| EX.NO:1 | BRANCHING AND ITERATIVE CONSTRUCTS |
|---|---|

**AIM:**

To write a C program to perform branching and iterative constructs.

**ALGORITHMS :**

**1. RESULT CALCULATION : PASS / FAIL**

Step 1 : START

Step 2 : DECLARE m1, m2, m3, m4, m5, per

Step 3 : GET m1, m2, m3, m4, m5

Step 4 : IF m1 >= 35 AND m2>=35 AND m3>=35 AND m4>=35 AND m5>=5 PRINT “PASS"

Step 5 : ELSE
PRINT “FAIL”

Step 6 : COMPUTE per = (m1 + m2 + m3 + m4 + m5) / 5

Step 7 : PRINT percentage per

Step 8 : STOP

**2. FIND THE GREATEST OF THREE NUMBERS**

Step 1 : START

Start 2 : GET A, B, C

Start 3 : LET max = A

Start 4 : IF B > max THEN max =B

Start 5 : IF C>max THEN max=C

Start 6 : PRINT (“%d is large”,max)

Start 7 : STOP

**3. FIND THE GIVEN NUMBER IS POSITIVE, NEGATIVE OR ZERO**

Step 1 : Start.

Start 2 : GET num

Start 3 : IF(num>0)

PRINT(“positive”)

Start 4 :ELSEIF (num<0)

PRINT(“Negative”)

Start 5 : ELSE

PRINT(“zero”)

Start 6 : STOP

**4. IDENTIFY THE GIVEN CHARACTER IS ALPHABET OR DIGIT OR SPECIAL CHARACTER.**

Step 1 : START

Step 2 : INPUT char

Step 3 : IF((c >= 'a' && c <= 'z')||(c >= 'A' && c <= 'Z'))

PRINT(“Alphabet”)

Step 4 : ELSE IF(c >= '0' && c <= '9')

PRINT(“Digit”)

Step 5 : ELSE

PRINT(“special char”)

Step 6 : STOP

**5. REVERSE THE GIVEN NUMBER**

Step 1 : START

Step 2 : INPUT n

Step 3 : INITIALIZE rev = 0

Step 4 : LOOP WHILE n>0

Step 5 : COMPUTE rev = rev * 10 + (n % 10)

Step 6 : COMPUTE n /= 10

Step 7 : END WHILE

Step 8 : PRINT rev

Step 9 : STOP

**6. PRINT THE FIRST DIGIT OF A GIVEN NUMBER**

Step 1 : START

Step 2 : GET num

Step 3 : WHILE n >= 10

Step 4 : COMPUTE n /= 10

Step 5 : END WHILE

Step 6 : PRINT ("The first digit is %d", n)

Step 7 : STOP

**7. CHECK IF THE NUMBER IS PERFECT OR NOT.**

Step 1 : START

Step 2 : DECLARE n, sum = 0

Step 3 : GET n

Step 4 : FOR(INT i=1; i<n; i++) DO

Step 5 : IF n%i == 0 THEN sum += i

Step 6 : END FOR

Step 7 : IF n == sum THEN

PRINT ("Perfect Number")

Step 8 : ELSE PRINT ("Not a Perfect Number")

Step 9 : STOP

**8. PRINT THE PATTERN**

Step 1 : START

Step 2 : DECLARE n

Step 3 : GET n

Step 4 : FOR (INT i = 1; i <= n; i++) DO

Step 5 : FOR(INT j = 1; j <= i; j++) DO PRINT ("*")

Step 6 : END FOR

Step 7 : PRINT("\n")

Step 8 : END FOR

Step 9 : STOP

**<u>PROGRAMS :</u>**

**1. RESULT CALCULATION : PASS / FAIL**

```
#include<stdio.h>
int main()
```

```
{
  int sub1,
  sub2,sub3,sub4,sub5; int
  total;
  float percentage;

  printf("Enter the five subject Marks:");
  scanf("%d %d %d %d %d",&sub1,&sub2,&sub3,&sub4,&sub5);
  if(sub1>50 && sub2>50 && sub3>50 && sub4>50 && sub5>50)
  {
     total=sub1+sub2+sub3+sub4+sub5;
     percentage=total*100 / 500;
     printf("Passed with  %.2f percentage",percentage);
  }
  else
  printf("Result is Fail");
  return 0;
}
```

**OUTPUT :**

```
Enter the five subject Marks:80 95 90 85 94
Passed with  88.00 percentage
```

**2. FIND THE GREATEST OF THREE NUMBERS**

```
#include <stdio.h>
int main()
  {
     int a,b,c;
     int largest;
     printf("Enter the value of a b and c:");
     scanf("%d %d %d",&a,&b,&c);
     if(a>b && a>c)
        printf("A is the largest number");
     else if(b>a && b>c)
```

```
        printf("B is the largest number");
      else
        printf("C is the largest number");
      return 0;
    }
```

**OUTPUT :**

```
Enter the value of a b and c:10 5 7
A is the largest number
```

**3. FIND THE GIVEN NUMBER IS POSITIVE, NEGATIVE OR ZERO**

```
#include <stdio.h>
int main()
{
   int num;
   char choice;
   printf("Enter an integer number :");
   scanf("%d",&num);
   if(num==0)
      printf("Number is ZERO.");
   else if(num>0)
      printf("Number is POSITIVE.");
    else
      printf("Number is NEGATIVE.");
    return 0;
}
```

**OUTPUT :**

```
Enter an integer number :-15
Number is NEGATIVE.
```

**4. IDENTIFY THE GIVEN CHARACTER IS ALPHABET OR DIGIT OR SPECIAL CHARACTER.**

```
#include <stdio.h>
int main()
{
```

```
    char ch;
    printf("Enter any character: ");
    scanf("%c", &ch);
    if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
      printf("'%c' is alphabet.", ch);
    else if(ch >= '0' && ch <= '9')
      printf("'%c' is digit.", ch);
    else
      printf("'%c' is special character.", ch);
    return 0;
  }
```

**OUTPUT :**

```
Enter any character: c
'c' is alphabet.
```

**5. REVERSE THE GIVEN NUMBER**

```
#include<stdio.h>
int main()
{
   int n, remind;
   int rev = 0;
   printf("Enter a number :");
   scanf("%d",&n);
   while (n != 0)
   {
      remind = n % 10;
      rev = rev * 10 + remind;
      n /= 10;
   }
   printf("\nThe Reversed Digit is %d",rev);
   return 0;
}
```

**OUTPUT :**

```
Enter a number :123

The Reversed Digit is 321
```

**6. PRINT THE FIRST DIGIT OF A GIVEN NUMBER.**

```
#include<stdio.h>
int main()
{
   int n;
   printf("Enter an integer: ");
   scanf("%d",&n);
   while (n >= 10)
      n /= 10;
   printf("The first digit of the given input is %d",n);
   return 0;
}
```

**OUTPUT :**

```
Enter an integer: 123
The first digit of the given input is 1
```

**7. CHECK IF THE NUMBER IS PERFECT OR NOT.**

```
#include<stdio.h>
int main()
{
   int n;
   int sum = 0;
   printf("Enter an integer:");
   scanf("%d",&n);
   for(int i=1; i<n; i++)
   {
      if (n % i == 0)
         sum += i;
   }
   if (n == sum)
      printf("The given number %d is a perfect number.",n);
   else
      printf("The given number %d is not a perfect number.",n);
   return 0;
}
```

**OUTPUT :**

```
Enter an integer:6
The given number 6 is a perfect number.
```

**8. PRINT THE PATTERN.**

```
#include<stdio.h>
int main()
{
    int n;
    printf("Enter the count :");
    scanf("%d",&n);
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= i; j++)
            printf("*");
        printf("\n");
    }
    return 0;
}
```

**OUTPUT :**

```
Enter the count :5
*
**
***
****
*****
```

**RESULT :**

Thus, the C program to perform branching and iterative constructs was executed and the output was verified successfully.

| EXP NO:2 | FUNCTIONS |
|---|---|

**AIM**

To write a C program to perform functions.

**ALGORITHMS**

**1. IDENTIFY THE GIVEN INPUT IS EITHER POSITIVE NEGATIVE OR ZERO USING FUNCTIONS**

Step 1 : Start

Step 2 : Prompt the user to enter a number.

Step 3 : Store the user entered value in a.

Step 4 : Declare and define a function num which takes single parameter of input number

Step 5: Call num function

Step 6: Construct an if else statement.

Step 7 : If the if part (a>0) is true ,then print the number is positive.

Step 8 : If the else if part (a<0) is true ,then print the number is negative.

Step 9 : If the else part is true, then print the number is zero

Step 10: End

**2. CHECK A GIVEN NUMBER IS A PERFECT NUMBER OR NOT USING FUNCTIONS**

Step 1: START

Step 2: INITIALIZE num, s

Step 3: GET num

Step 4: CALL perfect(num)function

Step 5: IF s==num THEN PRINT "Perfect number"

Step 6: ELSE PRINT "Not a Perfect number"

Step 7: STOP

perfect(n)

Step 1: INITIALIZE a=1, sum=0

Step 2: WHILE a<n

Step 3: IF n%a==0

Step 4: COMPUTE sum=sum+a

Step 5: COMPUTE a++

Step 6: RETURN sum

**3.PROGRAM TO ACCEPT AN INTEGER AND RETURN THE NUMBER OF FACTORS USING FUNCTIONS**

Step 1: Start

Step 2: Prompt the user to enter a number.

Step 3: Store the user entered value in number.

Step 4: Declare and define a function find_factors

Step 5: Call find_factors function

Step 6: Construct a for loop for (i = 1; i <= Number; i++)

Step 7: Using if statement if the entered number is divisible by 0,then print the number.

Step 8: Print the factors of the entered number.

Step 9: End

**4. PROGRAM TO SUM OF ITS DIGIT UP TO A SINGLE VALUE USING RECURSION**

Step 1: Start

Step 2: Read a input number

Step 3: Declare and define a function which takes single parameter of input number

Step 4: Call sum(value) function

Step 5: Print the value of total Step 5: Stop

sum(n)

Step 1: If n not equal to 0, then call the function by summing the least digit using modulo operator

Step 2: Set the condition where if total>9 and number !=0, if it is true then convert total value into 'n' and call the function again as a recursive one.

Step 3: Return the total value

**5. PROGRAM TO CONVERT DECIMAL NUMBER TO BINARY NUMBER USING RECURSION**

Step 1: Start

Step 2: Get input from the user(decimal number)

Step 3: Declare and define a function [convert(int dec)] which takes a single parameter of input number.

