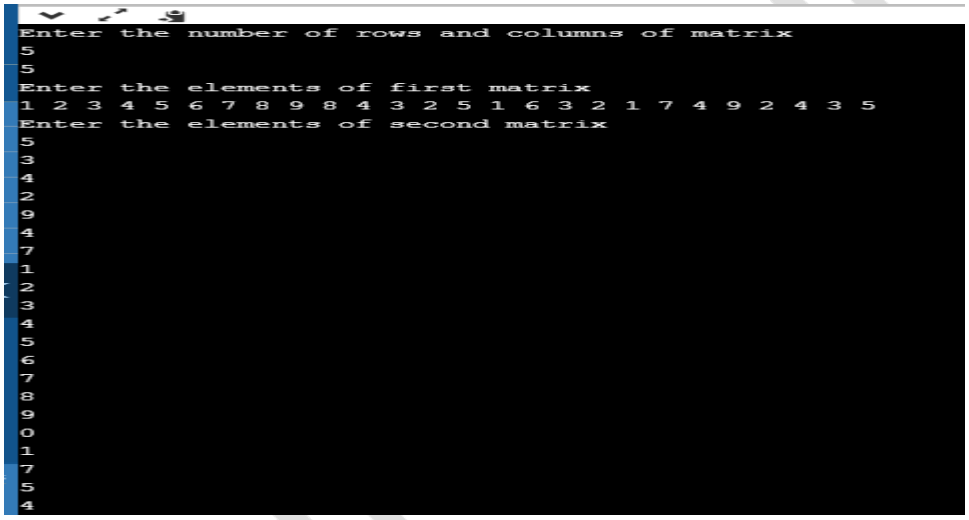
Input:-

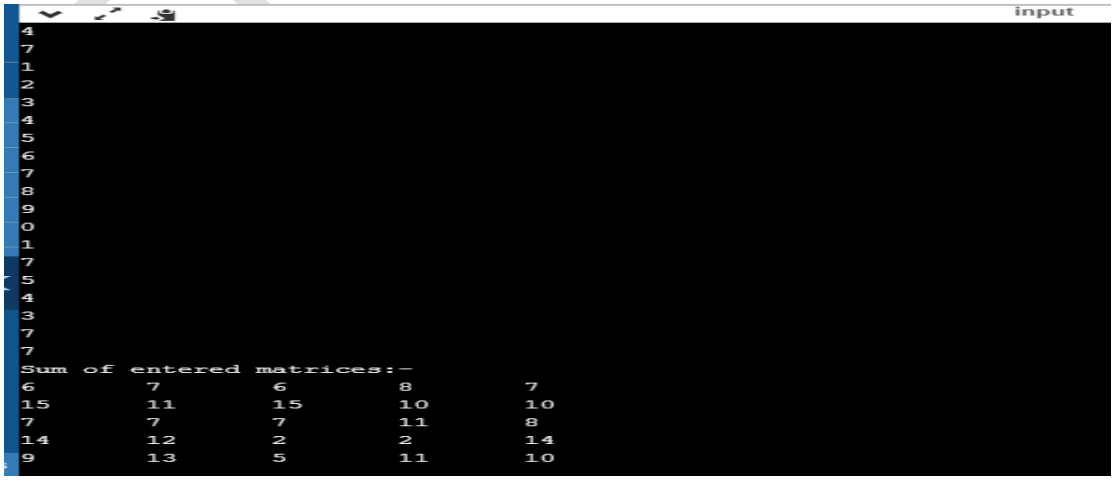
```
#include <stdio.h>


int main()
```

```
{
    int m, n, c, d, first[10][10], second[10][10], sum[10][10];

    printf("Enter the number of rows and columns of matrix\n");
    scanf("%d%d", &m, &n);
    printf("Enter the elements of first matrix\n");

    for (c = 0; c < m; c++)
        for (d = 0; d < n; d++)
            scanf("%d", &first[c][d]);

    printf("Enter the elements of second matrix\n");

    for (c = 0; c < m; c++)
        for (d = 0 ; d < n; d++)
            scanf("%d", &second[c][d]);

    printf("Sum of entered matrices:-\n");

    for (c = 0; c < m; c++) {
        for (d = 0 ; d < n; d++) {
            sum[c][d] = first[c][d] + second[c][d];
            printf("%d\t", sum[c][d]);
```

```
        }
        printf("\n");
    }

    return 0;
}
```

Output:-





Conclusion:-

Simulation of this program helped me to understand the logic for adding 2 matrices. Further it helps to understand the use of for loops.

21.))Objective:-

At the end of this activity, we shall be able to

- Use for loops
- Use 2 -D arrays in C- program.

Problem statement:-

In this problem we aim to understand and use 2D arrays and for loops to get the multiplication of 2 matrices. It is required to get the input from the admin such as:

Elements in the first matrix:

Elements in the second matrix:

Once they are collected, we print the output of the multiplied matrix.

Algorithm:-

START

DEFINE VARIABLES:- a=5,b=5,c=5,d=5, sum=0

INPUT : Read the input from the user.

COMPUTATION:- Multiplies the 2 matrices elements

DISPLAY: Prints the resultant matrix.

STOP.

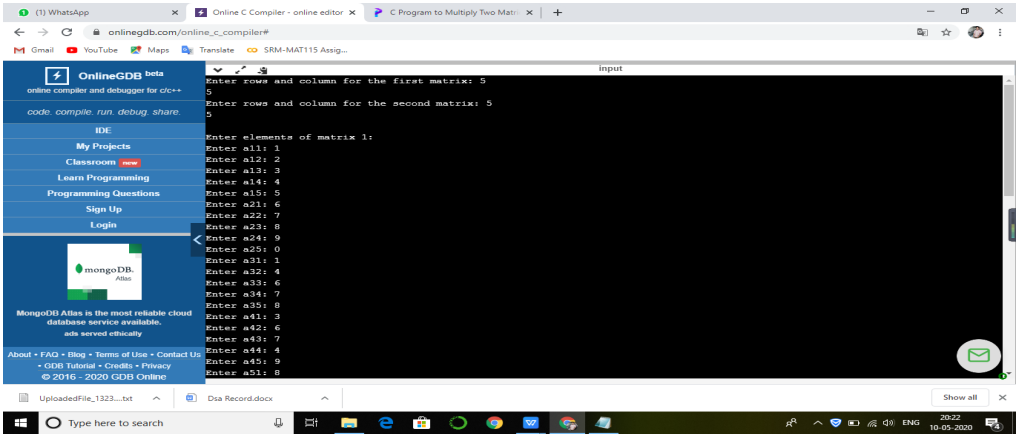
Input:-

```
#include <stdio.h>

void enterData(int first[][10], int second[][10], int r1, int c1, int r2, int c2);

void multiplyMatrices(int first[][10], int second[][10], int multResult[][10], int r1, int c1, int r2, int c2);

void display(int mult[][10], int r1, int c2);


int main() {
    int first[10][10], second[10][10], mult[10][10], r1, c1, r2, c2;
    printf("Enter rows and column for the first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and column for the second matrix: ");
    scanf("%d %d", &r2, &c2);
    while (c1 != r2) {
        printf("Error! Enter rows and columns again.\n");
        printf("Enter rows and columns for the first matrix: ");
        scanf("%d%d", &r1, &c1);
        printf("Enter rows and columns for the second matrix: ");
        scanf("%d%d", &r2, &c2);
    }
    enterData(first, second, r1, c1, r2, c2);
    multiplyMatrices(first, second, mult, r1, c1, r2, c2);
```

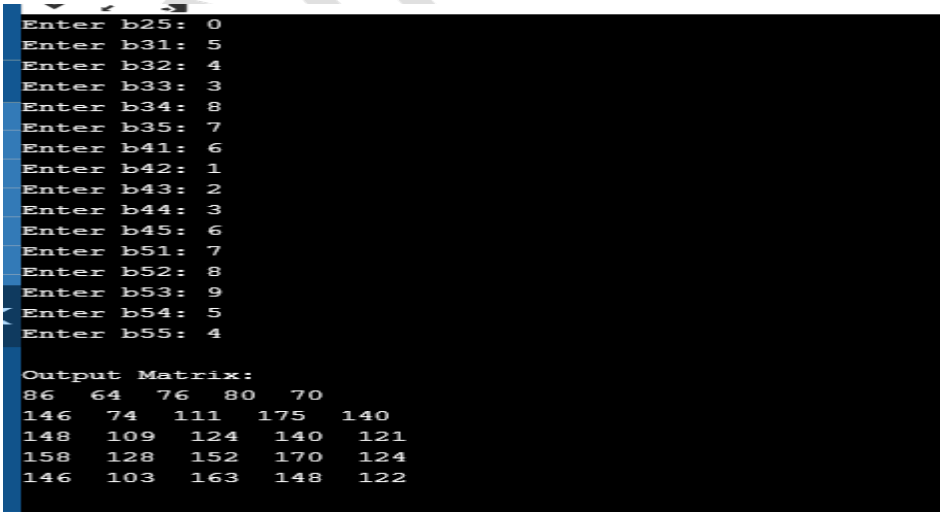
```
    display(mult, r1, c2);
    return 0;
}
void enterData(int first[][10], int second[][10], int r1, int c1, int r2, int
c2) {
    printf("\nEnter elements of matrix 1:\n");
    for (int i = 0; i < r1; ++i) {
        for (int j = 0; j < c1; ++j) {
            printf("Enter a%d%d: ", i + 1, j + 1);
            scanf("%d", &first[i][j]);
        }
    }
    printf("\nEnter elements of matrix 2:\n");
    for (int i = 0; i < r2; ++i) {
        for (int j = 0; j < c2; ++j) {
            printf("Enter b%d%d: ", i + 1, j + 1);
            scanf("%d", &second[i][j]);
        }
    }
}
void multiplyMatrices(int first[][10], int second[][10], int mult[][10],
int r1, int c1, int r2, int c2) {
    for (int i = 0; i < r1; ++i) {
        for (int j = 0; j < c2; ++j) {
```

```
                mult[i][j] = 0;
            }
        }
        for (int i = 0; i < r1; ++i) {
            for (int j = 0; j < c2; ++j) {
                for (int k = 0; k < c1; ++k) {
                    mult[i][j] += first[i][k] * second[k][j];
                }
            }
        }
}
void display(int mult[][10], int r1, int c2) {
        printf("\nOutput Matrix:\n");
        for (int i = 0; i < r1; ++i) {
            for (int j = 0; j < c2; ++j) {
                printf("%d    ", mult[i][j]);
                if (j == c2 - 1)
                    printf("\n");
            }
        }
}
```

