Exp no :1	BRANCHING AND ITERATIVE CONTRUCTS

AIM:

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

ALGORITHM :

1. Write a C program to read five subject marks and print the result status of "Pass or Fail " with percentage.

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. Write a C program to 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. Write a C program to identify the given input number is either 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. Write a C program to input any character and check whether it is the alphabet, 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. Write a C program to 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. Write a C program to the first digit of a given input: {Ex: 123 = 1(ans), Ex (67 = 6(ans)}.

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. Write a C program to check the given number is perfect number 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. Write a program to print the following pattern: n = 5

*

* *

* * *

* * * *

* * * * *

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

PROGRAMS :

1. Write a C program to read five subject marks and print the result status of “Pass or Fail “ with percentage.

CODE :

```c
#include <stdio.h>
void main(){
    int m1, m2, m3, m4, m5;
    float percentage;
    scanf("%d %d %d %d %d", &m1, &m2, &m3, &m4, &m5);
    if ((m1 > 34)&&(m2 > 34)&&(m3 > 34)&&(m4 > 34)&&(m5 > 34)){
```

```
        printf("Pass \n");
        percentage = ((m1 + m2 + m3 + m4 + m5)/500.00)*100;
        printf("%f", percentage);

    }
    else {
        printf("Fail");
    }


}
```

OUTPUT :

```
PS C:\Users\sugan\Documents\c
d Data structures\C programs\
90 95 90 87 63
Pass
85.000000
PS C:\Users\sugan\Documents\c
```

2. Write a C program to find the greatest of three numbers

CODE:

```
#include <stdio.h>
int main() {
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    if (a > b){
        if (a > c) {
            printf("%d is greater", a);
        }
        else {
            printf("%d is greater", c);
        }
    }
    else {
        if (b > c) {
            printf("%d is greater", b);
        }
        else {
            printf("%d is greater", c);
        }
        }
}
```

OUTPUT :

```
PS C:\Users\sugan\Documents\c
d Data structures\C programs\
2 5 8
8 is greater
PS C:\Users\sugan\Documents\c
```

3. Write a C program to identify the given input number is either positive, negative or zero.

CODE:

```c
#include <stdio.h>
int main() {
    int x;
    scanf("%d", &x);
    if (x > 0) {
        printf("Positive");
    }
    else if (x < 0)
    {
        printf("Negative");
    }
    else {
        printf("Zero");
    }

}
```

OUTPUT :

```
2\C and Data structures\
-2
Negative
PS C:\Users\sugan\Docume
```

```
2\C and Data structures\C
8
Positive
PS C:\Users\sugan\Document
```

```
0
Zero
PS C:\Users\sugan\
```

4. Write a C program to input any character and check whether it is the alphabet, digit, or special character.

CODE:

```c
#include <stdio.h>
int main() {
    char x;
    scanf("%c", &x );
    if (65 <= x && x <= 122)
    {
        printf("alphabet");
    }
    else if (48 <= x && x <= 57)
    {
        printf("number");
    }
    else
    {
        printf("special char");
    }
}
```

OUTPUT :

```
c
alphabet
PS C:\Users\sugan\Do
```

```
7
number
PS C:\Users\sugan\D
```

5. Write a C program to reverse the given number.

CODE:

```c
#include<stdio.h>
int main() {
    int num, result, remainder;
    scanf("%d", &num);
    result = 0;
    while (num != 0) {
        remainder = num % 10;
        result = result * 10 + remainder;
        num /= 10;
    }
    printf("%d", result);
}
```

OUTPUT :

```
123
321
PS C:\Users\sugan\Docum
```

6. Write a C program to the first digit of a given input: {Ex: 123 = 1(ans), Ex (67 = 6(ans)}.

CODE:

```c
#include<stdio.h>
int main() {
    int a;
    scanf("%d", &a);
    while (a >= 10) {
        a /= 10;
    }
    printf("%d", a);
}
```

OUTPUT :

```
123
1
PS C:\Users\sugan\Documents\college files\sem
2\C and Data structures\C programs\lab 2\" ; 
67
6
PS C:\Users\sugan\Documents\college files\sem
```

7. Write a C program to check the given number is perfect number or not

CODE:

```c
#include <stdio.h>
int main() {
    int a , res,i, temp;
    scanf("%d", &a);
    res = 0;
    for (i = 1;i < a;++i) {
        if ( a%i == 0) {
            res = res + i;
        }
    }
    if (res == a) {
        printf("The given number is a perfect number");
```

```
    }

}
```

OUTPUT :

```
28
The given number is a perfect number
PS C:\Users\sugan\Documents\college files
```

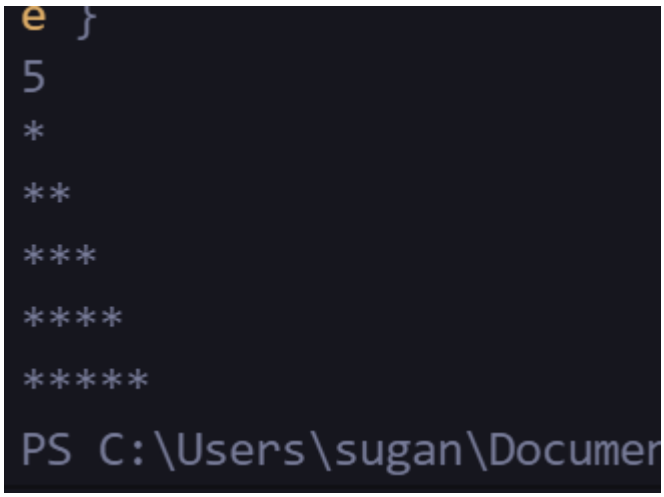
8. Write a program to print the following pattern: n = 5

```
*
* *
* * *
* * * *
* * * * *
```