Step 4: Call convert(dec)function

Step 5: Print the binary equivalent

convert (dec)

Step 1: Decide the input decimal number by 2 and store the remainder

Step 2: Store the quotient back to the input number variable

Step 3: Repeat the process till quotient becomes zero

**PROGRAMS**

**1.IDENTIFY THE GIVEN INPUT IS EITHER POSITIVE NEGATIVE OR ZERO USING FUNCTIONS**

```
#include<stdio.h>
void num(int a);
void main()
{
   int a;
   printf("Enter the number: ");
   scanf("%d",&a);
   num(a);
}
void num(int a)
{
   if(a>0)
   {
      printf("The number is positive");
   }
   else if(a<0)
   {
      printf("The number is negative");
   }
   else
   {
      printf("The number is zero");
   }
}
```

**OUTPUT:**

```
Enter the number: 90
The number is positive

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

**2. CHECK A GIVEN NUMBER IS A PERFECT NUMBER OR NOT USING FUNCTIONS**

```
#include <stdio.h>
int main()
{
   int i, num, sum = 0;
   printf("Enter a number: ");
   scanf("%d", &num);
   for(i = 1; i <= num / 2; i++)
   {
      if(num%i == 0)
      {
         sum=sum+ i;
      }
   }
   if(sum == num && num > 0)
   {
      printf("%d is a \" PERFECT NUMBER\"", num);
   }
   else
   {
      printf("%d is \"NOT PERFECT NUMBER\"", num);
   }

   return 0;
}
```


```
#include <stdio.h>
int main()
{
   int i, num, sum = 0;
   printf("Enter a number: ");
   scanf("%d", &num);
   for(i = 1; i <= num / 2; i++)
   {
      if(num%i == 0)
      {
         sum=sum+ i;
      }
   }
   if(sum == num && num > 0)
   {
      printf("%d is a \" PERFECT NUMBER\"", num);
   }
   else
   {
      printf("%d is \"NOT PERFECT NUMBER\"", num);
   }

   return 0;
}
```

**OUTPUT:**

```
Enter a number: 7
7 is "NOT PERFECT NUMBER"

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

**3.PROGRAM TO ACCEPT AN INTEGER AND RETURN THE NUMBER OF FACTORS USING FUNCTIONS**

```
#include <stdio.h>
void Find_Factors(int);
int main()
{
  int Number;
  printf("\nPlease Enter number to Find Factors\n");
  scanf("%d", &Number);
  printf("\nFactors of a Number are:\n");
  Find_Factors(Number);
  return 0;
}
void Find_Factors(int Number)
{
  int i;
  for (i = 1; i <= Number; i++)
   {
    if(Number%i == 0)
     {
      printf("%d ", i);
     }
   }
}
```

**OUTPUT:**

```
Please Enter number to Find Factors
100

Factors of a Number are:
1 2 4 5 10 20 25 50 100

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

## 4. PROGRAM TO SUM OF ITS DIGIT UP TO A SINGLE VALUE USING RECURSION

```
#include<stdio.h>
int main()
{
    long int num;
    int sum = 0, rem;
    printf("Enter a number: ");
    scanf("%ld", &num);
    while(num / 10 != 0)
    {
        sum = 0;
        while(num != 0)
        {
            rem = num % 10;
            sum += rem;
            num = num / 10;
        }
        num = sum;
    }
    printf("%d", sum);
    return 0;
}
```

**OUTPUT:**

```
Enter a number: 5678
8

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

**5. PROGRAM TO CONVERT DECIMAL NUMBER TO BINARY NUMBER USING RECURSION**

```
#include<stdio.h>
int binary_rec(int);
int binary(int);
int main()
{
   int num;
   printf("Enter a Decimal number:");
   scanf("%d", &num);
   printf("Binary Equivalent (Iterative) of %d is %d\n", num, binary(num));
   printf("Binary Equivalent (Recursive) of %d is %d\n", num, binary_rec(num));
   return 0;
}
int binary_rec(int num)
{
   if(num == 0)
      return 0;
   else
      return((num % 2) + 10 * binary_rec(num/2));
}
int binary(int num)
{
   int rem, bin = 0, place = 1;
   while(num)
   {
      rem   = num % 2;
      num   = num / 2;
      bin   = bin + (rem * place);
      place = place * 10;
   }
   return(bin);
}
```

**OUTPUT:**

```
Enter a Decimal number:15.5
Binary Equivalent (Iterative) of 15 is 1111
Binary Equivalent (Recursive) of 15 is 1111


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

**RESULT:**

Thus the program has been executed and tested successfully.

| EX.NO: 3 | ARRAYS AND STRINGS |
|---|---|

**AIM:**

To write a C Program to perform operations on Arrays and Strings.

**ALGORITHMS:**

1. **PRINT THE SECOND LARGEST NUMBER IN AN ARRAY**

Step:1: Start

Step:2: Initialize arr[100], size, i, max1, max2

Step:3: Get size

Step:4: Iterate a 'for' loop using 'i' from 0th index to last index

Step:5: Get arr[i]

Step:6: Set max1 = max2 = INT_MIN

Step:7: Iterate a 'for' loop using 'i' from 0th index to last index

Step:8: If arr[i] > max1

Step:9: Set max2 = max1, max1 = arr[i]

Step:10: Else If arr[i] > max2 && arr[i] < max1

Step:11: Set max2 = arr[i]

Step:12: Print max2

Step:13: Stop

2. **IDENTIFY MISSING NUMBER IN AN ARRAY**

Step:1: Start

Step:2: Read size for an array

Step 3: Declare supportive variables for loop

Step 4: Read runtime inputs for one dimensional array using a 'for' loop

Step 5: While reading input through scanf, identify smallest and largest

Step 6: Use, another loop for summing all the elements of an array

Step:7: Initialize the variable of t_sum and s_sum for getting answer for missing number

Step 8: Utilize the formula sum of ‘N’ numbers to produce the answer

Step:9: Print the result of missing number

Step:10: Stop

**3. REMOVE A PARTICULAR ELEMENT IN AN ARRAY**

Step:1: Start

Step:2: Set index value as -1 initially. i.e. index = -1

Step:3: Get key value from the user which needs to be deleted.

Step:4: Search and store the index of a given key

[ index will be -1, if the given key is not present in the array]

Step:5: If index not equal to -1then

Step:6: Shift all the elements from index + 1 by 1 position to the left.

Step:7: Else then print "Element Not Found"

Step:8: Stop

**4. PRINT THE LARGEST NUMBER IN EVERY COLUMN OF A MATRIX**

Step:1: Start

Step:2: Initialize m, n, i, j, mat1[m][n]

Step:3: Get m,n

Step:4: Iterate a ‘for’ loop using ‘i’ from 0th index to m

Step:5: Iterate a ‘for’ loop using ‘j’ from 0th index to n

Step:6: Get mat1[i][j]

Step:7: Call maxi_col(mat1,m,n) function

Step:8: Stop

maxi_col(mat1,m,n) function:

Step:1: Initialize i, j

Step:2: Iterate a ‘for’ loop using ‘i’ from 0th index to n

Step:3: Declare max = mat[0][i]

Step:4: Iterate a ‘for’ loop using ‘j’ from 0th index to n

Step:5: If mat[j][i] > max then

Step:6: Set max = mat[j][i]

Step:7: End If

Step:8: Print max

**5. CHECK THE EQUALITY OF TWO STRINGS WITHOUT USING PREDEFINED FUNCTIONS**

Step 1: Start.

Step 2: Read size separately for two character array

Step 3: Scan input in runtime for array1 and array2

Step 4: Declare supportive variable for a while loop ‘i'

Step 5: Set i value as 0, Check condition as array1 should not be ‘\0 and array2 should not be ‘\0’

Step 6: Within while loop, if array[i] !=array2[i] Then Increment the counter variable and break the loop

Step 7: End While

Step 8: If count ==0 , print “Both strings are same” Else print “Both strings are not same”

Step 9: Stop

**6. REMOVE UNWANTED WHITE SPACES IN A SENTENCE**

Step:1: Start

Step:2: Initialize inputString[100], outputArray[100], readIndex = 0, writeIndex

Step:3: Input inputstring

Step:4: Within while loop, if inputString[readIndex] == ' ' Then Increment the readIndex variable and break the loop

Step:5: Iterate a ‘for’ loop using ‘readIndex’ from 0th index to n

Step:6: if “inputString[readIndex]==' ' && inputString[readIndex-1]==' ' ” then

Step:7: continue

Step:8: Set outputArray[writeIndex] = inputString[readIndex]

Step:9: Compute writeIndex++

Step:10: Set outputArray[writeIndex] = '\0'

Step:11: Print outputArray

Step:12: Stop

7. **PRINT THE STRING WITHOUT VOWELS**

Step:1: Start

Step:2: Initialize the variables str[100], i, j, len=0

Step:3: Accept the input str

Step:4: Iterate a 'for' loop using 'i' from 0th index to len

Step:5:If

"str[i]=='a'||str[i]=='e'||str[i]=='i'||str[i]=='o'||str[i]=='u'||str[i]=='A'||str[i]=='E'||str[i]=='I'||str[i]=='O'||str[i]=='U' "

Step:6: Iterate a 'for' loop using 'j' from 0th index to len

Step:7: Compute str[j]=str[j+1]

Step:8: Compute len--

Step:9: Terminate both for loop.Print the string without vowels.

Step:10:Stop

**PROGRAMS:**

1. **PRINT THE SECOND LARGEST NUMBER IN AN ARRAY**

```
#include <stdio.h>
#include <limits.h>
 int main()
{
int arr[100], size, i;
int max1, max2;
printf("Enter size of the array: ");
scanf("%d", &size);
printf("Enter elements in the array: ");
for(i=0; i<size; i++)
{
      scanf("%d", &arr[i]);
}
```

```
max1 = max2 = INT_MIN;
for(i=0; i<size; i++)
{
    if(arr[i] > max1)
    {
        max2 = max1;
        max1 = arr[i];
    }
    else if(arr[i] > max2 && arr[i] < max1)
    {

        max2 = arr[i];
    }
}
printf("Second largest = %d", max2);

return 0;
}
```

**OUTPUT:**

```
Enter size of the array: 5
Enter elements in the array: 56 43 72 54 16
Second largest = 56
```

**2. IDENTIFY MISSING ELEMENT IN AN ARRAY**

```
#include <stdio.h>
int main ()
{
int number[30];
int i, n,sum=0,small,big;
printf("Enter the value of N:");
scanf("%d", &n);
```

```
printf("Enter the numbers:");
for (i = 0; i < n; ++i){
    scanf("%d", &number[i]);
    if(i==0)
    {
        sum=sum+number[i];
        small=number[i];
        big=number[i];
        continue;
    }
    if(number[i]<small)
        small=number[i];

    if(number[i]>big)
        big=number[i];
        sum=sum+number[i];
    }
int t_sum=(big*(big+1))/2;
int s_sum=((small-1)*(small))/2;
int ans=t_sum - (s_sum+sum);
printf("Missing number is :%d",ans);
return 0;
    }
```

**OUTPUT:**

```
Enter the value of N:5
Enter the numbers:10 15 13 12 14
Missing number is :11
```

3. **REMOVE A PARTICULAR ELEMENT IN AN ARRAY**