Output:-


input
Enter rows and column for the first matrix: 5
5
Enter rows and column for the second matrix: 5
5
Enter elements of matrix 1:
Enter a11: 1
Enter a12: 2
Enter a13: 3
Enter a14: 4
Enter a15: 5
Enter a21: 6
Enter a22: 7
Enter a23: 8
Enter a24: 9
Enter a25: 0
Enter a31: 1
Enter a32: 4
Enter a33: 6
Enter a34: 7
Enter a35: 8
Enter a41: 3
Enter a42: 6
Enter a43: 7
Enter a44: 4
Enter a45: 9
Enter a51: 8


Enter a34: 7
Enter a35: 8
Enter a41: 3
Enter a42: 6
Enter a43: 7
Enter a44: 4
Enter a45: 9
Enter a51: 8
Enter a52: 3
Enter a53: 2
Enter a54: 6
Enter a55: 8
Enter elements of matrix 2:
Enter b11: 4
Enter b12: 2
Enter b13: 8
Enter b14: 7
Enter b15: 5
Enter b21: 4
Enter b22: 3
Enter b23: 3
Enter b24: 6
Enter b25: 0
Enter b31: 5
Enter b32: 4


Enter b25: 0
Enter b31: 5
Enter b32: 4
Enter b33: 3
Enter b34: 8
Enter b35: 7
Enter b41: 6
Enter b42: 1
Enter b43: 2
Enter b44: 3
Enter b45: 6
Enter b51: 7
Enter b52: 8
Enter b53: 9
Enter b54: 5
Enter b55: 4
Output Matrix:
86 64 76 80 70
146 74 111 175 140
148 109 124 140 121
158 128 152 170 124
146 103 163 148 122

Conclusion:-

From this program I have learned how to multiply matrices using arrays.

30.))Objective:-

At the end of this activity, we shall be able to

- Find the Inorder, Preorder, Postorder transversals in a Binary search tree.

Problem Statement:-

In this problem, how the Inorder transversal, Preorder transversal and the Postorder

transversal works in a Binary search tree.

Algorithm:

START

DEFINE VARIABLES: data, node* left, node* right

INPUT: Input of a binary search tree was given in the code itself.

COMPUTATION: For the Inorder transversal First, It visits all the nodes in the left

subtree then the root node lastly all nodes on the right subtree

For Preorder transversal, first visit root node then all nodes of left subtree finally visits all

the nodes in the right subtree

For Post Transversal first it visits all the node in the left subtree then it visits all the

nodes in right subtree finally it visits the root node

DISPLAY: It displays all the three transversals in a binary search tree.

STOP

INput:-

```
#include <stdio.h>

#include <stdlib.h>

struct node {

   int data;

   struct node* left;

   struct node* right;

};

void inorder(struct node* root) {

   if (root == NULL) return;

   inorder(root->left);

   printf("%d ->", root->data);

   inorder(root->right);

}

void preorder(struct node* root) {

   if (root == NULL) return;

   printf("%d ->", root->data);
```

```
    preorder(root->left);
    preorder(root->right);
}
void postorder(struct node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ->", root->data);
}
struct node* createNode(value) {
    struct node* newNode = malloc(sizeof(struct node));
        newNode->data = value;
        newNode->left = NULL;
        newNode->right = NULL;
        return newNode;
}
struct node* insertLeft(struct node* root, int value) {
    root->left = createNode(value);
    return root->left;
}
struct node* insertRight(struct node* root, int value) {
    root->right = createNode(value);
    return root->right;
```

```
}
int main() {
    struct node* root = createNode(1);
    insertLeft(root, 12);
    insertRight(root, 9);
        insertLeft(root->left, 5);
        insertRight(root->left, 6);
        printf("Inorder traversal \n");
        inorder(root);
        printf("\nPreorder traversal \n");
        preorder(root);
        printf("\nPostorder traversal \n");
        postorder(root);
}
```

Output:-

```
Inorder traversal
5 ->12 ->6 ->1 ->9 ->
Preorder traversal
1 ->12 ->5 ->6 ->9 ->
Postorder traversal
5 ->6 ->12 ->9 ->1 ->
```

Conclusion:-WE have inorder and preorder traversal in binarfy search trees.

31.))Objective:

At the end of this activity, we shall be able to

- Search an element in a given binary search tree

Problem Statement:

In this problem, we aim to understand how to search a particular element in a binary

search tree.

Algorithm:

START

DEFINE VARIABLES: value, search_val

INPUT: Reads the input from the user.

COMPUTATION: Takes the search element from the user and searches in the binary

search tree

DISPLAY: It displays whether the element is present in a binary search tree or not.

STOP

Input:-

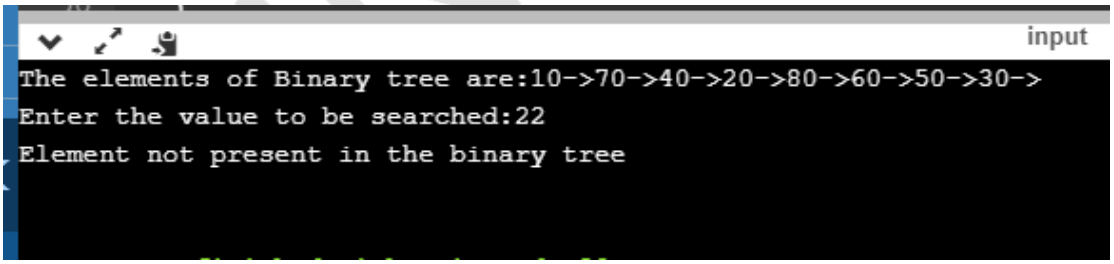
```
#include <stdio.h>
#include <malloc.h>
struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
```

```
};
struct btnode *root = NULL;
int flag;
void in_order_traversal(struct btnode *);
void in_order_search(struct btnode *,int);
struct btnode *newnode(int);
void main()
{
   int search_val;
   root = newnode(50);
   root->l = newnode(20);
   root->r = newnode(30);
   root->l->l = newnode(70);
   root->l->r = newnode(80);
   root->l->l->l = newnode(10);
   root->l->l->r = newnode(40);
   root->l->r->r = newnode(60);
      printf("The elements of Binary tree are:");
      in_order_traversal(root);
      printf("\nEnter the value to be searched:");
      scanf("%d", &search_val);
      in_order_search(root, search_val);
      if (flag == 0) // flag to check if the element is present in the
tree or not
```

```
        {
                printf("Element not present in the binary tree\n");
        }
}
struct btnode* newnode(int value)
{
    struct btnode *temp = (struct btnode *)malloc(sizeof(struct
btnode));
    temp->value = value;
    temp->l = NULL;
    temp->r = NULL;
      return temp;
}
void in_order_traversal(struct btnode *p)
{
   if (!p)
   {
        return;
   }
   in_order_traversal(p->l);
   printf("%d->", p->value);
   in_order_traversal(p->r);
}
void in_order_search(struct btnode *p, int val)
```

```
{
    if (!p)
    {
        return;
    }
    in_order_search(p->l, val);
    if(p->value == val)
    {
        printf("\nElement present in the binary tree.\n");
        flag = 1;
    }
    in_order_search(p->r, val);
}
```

Output:-



Conclusion:-we have searched the element in the binary search ytree.

33.))Objective:

At the end of this activity, we shall be able to To find all pairs shortest path problems from a given weighted graph. As a result of this algorithm, it will generate a matrix, which will represent the

minimum distance from any node to all other nodes in the graph.

Problem Statement:

In this problem, we aim to understand all pair shortest path problems from a given weighted graph.

Algorithm:

START

DEFINE VARIABLES: costMat

INPUT: Takes the input from the user in the matrix form

COMPUTATION: It is used to find all pairs shortest path problem from a given weighted

graph. As a result of this algorithm.

DISPLAY: It displays that the output matrix will be updated with all vertices k as the

intermediate vertex.