CODE:

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

OUTPUT :



RESULT :

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

Exp no :2	FUCTIONS

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 numberStep 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, sStep 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<nStep

3: IF

n%a==0

Step 4: COMPUTE

sum=sum+aStep 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**

CODE :

```c
#include<stdio.h>
void find_mod(int);
void main()
{
    int num;
    scanf("%d", &num);
    find_mod(num);
}
void find_mod(int num)
{
    if (num < 0)
    {
        printf("NEGATIVE");
    }
    else if (num > 0)
    {
        printf("POSITIVE");
    }
    else
    {
        printf("ZERO");
    }
}
```

OUTPUT :

```
7
POSITIVE
PS C:\Users\sugan\Documents\college files\sem 2\C and Data structure
2\C and Data structures\C programs\lab 3\" ; if ($?) { gcc tempCodeR
e }
-1
NEGATIVE
PS C:\Users\sugan\Documents\college files\sem 2\C and Data structure
2\C and Data structures\C programs\lab 3\" ; if ($?) { gcc tempCodeR
e }
0
ZERO
PS C:\Users\sugan\Documents\college files\sem 2\C and Data structure
```

2. Write a program to check the given number is a perfect number or not using functions

CODE :

```c
#include<stdio.h>
int is_perfect_num(int);
void main()
{
    int num;
    scanf("%d", &num);
    is_perfect_num(num);

}
int is_perfect_num(int num)
{
    int temp = 0, i;
    for (i = 1;i < num; i++)
    {
        if (num%i == 0)
        {
            temp += i;
        }
    }
    if (temp == num)
    {
        printf("It is a perfect number !");
    }
    else
    {
        printf("It is not a perfect number :(");
    }
}
```

OUTPUT :

```
28
It is a perfect number !
PS C:\Users\sugan\Documents\college files\s
2\C and Data structures\C programs\lab 3\"
67
It is not a perfect number :(
PS C:\Users\sugan\Documents\college files\s
```

3. Write a program to accept an integer and returns the number of factors using functions

CODE :

```c
#include<stdio.h>
void factors(int);
void main()
{
    int num;
    scanf("%d", &num);
    factors(num);
}
void factors(int num)
{
    int temp = 0, i;
    for (i = 1; i <= num ; i++)
    {
        if (num%i == 0)
        {
            temp++;
        }
    }
    printf("%d has %d factors", num, temp);
}
```

OUTPUT :

```
2\C and Data structures\C programs\
8
8 has 4 factors
PS C:\Users\sugan\Documents\college
```

4. Write a program to sum of its digit up to a single value using recursion

CODE :

```c
#include<stdio.h>
int sum_to_single(int);
void main()
{
    int num;
    int res;
    scanf("%d", &num);
    res = sum_to_single(num);
    printf("%d", res);
}
int sum_to_single(int num)
{
    int temp = 0, i;
    while (num)
    {
        i = num/10;
        temp += num - i*10;
        num = i;
    }
    if (temp/10 == 0)
    {
        return temp;
    }
    else
    {
        return sum_to_single(temp);
    }
}
```

OUTPUT :

```
e }
5678
8
PS C:\Users\sugan\Documents\college files\
```

5. Write a program to convert decimal number to binary number using recursion

CODE :

```c
#include<stdio.h>
int binary(int num, int a[8], int i);
void main()
{
    int num, a[8], i, b[8];
    scanf("%d", &num);
    binary(num, a, 0);
}
int binary(int num, int a[8], int i)
{
    a[i] = num%2;
    num = num/2;
    i += 1;
    printf("%d", a[i - 1]);
    if (num == 0)
    {
        return 0;
    }
    else
    {
        return binary(num, a, i);
    }

}
```

OUTPUT :

```
e }
87
1110101
PS C:\Users\sugan\Documents\college fil
```

RESULT:

Thus the program has been executed and tested successfully

Exp 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,

max2Step: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] <

max1Step:11: Set max2 = arr[i]

Step:12: Print

max2Step: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

Step1: Start

Step2: Initialize inputString[100], outputArray[100], readIndex = 0, writeIndex

Step3: Input inputstring

Step4: Within while loop, if inputString[readIndex] == ' ' Then Increment the readIndex variable and break the loop

Step5: Iterate a ‘for’ loop using ‘readIndex’ from 0th index to n

Step6: if “inputString[readIndex]==' ' && inputString[readIndex-1]==' ' ” then

Step7: continue

Step8: Set outputArray[writeIndex] = inputString[readIndex]

Step9: Compute writeIndex++

Step10: Set outputArray[writeIndex] = '\0'

Step11: Print outputArray Step:12: Stop

7. PRINT THE STRING WITHOUT VOWELS

Step1: Start

Step2: Initialize the variables str[100], i, j, len=0

Step3: Accept the input str

Step4: 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' ”

Step6: Iterate a ‘for’ loop using ‘j’ from 0th index to len

Step7: Compute str[j]=str[j+1]

Step8: Compute len--

Step9: Terminate both for loop.Print the string without vowels.