```
#include<stdio.h>
int main()
{
    int key, i, index = -1,size;
```

```
    int arr[size];
    printf("Enter no of elements in an array:");
    scanf("%d", &size);
    printf("Enter elements in an array:");
    for(i=0;i<size;i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("Enter element to delete\n");
    scanf("%d",&key);

    for(i = 0; i < size; i++)
    {
        if(arr[i] == key)
        {
            index = i;
            break;
        }
    }

    if(index != -1)
    {
        for(i = index; i < size - 1; i++)
            arr[i] = arr[i+1];
        printf("New Array : ");
        for(i = 0; i < size - 1; i++)
            printf("%d ",arr[i]);
    }
    else
    printf("Element Not Found\n");

    return 0;
}
```

**OUTPUT:**

```
Enter no of elements in an array:4
Enter elements in an array:54 67 45 34
Enter element to delete
67
New Array : 54 45 34
```

**4. PRINT THE LARGEST NUMBER IN EVERY COLUMN OF A MATRIX**

```
#include<stdio.h>
void maxi_col(int mat[][3], int m, int n)
{
   printf("The largest number in every column:\n");
   int i,j;
   for(i = 0; i < n; i++)
   {
        int max = mat[0][i];
        for(j = 1; j < m; j++)

        {
                if(mat[j][i] > max)
                        {
                                max = mat[j][i];
                        }
        }
   printf("%d\n",max);
   }
}
int main()
{
        int m, n;
        printf("Enter row and column:");
        scanf("%d %d",&m,&n);
        int i, j;
```

```
        int mat1[m][n];
        printf("Enter elements:\n");
        for(i = 0; i < m; i++)
            {
                for(j = 0; j < n; j++)
                    scanf("%d",&mat1[i][j]);
            }
        maxi_col(mat1,m,n);
    return 0;
    }
```

**OUTPUT:**

```
Enter row and column:3 3
Enter elements:
45 6 75
87 45 2
9 56 23
The largest number in every column:
87
56
75
```

5. **CHECK THE EQUALITY OF TWO STRINGS WITHOUT USING PREDEFINED FUNCTIONS**

```
#include<stdio.h>
int main()
{
char str1[20], str2[20];
int i=0, c=0;
printf("Enter first string :");
gets(str1);
printf("Enter second string :");
gets(str2);
while((str1[i]!='\0') || (str2[i]!='\0'))
{
```

```
if(str1[i]!=str2[i]){
c++;
break;
}
i++;
}
if(c==0)
printf("Strings are equal.");
else
printf("Strings are not equal.");
return 0;
}
```

**OUTPUT:**

```
Enter first string :Hello
Enter second string :Hello
Strings are equal.
```

6. **REMOVE UNWANTED WHITE SPACES IN A SENTENCE**

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

int main(){
char inputString[100], outputArray[100];
int readIndex = 0, writeIndex;
printf("Enter a String \n");
gets(inputString);
while(inputString[readIndex] == ' '){
    readIndex++;
}
```

```
 for(writeIndex = 0;inputString[readIndex] != '\0'; readIndex++){
     if(inputString[readIndex]==' ' && inputString[readIndex-1]==' '){
         continue;
     }
     outputArray[writeIndex] = inputString[readIndex];
     writeIndex++;
  }
  outputArray[writeIndex] = '\0';
  printf("String without extra spaces\n%s", outputArray);

  return 0;
}
```

**OUTPUT:**

```
Enter a String
I      am    an   Indian
String without extra spaces
I am an Indian
```

7. **PRINT THE STRING WITHOUT VOWELS**

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[100];
        int i,j,len=0;
        printf("Enter a string : ");
        gets(str);
        len=strlen(str);
        for(i=0; i<len; i++)
        {

if(str[i]=='a'||str[i]=='e'||str[i]=='i'||str[i]=='o'||str[i]=='u'||str[i]=='A'||str[i]=='E'||str[i]=='I'||str[i]=='O'||str[
i]=='U')
```

```
                {
                        for(j=i; j<len; j++)
                        {
                                str[j]=str[j+1];
                        }
                len--;
                }
        }
        printf("After deleting the vowels, the string will be : %s",str);


    return 0;
}
```

**OUTPUT:**

```
Enter a string : engineer
After deleting the vowels, the string will be : ngner
```

**RESULT:**

Thus, the program implementation using strings and arrays has been executed and tested successfully.

| EX.NO: 4 | STRUCTURES AND POINTERS |
|---|---|

**AIM:**

To write a C program to perform structures and pointers.

**ALGORITHMS :**

**1.STUDENTS INFORMATION {NAME, ROLL_NO, DEPART, MOBILE}**

Step 1: Start.

Step 2: Define a structure with basic fields like name, roll_no, department and mobile_no

Step3: Declare a size for a structure variable

Step 4: Declare a structure variable and collect run-time input through a 'for' loop

Step 5: Use another loop for printing the 'N' number of students information using structure variable and member access operators

Step 6: Stop

**2.TOP-3 RANKS IN A CLASS**

Step 1: START

Step 2: INITIALIZE arr[] ={5, 2, 8, 7, 1 }.

Step 3: SET temp =0

Step 4: length= sizeof(arr)/sizeof(arr[0])

Step 5: PRINT "Elements of Original Array"

Step 6: SET i=0. REPEAT STEP 7 and STEP 8 UNTIL i<length

Step 7: PRINT arr[i]

Step 8: i=i+1.

Step 9: SET i=0. REPEAT STEP 10 to STEP 13 UNTIL i<n

Step 10: SET j=i+1. REPEAT STEP 11 and STEP 12 UNTIL j<length

Step 11: if(arr[i]

temp = arr[i]

arr[i]=arr[j]

arr[j]=temp

Step 12: j=j+1.

Step 13: i=i+1.

Step 14: PRINT new line

Step 15: PRINT "Elements of array sorted in ascending order"

Step 16: SET i=0. REPEAT STEP 17 and STEP 18 UNTIL i<length

Step 17: PRINT arr[i], uptoarr[2]

Step 18: i=i+1.

Step 19: RETURN 0.

Step 20: END.

**3.COPY ONE ARRAY TO ANOTHER USING POINTER**

Step 1: Start.

Step 2: Input size and elements in first array, store it in some variable say size and source_array.

Step 3: Declare another array say dest_array to store copy of source_array.

Step 4: Declare a pointer to source_array say *source_ptr = source_array and one more pointer to dest_array say *dest_ptr = dest_array.

Step 5: Copy elements from source_ptr to desc_ptr using *desc_ptr = *source_ptr.

Step 6: Increment pointers source_ptr and desc_ptr by 1.

Step 7: Repeat step 4 and 5 till source_ptr exists in source_arr memory range.

Step 8: Print the dest_ptr elements using a 'for' loop

Step 9: Stop

**4.SEARCH OF A CHARACTER**

Step 1: Start.

Step 2: Declare all the variables for string input and looping statements

Step 3: Collect the input string from the user

Step 4: Calculate the length of input string

Step 5: Get an runtime input character to search for its frequency

Step 6: Iterate a 'for' loop using 'i' from $0^{th}$ index to last index

Step 7: If input character is equal to the read character array, then increment the counter variable by 1

Step 8: Terminate the loop if it reaches last position

Step 9: Display the frequency of an input character

Step 10: Stop

**5.SWAP TWO NUMBERS USING CALL BY REFERENCE**

Step 1: Start.

Step 2: Declare two variables n1 and n2

Step 3: Get input in runtime using scanf() function

Step 4: Decalre and define a function with two parameters

Step 5: From main method, Call a function of swap(&int,&int)

Step 6:In function definition,

Use function formal parameter as a pointer variable in which swap the these two variables using temporary variable

Step 7: Updation in function definition will be reflected in main method

Step 8: Display the output of swapped values using printf function

Step 9:Stop

**PROGRAMS**

**1.STUDENTS INFORMATION {NAME, ROLL_NO, DEPART, MOBILE}**

```
#include<stdio.h>

int main()
  {
    char name[20],dept[20];
    int roll_No,age;
    long long unsigned int m_No;
    printf("Enter your name:");
    gets(name);
    printf("Enter your roll no:");
    scanf("%d",&roll_No);
    printf("Enter your age:");
    scanf("%d",&age);
    printf("Enter your mobile no:");
    scanf("%llu",&m_No);
    printf("Enter your department name:");
    scanf("%s",dept);
    printf("Name is: %s",name);
    printf("\nRoll Number is: %d",roll_No);
    printf("\nDepartment Name is: %s",dept);
    printf("\nAge is: %d",age);
    printf("\nMobile Number is: %llu",m_No);
```

```
return 0;
}
```

**OUTPUT:**

```
Enter your name:Akilesh
Enter your roll no:23
Enter your age:18
Enter your mobile no:9454443443
Enter your department name:EEE
Name is: Akilesh
Roll Number is: 23
Department Name is: EEE
Age is: 18
Mobile Number is: 9454443443
```

**2.TOP-3 RANKS IN A CLASS:**

```
#include <stdio.h>

struct student
{
    char Name[100];
    int total;
};

void main()
{
    int i, n, max = 0, temp = 0;
    printf("Enter the number of students :");
    scanf("%d", &n);
    struct student stud[n];
    for (i = 0; i < n; i++)
    {
        printf("\nEnter the student %d name :",
i + 1);
        scanf("%s", stud[i].Name);
        printf("\nEnter the student %d total :", i
+ 1);
        scanf("%d", &stud[i].total);
```

```
        }
        for (i = 0; i < n; i++)
        {
            if (stud[i].total > max)
            {
                max = stud[i].total;
                temp = i;
            }
        }
        printf("\n First ranker is %s",
stud[temp].Name);
        stud[temp].total = 0;
        max = 0;
        for (i = 0; i < n; i++)
        {
            if (stud[i].total > max)
            {
                max = stud[i].total;
                temp = i;
            }
        }
        printf("\n second ranker is %s",
stud[temp].Name);
        stud[temp].total = 0;
        max = 0;
        for (i = 0; i < n; i++)
        {
            if (stud[i].total > max)
            {
                max = stud[i].total;
                temp = i;
            }
        }
        printf("\n third ranker is %s",
stud[temp].Name);
```

```
    }



}
```

**OUTPUT:**

```
Enter the student 1 name :sri

Enter the student 1 total :555

Enter the student 2 name :neveda

Enter the student 2 total :444

Enter the student 3 name :suba

Enter the student 3 total :333

 First ranker is sri
 second ranker is neveda
 third ranker is suba
```

**3.COPY ONE ARRAY TO ANOTHER USING POINTER:**

```
#include<stdio.h>
void main()
{
   int arr[10], n =
10, i, res[n];
   for (i = 0; i < n;
i++)
   {
      printf("Enter
the %d th
number:", i + 1);
      scanf("%d",
&arr[i]);
   }
```

```
    for (i = 0; i < n;
i++)
    {
        res[i] = *(arr
+ i);
    }


    for (i = 0; i < n;
i++)
    {
        printf("The
number is %d \n",
res[i]);
    }





}
```

**OUTPUT:**

```
Enter the 1 th number:1
Enter the 2 th number:2
Enter the 3 th number:3
Enter the 4 th number:4
Enter the 5 th number:5
Enter the 6 th number:6
Enter the 7 th number:7
Enter the 8 th number:8
Enter the 9 th number:9
Enter the 10 th number:10
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