STOP

Input:-

```
#include<iostream>

#include<iomanip>

#define NODE 7

#define INF 999

using namespace std;

//Cost matrix of the graph

int costMat[NODE][NODE] = {

    {0, 3, 6, INF, INF, INF, INF},
```

```
    {3, 0, 2, 1, INF, INF, INF},
    {6, 2, 0, 1, 4, 2, INF},
    {INF, 1, 1, 0, 2, INF, 4},
    {INF, INF, 4, 2, 0, 2, 1},
    {INF, INF, 2, INF, 2, 0, 1},
    {INF, INF, INF, 4, 1, 1, 0}
};
void floydWarshal(){
    int cost[NODE][NODE]; //defind to store shortest distance from any node to any node
 for(int i = 0; i<NODE; i++)
    for(int j = 0; j<NODE; j++)
        cost[i][j] = costMat[i][j]; //copy costMatrix to new matrix
        for(int k = 0; k<NODE; k++){
           for(int i = 0; i<NODE; i++)
               for(int j = 0; j<NODE; j++)
                  if(cost[i][k]+cost[k][j] < cost[i][j])
                      cost[i][j] = cost[i][k]+cost[k][j];
 }
 cout << "The matrix:" << endl;
 for(int i = 0; i<NODE; i++){
    for(int j = 0; j<NODE; j++)
        cout << setw(3) << cost[i][j];
    cout << endl;
```

```
  }
}
int main(){
   floydWarshal();
}
```

Output:-

Conclusion:-At the end we will find all pairs shortpaths in graphs.

34.))Objective:

At the end of this activity, we shall be able to

- Push an element to the stack
- Pop an element to the stack
- Peek element in the stack

Problem Statement:

Stack is basically a data object. A stack is a data structure in which items can be inserted only from one end and get items back from the same end. There , the last item inserted into stack, is the first item to be taken out from the stack. In short it's also called Last in First out.

Algorithm:

START

DEFINE VARIABLES: value, choice

INPUT: Takes the input from the user

COMPUTATION: Push, Add an element to the top of the stack.

Pop, Remove the element at the top of the stack.

Peek, prints the value of the top most element of the stack without deleting that

element from the stack.

DISPLAY: It displays the elements in the stack after the operations.

STOP

Input:-

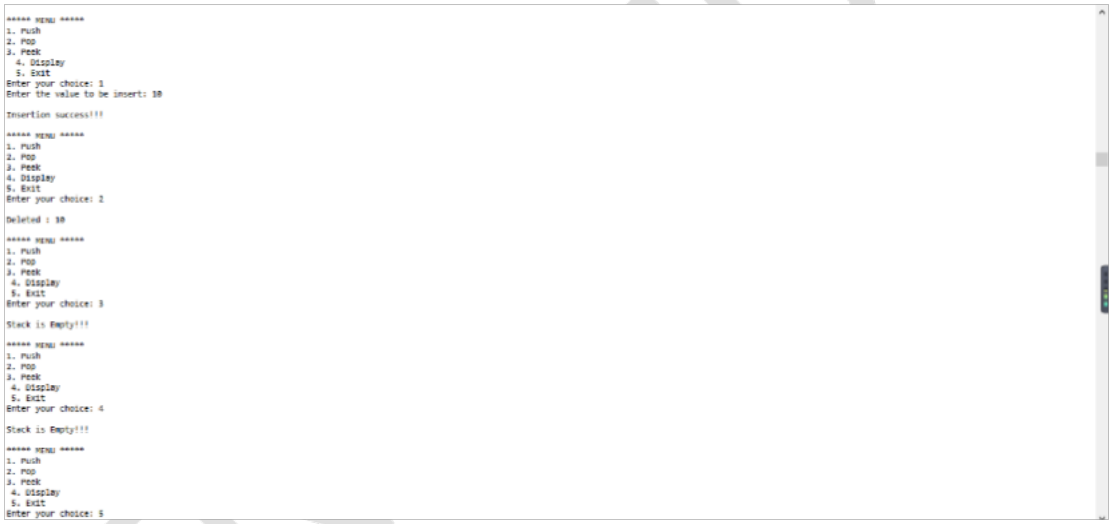
```
#include<stdio.h>
#define SIZE 10
void push(int);
void pop();
void display();
int stack[SIZE], top = -1;
void main()
{
   int value, choice;
   while(1){
        printf("\n\n***** MENU *****\n");
```

```
        printf("1. Push\n2. Pop\n3. Peek \n 4. Display \n 5. Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter the value to be insert: ");
                    scanf("%d",&value);
                    push(value);
                    break;
            case 2: pop();
                    break;
           case 3: peek();
                    break;
            case 4: display();
                    break;
            case 5: exit(0);
            default: printf("\nWrong selection!!! Try again!!!");
            }
            }
}
void push(int value){
   if(top == SIZE-1)
        printf("\nStack is Full!!! Insertion is not possible!!!");
   else{
```

```
        top++;
        stack[top] = value;
        printf("\nInsertion success!!!");
    }
}
void pop(){
    if(top == -1)
        printf("\nStack is Empty!!! Deletion is not possible!!!");
  else{
      printf("\nDeleted : %d", stack[top]);
      top--;
  }
}
void display(){
    if(top == -1)
        printf("\nStack is Empty!!!");
    else{
        int i;
        printf("\nStack elements are:\n");
        for(i=top; i>=0; i--)
                printf("%d\n",stack[i]);
    }
}
```

```
void peek(){
   if(top == -1)
        printf("\nStack is Empty!!!");
   else{
        int i;
        printf("\nStack top most element is: %d\n",stack[top]);
}
}
```

Output:-



Conclusion:-we will use stack here.

35.))Objective:

At the end of this activity, we shall be able to

- Reverse a string using stack

Problem Statement:

In a data structure stack allows you to access the last data element that you inserted to stack,if you remove the last element of the stack,you will be able to access the next to last element. We can use this method or operation to reverse a string value.

Algorithm:

START

DEFINE VARIABLES: top, stack

INPUT: Takes the input from the user

COMPUTATION: Creates an empty stack. One by one push all characters of string to

stack. One by one pop all characters from stack and put them back to string.

DISPLAY: It displays the reverse of the given string

STOP

Input:-

```
#include <stdio.h>

#include <string.h>


#define max 100

int top,stack[max];


void push(char x){


    // Push(Inserting Element in stack) operation

    if(top == max-1){
```

```
            printf("stack overflow");
        } else {
            stack[++top]=x;
          }

}

void pop(){
   // Pop (Removing element from stack)
      printf("%c",stack[top--]);
}



main()
{
   char str[50];
   printf("Enter the string\n");
   scanf("%s",str);
   int len = strlen(str);
   int i;

      for(i=0;i<len;i++)
```

```
            push(str[i]);


        for(i=0;i<len;i++)
            pop();
}
```

OUTPUT:-

36.))Objective: By the end of this activity we shall be able to use stack operations to convert a given infix expression to a post fix expression.

Problem statement: The problem statement is being able to use stack for conversion of infix to post fix

Algorithm:

- START
- DEFINE size of stack and top variable and function
- INPUT the infix expression
- COMPUTATION using the defined function and stack operators, perform the conversion
- DISPLAY post fix expression