Step10:Stop

PROGRAMS :

1. Write a program to print the second largest number in an array

CODE :

```
#include<stdio.h>
void main()
{
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, temp = 0, i, ival;
    for(i = 0; i < 10; i++)
    {
        if (a[i] > temp)
        {
            temp = a[i];
            ival = i;
        }
    }
    temp = 0;
    for(i = 0; i < 10; i++)
    {
        if ((a[i] > temp)&(i != ival))
        {
            temp = a[i];
        }
    }
```

```
    printf("%d ", temp);
    //20EUCS147
}
```

OUTPUT:

```
d Data structures\C programs\lab
9
PS C:\Users\sugan\Documents\colle
```

2. Write a program identify the missing number in an array

Example:

Input: N = 5

10 15 13 12 14

Output: 11

CODE :

```
#include<stdio.h>
void main()
{
    int size, first, last, i, count = 0, boo;
    printf("Enter the size of array :");
    scanf("%d", &size);
    int arr[size];
    for (i = 0; i < size; i++)
    {
        printf("Enter the %d th element", i);
        scanf("%d",&arr[i]);
    }
```

```
    first = arr[0];
    for(i = 0; i< size; i++)
    {
        if(arr[i] < first)
        {
            first = arr[i];
        }
    }
    last = 0;
    for(i = 0; i< size; i++)
    {
        if(arr[i] > last)
        {
            last = arr[i];
        }
    }
    last = arr[size - 1];
    for (i = first; i < last ; i++)
    {
        boo = 1;
        for(count = 0; count < size; count++)
        {
            if (i == arr[count])
            {
                boo = 0;
            }
        }
        if (boo)
        {
            printf("%d is the missing number ", i);
        }
        count++;
    }
    //20EUCS147

}
```

OUTPUT:

```
Enter the size of array :5
Enter the 0 th element10
Enter the 1 th element15
Enter the 2 th element13
Enter the 3 th element12
Enter the 4 th element14
11 is the missing number
PS C:\Users\sugan\Documents\college files\se
```

3. Write a program to remove a particular element in an array

CODE :

```c
#include<stdio.h>
void main()
{
    int arr[5] = {1, 2, 3, 4, 5}, i;
    printf("array is[ ");
    for ( i = 0; i < 5; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("]");
    int res[4] , rm_idx, j = 0;
    printf("Enter the index to remove :");
    scanf("%d", &rm_idx);
```

```
    for ( i = 0; i < 4; i++)
    {
        if( i == rm_idx)
        {
            j++;
        }
        res[i] = arr[j];
        j++;
    }
    printf("resultant array is[ ");
    for ( i = 0; i < 4; i++)
    {
        printf("%d ", res[i]);
    }
    printf("]");

}
```

OUTPUT:

```
array is[ 1 2 3 4 5 ]Enter the index to remove :2
resultant array is[ 1 2 4 5 ]
PS C:\Users\sugan\Documents\college files\sem 2\C and Data
```

4. Write a program to print the largest number in every column of a matrix

CODE :

```
#include<stdio.h>
void main()
{
    int mat[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}, m = 3, n = 3, i, j, max = 0;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            if(max < mat[i][j])
            {
                max = mat[i][j];
            }
```

```
        }
        printf("\n The biggest number in %d column is %d ", i + 1, max);
    }
}
```

OUTPUT:

```
 The biggest number in 1 column is 3
 The biggest number in 2 column is 6
 The biggest number in 3 column is 9
PS C:\Users\sugan\Documents\college files\sem 2\C
```

5. Write a program to check the equality of two strings without using pre-defined functions

CODE :

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[100], s2[100], i, boo = 1 ;
    printf("Enter the first string :");
    scanf("%s", s1);
    printf("Enter the second string :");
```

```
    scanf("%s", s2);
    if (strlen(s1) == strlen(s2))
    for ( i = 0; i < strlen(s1); i++)
    {
        if(s1[i] != s2[i])
        {
            printf("Not the Same");
            boo = 0;
            break;
        }
    }
    else
    {
        printf("Not the same");
        boo = 0;
    }
    if(boo)
    {
        printf("Both are same ");
    }

}
```

OUTPUT:

```
Enter the first string :sugan
Enter the second string :sugan
Both are same
PS C:\Users\sugan\Documents\college file
```

6. Write a program to remove unwanted white spaces in a sentence

CODE :

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