**4.SEARCH OF A CHARACTER:**

```
#include <stdio.h>
#include <string.h>
void main()
{
    char usr_str[100], compare[1], *stp;
    int i, n, count = 0;
    printf("Enter the string :");
    scanf("%c",&usr_str);
    printf("Enter the char :");
    scanf("%c", compare);
    n = strlen(usr_str);
    stp = &compare;
    for (i = 0; i < n; i++)
    {
        if(*stp == usr_str[i])
        {
            count += 1;
        }
    }
    printf("The count is %d", count);

}
```

**OUTPUT:**

```
Enter a string : department
Enter the character to be searched : e
Occurence of character 'e' : 2
```

**5. SWAP TWO NUMBERS USING CALL BY REFERENCE:**

```
#include <stdio.h>
void swp(int*, int*);
int main()
{
  int x, y;
  printf("Enter the value of x and y\n");
  scanf("%d %d",&x,&y);
  printf("Before Swapping\nx = %d\ny = %d\n", x, y);
  swp(&x, &y);
```

```
  printf("After Swapping\nx = %d\ny = %d\n", x, y);
  return 0;
}
void swp(int *a, int *b)
{
  int temp;
  temp = *b;
  *b = *a;
  *a = temp;
}
```

**OUTPUT:**

```
Enter the value of x and y
2  22
Before Swapping
x = 2
y = 22
After Swapping
x = 22
y = 2
```

**RESULT:**

Thus the program implementation using structures and pointers has been executed and tested successfully.

| EX.NO:5 | PROGRAMS USING FILES |
|---|---|

**AIM:**

To write a C program to perform operation using files

**ALGORITHMS**

**1. PROGRAM TO READ THE CONTENT OF A FILE AND DISPLAY**

Step 1:START

Step 2: OPEN a file using fopen() function and store its

reference in a FILE pointer say fp.

Step 3:READ content from file using read option.

Step 4: CLOSE the file using fclose(fp) .

Step 5:STOP

**2. PROGRAM TO COPY THE CONTENT OF ONE FILE TO ANOTHER FILE**

Step 1:START

Step 2:OPEN a file using fopen() with *fp1 in

read mode

Step 3:OPEN a file using fopen() with *fp2 in

write mode

Step 4:GET the content of the file using getc()

Step 5:COPY the content of the file using putc()

Step 6:The file is copied

Step 7:CLOSE both files using

fclose(fp1),fclose(fp2)

Step 8:STOP

**3. PROGRAM TO CAPITALIZE FIRST LETTER OF EVERY WORD IN A FILE**

Step 1:START

Step 2:DECLARE ch,pre

Step 2:OPEN a file using fopen() with *fp in read mode

Step 3: OPEN a file using fopen() with *fp2 in write mode.

Step 4: GET the content of the file fp using

getc()

Step 5**:** ch = pre - 32;

putc(ch, fp2);

boo = 0; to capitalize the first letter of every word

Step 6:STOP

**4. PROGRAM TO FIND THE NUMBER OF WORDS AND LINES IN A TEXT FILE**

Step 1:START

Step 2:DECLARE count=1,lines=1

Step 3:OPEN a file using fopen() with fp* in read mode

Step 4:GET the content of the file using getc()

Step 5**:** if(c ==' ' || c == '\n')

Step 6 :INCREMENT ,count

Step 7: if(c=='\n')

Step 8:INCREMENT, lines

Step 9:DISPLAY,count,lines

Step 10:STOP

**PROGRAMS**

**1. PROGRAM TO READ THE CONTENT OF A FILE AND DISPLAY**

```
#include<stdio.h>
int main()
{
   FILE *fp;
   fp=fopen("C:/Users/dava/Desktop/s1.txt","r");
   if(fp==NULL)
   printf("file is not present");
   else
   printf("Opening the file\n");
   fclose(fp);
   return 0;
   }
```

**OUTPUT:**

**2. PROGRAM TO COPY THE CONTENT OF ONE FILE TO ANOTHER FILE**

```
#include <stdio.h>
int main()
{
FILE *fp1, *fp2;
char ch;
fp1 = fopen("E:/c program/sample.txt","r");
fp2 = fopen("E:/c program/sample2.txt","w");
do {
ch = getc(fp1);
putc(ch, fp2);
}while(ch != EOF);
fclose(fp1);
fclose(fp2);
printf("File copied successfully...");
}
```

**OUTPUT:**

**3. PROGRAM TO CAPITALIZE FIRST LETTER OF EVERY WORD IN A FILE**

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

int main()
{
    FILE *fp, *fp2;
    char pre, ch;
    int boo = 1;
    fp = fopen("C:/Users/dava/Desktop/sample.txt", "r");
    fp2 = fopen("C:/Users/dava/Desktop/sample1.txt", "w");
    do
    {
        pre = getc(fp);
        if (!boo)
        {
```

```
            putc(pre, fp2);
        }
        if(boo)
        {
            ch = pre - 32;
            putc(ch, fp2);
            boo = 0;
        }
        if (pre == ' ')
        {
            boo = 1;
        }

    }while (pre != EOF);
     printf("Operation Successful");
    fclose(fp);}
```

**OUTPUT:**

**4. PROGRAM TO FIND THE NUMBER OF WORDS AND LINES IN A TEXT FILE**

```
#include<stdio.h>
int main()
{   char c;
    FILE *f;
    int count = 1;
    int lines=1;
    f = fopen("sample.txt","r");
    do{
        c=getc(f);
        if(c ==' ' || c == '\n')
            count++;
```

```
        if(c=='\n')
            lines++;
    }while(c!=EOF);
    printf("Number of words: %d\n", count);
    printf("Number of lines: %d",lines);
    fclose(f);