Input:-

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

```
#include<stdlib.h>          /* for exit() */
#include<ctype.h>         /* for isdigit(char ) */
#include<string.h>

#define SIZE 100



/* declared here as global variable because stack[]
* is used by more than one fucntions */
char stack[SIZE];
int top = -1;

/* define push operation */


void push(char item)
{
    if(top >= SIZE-1)
    {
        printf("\nStack Overflow.");
    }
```

```
    else
    {
        top = top+1;
        stack[top] = item;
    }
}

/* define pop operation */
char pop()
{
    char item ;

    if(top <0)
    {
        printf("stack under flow: invalid infix expression");
        getchar();
        /* underflow may occur for invalid expression */
        /* where ( and ) are not matched */
        exit(1);
    }
```

```
    else
    {
        item = stack[top];
        top = top-1;
        return(item);
    }
}

/* define function that is used to determine whether any symbol is
operator or not
(that is symbol is operand)
* this fucntion returns 1 if symbol is opreator else return 0 */

int is_operator(char symbol)
{
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' ||
symbol =='-')
    {
        return 1;
    }
```

```
    else
    {
    return 0;
    }
}

/* define fucntion that is used to assign precendence to operator.
* Here ^ denotes exponent operator.
* In this fucntion we assume that higher integer value
* means higher precendence */

int precedence(char symbol)
{
    if(symbol == '^')/* exponent operator, highest precedence*/
    {
        return(3);
    }
    else if(symbol == '*' || symbol == '/')
    {
        return(2);
```

```
    }
    else if(symbol == '+' || symbol == '-')                /* lowest
precedence */
    {
        return(1);
    }
    else
    {
        return(0);
    }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item;
    char x;

    push('(');                                             /* push '(' onto
stack */
```

```
    strcat(infix_exp,")");                          /* add ')' to infix
expression */

    i=0;
    j=0;
    item=infix_exp[i];              /* initialize before loop*/

    while(item != '\0')             /* run loop till end of infix expression
*/
    {
        if(item == '(')
        {
            push(item);
        }
        else if( isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;                  /* add operand
symbol to postfix expr */
            j++;
        }
```

```
        else if(is_operator(item) == 1)          /* means symbol is
operator */
        {
            x=pop();
            while(is_operator(x) == 1 && precedence(x)>=
precedence(item))
            {
                postfix_exp[j] = x;                    /* so pop all
higher precendence operator and */
                j++;
                x = pop();                         /* add them to
postfix expresion */
            }
            push(x);
            /* because just above while loop will terminate we have
            oppped one extra item
            for which condition fails and loop terminates, so that
one*/

            push(item);                    /* push current oprerator
symbol onto stack */
```

```
        }
        else if(item == ')')              /* if current symbol is ')' then */
        {
            x = pop();                    /* pop and keep popping
until */
            while(x != '(')               /* '(' encounterd */
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
        else
        { /* if current symbol is neither operand not '(' nor ')' and nor
            operator */
            printf("\nInvalid infix Expression.\n");          /* the it is
illegeal    symbol */
            getchar();
            exit(1);
        }
```

```
        i++;



        item = infix_exp[i]; /* go to next symbol of infix expression
*/
    } /* while loop ends here */
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");          /* the it is
illegeal    symbol */
        getchar();
        exit(1);
    }
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");          /* the it is
illegeal    symbol */
        getchar();
        exit(1);
    }
```

```
    postfix_exp[j] = '\0'; /* add sentinel else puts() fucntion */
    /* will print entire postfix[] array upto SIZE */

}

/* main function begins */
int main()
{
    char infix[SIZE], postfix[SIZE];          /* declare infix string
and postfix string */

    /* why we asked the user to enter infix expression
    * in parentheses ( )
    * What changes are required in porgram to
    * get rid of this restriction since it is not
    * in algorithm
    * */
    printf("ASSUMPTION: The infix expression contains single
```

```
letter variables and single digit constants only.\n");
    printf("\nEnter Infix expression : ");
    gets(infix);

    InfixToPostfix(infix,postfix);              /* call to
convert */
    printf("Postfix Expression: ");
    puts(postfix);                              /* print postfix expression
*/

    return 0;
}
```

Output:-

ASSUMPTION: The infix expression contains single letter variables and single digit constants only.

Enter Infix expression : A=B +B*C*D/E

Postfix Expression: ABC*D*E/+

Conclusion:

By simulating the above program we learned how to convert a infix expression into the postfix expression by following the order of proceedings.

37.))Objective:

At the end of this activity, we shall be able to

- Convert an Infix expression to an Prefix expression using stack

Problem Statement:

While we use infix expressions in our day to day lives. Computers have trouble understanding this format because they need to keep in mind rules of operator precedence and also brackets. Prefix and Postfix expressions are easier for a computer to understand and evaluate.

Algorithm:

START

Step 2. Scan A from right to left and repeat step 3 to 6 for each element of A until the

STACK is empty

Step 3. If an operand is encountered add it to B

Step 4. If a right parenthesis is encountered push it onto STACK

Step 5. If an operator is encountered then:

a. Repeatedly pop from STACK and add to B each operator (on the top of STACK)

which has the same or higher precedence than the operator.

b. Add operator to STACK

Step 6. If left parenthesis is encountered then

a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a

left parenthesis is encountered)

b. Remove the left parenthesis

Step 7. STOP

Input:-

```
#define SIZE 50 /* Size of Stack */

#include<string.h>

#include <ctype.h>

#include<stdio.h>

char s[SIZE]; int top=-1; /* Global declarations */

push(char elem)

{ /* Function for PUSH operation */

s[++top]=elem;

}
```

```
char pop()
{ /* Function for POP operation */
return(s[top--]);
}
int pr(char elem)
{ /* Function for precedence */
switch(elem)
{
case '#': return 0;
case ')': return 1;
case '+':
case '-': return 2;
case '*':
case '/':return 3;
}
}
main()
{ /* Main Program */
char infx[50],prfx[50],ch,elem;
int i=0,k=0;
printf("\n\nInfix Expression: ");
```

```
scanf("%s",infx);
push('#');
strrev(infx);
while( (ch=infx[i++]) != '\0')
{
if( ch == ')')
push(ch);
else if(isalnum(ch))
prfx[k++]=ch;
else if( ch == '(')
{
while( s[top] != ')')
prfx[k++]=pop();
elem=pop(); /* Remove ) */
}
else
{ /* Operator */
while( pr(s[top]) >= pr(ch) )
prfx[k++]=pop(); push(ch);
}
}
```

```
while( s[top] != '#') /* Pop from stack till empty */

prfx[k++]=pop();

prfx[k]='\0'; /* Make prfx as valid string */

strrev(prfx);

strrev(infx);

printf("\n\nGiven Infix Expn: %s \nPrefix Expn: %s\n",infx,prfx);

}
```

OUTPUT:-

```
 Infix Expression: (A+B)*(B-C)

Given Infix Expn: (A+B)*(B-C)

Prefix Expn: *+AB-BC
```

38.))Objective:

At the end of this activity, we shall be able to

- Prefix and Postfix expressions can be evaluated faster than an infix expression.This is because we don't need to process any brackets or follow operator precedence rules. In postfix and prefix expressions whichever operator comes before will be evaluated first, irrespective of its priority. Also, there are no brackets in these expressions.

Algorithm:

START

DEFINE VARIABLES: n1, n2, n3, num

1) Create a stack to store operands (or values).

2) Scan the given expression and do the following for every scanned element.

- If the element is a number, push it into the stack

- If the element is an operator, pop operands for the operator from stack. Evaluate

the operator and push the result back to the stack

3) When the expression is ended, the number in the stack is the final answer and prints

the answer

STOP

Input

```
#include<stdio.h>

int stack[20];

int top = -1;

void push(int x)

{

    stack[++top] = x;

}

int pop()
```

```
{
        return stack[top--];
}
int main()
{
        char exp[20];
        char *e;
        int n1,n2,n3,num;
        printf("Enter the expression :: ");
        scanf("%s",exp);
        e = exp;
        while(*e != '\0')
        {
                if(isdigit(*e))
                {
                        num = *e - 48;
                        push(num);
                }
                else
                {
                        n1 = pop();
```

```
        n2 = pop();
        switch(*e)
        {
                case '+':
                {
                        n3 = n1 + n2;
break;
                }
                case '-':
                {
                        n3 = n2 - n1;
                        break;
                }
                case '*':
                {
                        n3 = n1 * n2;
                        break;
                }
                case '/':
                {
                    n3 = n2 / n1;
```

```
                        break;
                    }
                }
                push(n3);
            }
            e++;
        }
        printf("\nThe result of expression %s = %d\n\n",exp,pop());
        return 0;

}
```

Testcase

Enter the expression :: 245+*

The result of expression 245+* = 18

39.))Objective:

At the end of this activity, we shall be able to learn about how to

- Push an element to the queue
- Pop an element to the queue
- Peek element in the queue
- Displaying queue

Problem statement:

A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence. And we will also learn about applications of the queues.

Algorithm:

START

DEFINE VARIABLES: size, choice

INPUT: Takes the input from the user

COMPUTATION:

Push, Add an element to the top of the queue.

Pop, Remove the element at the top of the queue.

Peek, prints the value of the top most element of the queue

DISPLAY: It displays the elements in the queue after the operations.

STOP

Input:-

```
#include<stdio.h>

#include<stdlib.h>

#define SIZE 5

void enq(int);
```

```
void deq();
void display();
int queue[SIZE], front=-1,rear=-1;
int main()
{
    while(1)
    {
        int choice,element;
        printf("\n1.enq\n2.deq\n3.display\n4.exit");
        printf("\nEnter your choice");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("Enter the element you want to add");
            scanf("%d",&element);
            enq(element);
            break;
            case 2: deq();
            break;
            case 3: display();
```

```
            break;
            case 4: exit(0);
            break;
            default: printf("You have entered invalid number");
        }
    }
}
void enq(int element)
{
    if(rear==SIZE-1)
    {
        printf("Overflow");
    }
    else
    {
        if(front==-1)
            front=0;
        rear++;
        queue[rear]=element;
        printf("Insertion Success");
```

```
    }
}
void deq()
{
    if(front==-1&&rear==-1)
    {
        printf("Underflow");
    }
    else
    {
        printf("Deleted = %d", queue[front]);
        front++;
        if(front==rear)
            front=rear=-1;
    }
}
void display()
{
    int i;
    if(front==-1&&rear==-1)
```

```
    {
        printf("Underflow");
    }
    else
    {
        for(i=front;i<=rear;i++)
        printf("%d", queue[i]);
    }

}
```

Output:

```
1.enq
2.deq
3.display
4.exit
Enter your choice1
Enter the element you want to add 18
Insertion Success
1.enq
2.deq
```

3.display

4.exit

Enter your choice 1

Enter the element you want to add 22

Insertion Success

1.enq

2.deq

3.display

4.exit

Enter your choice 3

18 22

1.enq

2.deq

3.display

4.exit

Enter your choice 2

Deleted = 18

1.enq

2.deq

3.display

4.exit

Enter your choice 4

Conclusion:

By simulating the above program we learned how to use queues and its applications and how to perform different tasks on the queues.

40.))