```
    char s1[100], s2[100], i, boo = 1 , j = 0;
    printf("Enter the first string :");
    gets(s1);
    for ( i = 0; i < strlen(s1); i++)
    {
        printf("\n i %d j %d", i, j);
        printf("\n Start");
        if (s1[i] == ' ' && !boo)
        {
            printf("\nif 1");
            i++;
        }
        if (s1[i] == ' ' && boo)
        {
            boo = 0;
            s2[j] = s1[i];
        }
        if (!(s1[i] == ' '))
        {
            printf("\nif 2");
            boo = 1;
            s2[j] = s1[i];
        }

        j++;
        printf("\n i %d j %d", i, j);
        printf("\nTermination \n");
    }
    s2[j + 1] == '\0';
    for (i = 0; i < j; i++)
    {
        printf("%c", s2[i]);
    }
}
```

OUTPUT:

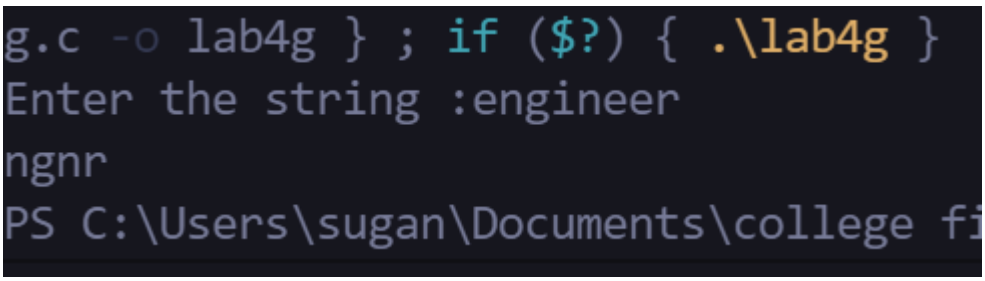
```
Enter the string :I    am    an    Indian

I am an Indian
PS C:\Users\sugan\Documents\college files\sem 2\C
```

7. Write a program to print the string without vowels

CODE :

```c
#include<stdio.h>
#include<string.h>
int check_vow(char);
void main()
{
    char s1[100], i ;
    printf("Enter the string :");
    scanf("%s", s1);
    for ( i = 0; i < strlen(s1); i++)
    {
        if(!check_vow(s1[i]))
        {
            printf("%c", s1[i]);
        }
    }

}
int check_vow(char t)
{
    if (t == 'a' || t == 'A' || t == 'e' || t == 'E' || t == 'i' || t == 'I' || t =='o' ||
 t=='O' || t == 'u' || t == 'U')
        return 1;
    else
        return 0;
}
```

OUTPUT:



RESULT:

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

Exp 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

20EUCS147-SUGAVANESH M
PROBLEMS :

1. Program to read and print ‘N’ number of students information {Name, Roll_no,Depart, Mobile}

CODE :

```c
#include <stdio.h>

struct student {
    char Name[100];
    int Roll_no;
    char dept[10];
    char mobile[10];

};

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

    }
    for (i = 0; i <= n; i++)
    {
        printf("\nthe student %d name is : %s", i + 1, lstarr[i].Name);
        printf("\nthe student %d Roll_no is : %d", i + 1, lstarr[i].Roll_no);
        printf("\nthe student %d dept is : %s", i + 1, lstarr[i].dept);
        printf("\nthe student %d mobile is : %s ", i + 1, lstarr[i].mobile);

    }
}
```

OUTPUT :

```
Enter the number of students :1

Enter the student 1 name :sugan

Enter the student 1 Roll_no :147

Enter the student 1 dept :CSE

Enter the student 1 mobile :8778559307

the student 1 name is : sugan
the student 1 Roll_no is : 147
the student 1 dept is : CSE
the student 1 mobile is : 8778559307
```

2. Program to find the top-3 ranks in a class based on marks using structures

CODE :

```c
#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 lstarr[n];
    for (i = 0; i < n; i++)
    {
        printf("\nEnter the student %d name :", i + 1);
        scanf("%s", lstarr[i].Name);
        printf("\nEnter the student %d total :", i + 1);
        scanf("%d", &lstarr[i].total);

    }
    for (i = 0; i < n; i++)
    {
        if (lstarr[i].total > max)
        {
            max = lstarr[i].total;
            temp = i;
        }
```

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

OUTPUT :

```
Enter the number of students :3

Enter the student 1 name :sugan

Enter the student 1 total :400

Enter the student 2 name :kali

Enter the student 2 total :540

Enter the student 3 name :don

Enter the student 3 total :230

 First ranker is kali
 second ranker is sugan
 third ranker is don
PS C:\Users\sugan\Documents\college files\
```

3. Program to copy one integer array to another array using pointer

CODE :

```c
#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 :

```
d Data structures\C programs\lab 5\" ; if ($?) { gcc tempCodeRunnerFil
Enter the 1 th number10
Enter the 2 th number9
Enter the 3 th number8
Enter the 4 th number7
Enter the 5 th number6
Enter the 6 th number5
Enter the 7 th number4
Enter the 8 th number3
Enter the 9 th number2
Enter the 10 th number1
The number is 10
The number is 9
The number is 8
The number is 7
The number is 6
The number is 5
The number is 4
The number is 3
The number is 2
The number is 1
PS C:\Users\sugan\Documents\college files\sem 2\C and Data structures\
```

4. Program to search a particular character in a string for its count using pointer

CODE :

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

void main()
{
    char usr_str[100], compare[1], *stp;
    int i, n, count = 0;
    printf("Enter the string :");
    gets(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 the string :hihoware
Enter the char :h
The count is 2
PS C:\Users\sugan\Documents\college files\sem 2\C and
```

5. Program to swap two numbers using call by reference

CODE :

```
#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
8 9
Before Swapping
x = 8
y = 9
After Swapping
x = 9
y = 8
PS C:\Users\sugan\Documents\college files\sem 2\C and Data stru
```

RESULT:

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

Exp 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

20EUCS147-SUGAVANESH M

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 it.

CODE:

```c
#include<stdio.h>
int main()
{
    FILE *f;
    char a;
    f=fopen("test.txt","r");
    while(a!=EOF){
        a=getc(f);
        printf("%c",a);
    }
    fclose(f);
    return 0;
}
```

OUTPUT:

```
d Data structures\C programs\l
hi how are
you
there dude
PS C:\Users\sugan\Documents\co
```

2. Program to copy the content of one file to another file.

CODE:

```c
#include<stdio.h>
int main()
{
    char a;
    FILE *f1,*f2;
    f1=fopen("test.txt","r");
    f2=fopen("test1.txt","w");
    do
    {
        a=getc(f1);
        putc(a,f2);
    }while(a!=EOF);
    printf("Content of the file is transferred.");
    fclose(f1);
    fclose(f2);
    return 0;
}
```

OUTPUT:

```
PS C:\Users\sugan\Documents\college files
d Data structures\C programs\lab 6\" ; if
Content of the file is transferred.
PS C:\Users\sugan\Documents\college files
```

3. Program to Capitalize First Letter of every Word in a File.

CODE:

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

void main()
{
    FILE *fp, *fp2;
    char pre, ch;
    int boo = 1;
```

```
    fp = fopen("test.txt", "r");
    fp2 = fopen("test1.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);
    fclose(fp);

}
```

OUTPUT:

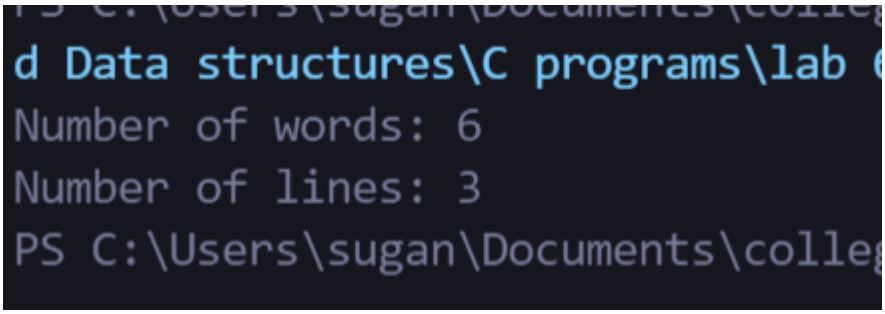
4. Program to Find the Number of words and lines in a Text File

CODE:

```
#include<stdio.h>
int main()
{   char c;
    FILE *f;
    int count = 0;
```

```
    int lines=1;
    f = fopen("test.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.

Exp no :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

PROGRAMS :

1. SINGLY LINKED LIST

CODE :

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

struct Node
{
    int data ;
    struct Node *next;

};


void main(){

    int choice, n, pos_val, i;
    struct Node *Head = NULL, *new, *temp, *pre;
    printf("1 for creating new Linked List \n2 for printing Linked List \n3 for adding ele
ment to first index \n4 for appending\n5 for adding element at desired place\n6 for deleti
ng an element");
    do
    {
        printf("\nEnter the option : ");
        /* 2 for creating new LinkedList
           2 for printing LinkedList
           3 for adding a element to the begining of the list
           4 for appending a element to the end of tha list
           5 for adding a node before another node
           6 for deleing an element
        */
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            // creation of LinkedList
            printf("\n Enter the node data :");
            scanf("%d", &n);
            while (n!=0)
            {
                new = (struct Node *)malloc(sizeof(struct Node));
                new->data = n;
                new->next = NULL;
                if(Head == NULL)
                {
                    Head = new;
                    temp = Head;
                }
                else
                {
                    temp->next = new;
                    temp = new;
                }
```

```
            printf(" Enter the node data :");
            scanf("%d", &n);
        }
        break;


    case 2:
        // Printing the LinkedList
        temp = Head;
        printf("%d", temp->data);
        temp = temp->next;
        while (temp != NULL)
        {
            printf(" -> %d", temp->data);
            temp = temp->next;
        }
        printf(" - > NULL");
        break;


    case 3:
        // adding element at the begining
        temp = Head;
        printf("\n Enter the num to be inserted :");
        scanf("%d", &n);
        new = (struct Node *)malloc(sizeof(struct Node));
        new->data = n;
        new->next = temp;
        Head = new;
        break;


    case 4:
        // appending a element
        temp = Head;
        printf("\n Enter the num to be append :");
        scanf("%d", &n);
        new = (struct Node *)malloc(sizeof(struct Node));
        new->data = n;
        new->next = NULL;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = new;
        break;


    case 5:
        // adding element at a postion
```

```
            temp = Head;
            printf("\n Enter the num to be inserted :");
            scanf("%d", &n);
            printf("\n Enter the node to be inserted  before :");
            scanf("%d", &pos_val);
            new = (struct Node *)malloc(sizeof(struct Node));
            new->data = n;
            new->next = NULL;
            while (temp != NULL)
            {
                // comparing that pos_val to get access to it's previous
                if (temp->data == pos_val)
                {
                    new->next = temp;
                    pre->next = new;
                    break;
                }
                pre = temp;
                temp = temp->next;
            }
            break;


        case 6:
            // removing element
            temp = Head;
            printf("\n Enter the num to be deleted :");
            scanf("%d", &n);
            while (temp != NULL)
            {
                // comparing that pos_val to get access to it's previous
                if (temp->data == n)
                {
                    pre->next = temp->next;
                    break;
                }
                pre = temp;
                temp = temp->next;
            }
            break;


        default:
            break;
        }

    } while (choice > 0);
    printf("----*----*----*----*----*----");
    printf("\nThanks for using this code :)\n");
    printf("----*----*----*----*----*----");
```

```
    //Done by 20EUCS147

}
```