    return 0;
}
```

**OUTPUT:**

**RESULT:**

Thus the program implementation using files has been executed and tested successfully.

| EXPNO: 6 | SINGLY,DOUBLY AND CIRCULAR LINKED LIST |
|---|---|

**AIM :**

To write a C program to perform operations on singly, doubly and circular linked lists.

**ALGORITHMS :**

**1. SINGLY LINKED LIST**

**A) CREATION**

Step 1 : START

Step 2 : Include all the header files which are used in the program.

Step 3 : Declare all the user defined functions.

Step 4 : Define a Node structure with two members data and next

Step 5 : Define a Node pointer 'head' and set it to NULL.

Step 6 : Implement the main method by displaying the operations menu and make suitable function calls in the main method to perform user selected operations.

**B) INSERTION**

**Inserting At Beginning of the list**

Step 1 : Create a newNode with a given value.

Step 2 : Check whether list is Empty (head == NULL)

Step 3 : If it is Empty then, set newNode→next = NULL and head = newNode.

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

**Inserting At End of the list**

Step 1 - Create a newNode with given value and newNode → next as NULL.

Step 2 - Check whether the list is Empty (head == NULL).

Step 3 - If it is Empty then, set head = newNode.

Step 4 : If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 : Keep moving the temp to its next node until it reaches the last node in the list (until temp → next is equal to NULL).

Step 6 : Set temp → next = newNode.

**Inserting At Specific location in the list (After a Node)**

Step 1 : Create a newNode with a given value.

Step 2 : Check whether list is Empty (head == NULL)

Step 3 : If it is Empty then, set newNode → next = NULL and head = newNode.

Step 4 : If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 : Keep moving the temp to its next node until it reaches the node after which we want to insert the newNode.

Step 6 : Every time check whether temp is reached to the last node or not. If it is reached to the last node then displays 'Given node is not found in the list!!!' and terminate the function. Otherwise move the temp to the next node.

Step 7 : Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode'

**C) DELETION**

**Deleting from Beginning of the list**

Step 1 : Check whether list is Empty (head == NULL)

Step 2 : If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminates the function.

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

Step 4 : Check whether list is having only one node (temp → next == NULL)

Step 5 : If it is TRUE then set head = NULL and delete temp.

Step 6 : If it is FALSE then set head = temp → next, and delete temp.

**Deleting from End of the list**

Step 1 : Check whether list is Empty (head == NULL)

Step 2 : If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminates the function.

Step 3 : If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 : Check whether list has only one Node (temp1 → next == NULL)

Step 5 : If it is TRUE. Then, set head = NULL and delete temp1. And terminate the function.

Step 6 : If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until it reaches the last node in the list.

Step 7 : Finally, Set temp2 → next = NULL and delete temp1.

**Deleting a Specific Node from the list**

Step 1 : Check whether list is Empty (head == NULL)

Step 2 : If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminates the function.

Step 3 : If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 : Keep moving the temp1 until it reaches the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.

Step 5 : If it is reached to the last node then display 'Given node not found in the list!'. And terminate the function.

Step 6 : If it is reached to the exact node which we want to delete, then check whether list is having only one node or not

Step 7 : If the list has only one node and that is the node to be deleted, then set head = NULL and delete temp1 (free(temp1)).

Step 8 : If the list contains multiple nodes, then check whether temp1 is the first node in the list.

Step 9 : If temp1 is the first node then move the head to the next node (head = head → next) and delete temp1.

Step 10 : If temp1 is not first node then check whether it is last node in the list.

Step 11 : If temp1 is the last node then set temp2 → next = NULL and delete temp1.

Step 12 : If temp1 is not the first node and not the last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

**D) SEARCH**

Step 1 : Initialise the Current pointer with the beginning of the List.

Step 2 : Compare the KEY value with the Current node value; if they match then quit there else go to step-3.

Step 3 : Move the Current pointer to point to the next node in the list and go to step-2, till the list is not over or else quit.

**E) DISPLAY**

Step 1 : Check whether list is Empty (head == NULL)

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

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

Step 4 : Keep displaying temp → data with an arrow (--->) until temp reaches to the last node

Step 5 : Finally display temp → data with an arrow pointing to NULL (temp → data ---> NULL).

**2. DOUBLY LINKED LIST**

**A) CREATION**

Step 1 - Include all the header files which are used in the program.

Step 2 - Declare all the user defined functions.

Step 3 - Define a Node structure with two members data and next

Step 4 - Define a Node pointer 'head' and set it to NULL.

Step 5 - Implement the main method by displaying the operations menu and make suitable function calls in the main method to perform user selected operations.

**B) INSERTION**

**Inserting At Beginning of the list**

Step 1 - Create a newNode with given value and newNode → previous as NULL.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, assign NULL to newNode → next and newNode to head.

Step 4 - If it is not Empty then, assign head to newNode → next and newNode to head.

**Inserting At End of the list**

Step 1 - Create a newNode with given value and newNode → next as NULL.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty, then assign NULL to newNode → previous and newNode to head.

Step 4 - If it is not Empty, then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches the last node in the list (until temp → next is equal to NULL).

Step 6 - Assign newNode to temp → next and temp to newNode → previous.

**Inserting At Specific location in the list (After a Node)**

Step 1 - Create a newNode with a given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, assign NULL to both newNode → previous & newNode → next and set newNode to head.

Step 4 - If it is not Empty then, define two node pointers temp1 & temp2 and initialize temp1 with head.

Step 5 - Keep moving the temp1 to its next node until it reaches the node after which we want to insert the newNode.

Step 6 - Every time check whether temp1 is reached to the last node. If it is reached to the last node then display 'Given node is not found in the list!!! Insertion is not possible!!!' and terminate the function. Otherwise move the temp1 to the next node.

Step 7 - Assign temp1 → next to temp2, newNode to temp1 → next, temp1 to newNode → previous, temp2 to newNode → next and newNode to temp2 → previous.

**C) DELETION**

**Deleting from Beginning of the list**

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminates the function.

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

Step 4 - Check whether list is having only one node (temp → previous is equal to temp → next)

Step 5 - If it is TRUE, then set head to NULL and delete temp (Setting Empty list conditions)

Step 6 - If it is FALSE, then assign temp → next to head, NULL to head → previous and delete temp.

**Deleting from End of the list**

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty, then display 'List is Empty!!! Deletion is not possible' and terminates the function.

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

Step 4 - Check whether list has only one Node (temp → previous and temp → next both are NULL)

Step 5 - If it is TRUE, then assign NULL to head and delete temp. And terminate from the function.

Step 6 - If it is FALSE, then keep moving temp until it reaches the last node in the list. (until temp → next is equal to NULL)

Step 7 - Assign NULL to temp → previous → next and delete temp.

**Deleting a Specific Node from the list**

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

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

Step 4 - Keep moving the temp until it reaches the exact node to be deleted or to the last node.

Step 5 - If it is reached to the last node, then display 'Given node not found in the list! Deletion not possible!!!' and terminate the function.

Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not

Step 7 - If list has only one node and that is the node which is to be deleted then set head to NULL and delete temp (free(temp)).

Step 8 - If list contains multiple nodes, then check whether temp is the first node in the list (temp == head).

Step 9 - If temp is the first node, then move the head to the next node (head = head → next), set head of previous to NULL (head → previous = NULL) and delete temp.

Step 10 - If temp is not the first node, then check whether it is the last node in the list.

Step 11 - If temp is the last node then set temp of previous or next to NULL.

Step 12 - If temp is not the first node and not the last node, then set temp of previous of next to temp of next , temp of next of previous to temp of previous.

**Displaying a Double Linked List**

Step 1 - Check whether list is Empty (head == NULL)

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

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

Step 4 - Display 'NULL <--- '.

Step 5 - Keep displaying temp → data with an arrow (<===>) until temp reaches to the last node

Step 6 - Finally, display temp → data with an arrow pointing to NULL (temp → data ---> NULL).

**3. CIRCULAR LINKED LIST**

**A) CREATION**

Step 1 - Include all the header files which are used in the program.

Step 2 - Declare all the user defined functions.

Step 3 - Define a Node structure with two members data and next

Step 4 - Define a Node pointer 'head' and set it to NULL.

Step 5 - Implement the main method by displaying the operations menu and make suitable function calls in the main method to perform user selected operation.

**B) INSERTION**

**Inserting At Beginning of the list**

Step 1 - Create a newNode with a given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set head = newNode and newNode→next = head .

Step 4 - If it is Not Empty then, define a Node pointer 'temp' and initialize with 'head'.

Step 5 - Keep moving the 'temp' to its next node until it reaches the last node.

Step 6 - Set 'newNode → next =head', 'head = newNode' and 'temp → next = head'.

**Inserting At End of the list**

Step 1 - Create a newNode with a given value.

Step 2 - Check whether the list is Empty (head == NULL).

Step 3 - If it is Empty then, set head = newNode and newNode → next = head.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches the last node in the list (until temp → next == head).

Step 6 - Set temp → next = newNode and newNode → next = head.

**Inserting At Specific location in the list (After a Node)**

Step 1 - Create a newNode with a given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set head = newNode and newNode → next = head.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode.

Step 6 - Every time check whether temp is reached to the last node or not. If it is reached to last node then display 'Given node is not found in the list!!!' and terminate the function. Otherwise move the temp to the next node.

Step 7 - If temp is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).

Step 8 - If temp is last node then set temp → next = newNode and newNode → next = head.

Step 9 - If temp is not last node then set newNode → next = temp → next and temp → next = newNode.

## C) DELETION

### Deleting from Beginning of the list

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize both 'temp1' and 'temp2' with head.

Step 4 - Check whether list is having only one node (temp1 → next == head)

Step 5 - If it is TRUE then set head = NULL and delete temp1 (Setting Empty list conditions)

Step 6 - If it is FALSE move the temp1 until it reaches to the last node. (until temp1 → next == head )

Step 7 - Then set head = temp2 → next, temp1 → next = head and delete temp2.

### Deleting from End of the list

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 - Check whether list has only one Node (temp1 → next == head)

Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate from the function.

Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until temp1 reaches to the last node in the list. (until temp1 → next == head)

Step 7 - Set temp2 → next = head and delete temp1.

### Deleting a Specific Node from the list

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1'

with head.

Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.

Step 5 - If it is reached to the last node then display 'Given node not found in the list!'. And terminate the function.

Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node (temp1 → next == head)

Step 7 - If list has only one node and that is the node to be deleted then set head = NULL and delete temp1 (free(temp1)).

Step 8 - If list contains multiple nodes then check whether temp1 is the first node in the list (temp1 == head).

Step 9 - If temp1 is the first node then set temp2 = head and keep moving temp2 to its next node until temp2 reaches the last node. Then set head = head→next, temp2 → next = head and delete temp1.

Step 10 - If temp1 is not first node then check whether it is last node in the list.

Step 11- If temp1 is last node then set temp2 → next = head and delete temp1 (free(temp1)).

Step 12 - If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

**Displaying a circular Linked List**

Step 1 - Check whether list is Empty (head == NULL)

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

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

Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last node

Step 5 - Finally display temp → data with arrow pointing to head → data

**<u>PROGRAMS :</u>**

**1. SINGLY LINKED LIST**

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

```
struct Node
{
   int data;
   struct Node *next
};
int main()
{
   int choice,n,nodeaft,deletel, count;
   struct Node *head=NULL,*temp,*neww,*temp2,*prev;
   do
   {
      printf("\n1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
          8.Del'@last 9.Search");
      scanf("%d",&choice);
      switch(choice)
      {
        case 1:
        printf("ENter the number other than 0");
        scanf("%d",&n);
        while(n!=0)
        {
           neww=(struct Node *)malloc(sizeof(struct Node));
           neww->data=n;
           neww->next=NULL;
           if(head==NULL)
           {
              head=neww;
              temp=head;
           }
           else
           {
              temp->next = neww;
              temp = neww;
```

```
    }
    printf("ENter the number other than 0");
    scanf("%d",&n);
}
break;
case 2:
  temp=head;
  while(temp!=NULL)
  {
      printf("%d",temp->data);
      temp=temp->next;
  }
  printf("NULL");
  break;
case 3:
    printf("ENter the new value");
    scanf("%d",&n);
    neww=(struct Node *)malloc(sizeof(struct Node));
    neww->data=n;
    neww->next = head;
    head = neww;

break;
case 4:
    printf("ENter the new value");
    scanf("%d",&n);
    neww=(struct Node *)malloc(sizeof(struct Node));
    neww->data=n;
    neww->next=NULL;
    temp = head;
    while(temp->next!=NULL)
      temp = temp->next;
    temp->next=neww;
```

```
 break;
 case 5:
    printf("ENter the new value");
    scanf("%d",&n);
    printf("Enter the node after:");
    scanf("%d",&nodeaft);
    neww=(struct Node *)malloc(sizeof(struct Node));
    neww->data=n;
    temp = head;
    while(temp!=NULL)
    {
       if(temp->data==nodeaft)
             break;
       temp = temp->next;
    }
      neww->next= temp->next;;
      temp->next = neww;
    break;
case 6:
    temp = head;
    head = temp->next;
    free(temp);
    break;
case 7:
    temp = head;
    printf("Enter the node to delete :");
    scanf("%d",&n);
    while(temp->data != n)
    {
       neww = temp;
       temp = temp->next;
    }
    neww->next = temp->next;
```

```
        break;
    case 8:
        temp = head;
        while(temp->next != NULL)
        {
            neww = temp;
            temp = temp->next;
        }
        neww->next = NULL;
        free(temp);
        break;
    case 9:
        temp = head;
        printf("Enter the element to search :");
        scanf("%d",&n);
        count =1;
        while(temp != NULL)
        {
            if(temp->data == n)
            {
                printf("The ele %d is found %d position",temp->data,count);
                break;
            }
        temp = temp->next;
        count += 1
        }
        if(temp==NULL)
            printf("Not found");
        break;
    }
}while(choice<=8);

return 0;
```

```
}
```

**OUTPUT :**

```
1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search1
ENter the number other than 01
ENter the number other than 02
ENter the number other than 00

1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search3
ENter the new value6

1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search4
ENter the new value6

1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search2
6126NULL
1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search6

1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search8

1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search2
12NULL
1.creation 2.Display 3.Insert@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8.Del'@last 9.Search9
Enter the element to search :2
The ele 2 is found 1 position
```

**2. DOUBLY LINKED LIST**

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
```

```
    int data;
    struct Node *next;
    struct Node *prev;
};
int main()
{
    int choice,n,nodeaft,nodedel;
    struct Node*head=NULL,*neww,*temp,*temp2,*temp3;
    do
    {
        printf("Enter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last
5.Ins@mid6.Del@first7.Del@last8.Del@mid");
        scanf("%d",&choice);
        switch(choice)
        {
          case 1:
          printf("Enter the number(other than 0): ");
          scanf("%d",&n);
          while(n!=0){
           neww=(struct Node*)malloc(sizeof(struct Node));
           neww->data=n;
           neww->next=NULL;
           neww->prev=NULL;
           if(head==NULL)
           {
              head=neww;
              temp = head;
           }
           else
           {
              temp->next=neww;
              neww->prev=temp;
              temp = neww;
```

```
 }
 printf("Enter the number(other than 0): ");
 scanf("%d",&n);
}
break;
case 2:
 temp=head;
 while(temp!=NULL)
 {
    printf("%d->",temp->data);
    temp = temp->next;
 }
 printf("NULL");
 break;
 case 3:
 printf("Enter the number(other than 0): ");
 scanf("%d",&n);
 neww=(struct Node*)malloc(sizeof(struct Node));
 neww->data=n;
 neww->next=NULL;
 neww->prev=NULL;
 temp = head;
 neww->next=head;
 head->prev=neww;
 head = neww;
 break;
 case 4:
 printf("Enter the number(other than 0): ");
 scanf("%d",&n);
 neww=(struct Node*)malloc(sizeof(struct Node));
 neww->data=n;
 neww->next=NULL;
 neww->prev=NULL;
```

```
temp=head;
while(temp->next!=NULL)
   temp=temp->next;
temp->next = neww;
neww->prev = temp;
break;
case 5:
printf("Enter the number(other than 0): ");
scanf("%d",&n);
printf("Enter the nodeaft element:");
scanf("%d",&nodeaft);
neww=(struct Node*)malloc(sizeof(struct Node));
neww->data=n;
neww->next=NULL;
neww->prev=NULL;
temp=head;
while(temp->next!=NULL)
{
   if(temp->data==nodeaft)break;
     temp=temp->next;
}
temp2 = temp->next;
temp->next = neww;
neww->prev = temp;
neww->next = temp2;
temp2->prev=neww;
break;
case 6:
   temp = head;
   head= head->next;
   head->prev=NULL;
   free(temp);
break;
```

```
        case 7:
            temp=head;
            while(temp->next!=NULL)
                temp=temp->next;
            temp2=temp->prev;
            temp2->next=NULL;
            free(temp);
            break;
        case 8:
            printf("Enter the node to be deleted:");
            scanf("%d",&nodedel);
            temp=head;
            while(temp->next!=NULL)
            {
                if(temp->data==nodedel)
                    break;
                temp=temp->next;
            }
            temp2=temp->prev;
            temp3=temp->next;
            temp2->next=temp3;
            temp3->prev=temp2;
            free(temp);
            break;

        }
    }while(choice<=9);
    return 0;
}
```