Objective:

At the end of this activity, we shall be able to learn about how to

- Push an element to the circular queue
- Pop an element to the circular queue
- Peek element in the circular queue
- Displaying circular queue

Problem statement:

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. And we will also learn about its applications.

Algorithm:-

START

DEFINE VARIABLES: size, choice

INPUT: Takes the input from the user

COMPUTATION:

Push, Add an element to the circular queue.

Pop, Remove the element from the circular queue.

Peek, prints the value of the top most element of the circular queue

DISPLAY: It displays the elements in the circular queue after the operations.

STOP

INPUT:-

```
#include<stdio.h>

#include<stdlib.h>

#define SIZE 5

void enq(int);

void deq();

void display();

int queue[SIZE], front=-1,rear=-1;
```

```
int main()
{
    while(1)
    {
        int choice,element;
        printf("\n1.enq\n2.deq\n3.display\n4.exit");
        printf("\nEnter your choice");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("Enter the element you want to add");
            scanf("%d",&element);
            enq(element);
            break;
            case 2: deq();
            break;
            case 3: display();
            break;
            case 4: exit(0);
            break;
```

```
            default: printf("You have entered invalid number");
        }
    }
}
void enq(int element)
{
    if(front==0&&rear==SIZE-1||front==rear+1)
    {
        printf("Overflow");
    }
    else
    {
        if(front==-1)
        front=0;
        rear=rear+1%SIZE;
        queue[rear]=element;
        printf("Insertion Success");
    }
}
void deq()
```

```
{
    if(front==-1)
    {
        printf("Underflow");
    }
    else
    {
        printf("Deleted: %d", queue[front]);
        front=front+1%SIZE;
        if(front==rear)
        front=rear=-1;
    }
}
void display()
{
    int i;
    if(front==-1)
    {
        printf("Underflow");
    }
```

```
    else
    {
        for(i=front;i!=rear;(i+1)%SIZE)
        {
            printf("%d", queue[i]);
        }
    }
}
```

Output:

```
1.enq
2.deq
3.display
4.exit
Enter your choice 1
Enter the element you want to add 30
Insertion Success
1.enq
2.deq
```

3.display

4.exit

Enter your choice 3

30

1.enq

2.deq

3.display

4.exit

Enter your choice 2

Deleted: 30

1.enq

2.deq

3.display

4.exit

Enter your choice 4

Conclusion:

By simulating the program we learned about applications and how to use circular queues and different tasks in the circular queues.

41.))Objective:

At the end of this activity we shall learn the linked lists and applications of the linked lists and how to insert elements and display them.

Problem statement:

A linked list is a linear data structure where each element is a separate object. Linked list elements are not stored at contiguous locations the elements are linked using pointers. Each node of a list is made up of two items: the data and a reference to the next node. The last node has a reference to null and we will also learn about applications of linked lists.

Algorithm:

START

A linked list is a series of connected nodes. Each node contains at least. A piece of data

(any type). Pointer to the next node in the list. Head: pointer to the first node. The last

node points to NULL

Empty Linked list is a single pointer having the value of NULL.

head = NULL; head

Let’s assume that the node is given by the following type declaration:

struct Node{

int data;

struct Node *next;

};

To start with, we have to create a node (the first node), and make a head point to it.

head = (struct Node*)malloc(sizeof(struct Node));

STOP

INPUT:-

```
#include <stdio.h>

#include <stdlib.h>


struct node

{

    int num;                        //Data of the node

    struct node *nextptr;           //Address of the next node

}*stnode;
```

```
void createNodeList(int n); // function to create the list
void displayList();                // function to display the list


int main()
{
      int n;
         printf("\n\n Linked List : To create and display Singly Linked
List :\n");


      printf(" Input the number of nodes : ");
      scanf("%d", &n);
      createNodeList(n);
      printf("\n Data entered in the list : \n");
      displayList();
      return 0;
}
void createNodeList(int n)
{
      struct node *fnNode, *tmp;
```

```
    int num, i;
    stnode = (struct node *)malloc(sizeof(struct node));

    if(stnode == NULL) //check whether the fnnode is NULL and if so no memory allocation
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
// reads data for the node through keyboard

        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode->num = num;
        stnode->nextptr = NULL; // links the address field to NULL
        tmp = stnode;
// Creating n nodes and adding to linked list
        for(i=2; i<=n; i++)
        {
```

```
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode == NULL)
            {
                printf(" Memory can not be allocated.");
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);

                fnNode->num = num;          // links the num field
of fnNode with num
                fnNode->nextptr = NULL; // links the address
field of fnNode with NULL

                tmp->nextptr = fnNode; // links previous node i.e.
tmp to the fnNode
                tmp = tmp->nextptr;
            }
```

```
            }
        }
}
void displayList()
{
        struct node *tmp;
        if(stnode == NULL)
        {
            printf(" List is empty.");
        }
        else
        {
            tmp = stnode;
            while(tmp != NULL)
            {
                printf(" Data = %d\n", tmp->num);
                tmp = tmp->nextptr;
            }
        }
}
```

Output:

```
Linked List : To create and display Singly Linked List :

Input the number of nodes : 5

Input data for node 1 : 1

Input data for node 2 : 2

Input data for node 3 : 3

Input data for node 4 : 4

Input data for node 5 : 5


Data entered in the list :

Data = 1

Data = 2

Data = 3

Data = 4

Data = 5
```

Conclusion:

By simulating the above program we learned how to create a singly linked list and some of its applications in c language.

42.))Objective:

In this we aim to understand about linked lists and its applications and how we will search the element in the linked list.

Problem statement:

A linked list is a linear data structure where each element is a separate object. Linked list elements are not stored at contiguous locations the elements are linked using pointers. Each node of a list is made up of two items: the data and a reference to the next node. The last node has a reference to null and we will also learn about applications of linked lists and how to search an element in the linked list.

Algorithm:

START

Input element to search from user. Store it in some variable say keyToSearch. Declare two variables one to store the index of the found element and other to iterate through the list. Say index = 0; and struct node *curNode = head; If curNode is not NULL and its data is not equal to keyToSearch. Then, increment the index and move curNode to its next node. Repeat step 3 till curNode != NULL and element is not found, otherwise move to 5th step. If curNode is not NULL, then element is found hence return index otherwise -1.

STOP

Input:-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int num;
struct node *nextptr;
}
stnode, *ennode;
int FindElement(int);
void main()
{
int n,i,FindElem,FindPlc;
stnode.nextptr=NULL;
ennode=&stnode;
printf("\n\n Linked List : Search an element in a Singly Linked List
  :\n");
printf(" Input the number of nodes : ");
scanf("%d", &n);
```

```
printf("\n");
for(i=0;i< n;i++)
{
ennode->nextptr=(struct node *)malloc(sizeof(struct node));
printf(" Input data for node %d : ",i+1);
scanf("%d",&ennode->num);
ennode=ennode->nextptr;
}
ennode->nextptr=NULL;
printf("\n Data entered in the list are :\n");
ennode=&stnode;
while(ennode->nextptr!=NULL)
{
printf(" Data = %d\n",ennode->num);
ennode=ennode->nextptr;
}
printf("\n");
printf(" Input the element to be searched : ");
scanf("%d",&FindElem);
FindPlc=FindElement(FindElem);
```

```
if(FindPlc<=n)
printf(" Element found at node %d \n\n",FindPlc);
else
printf(" This element does not exists in linked list.\n\n");
}
int FindElement(int FindElem)
{
int ctr=1;
ennode=&stnode;
while(ennode->nextptr!=NULL)
{
if(ennode->num==FindElem)
break;
else
ctr++;
ennode=ennode->nextptr;
}
return ctr;
}
```

Output:

Linked List : Search an element in a Singly Linked List :

Input the number of nodes : 5

Input data for node 1 : 1

Input data for node 2 : 2

Input data for node 3 : 4

Input data for node 4 : 6

Input data for node 5 : 7

Data entered in the list are :

Data = 1

Data = 2

Data = 4

Data = 6

Data = 7

Input the element to be searched : 2

Element found at node 2

Conclusion:

By simulating the above program we learned about the linked lists

and how to search an element in the given linked lists.

43.))Objective:

At the end of this activity, we shall be able to

- Insert a node, Delete a node at the beginning of a singly linked list.

- Insert a node, Delete a node at the middle of a singly linked list.

- Insert a node, Delete a node at the end of a singly linked list.

Problem Statement:

There are three different possibilities for inserting a node into a linked list. These three possibilities are:

Insertion at the beginning of the list.

Insertion at the end of the list

Inserting a new node except the above-mentioned positions.

Algorithm:

START

A) Insert node at beginning of linked list

Step1: Create a Node

Step2: Set the node data Value in the node just created

Step3: Connect the pointers

B) Insert node at end of linked list

Step1: Create a Node

Step2: Set the node data Values

Step3: Connect the pointers

C) Insert node at middle of linked list

Step1: Create a Node

Step2: Set the node data Values

Step3: Break pointer connection

Step 4: Re-connect the pointers

D) Delete node at beginning of the linked list

Step1: Break the pointer connection

Step2: Re-connect the nodes

Step3: Delete the node

E) Delete node at end of linked list

Step1: Break the pointer connection

Step2: Set previous node pointer to NULL

Step3: Delete the node

Input:-

A)

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