OUTPUT:

```
1 for creating new Linked List
2 for printing Linked List
3 for adding element to first index
4 for appending
5 for adding element at desired place
6 for deleting an element
Enter the option : 1

 Enter the node data :1
 Enter the node data :2
 Enter the node data :3
 Enter the node data :4
 Enter the node data :5
 Enter the node data :0

Enter the option : 2
1 -> 2 -> 3 -> 4 -> 5 - > NULL
Enter the option : 3

 Enter the num to be inserted :9

Enter the option : 4

 Enter the num to be append :8

Enter the option : 5

 Enter the num to be inserted :8

 Enter the node to be inserted  before :2

Enter the option : 6

 Enter the num to be deleted :4

Enter the option : 2
9 -> 1 -> 8 -> 2 -> 3 -> 5 -> 8 - > NULL
```

2. DOUBLY LINKED LIST

CODE :

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data ;
    struct Node *next;
```

```
    struct Node *prev;

};


void main(){

    int choice, n, pos_val, i;
    struct Node *Head = NULL, *new, *temp, *pre;
    printf("1 for creating new Double Linked List \n2 for printing Double Linked List \n3
for adding element to first index \n4 for appending\n5 for adding element at desired place
\n6 for deletion");
    do
    {
        printf("\nEnter the option : ");
        /* 1 for creating new Double LinkedList
           2 for printing Double LinkedList
           3 for adding a element to the begining of the list
           4 for appending an element to the end of tha list
           5 for adding a node before another node
           6 for removing an element
        */
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            // creation of Double LinkedList
            printf("\n Enter the node data :");
            scanf("%d", &n);
            while (n!=0)
            {
                new = (struct Node *)malloc(sizeof(struct Node));
                new->data = n;
                new->next = NULL;
                new->prev = NULL;
                if(Head == NULL)
                {
                    Head = new;
                    Head->prev = NULL;
                    temp = Head;
                }
                else
                {
                    new->prev = temp;
                    temp->next = new;
                    temp = new;
                }
                printf(" Enter the node data :");
                scanf("%d", &n);
            }
```

```
            break;


        case 2:
            // Printing the Double LinkedList
            printf("\n by ascending ");
            temp = Head;
            printf("%d", temp->data);
            temp = temp->next;
            while (temp != NULL)
            {
                printf(" -> %d", temp->data);
                if (temp->next == NULL)
                {
                    pre = temp;
                }
                temp = temp->next;
            }
            printf(" - > NULL");

            printf("\n by desending ");
            temp = pre;
            printf("%d", temp->data);
            temp = temp->prev;
            while (temp != NULL)
            {
                printf(" -> %d", temp->data);
                temp = temp->prev;
            }
            printf(" - > NULL");
            break;


        case 3:
            // adding element at the begining
            temp = Head;
            printf("\n Enter the num to be inserted :");
            scanf("%d", &n);
            new = (struct Node *)malloc(sizeof(struct Node));
            new->data = n;
            temp->prev = new;
            new->next = temp;
            Head = new;
            Head->prev = NULL;
            break;


        case 4:
            // appending a element
            temp = Head;
```

```
            printf("\n Enter the num to be append :");
            scanf("%d", &n);
            new = (struct Node *)malloc(sizeof(struct Node));
            new->data = n;
            new->next = NULL;
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            new->prev = temp;
            temp->next = new;
            break;


        case 5:
            // adding element at a postion
            temp = Head;
            printf("\n Enter the num to be inserted :");
            scanf("%d", &n);
            printf("\n Enter the node to be inserted  before :");
            scanf("%d", &pos_val);
            new = (struct Node *)malloc(sizeof(struct Node));
            new->data = n;
            new->next = NULL;
            while (temp != NULL)
            {
                // comparing that pos_val to get access to it's previous
                if (temp->data == pos_val)
                {
                    new->prev = pre;
                    temp->prev = new;
                    new->next = temp;
                    pre->next = new;
                    break;
                }
                pre = temp;
                temp = temp->next;
            }
            break;


        case 6:
            // removing element
            temp = Head;
            printf("\n Enter the num to be deleted :");
            scanf("%d", &n);
            while (temp != NULL)
            {
                // comparing that pos_val to get access to it's previous
                if (temp->data == n)
```

```
                {
                    pre->next = temp->next;
                    temp->next->prev = pre;
                    break;
                }
                pre = temp;
                temp = temp->next;
            }
            break;


        default:
            break;
        }

    } while (choice > 0);
    printf("----*----*----*----*----*----");
    printf("\nThanks for using this code :)\n");
    printf("----*----*----*----*----*----");
    // made by 20EUCS147

}
```

OUTPUT:

```
1 for creating new Double Linked List
2 for printing Double Linked List
3 for adding element to first index
4 for appending
5 for adding element at desired place
6 for deletion
Enter the option : 1

 Enter the node data :1
 Enter the node data :2
 Enter the node data :3
 Enter the node data :4
 Enter the node data :5
 Enter the node data :0
```

```
Enter the option : 2

 by ascending 1 -> 2 -> 3 -> 4 -> 5 - > NULL
 by desending 5 -> 4 -> 3 -> 2 -> 1 - > NULL
Enter the option : 3

 Enter the num to be inserted :9

Enter the option : 4

 Enter the num to be append :8

Enter the option : 5

 Enter the num to be inserted :7

 Enter the node to be inserted  before :2

Enter the option : 6
```

3. CIRCULAR LINKED LIST

CODE:

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

struct Node {
    int  data;
    struct Node *next;
};

void main() {
    int choice, n, pos_val, i;
    struct Node *Head = NULL, *new, *temp, *pre;
    printf("1 for creating new Circular Linked List \n2 for printing Circular Linked List
\n3 for adding element to first index \n4 for appending\n5 for adding element at desired p
lace\n6 for removing a element");
    do
    {
        printf("\nEnter the option : ");
        /* 1 for creating new LinkedList
           2 for printing LinkedList
           3 for adding a element to the begining of the list
           4 for appending a element to the end of tha list
           5 for adding a node before another node
           6 for removing an element
        */
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            // creation of LinkedList
            printf("\n Enter the node data :");
            scanf("%d", &n);
```

```
            while (n!=0)
            {
                new = (struct Node *)malloc(sizeof(struct Node));
                new->data = n;
                new->next = NULL;
                if(Head == NULL)
                {
                    Head = new;
                    Head->next = Head;
                    temp = Head;
                }
                else
                {
                    new->next = Head;
                    temp->next = new;
                    temp = new;
                }
                printf(" Enter the node data :");
                scanf("%d", &n);
            }
            break;


        case 2:
            // Printing the LinkedList
            temp = Head;
            while (temp->next != Head)
            {
                printf(" %d -> ", temp->data);
                temp = temp->next;
            }
            printf("%d", temp->data);
            printf(" -> Head (%d)", temp->next->data);
            break;


        case 3:
            // adding element at the begining
            temp = Head;
            printf("\n Enter the num to be inserted :");
            scanf("%d", &n);
            new = (struct Node *)malloc(sizeof(struct Node));
            new->data = n;
            new->next = temp;
            while (temp->next != Head)
            {
                temp = temp->next;
            }
            temp->next = new;
            Head = new;
```

```
            break;


        case 4:
            // appending a element
            temp = Head;
            printf("\n Enter the num to be append :");
            scanf("%d", &n);
            new = (struct Node *)malloc(sizeof(struct Node));
            new->data = n;
            new->next = NULL;
            while (temp->next != Head)
            {
                temp = temp->next;
            }
            temp->next = new;
            new->next = Head;
            break;


        case 5:
            // adding element at a postion
            temp = Head;
            printf("\n Enter the num to be inserted :");
            scanf("%d", &n);
            printf("\n Enter the node to be inserted  before :");
            scanf("%d", &pos_val);
            new = (struct Node *)malloc(sizeof(struct Node));
            new->data = n;
            new->next = NULL;
            while (temp->next != Head)
            {
                // comparing that pos_val to get access to it's previous
                if (temp->data == pos_val)
                {
                    new->next = temp;
                    pre->next = new;
                    break;
                }
                pre = temp;
                temp = temp->next;
            }
            break;


        case 6:
            // removing element
            printf("\n---Note you cant delete Head---");
            temp = Head->next;
```

```
            printf("\n Enter the num to be deleted :");
            scanf("%d", &n);
            while (temp != Head)
            {
                // comparing that pos_val to get access to it's previous
                if (temp->data == n)
                {
                    pre->next = temp->next;
                    break;
                }
                pre = temp;
                temp = temp->next;
            }
            break;


        default:
            break;
        }

    } while (choice > 0);
    printf("----*----*----*----*----*----");
    printf("\nThanks for using this code :)\n");
    printf("----*----*----*----*----*----");
    // Made by 20EUCS147

}
```

OUTPUT:

```
1 for creating new Circular Linked List
2 for printing Circular Linked List
3 for adding element to first index
4 for appending
5 for adding element at desired place
6 for removing a element
Enter the option : 1

 Enter the node data :1
 Enter the node data :2
 Enter the node data :3
 Enter the node data :4
 Enter the node data :5
 Enter the node data :0
```

```
Enter the option : 2
 1 ->  2 ->  3 ->  4 -> 5 -> Head (1)
Enter the option : 3

 Enter the num to be inserted :8

Enter the option : 4

 Enter the num to be append :9

Enter the option : 5

 Enter the num to be inserted :8

 Enter the node to be inserted  before :2
```

```
Enter the option : 2
 8 ->  1 ->  8 ->  3 ->  4 ->  5 -> 9 -> Head (8)
```

RESULT :

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

Exp no :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

Step6: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 :