**OUTPUT :**

```
Enter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid6.Del@first7.
Del@last8.Del@mid1
Enter the number(other than 0): 1
Enter the number(other than 0): 2
Enter the number(other than 0): 0
Enter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid6.Del@first7.
Del@last8.Del@mid3
Enter the number(other than 0): 5
Enter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid6.Del@first7.
Del@last8.Del@mid4
Enter the number(other than 0): 6
Enter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid6.Del@first7.
Del@last8.Del@mid7
Enter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid6.Del@first7.
Del@last8.Del@mid2
5->1->2->NULLEnter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid
6.Del@first7.Del@last8.Del@mid8
Enter the node to be deleted:1
Enter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid6.Del@first7.
Del@last8.Del@mid2
5->2->NULLEnter ur choice 1.Create2.Display 3.Ins@first 4.Ins@last 5.Ins@mid6.D
el@first7.Del@last8.Del@mid
```

**3. CIRCULAR LINKED LIST**

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
int main()
{
   int choice,n,nodeaft,noddel;
```

```
struct Node *head=NULL,*neww,*temp, *temp1;
do
{
  printf("1.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid
8Del@last");
  scanf("%d",&choice);
  switch(choice)
  {
    case 1:

        printf("Enter the number(other than 0):");
        scanf("%d",&n);
        while(n!=0)
        {
          neww=(struct Node*)malloc(sizeof(struct Node));
          neww->data=n;
          if(head==NULL)
          {
            head = neww;
            neww->next = head;
            temp=head;
          }
          else
          {
             temp->next = neww;
             neww->next=head;
             temp = neww;
          }
          printf("Enter the number(other than 0):");
          scanf("%d",&n);
        }
       break;
    case 2:
        temp = head;
        while(temp->next!=head)
        {
           printf("%d->",temp->data);
           temp=temp->next;
        }
        printf("%d",temp->data);
        break;
```

```
case 3:
      printf("Enter the number:");
      scanf("%d",&n);
      neww=(struct Node*)malloc(sizeof(struct Node));
      neww->data=n;
      temp = head;
      while(temp->next!=head)
        temp = temp->next;
      neww->next=head;
      head = neww;
      temp->next = head;
      break;
  case 4:
      printf("Enter the number:");
      scanf("%d",&n);
      neww=(struct Node*)malloc(sizeof(struct Node));
      neww->data=n;
      temp = head;
      while(temp->next!=head)
        temp = temp->next;
      temp->next= neww;
      neww->next= head;
      break;
  case 5:
      printf("Enter the number:");
      scanf("%d",&n);
      printf("Enter the nodeafter:");
      scanf("%d",&nodeaft);
      neww=(struct Node*)malloc(sizeof(struct Node));
      neww->data=n;
      temp = head;
      while(temp->next!=head){
            if(temp->data==nodeaft)
            break;
      temp = temp->next;
      }
       neww->next = temp->next;
       temp->next = neww;
      break;
  case 6:
     temp = head;
```

```
            temp1 = head;
            while(temp->next!=head){
                temp = temp->next;
            }
            head = temp1->next;
            temp->next=head;
            free(temp);
            break;
        case 7:
            temp = head;
            printf("Enter the node to delete:");
            scanf("%d",&noddel);
            while(temp->data != noddel){
                temp1 = temp;
                temp = temp->next;
            }
            temp1 ->next = temp ->next;
            free(temp);
        case 8:
            temp = head;
            while(temp->next != head) {
                temp1 = temp;
                temp = temp->next;
            }
            temp1->next = head;
            free(temp);
        }
    }while(choice<=8);
}
```

**OUTPUT :**

```
1.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid 8Del@
last1
Enter the number(other than 0):1
Enter the number(other than 0):2
Enter the number(other than 0):0
1.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid 8Del@
last3
Enter the number:5
1.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid 8Del@
last5
Enter the number:8
Enter the nodeafter:2
1.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid 8Del@
last2
5->1->2->81.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del
@mid 8Del@last6
1.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mid 8Del@
last2
1->2->01.Creation2.Display3.Ins@first 4.Ins@last 5.Ins@mid 6.Del@first 7.Del@mi
d 8Del@last
```

**RESULT :**

Thus, the C program to implement the operations of singly, doubly and circular linked list has been executed and tested successfully.

| EXPNO: 7 | STACK IMPLEMENTATION USING ARRAY |
|---|---|

**AIM:**

To write a C program to perform stack implementation using array

**ALGORITHMS**

**1. PUSH() OPEARTION**

Step 1: Start

Step 2: Declare Stack[MAX]; //Maximum size of Stack

Step 3: Check if the stack is full or not by comparing top with (MAX-1)

If the stack is full, Then print "Stack Overflow" i.e, stack is full and cannot be pushed with another element

Step 4: Else, the stack is not full

Increment top by 1 and Set, a[top] = x

which pushes the element x into the address pointed by top.

// The element x is stored in a[top]

Step 5: Stop

**2. POP()**

Step 1: Start

Step 2: Declare Stack[MAX]

Step 3: Push the elements into the stack

Step 4: Check if the stack is empty or not by comparing top with base of array i.e 0

If top is less than 0, then stack is empty, print "Stack Underflow"

Step 5: Else, If top is greater than zero the stack is not empty, then store the value pointed by top in a variable x=a[top] and decrement top by 1. The popped element is x

Step:Stop

**3. DISPLAY()**

Step 1:START

Step 2 - Check whether stack is EMPTY. (top == -1)

Step 3 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

Step 4 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).

Step 5 - Repeat above step until i value becomes '0'.

Step 6:STOP

**PROGRAMS**

**1.POP() OPERATION:**

```
#include <stdio.h>
int stack[100],i,j,choice=0,n,top=-1;
void push();
void pop();
void show();
void main ()
{

   printf("Enter the number of elements in
the stack ");
   scanf("%d",&n);

   while(choice != 4)
   {
      printf("Chose one from the below
options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit")
;
      printf("\n Enter your choice \n");
      scanf("%d",&choice);
      switch(choice)
```

```
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                show();
                break;
            }
            case 4:
            {
                printf("Exiting....");
                break;
            }
            default:
            {
                printf("Please Enter valid choice
");
            }
        };
    }
}

void push ()
{
    int val;
```

```
    if (top == n )
    printf("\n Overflow");
    else
    {
        printf("Enter the value?");
        scanf("%d",&val);
        top = top +1;
        stack[top] = val;
    }
}

void pop ()
{
    if(top == -1)
    printf("Underflow");
    else
    top = top -1;
}
void show()
{
    for (i=top;i>=0;i--)
    {
        printf("%d\n",stack[i]);
    }
    if(top == -1)
    {
        printf("Stack is empty");
    }
}
```

**OUTPUT:**

```
1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value?2
Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value?3
Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3
3
2
```

**RESULT:**

Thus the program implementation using stack using array has been executed and tested successfully.

| EX.NO : 8 | STACK USING LINKED LIST |
|---|---|

**AIM :**

To write a C program to implement stack using linked list.

**ALGORITHMS :**

**1. PUSH ( ) OPERATION**

Step 1 : START

Step 2 : Create a node new and declare variable top

Step 3 : Set new data part to be Null
// The first node is created, having null value and top pointing to it

Step 4 : Read the node to be inserted.

Step 5 : Check if the node is Null, then print "Insufficient Memory"

Step 6 : If node is not Null, assign the item to data part of new and assign top to link part of new and also point stack head to new.

Step 7 : STOP

**2. POP ( ) OPERATION**

Step 1 : START

Step 2 : Check if the top is NULL, then print “Stack Underflow”.

Step 3 : If top is not NULL, assign top’s link part to temp and assign temp to stack_head’s link part.

Step 4 : STOP

**3. DISPLAY ( ) OPERATION**

Step 1 : START

Step 2 : Check if the top is NULL, then print “Stack is Empty”.

Step 3 : If it is not NULL, assign temp pointer to top.

Step 4 : Display temp →data and move it to the next node. Repeat the same until temp reaches to the first node in the stack (temp→next != NULL).

Step 5 : STOP

**PROGRAM :**

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
int main()
{
   int choice,ele;
   struct Node *Top=NULL,*neww,*temp;
   do
   {
      printf("Enter ur option1.Push 2.Pop 3.DIsplay\n");
      scanf("%d",&choice);
      switch(choice)
      {
        case 1:
            printf("Enter the data to push:");
            scanf("%d",&ele); //10
            neww=(struct Node*)malloc(sizeof(struct Node));
            neww->data=ele;
            neww->next=NULL;
            if(Top==NULL)
            {
               Top = neww;
               temp = Top;
            }
            else
            {
               neww->next = Top;
               Top = neww;
            }
        break;
        case 2:
             if(Top==NULL)
               printf("Stack is underflow");
             else
             {
                temp = Top;
                Top= Top->next;
```

```
            }
        break;
        case 3:
            if(Top==NULL)
            {
               printf("Stack is Empty");
            }
            else{
            temp=Top;
            while(temp!=NULL){
              printf("\n%d",temp->data);
              temp = temp->next;
            }
            }
        break;
      }
   }while(choice<=3);
   return 0;
}
```

**OUTPUT :**

```
Enter ur option1.Push 2.Pop 3.DIsplay
1
Enter the data to push:1
Enter ur option1.Push 2.Pop 3.DIsplay
1
Enter the data to push:2
Enter ur option1.Push 2.Pop 3.DIsplay
1
Enter the data to push:3
Enter ur option1.Push 2.Pop 3.DIsplay
3
3-->2-->1-->Enter ur option1.Push 2.Pop 3.DIsplay
2
Enter ur option1.Push 2.Pop 3.DIsplay
3
2-->1-->Enter ur option1.Push 2.Pop 3.DIsplay
```

**RESULT :**

Thus, the program to implement stack operations using linked list has been executed and tested successfully.

| EX.NO:9 | STACK APPLICATIONS |
|---|---|

**AIM:**

To write a C program to perform stack applications.