```
#include <stdlib.h>


struct node
{
    int num;                    //Data of the node
    struct node *nextptr;       //Address of the node
}*stnode;


void createNodeList(int n); //function to create the list
void NodeInsertatBegin(int num);                //function to insert
node at the beginning
void displayList();         //function to display the list


int main()
{
    int n,num;
                    printf("\n\n Linked List : Insert a new node at the
beginning of a Singly
Linked List:\n");
```

```
printf("--------------------------------------------------------------------\n");
    printf(" Input the number of nodes : ");
    scanf("%d", &n);
    createNodeList(n);
    printf("\n Data entered in the list are : \n");
    displayList();
    printf("\n Input data to insert at the beginning of the list : ");
    scanf("%d", &num);
    NodeInsertatBegin(num);
    printf("\n Data after inserted in the list are : \n");
    displayList();
  return 0;
}
void createNodeList(int n)
{
  struct node *fnNode, *tmp;
  int num, i;

    stnode = (struct node *)malloc(sizeof(struct node));
    if(stnode == NULL) //check whether the stnode is NULL and if
```

```
so no memory
allocation
        {
            printf(" Memory can not be allocated.");
        }
        else
        {
// reads data for the node through keyboard
            printf(" Input data for node 1 : ");
            scanf("%d", &num);
            stnode-> num = num;
            stnode-> nextptr = NULL; //Links the address field to
NULL
            tmp = stnode;

//Creates n nodes and adds to linked list
         for(i=2; i<=n; i++)
         {
            fnNode = (struct node *)malloc(sizeof(struct node));
```

```
        if(fnNode == NULL) //check whether the fnnode is NULL
and if so no memory
allocation
        {
            printf(" Memory can not be allocated.");
            break;
        }
        else
        {
            printf(" Input data for node %d : ", i);
            scanf(" %d", &num);
            fnNode->num = num;              // links the num field
of fnNode with num
                fnNode->nextptr = NULL; // links the address
field of fnNode with NULL
                tmp->nextptr = fnNode; // links previous node i.e.
tmp to the fnNode
                tmp = tmp->nextptr;
            }
        }
    }
```

```
}

void NodeInsertatBegin(int num)
{
   struct node *fnNode;
   fnNode = (struct node*)malloc(sizeof(struct node));
   if(fnNode == NULL)
   {
        printf(" Memory can not be allocated.");
   }
   else
   {
        fnNode->num = num; //Links the data part
        fnNode->nextptr = stnode; //Links the address part
        stnode = fnNode; //Makes stnode as first node
   }
}

void displayList()
{
```

```
    struct node *tmp;
    if(stnode == NULL)
    {
        printf(" No data found in the list.");
    }
    else
    {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num); // prints the data of
current node
            tmp = tmp->nextptr;                    // advances the
position of current node
        }
      }
}
```

Testcase:

Linked List : Insert a new node at the beginning of a Singly Linked List:

Input the number of nodes : 3

Input data for node 1 : 10

Input data for node 2 : 20

Input data for node 3 : 30

Data entered in the list are :

Data = 10

Data = 20

Data = 30

Input data to insert at the beginning of the list : 5

Data after inserted in the list are :

Data = 5

Data = 10

Data = 20

Data = 30

B.

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

struct node
{
    int num;                    //Data of the node
    struct node *nextptr;       //Address of the node
}*stnode;

void createNodeList(int n); //function to create the list
void NodeInsertatEnd(int num); //function to insert node at the end
void displayList();          //function to display the list
int main()
{
    int n,num;
                    printf("\n\n Linked List : Insert a new node at the
end of a Singly Linked
List :\n");
```

```
printf("--------------------------------------------------------------------\n");


    printf(" Input the number of nodes : ");
    scanf("%d", &n);
    createNodeList(n);
    printf("\n Data entered in the list are : \n");
    displayList();
    printf("\n Input data to insert at the end of the list : ");
    scanf("%d", &num);
    NodeInsertatEnd(num);
    printf("\n Data, after inserted in the list are : \n");
    displayList();
    return 0;
}
void createNodeList(int n)
{
     struct node *fnNode, *tmp;
     int num, i;
     stnode = (struct node *)malloc(sizeof(struct node));
```

```
    if(stnode == NULL) //check whether the stnode is NULL and if so no memory
allocation
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
// reads data for the node through keyboard
        printf(" Input data for node 1 : ");
        scanf("%d", &num);

        stnode-> num = num;
        stnode-> nextptr = NULL; //Links the address field to NULL
        tmp = stnode;
//Creates n nodes and adds to linked list
        for(i=2; i<=n; i++)
        {
            fnNode = (struct node *)malloc(sizeof(struct node));
            if(fnNode == NULL) //check whether the fnnode is NULL
```

```
and if so no memory
allocation
            {
                printf(" Memory can not be allocated.");
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);
                fnNode->num = num;              // links the num field
of fnNode with num
                fnNode->nextptr = NULL; // links the address field of
fnNode with NULL
                tmp->nextptr = fnNode; // links previous node i.e.
tmp to the fnNode
                tmp = tmp->nextptr;
            }
        }
    }
```

```
}

void NodeInsertatEnd(int num)
{
   struct node *fnNode, *tmp;
   fnNode = (struct node*)malloc(sizeof(struct node));
   if(fnNode == NULL)
   {
        printf(" Memory can not be allocated.");
   }
   else
   {
        fnNode->num = num; //Links the data part
        fnNode->nextptr = NULL;
        tmp = stnode;
        while(tmp->nextptr != NULL)
            tmp = tmp->nextptr;
        tmp->nextptr = fnNode; //Links the address part
   }
}
```

```
void displayList()
{
   struct node *tmp;
   if(stnode == NULL)
   {
        printf(" No data found in the empty list.");
   }
   else
   {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num); // prints the data of
current node
            tmp = tmp->nextptr;                    // advances the
position of current node
        }
   }
}
```

Testcase:

Linked List : Insert a new node at the end of a Singly Linked List :

Input the number of nodes : 3

Input data for node 1 : 10

Input data for node 2 : 20

Input data for node 3 : 30

Data entered in the list are :

Data = 10

Data = 20

Data = 30

Input data to insert at the end of the list : 5

Data, after inserted in the list are :

Data = 10

Data = 20

Data = 30

Data = 5

C.

```
#include <stdio.h>

#include <stdlib.h>


struct node

{

    int num;                    //Data of the node

    struct node *nextptr;       //Address of the node

}*stnode;


void createNodeList(int n);

//function to create the list

void insertNodeAtMiddle(int num, int pos);

//function to insert node at the middle

void displayList();                                         //function
```

to display the list

```
int main()
{
    int n,num,pos;
                    printf("\n\n Linked List : Insert a new node at the
middle of the Linked List
:\n");

printf("-----------------------------------------------------------------\n");

   printf(" Input the number of nodes (3 or more) : ");
   scanf("%d", &n);
   createNodeList(n);
   printf("\n Data entered in the list are : \n");
   displayList();
   printf("\n Input data to insert in the middle of the list : ");
   scanf("%d", &num);
   printf(" Input the position to insert new node : " );
   scanf("%d", &pos);
```

```
        if(pos<=1 || pos>=n)
   {
     printf("\n Insertion can not be possible in that position.\n ");
   }
        if(pos>1 && pos<n)
    {
                insertNodeAtMiddle(num, pos);
      printf("\n Insertion completed successfully.\n ");
    }
   printf("\n The new list are : \n");
   displayList();
   return 0;
}
void createNodeList(int n)
{
      struct node *fnNode, *tmp;
      int num, i;
      stnode = (struct node *)malloc(sizeof(struct node));
      if(stnode == NULL) //check whether the stnode is NULL and if
so no memory
```

```
allocation
        {
                printf(" Memory can not be allocated.");
        }
        else
        {
// reads data for the node through keyboard
                printf(" Input data for node 1 : ");
                scanf("%d", &num);
                stnode-> num = num;
                stnode-> nextptr = NULL; //Links the address field to
NULL
                tmp = stnode;
//Creates n nodes and adds to linked list
                for(i=2; i<=n; i++)
                {
                    fnNode = (struct node *)malloc(sizeof(struct node));
                    if(fnNode == NULL) //check whether the fnnode is
NULL and if so no memory
allocation
```

```
            {
                printf(" Memory can not be allocated.");
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                 scanf(" %d", &num);

                 fnNode->num = num;         // links the num field
of fnNode with num
                 fnNode->nextptr = NULL; // links the address
field of fnNode with NULL

                 tmp->nextptr = fnNode; // links previous node i.e.
tmp to the fnNode
                 tmp = tmp->nextptr;
              }
        }
    }
```

```
}

void insertNodeAtMiddle(int num, int pos)
{
    int i;
    struct node *fnNode, *tmp;
    fnNode = (struct node*)malloc(sizeof(struct node));
    if(fnNode == NULL)
    {
        printf(" Memory can not be allocated.");
    }
    else
    {
        fnNode->num = num; //Links the data part
        fnNode->nextptr = NULL;
        tmp = stnode;
        for(i=2; i<=pos-1; i++)
        {
            tmp = tmp->nextptr;
```

```
                if(tmp == NULL)
                        break;
                }
                if(tmp != NULL)
                {
                        fnNode->nextptr = tmp->nextptr; //Links the address
part of new node
                        tmp->nextptr = fnNode;
                }
                else
                {
                        printf(" Insert is not possible to the given position.\n");
                }
        }
}

void displayList()
{
   struct node *tmp;
   if(stnode == NULL)
```

```
    {
        printf(" No data found in the empty list.");
    }
    else
    {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num); // prints the data of
current node
            tmp = tmp->nextptr;                    // advances the
position of current node
        }
    }
}
```

Testcase:

Linked List : Insert a new node at the middle of the Linked List :

Input the number of nodes (3 or more) : 3

Input data for node 1 : 10

Input data for node 2 : 20

Input data for node 3 : 30

Data entered in the list are :

Data = 10

Data = 20

Data = 30

Input data to insert in the middle of the list : 5

Input the position to insert new node : 2

Insertion completed successfully.