**ALGORITHMS**

**1. CONVERSION OF INFIX TO POSTFIX EXPRESSION**

Step 1: START

Step 2: DECLARE stack, top, push, pop, priority, exp, *e, x

Step 3: DECLARE stack[100], top=-1

Step 4: DECLARE VOID function push with argument x and perform stack[++top] = x;

Step 5: DECLARE CHAR function pop with no argument and perform IF(top -1) - return -1; or ELSE - return stack[top--];

Step 6: DECLARE INT function priority with argument x and perform IF(x == '(') - return 0; IF(x '+' || x == '-') - RETURN 1; IF(x == '*' || x == '/') - RETURN 2; or ELSE return 0;

Step 7: In main function GET the value of the expression and store it in variable exp

Step 8: Assign e=exp;

Step 9: WHILE(*e != '\0') - IF(isa1num(*e)) - PRINT("%c ",*e); ELSE IF(*e == '(') - push(*e); ELSE IF(*e == ')') - WHILE((x = pop()) != '(') - PRINT("%c ", x); ELSE — WHILE(priority(stack[top])>= priority(*e)) - PRINT ("%c ",pop()); push(*e); e++; WHILE(top != -1) PRINT (”%c ",pop());

Step 10: ENDWHILE

Step 11: RETURN 0;

Step 12: STOP

**2. POSTFIX EXPRESSION EVALUATION**

Step 1 : START
Step 2 : DECLARE stack, top, push, pop, priority, exp, *e, x, n1, n2, n3, num
Step 3 : DECLARE stack[20], top=-1
Step 4 : DECLARE VOID function push with argument x and perform stack[++top] = x;
Step 5 : DECLARE INT function pop with no argument and perform RETURN stack[top--];
Step 6 : In main function GET the value of the expression and store it in variable exp
Step 7 : Assign e=exp;
Step 8 : 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++;
Step 9 : ENDWHILE
Step 10 : PRINT The result of expression %s %d\n\n",exp,pop());
Step 11 : RETURN 0

Step 12 : STOP

**PROGRAMS**

**1. CONVERSION OF INFIX TO POSTFIX EXPRESSION**

```
#inc1ude<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;

void push(char x)

   stack[++top] = x;


char pop()

   if(top == -1)
      return -1;
   else
      return stack[top--];


int priority(char x)

   if(x == '(')
      return 0;
   if(x == '+' || x == '-')
      return 1;
   if(x == '*' || x == '/')
      return 2;
   return 0;


int main()

   char exp[100];
   char *e, x;
   printf("Enter the expression : ");
   scanf("%s",exp);
   printf("\n");
   e = exp;

   while(*e != '\0')

      if(isa1num(*e))
         printf("%c ",*e);
      else if(*e == '(')
         push(*e);
      else if(*e == ')')
```

```
        while((x = pop()) != '(')

         printf("%c ", x);

    else

        while(priority(stack[top]) >= priority(*e))
            printf(”°%c ",pop());
        push(*e);

    e++;


while(top != -1)

    printf("%c ",pop());
}return 0;
```

**OUTPUT :**

```
Enter the expression : (A/(B-C)*D+E)

A B C - / D * E +
```

## 2. POSTFIX EXPRESSION EVALUATION

```
#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 - nl;
                break;
            }
            case '*':
            {
                n3 = n1 * n2;
                break;
            }
            case'/':
            {
                n3 = n2 / nl;
                break;
            }
            }
            push(n3);
        }
        e++;
    }
    printf("uiThe result of expression %s = %duiui",exp,pop());
    return 0;
}
```

**OUTPUT :**

```
Enter the expression :: (A+B)*C-(D-E)*(F+G)

The result of expression (A+B)*C-(D-E)*(F+G)  =  0
```

**RESULT:**

Thus the program implementation using stack applications has been executed and tested successfully.

| EX.NO:10 | QUEUE USING ARRAY |
|---|---|

**AIM:**

To write a C program to perform queue using array.

**ALGORITHM**

**QUEUE USING ARRAY**

Step 1 : START

Step 2 : DEFINE rear = -1, front = -1, size = 10, queue[size]

Step 3 : DISPLAY "1 for enqueue, 2 for dequeue, 3 for display"

Step 3 : DEFINE i, option

Step 4 : GET option

Step 5 : DO

Step 6 : SWITCH (option) DO

Step 7 : CASE 1 do

Step 8 : DEFINE int n

Step 9 : GET n

Step 10: IF (front == -1) do queue[0] = n, rear++, front++

Step 11: ELSE DO queue[++rear] = n;

Step 12: ENDIF

Step 13: CASE 2 do

Step 14: DEFINE temp

Step 15: Rear = rear -1

Step 16: EXIT;

Step 17: CASE 3 do

Step 18: DEFINE int i

Step 19: FOR i in range(front, rear) DO

Step 20: DISPLAY queue[i]

Step 21: EXIT

Step 22: STOP

## PROGRAM

**QUEUE USING ARRAY**

```
#include<stdio.h>
#define size 10
int rear = -1, front = -1, qeue[size];

//function declaration definition will be at the bottom
void enqeue();
void deqeue();
void print();

void main()
{
   int i, boo;
   printf("1 for enqueue\n2 for printing\n3 for dequeue\n");
   printf("\n Enter the option:");
   scanf("%d", &boo);
   do
   {
      switch (boo)
      {
         case 1:
            enqeue();
            break;


         case 2:
            print();
            break;


         case 3:
            deqeue();
            break;


      }


      printf("\nEnter opt :");
      scanf("%d", &boo);
   }

void enqeue()
{
   int n;
   printf("\nEnter the num");
   scanf("%d", &n);
   if (front == -1)
   {
      qeue[0] = n;
      rear++;
      front++;
```

```
    }
    else
    {
        qeue[++rear] = n;
    }

}

void deqeue()
{
    int temp;
    rear--;
}

void print()
{
    int i;
    for ( i = front; i <= rear; i++)
    {
        printf(" %d", qeue[i]);
    }

    printf("\nFront : %d Rear : %d", qeue[front], qeue[rear]);
}
```

**OUTPUT :**

```
1.Enqueue 2.Dequeue 3.Display1
Enter the element:11
1.Enqueue 2.Dequeue 3.Display1
Enter the element:6
1.Enqueue 2.Dequeue 3.Display1
Enter the element:5
1.Enqueue 2.Dequeue 3.Display3
11 6 5 1.Enqueue 2.Dequeue 3.Display
```

**RESULT:**

Thus the program implementation of queue using array has been executed and tested successfully.

| EXP NO:11 | QUEUE USING LINKED LIST |
|---|---|

**AIM:**

To write a C program to perform queue using linked list

**ALGORITHMS**

**1. CREATION**

Step 1-START

Step 2 - Include all the header files which are used in the program. And declare all the user defined functions.

Step 3 - Define a 'Node' structure with two members data and next.

Step 4 - Define two Node pointers 'front' and 'rear' and set both to NULL.

Step 5 - Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

Step 6-STOP

**2. ENQUEUE()**

Step 1-START

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.

Step 5-STOP

**3. DEQUEUE()**

Step 1:START

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

Step 3 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function

Step 4 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.

Step 5 - Then set 'front = front → next' and delete 'temp' (free(temp)).

Step 6-STOP

**4. DISPLAY**

Step 1:START

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

Step 3 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.

Step 4 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.

Step 5- Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).

Step 6 - Finally! Display 'temp → data ---> NULL'.

Step 7:STOP

**PROGRAMS**

```
#include<stdio.h>
struct Node
{
   int data;
   struct Node *next;
};
int main()
{
                                    struct          Node
*front=NULL,*rear=NULL,*neww,*temp;
   int choice,ele;
   do
   {
          printf("\nEnter ur option 1.Enqueue
2.Dequeue 3.Display\n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
                                          neww=(struct
Node*)malloc(sizeof(struct Node));
                printf("Enter the data:");
```

```
        scanf("%d",&ele);
        neww->data=ele;
        neww->next=NULL;
        if(front==NULL)
        {
            front = neww;
            rear = front;
        }
        else
        {
           rear->next=neww;
           rear = neww;
        }
break;
case 2:
        if(front==NULL)
          printf("Queue is empty");
        else{
         temp=front;
         front = temp->next;
         free(temp);
        }
break;
case 3:
         if(front==NULL)
           printf("Queue is empty");

         else{temp=front;
         while(temp!=NULL)
         {
            printf("%d->",temp->data);
            temp=temp->next;
         }
         }
break;
```

```
        }
    }while(choice<=3);
  return 0;
}
```

**OUTPUT:**

```
Enter ur option 1.Enqueue 2.Dequeue 3.Display
1
Enter the data:13

Enter ur option 1.Enqueue 2.Dequeue 3.Display
1
Enter the data:24

Enter ur option 1.Enqueue 2.Dequeue 3.Display
1
Enter the data:45

Enter ur option 1.Enqueue 2.Dequeue 3.Display
3
13->24->45->
Enter ur option 1.Enqueue 2.Dequeue 3.Display
2

Enter ur option 1.Enqueue 2.Dequeue 3.Display
3
24->45->
```

**RESULT:**

Thus the C program to perform queue using linked list is implemented, executed and verified successfully

| EX.NO:12 | PRIORITY QUEUE |
|---|---|

## Aim:

To create a program for priority Queue Implementation using linked list in ascending order and descending order.

## Algorithm:

### 1. Priority Queue Implementation using linked list in ascending order.

Step 1: Create new node with DATA and PRIORITY

Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end. Else goto Step 5.

Step 3: NEW -> NEXT = HEAD

Step 4: HEAD = NEW

Step 5: Set TEMP to head of the list

Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY > PRIORITY

Step 7: TEMP = TEMP -> NEXT

[END OF LOOP]

Step 8: NEW -> NEXT = TEMP -> NEXT

Step 9: TEMP -> NEXT = NEW

Step 10: End

### 2.Priority Queue Implementation using linked list in descending order.

Step 1: Create new node with DATA and PRIORITY

Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end. Else goto Step 5.