The new list are :

Data = 10

Data = 5

Data = 20

Data = 30

D.

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

struct node
{
    int num;                    //Data of the node
    struct node *nextptr;       //Address of the node
}*stnode;

void createNodeList(int n); //function to create the list
void FirstNodeDeletion();       //function to delete the first node
void displayList();         //function to display the list

int main()
{
    int n,num,pos;
                    printf("\n\n Linked List : Delete first node of
```

```
Singly Linked List :\n");
                        printf("----------------------------------------------------\n");
    printf(" Input the number of nodes : ");
    scanf("%d", &n);
    createNodeList(n);
    printf("\n Data entered in the list are : \n");
    displayList();
    FirstNodeDeletion();
    printf("\n Data, after deletion of first node : \n");
    displayList();
    return 0;
}
void createNodeList(int n)
{
      struct node *fnNode, *tmp;
      int num, i;
      stnode = (struct node *)malloc(sizeof(struct node));
      if(stnode == NULL)                                    //check
whether the stnode is NULL and if so
no memory allocation
```

```
        {
            printf(" Memory can not be allocated.");
        }
        else
        {
// reads data for the node through keyboard
            printf(" Input data for node 1 : ");
            scanf("%d", &num);
            stnode-> num = num;
            stnode-> nextptr = NULL; //Links the address field to
NULL
            tmp = stnode;
//Creates n nodes and adds to linked list
            for(i=2; i<=n; i++)
            {
                fnNode = (struct node *)malloc(sizeof(struct node));
                if(fnNode == NULL)
//check whether the fnnode is NULL and
if so no memory allocation
                {
```

```
                printf(" Memory can not be allocated.");
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);
                fnNode->num = num;              // links the num
field of fnNode with num
                fnNode->nextptr = NULL; // links the address field
of fnNode with NULL
                 tmp->nextptr = fnNode; // links previous node i.e.
tmp to the fnNode
                tmp = tmp->nextptr;
            }
        }
    }
}

void FirstNodeDeletion()
```

```
{
   struct node *toDelptr;
   if(stnode == NULL)
   {
        printf(" There are no node in the list.");
   }
   else
   {
        toDelptr = stnode;
        stnode = stnode->nextptr;
        printf("\n Data of node 1 which is being deleted is : %d\n",
toDelptr->num);
        free(toDelptr); // Clears the memory occupied by first node
   }
}



void displayList()
{
```

```
    struct node *tmp;
    if(stnode == NULL)
    {
        printf(" No data found in the list.");
    }
    else
    {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num); // prints the data of
current node
            tmp = tmp->nextptr;                    // advances the
position of current node
        }
    }
}
```

Testcase:

Linked List : Delete first node of Singly Linked List :

Input the number of nodes : 3

Input data for node 1 : 10

Input data for node 2 : 20

Input data for node 3 : 30

Data entered in the list are :

Data = 10

Data = 20

Data = 30

Data of node 1 which is being deleted is : 10

Data, after deletion of first node :

Data = 20

Data = 30

E.

```
#include <stdio.h>

#include <stdlib.h>
```

```
struct node
{
    int num;                    //Data of the node
    struct node *nextptr;       //Address of the node
}*stnode;

void createNodeList(int n); //function to create the list
void LastNodeDeletion();          //function to delete the last nodes
void displayList();          //function to display the list

int main()
{
    int n,num,pos;
                    printf("\n\n Linked List : Delete the last node of
Singly Linked List :\n");
                    printf("-----------------------------------------------------------\n");
   printf(" Input the number of nodes : ");
   scanf("%d", &n);
   createNodeList(n);
```

```
   printf("\n Data entered in the list are : \n");
   displayList();
   LastNodeDeletion();
            printf("\n The new list after deletion the last node are :
\n");
   displayList();
   return 0;
}
void createNodeList(int n)
{
   struct node *fnNode, *tmp;
   int num, i;

      stnode = (struct node *)malloc(sizeof(struct node));
      if(stnode == NULL) //check whether the stnode is NULL and if
so no memory
allocation
      {
            printf(" Memory can not be allocated.");
```

```
    }
    else
    {
// reads data for the node through keyboard
        printf(" Input data for node 1 : ");
        scanf("%d", &num);

    stnode-> num = num;
    stnode-> nextptr = NULL; //Links the address field to NULL
    tmp = stnode;

//Creates n nodes and adds to linked list
     for(i=2; i<=n; i++)
     {
        fnNode = (struct node *)malloc(sizeof(struct node));
        if(fnNode == NULL) //check whether the fnnode is NULL
and if so no memory
allocation
            {
                printf(" Memory can not be allocated.");
```

```
                break;
            }
            else
            {
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);
                fnNode->num = num;              // links the num field
of fnNode with num
                fnNode->nextptr = NULL; // links the address field of
fnNode with NULL
                tmp->nextptr = fnNode; // links previous node i.e.
tmp to the fnNode
                tmp = tmp->nextptr;
            }
        }
    }
}
// Deletes the last node of the linked list
void LastNodeDeletion()
{
```

```
struct node *toDelLast, *preNode;
if(stnode == NULL)
{
    printf(" There is no element in the list.");
}
else
{
    toDelLast = stnode;
    preNode = stnode;
    /* Traverse to the last node of the list*/
    while(toDelLast->nextptr != NULL)
    {
        preNode = toDelLast;
        toDelLast = toDelLast->nextptr;
    }
    if(toDelLast == stnode)
    {
        stnode = NULL;
    }
    else
```

```
        {


                /* Disconnects the link of second last node with last
node */
                preNode->nextptr = NULL;
        }


        /* Delete the last node */
        free(toDelLast);
   }
}
// function to display the entire list
void displayList()
{
      struct node *tmp;
      if(stnode == NULL)
      {
            printf(" No data found in the empty list.");
      }
      else
```

```
    {
        tmp = stnode;
        while(tmp != NULL)
        {
            printf(" Data = %d\n", tmp->num); // prints the data of
current node
            tmp = tmp->nextptr;                    // advances the
position of current node
        }
    }
}
```

Testcase:

```
  Linked List : Delete the last node of Singly Linked List :
-----------------------------------------------------------
  Input the number of nodes : 3
  Input data for node 1 : 10
  Input data for node 2 : 20
Input data for node 3 : 30
```

Data entered in the list are :

Data = 10

Data = 20

Data = 30

The new list after deletion the last node are :

Data = 10

Data = 20

44.))Objective:

At the end of this activity, we shall be able to

- Travers in both forward and backward direction. The delete operation in DLL is more efficient if a pointer to the node to be deleted is given. We can quickly insert a new node before a given node.

Problem Statement:

A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains three fields: two link fields (references to the previous and to the next node in the sequence of nodes) and one data field.

Algorithm:

START

DEFINE VARIABLES: num, n, *fnNode, *temp

INPUT: Takes the input from the user

COMPUTATION: Navigation is possible in both ways either forward and backward.

DISPLAY: It displays the data entered in the doubly linked list.

STOP

Input:

```
#include <stdio.h>

#include <stdlib.h>


struct node {

    int num;

    struct node * preptr;

    struct node * nextptr;

}*stnode, *ennode;




void DlListcreation(int n);

void displayDlList();
```

```
int main()
{
    int n;
    stnode = NULL;
    ennode = NULL;
        printf("\n\n Doubly Linked List : Create and display a
doubly linked list :\n");
        printf("------------------------------------------------------------\n");

     printf(" Input the number of nodes : ");
     scanf("%d", &n);

     DlListcreation(n);
     displayDlList();
     return 0;
}

void DlListcreation(int n)
{
```

```
    int i, num;
    struct node *fnNode;

      if(n >= 1)
      {
            stnode = (struct node *)malloc(sizeof(struct node));
        if(stnode != NULL)
        {
              printf(" Input data for node 1 : "); // assigning data in
the first node
              scanf("%d", &num);

            stnode->num = num;
            stnode->preptr = NULL;
            stnode->nextptr = NULL;
            ennode = stnode;
// putting data for rest of the nodes
            for(i=2; i<=n; i++)
            {
                fnNode = (struct node *)malloc(sizeof(struct node));
```

```
            if(fnNode != NULL)
            {
                printf(" Input data for node %d : ", i);
                scanf("%d", &num);
                fnNode->num = num;
                fnNode->preptr = ennode; // new node is linking with the previous node
                fnNode->nextptr = NULL;

               ennode->nextptr = fnNode; // previous node is linking with the new node
               ennode = fnNode;          // assign new node as last node
            }
            else
            {
                printf(" Memory can not be allocated.");
                break;
            }
        }
```

```
        }
        else
        {
            printf(" Memory can not be allocated.");
        }
    }
}
void displayDlList()
{
    struct node * tmp;
    int n = 1;
    if(stnode == NULL)
    {
        printf(" No data found in the List yet.");
    }
    else
    {
        tmp = stnode;
        printf("\n\n Data entered on the list are :\n");
```

```
            while(tmp != NULL)
            {
               printf(" node %d : %d\n", n, tmp->num);
               n++;
               tmp = tmp->nextptr; // current pointer moves to the
next node
            }
      }
}
```

OUTPUT:-

```
   Doubly Linked List : Create and display a doubly linked list :
------------------------------------------------------------
   Input the number of nodes : 5
   Input data for node 1 : 10
   Input data for node 2 : 20
   Input data for node 3 : 30
   Input data for node 4 : 40
   Input data for node 5 : 50
Data entered on the list are :
node 1 : 10
```

node 2 : 20

node 3 : 30

node 4 : 40

node 5 : 50

45.))Objective:

At the end of this activity, we shall be able to - Accessing any node of the linked list, we start traversing from the first node. If we are at any node in the middle of the list, then it is not possible to access nodes that precede the given node. This problem can be solved by slightly altering the structure of singly linked lists.

Problem Statement:

Implement a circular singly linked list, we take an external pointer that points to the last

node of the list. If we have a pointer last pointing to the last node, then last -> next will

point to the first node.

Algorithm:

START

Step 1- To implement a circular singly linked list, we take an external pointer that points

Step 2- To the last node of the list. If we have a pointer last

pointing to the last node

Step 3- Then last -> next will point to the first node.

Step 4- The pointer last points to node Z and last -> next points to node P.

STOP

INPUT:-

```
#include <stdio.h>

#include <stdlib.h>

/*

 * Basic structure of Node

 */

struct node {

      int data;

      struct node * next;

}*head;

/*

 * Functions used in this program

 */

void createList(int n);

void displayList();
```

```
int main()
{
    int n, data, choice=1
   head = NULL
   /*
    * Run forever until user chooses 0
    */
   while(choice != 0)
   {

printf("=========================================\n");
        printf("CIRCULAR LINKED LIST PROGRAM\n");

printf("=========================================\n");
        printf("1. Create List\n");
        printf("2. Display list\n");
        printf("0. Exit\n");
        printf("-----------------------------------\n");
        printf("Enter your choice : ");
       scanf("%d", &choice);
```

```
        switch(choice)
            {
                case 1:
                    printf("Enter the total number of nodes in list: ");
                    scanf("%d", &n);
                    createList(n);
                    break;
                case 2:
                    displayList();
                    break;
                case 0:
                    break;
                default:
                    printf("Error! Invalid choice. Please choose
between 0-2");
            }
            printf("\n\n\n\n\n");
        }
        return 0;
}
```

```
void createList(int n)
{
   int i, data;
   struct node *prevNode, *newNode;
      if(n >= 1)
      {
                for(i=2; i<=n; i++)
            {
               newNode = (struct node *)malloc(sizeof(struct node));

                printf("Enter data of %d node: ", i);
                scanf("%d", &data);

                newNode->data = data;
                newNode->next = NULL;

                // Link the previous node with newly created node
                prevNode->next = newNode;

                // Move the previous node ahead
```

```
            prevNode = newNode;
        }

        // Link the last node with first node
        prevNode->next = head;

        printf("\nCIRCULAR LINKED LIST CREATED
SUCCESSFULLY\n");
    }
}

/**
 * Display the content of the list
 */
void displayList()
{
    struct node *current;
    int n = 1;
```

```
    if(head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        current = head;
        printf("DATA IN THE LIST:\n");

        do {
           printf("Data %d = %d\n", n, current->data);

           current = current->next;
           n++;
        }while(current != head);
    }
}
```

OUTPUT:-

==========

CIRCULAR LINKED LIST PROGRAM

==========

1. Create List

2. Display list

0. Exit

Enter your choice : 1

Enter the total number of nodes in list: 5

Enter data of 1 node: 10

Enter data of 2 node: 20

Enter data of 3 node: 30

Enter data of 4 node: 40

Enter data of 5 node: 50

CIRCULAR LINKED LIST CREATED SUCCESSFULLY

==========

CIRCULAR LINKED LIST PROGRAM

==========

1. Create List

2. Display list

0. Exit

Enter your choice : 2

DATA IN THE LIST:

Data 1 = 10

Data 2 = 20

Data 3 = 30

Data 4 = 40

Data 5 = 50

==========

CIRCULAR LINKED LIST PROGRAM

==========

1. Create List

2. Display list

0. Exit

Enter your choice : 0

46.))Objective:

At the end of this activity, we shall be able to

- Create a linked list and implement the stack using a linked list.

Problem Statement:

This C Program implements a stack using linked lists. Stack is a type of queue that in practice is implemented as an area of memory that holds all local variables and parameters used by any function, and remembers the order in which functions are called so that function returns occur correctly.

Algorithm:

START

push

The steps for push operation are:

1.Make a new node.

2.Give the ‘data’ of the new node its value.

3.Point the ‘next’ of the new node to the top of the stack.

4.Make the ‘top’ pointer point to this new node

pop

1.Make a temporary node.

2.Point this temporary node to the top of the stack

3.Store the value of ‘data’ of this temporary node in a variable.

4.Point the ‘top’ pointer to the node next to the current top node.

5.Delete the temporary node using the ‘free’ function.

6.Return the value stored in step 3.

STOP

INPUT:-

```
#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0


struct node

{

    int data;

    struct node *next;

};

typedef struct node node;
```

```
node *top;


void initialize()
{
   top = NULL;
}


void push(int value)
{
   node *tmp;
   tmp = malloc(sizeof(node));
   tmp -> data = value;
   tmp -> next = top;
   top = tmp;
}


int pop()
{
     node *tmp;
     int n;
```

```
    tmp = top;
    n = tmp->data;
    top = top->next;
    free(tmp);
    return n;
}

int Top()
{
    return top->data;
}

int isempty()
{
    return top==NULL;
}

void display(node *head)
{
   if(head == NULL)
```

```
    {
        printf("NULL\n");
    }
    else
    {
        printf("%d\n", head -> data);
        display(head->next);
    }
}

int main()
{
    initialize();
      push(10);
      push(20);
      push(30);
      printf("The top is %d\n",Top());
      pop();
      printf("The top after pop is %d\n",Top());
      display(top);
```

```
    return 0;
}
```

OUTPUT:-

```
The top is 30
The top after pop is 20
20
10
NULL
```

47.))Objective:

At the end of this activity, we shall be able to

- Making a queue using a linked list is obviously a linked list.

Problem Statement:

The major problem with the queue implemented using an array is, It will work for an only fixed number of data values. That means, the amount of data must be specified at the beginning itself. Queue using an array is not suitable when we don't know the size of data which we are going to use. A queue data structure can be implemented using a linked list data structure.

Algorithm:

START

enQueue

Step 1 - Create a newNode with given value and set 'newNode → next' to NULL.

Step 2 - Check whether queue is Empty (rear == NULL)

Step 3 - If it is Empty then, set front = newNode and rear = newNode.

Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode.

deQueue

Step 1 - Check whether the queue is Empty (front == NULL).

Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and

terminate from the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.

Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

Display

Step 1 - Check whether the queue is Empty (front == NULL).

Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.

Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until

'temp' reaches to 'rear' (temp → next != NULL).

Step 5 - Finally! Display 'temp → data ---> NULL'.

INPUT:-

```
#include<stdio.h>

#include<conio.h>


struct Node

{

   int data;

   struct Node *next;

}*front = NULL,*rear = NULL;


void insert(int);

void delete();

void display();

void main()

{

   int choice, value;
```

```
    printf("\n:: Queue Implementation using Linked List ::\n");
    while(1){
        printf("\n****** MENU ******\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice){
                case 1: printf("Enter the value to be insert: ");
                        scanf("%d", &value);
                        insert(value);
                        break;
                case 2: delete(); break;
                case 3: display(); break;
                case 4: exit(0);
                default: printf("\nWrong selection!!! Please try
again!!!\n");
        }
    }
}
```

```
void insert(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode -> next = NULL;
   if(front == NULL)
        front = rear = newNode;
   else{
        rear -> next = newNode;
        rear = newNode;
   }
   printf("\nInsertion is Success!!!\n");
}
void delete()
{
   if(front == NULL)
     printf("\nQueue is Empty!!!\n");
  else{
     struct Node *temp = front;
```

```
        front = front -> next;
        printf("\nDeleted element: %d\n", temp->data);
        free(temp);
    }
}
void display()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        while(temp->next != NULL){
            printf("%d--->",temp->data);
            temp = temp -> next;
        }
        printf("%d--->NULL\n",temp->data);
    }
}
```

Output:-

Insertion is Success!!!

****** MENU ******

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 2

Deleted element: 10

****** MENU ******

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 3

20--->30--->NULL

****** MENU ******

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 4