Step 3: NEW -> NEXT = HEAD

Step 4: HEAD = NEW

Step 5: Set TEMP to head of the list

Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY > PRIORITY

Step 7: TEMP = TEMP -> NEXT

[END OF LOOP]

Step 8: NEW -> NEXT = TEMP -> NEXT

Step 9: TEMP -> NEXT = NEW

Step 10: End

**Program:**

**1. Priority Queue Implementation using linked list in ascending order.**

```
#include<stdio.h>
#define MAX 10
struct P_Queue
{
    int data;
    int priority;
};
int main()
{
   struct P_Queue p[MAX];
   int choice,prior,ele,i,rear=-1;
   do
   {
       printf("Option 1.Insertion 2.Deletion 3.Display");
       scanf("%d",&choice);
       switch(choice)
       {
         case 1:
                printf("Enter the data:");
                scanf("%d",&ele);
                printf("Enter the priority");
                scanf("%d",&prior);
                if(rear==-1){
                   p[++rear].data=ele;
                   p[rear].priority=prior;
```

```
            }
            else{
                p[++rear].data=ele;
                p[rear].priority=prior;
            }
            Sort(p,rear);

        break;
        case 2:
            if(rear==-1)
                printf("Queue is empty");
            else
            {
                for(i=0;i<=rear;i++){
                    p[i].data = p[i+1].data;
                    p[i].priority=p[i+1].priority;
                }
                rear--;
            }

        break;
        case 3:
            for(i=rear;i>=0;i--)
            {
                printf("%d %d",p[i].data,p[i].priority);
                printf("\n");
            }
        break;
    }
}while(choice<=3);
```

```
  return 0;
}
void Sort(struct P_Queue p[],int rear)
{
 int i,j;
   for(i=0;i<=rear;i++)
   {
      for(j=i+1;j<=rear;j++)
      {
         if(p[i].priority<p[j].priority){
         int data=p[i].data;
         int prior=p[i].priority;
         p[i].data=p[j].data;
         p[i].priority=p[j].priority;
         p[j].data=data;
         p[j].priority=prior;
        }
      }
   }
}
```

**OUTPUT:**

```
Option 1.Insertion 2.Deletion 3.Display :1
Enter the data:10
Enter the priority :2
Option 1.Insertion 2.Deletion 3.Display :1
Enter the data:5
Enter the priority :1
Option 1.Insertion 2.Deletion 3.Display :3
5 1
10 2
```

**2.Priority Queue Implementation using linked list in descending order.**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node* left, *right;
};
void printCurrentLevel(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);
 void printLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i=1; i<=h; i++)
        printCurrentLevel(root, i);
}
 void printCurrentLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1)
    {
        printCurrentLevel(root->left, level-1);
        printCurrentLevel(root->right, level-1);
    }
```

```
}
int height(struct node* node)
{
    if (node==NULL)
        return 0;
    else
    {
        int lheight = height(node->left);
        int rheight = height(node->right);
        if (lheight > rheight)
            return(lheight+1);
        else return(rheight+1);
    }
}
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}
int main()
{
    struct node *root = newNode(1);
```

**OUTPUT:**

```
Option 1.Insertion 2.Deletion 3.Display :1
Enter the data:10
Enter the priority :3
Option 1.Insertion 2.Deletion 3.Display :1
Enter the data:20
Enter the priority :2
Option 1.Insertion 2.Deletion 3.Display :3
10 3
20 2
Option 1.Insertion 2.Deletion 3.Display :
```

**RESULT:**

Thus the program implementation using priority queue has been executed and tested successfully.

| EX.NO: 13 | QUEUE APPLICATION |
|---|---|

**AIM:**

To level order traversal of a binary tree using queue.

**ALGORITHMS:**

Step 1 : Create an empty queue q

Step 2 : temp_node = root /start from root/

Step 3 : Loop while temp_node is not NULL

Step 4 : print temp_node->data.

Step 5 : Enqueue temp_node's children (first left then right children) to q

Step 6 : Dequeue a node from q and assign it's value to temp_node

**PROGRAM:**

**Level order Traversal of a Binary Tree using Queue**

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

/* A binary tree node has data,
   pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left, *right;
};

/* Function protoypes */
void printCurrentLevel(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);

/* Function to print level order traversal a tree*/
```

```
void printLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i=1; i<=h; i++)
        printCurrentLevel(root, i);
}

/* Print nodes at a current level */
void printCurrentLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1)
    {
        printCurrentLevel(root->left, level-1);
        printCurrentLevel(root->right, level-1);
    }
}

/* Compute the "height" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
int height(struct node* node)
{
    if (node==NULL)
        return 0;
    else
```

```
    {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return(lheight+1);
        else return(rheight+1);
    }
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root = newNode(1);
    root->left        = newNode(2);
```

```
    root->right       = newNode(3);
    root->left->left  = newNode(4);
    root->left->right = newNode(5);

    printf("Level Order traversal of binary tree is \n");
    printLevelOrder(root);

    return 0;
}
```

**OUTPUT:**

```
Level Order traversal of binary tree is
1 2 3 4 5
```

**RESULT:**

Thus the level order traversal of a Binary Tree using Queue.

| EX.NO:14 | BINARY SEARCH TREE |
|---|---|

**AIM :**

TowriteaCprogramtoimplement Binary Search Tree Operations.

**ALGORITHMS :**

1. **TO INSERT A NODE**

Step 1: Create a new BST node and assign values to it.

Step 2:insert(node, key)

i) If root == NULL,

return the new node to the calling function.

ii) if root=>data < key

call the insert function with root=>right and assign the return value in root=>right.

root->right = insert(root=>right,key)

iii) if root=>data > key

call the insert function with root->left and assign the return value in root=>left.

root=>left = insert(root=>left,key)

Step 3: Finally, return the original root pointer to the calling function.

2. **DISPLAY USING PRE-ORDER**

Step 1:Visit or print the root.

Step 2:Traverse the left subtree.

Step 3:Traverse the right subtree.

3. **SEARCH IN BINARY TREE**

Step 1: Start from the root.

Step 2: Compare the searching element with root, if less than root, then recurse for left, else recurse for right.

Step 3:If the element to search is found anywhere, return true, else return false.

**4. FIND MINIMUM IN SEARCH TREE**

Step 1:Traverse the node from root to left recursively until left is NULL.

Step 2:The node whose left is NULL is the node with minimum value.

**5. FIND MAXIMUM IN SEARCH TREE**

Step 1:Traverse the node from root to right recursively until right is NULL.

Step 2:The node whose right is NULL is the node with maximum value.

**PROGRAMS :**

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

struct node {
   int key;
   struct node *left, *right;
};


struct node* newNode(int item)
{
   struct node* temp
      = (struct node*)malloc(sizeof(struct node));
   temp->key = item;
   temp->left = temp->right = NULL;
   return temp;
}


void inorder(struct node* root)
{
   if (root != NULL) {
inorder(root->left);
printf("%d \n", root->key);
inorder(root->right);
   }
}


struct node* insert(struct node* node, int key)
{

   if (node == NULL)
      return newNode(key);


   if (key < node->key)
```

```
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);


    return node;
}


int main()
{

    struct node* root = NULL;
    root = insert(root, 27);
insert(root, 14);
insert(root, 19);
insert(root, 10);
insert(root, 35);
insert(root, 31);
insert(root, 42);


inorder(root);

    return 0;
}
```

**OUTPUT:**

```
10
14
19
27
31
35
42

--------------------------------
Process exited after 0.08055 seconds with return value 0
Press any key to continue . . .
```

**RESULT:**

Thus, the program to implement Binary Search Tree has beenexecutedand tested successfully.

| Ex.No: 15 | HASHING TECHNIQUES |
|---|---|

**AIM:**

To write aC program to implement separate chaining in hashing.

**ALGORITHMS :**

1. **Algorithm to insert a value in hash table using separate chaining collision resolutiontechnique**

Hashtable is an array of pointers. All pointers are initialized to NULL ( head[ TABLE_SIZE] = NULL)

Step1: Read the value to be inserted
step 2: create a new node using malloc function
step 3: Assign the value read to the data field of newnode (newnode -> data =value)
step 4: compute the index index = value % TABLE_SIZE
step 5: if head[ index] is NULL then
step 5.1: attach the newnode as first node
step 6: else
step 6.1 : attach the newnode as lastnode

2.Algorithm to insert a value in linear probing

Hashtable is an array of size = TABLE_SIZE
Step 1: Read the value to be inserted, key
Step 2: let i = 0
Step 3: hkey = key% **TABLE_SIZE**
Step 4 :compute the index at which the key has to be inserted in hash table
**index = (hkey + i) % TABLE_SIZE**
Step 5: if there is no element at that index then insert the value at index and STOP
Step 6: If there is already an element at that index
step 4.1: i = i+1
step 7: if i< TABLE_SIZE then go to step 4

**PROGRAMS :**

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
struct node
{
        int data;
        struct node *next;
};
struct node *head[TABLE_SIZE]={NULL},*c;
void insert()
{
   int i,key;
   printf("\nenter a value to insert into hash table\n");
   scanf("%d",&key);
```

```
   i=key%TABLE_SIZE;
   struct node * newnode=(struct node *)malloc(sizeof(struct node));
   newnode->data=key;
   newnode->next = NULL;
   if(head[i] == NULL)
      head[i] = newnode;
   else
   {
      c=head[i];
      while(c->next != NULL)
      {
         c=c->next;
      }




      c->next=newnode;
   }
}
void search()
{
   int key,index;
   printf("\nenter the element to be searched\n");
   scanf("%d",&key);
   index=key%TABLE_SIZE;
   if(head[index] == NULL)
      printf("\n Search element not found\n");
   else
   {
      for(c=head[index];c!=NULL;c=c->next)
      {
         if(c->data == key)
            {
               printf("search element found\n");
               break;
            }
      }
      if(c==NULL)
         printf("\n Search element not found\n");

   }
}
void display()
{
   int i;
   for(i=0;i<TABLE_SIZE;i++)
        {
         printf("\nentries at index %d\n",i);
            if(head[i] == NULL)
            {
            printf("No Hash Entry");

            }
            else
            {
                        for(c=head[i];c!=NULL;c=c->next)
                        printf("%d->",c->data);
            }
        }
}
```

```
main()
{
   int opt,key,i;
   while(1)
   {
      printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
      scanf("%d",&opt);
      switch(opt)
      {
         case 1:
            insert();
            break;
         case 2:
            display();
            break;



         case 3:
            search();
            break;
         case 4:exit(0);

      }
   }
}
```

**OUTPUT:**

```
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
20

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
25

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
45

Press 1. Insert  2. Display     3. Search       4.Exit
2

entries at index 0
20->
entries at index 1
No Hash Entry
entries at index 2
No Hash Entry
entries at index 3
No Hash Entry
```

.

```
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10

int h[TABLE_SIZE]={NULL};

void insert()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
    {

     index=(hkey+i)%TABLE_SIZE;

     if(h[index] == NULL)
     {
        h[index]=key;
         break;
     }

    }

    if(i == TABLE_SIZE)

     printf("\nelement cannot be inserted\n");
}
void search()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
    index=(hkey+i)%TABLE_SIZE;
    if(h[index]==key)
    {
      printf("value is found at index %d",index);
      break;
    }
  }
  if(i == TABLE_SIZE)
    printf("\n value is not found\n");
}
void display()
{

  int i;

  printf("\nelements in the hash table are \n");
```

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

 printf("\nat index %d \t value =  %d",i,h[i]);

}
main()
{
   int opt,i;
   while(1)
   {
      printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
      scanf("%d",&opt);
      switch(opt)
      {
         case 1:
            insert();
            break;
         case 2:
            display();
            break;
         case 3:
            search();
            break;
         case 4:exit(0);
      }
   }
}
```

**OUTPUT:**

```
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
25

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
23

Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
12

Press 1. Insert  2. Display     3. Search       4.Exit
2

elements in the hash table are

at index 0       value =  0
at index 1       value =  0
at index 2       value =  12
at index 3       value =  23
at index 4       value =  0
at index 5       value =  25
```

**RESULT:**

Thus,the Cprogram to implement linear probing collision avoidance technique Executed and the output was verified successfully.