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define limit  5


void main()
{
    int choice, i = 0, j;
    int arr[limit];
    printf("\nEnter the choice 1 for adding , 2 for printing, 3 for pop :");
    scanf("%d", &choice);
    do
    {

    switch(choice)
    {
        case 1:
            if (i >= limit)
            {
                printf("\nStack  over flow\n");
                break;
            }

            printf("\nEnter the new Top :");
            scanf("%d", &arr[i]);
            i++;
            break;

        case 2:
        printf("\nPrinting the stack \n");
            printf("Top\n");
            for (j = i; j > 0; j--)
            {
                printf("%d \n", arr[j - 1]);
            }
            printf("Base\n");
```

```
            break;

        case 3:
            if(i <= 0)
            {
                printf("\nStack in underflow \n");
                break;
            }
            printf("\nTop element is popped \n");
            i--;
            break;
        default:
            break;
    }
    printf("\nEnter the choice 1 for adding , 2 for printing, 3 for pop :");
    scanf("%d", &choice);
    } while (choice > 0);
}
```

OUTPUT :

```
Enter the choice 1 for adding , 2 for printing, 3 for pop :1

Enter the new Top :1

Enter the choice 1 for adding , 2 for printing, 3 for pop :1

Enter the new Top :2

Enter the choice 1 for adding , 2 for printing, 3 for pop :1

Enter the new Top :3

Enter the choice 1 for adding , 2 for printing, 3 for pop :3

Top element is popped

Enter the choice 1 for adding , 2 for printing, 3 for pop :2

Printing the stack
Top
2
1
Base
```

RESULT:

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

Exp no :8	STACK IMPLEMENTATION 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

CODE:

```c
#include<stdio.h>
#include<stdlib.h>
#define size 5

struct Node
{
    int data;
    struct Node *next;
};

void main()
{
    int choice, n, check = 0;
    struct Node *Top=NULL,*new,*temp;
    do
    {
        printf("\nEnter the choice 1 for adding , 2 for printing, 3 for pop :");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                if(check >= 5)
                {
                    printf("\nStack overflow :(\n");
                    break;
                }
                printf("\nEnter the new Top :");
                scanf("%d", &n);
                new = (struct Node *)malloc(sizeof(struct Node));
                new->data = n;
                new->next = NULL;

                if ( Top == NULL)
                {
                    Top = new;
                    temp = Top;
                }

                else
                {
                    new->next = Top;
                    Top = new;
                }
                check++;
                break;


            case 2:
```

```
                printf("\nPrinting The Stack \n");
                temp = Top;
                printf("Top\n");

                while (temp != NULL)
                {
                    printf("%d\n", temp->data);
                    temp = temp->next;
                }

                printf("Base\n");
                break;


            case 3:
                if(check <= 0)
                {
                    printf("\nStack in Underflow\n");
                    exit;
                }
                printf("\nTop element is poped");
                temp = Top->next;
                Top = temp;
                check--;
                break;


            default:
                break;
        }

    }while(choice > 0);


}
```

20EUCS147-SUGAVANESH M
OUTPUT:

```
Enter the choice 1 for adding , 2 for printing, 3 for pop :1

Enter the new Top :1

Enter the choice 1 for adding , 2 for printing, 3 for pop :1

Enter the new Top :2

Enter the choice 1 for adding , 2 for printing, 3 for pop :1

Enter the new Top :3

Enter the choice 1 for adding , 2 for printing, 3 for pop :3

Top element is poped
Enter the choice 1 for adding , 2 for printing, 3 for pop :2

Printing The Stack
Top
2
1
Base
```

Exp no :9	STACK APPLICATIONS

AIM :

To write a C program to perform stack application.

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

Step7 : Assign e=exp;

Step8 : 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. Convension of infix to postfix

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
char stack[size], eq[20], res[20];
int Top = -1, base = 0, i, j, boo = 1;


void push(char);
char pop();
void display();
int opre(char);
void validate(char);
void special(char);


void main()
{
    printf("\n Enter the string :");
    scanf("%s", eq);
    printf("\n the string %s\n", eq);
    for(i = 0; i < 10; i++)
    {

        if(opre(eq[i]) < 0)
        {
            printf("%c", eq[i]);
        }
        else if(eq[i] == '(')
        {
            boo = 0;
```

```
        }
        else if(eq[i] == ')')
        {
            boo = 1;
        }
        else if (!boo)
        {
            special(eq[i]);
        }
        else if(opre(eq[i]) > 0 && boo)
        {
            validate(eq[i]);
        }
    }
    for(i = Top; i >= 0; i--)
    {

        printf("%c", stack[i]);
    }


}


void push(char ch)
{
    if (Top == size - 1)
    {
        printf("Stack overflow");
    }
    else
    {
        Top++;
        stack[Top] = ch;
    }
}

char pop()
{
    if(Top == -1)
    {
        printf("error occured ");
    }
    else
    {
        Top--;
        return stack[Top + 1];
    }
}
```

```c
void display()
{
    int i;
    for(i = 0; i <= Top; i++)
    {
        printf("%c", stack[i]);
    }
}

int opre(char ch)
{
    switch(ch)
    {
        case '-':
            return 1;
        case '+':
            return 1;
        case '*':
            return 2;
        case '/':
            return 2;
        case '^':
            return 3;
        case '(':
            return 4;
        case ')':
            return 4;
        default:
            return -1;
    }
}
void validate(char ch)
{
    First:
    if (Top == -1)
    {
        push(ch);
    }
    else if (opre(ch) > opre(stack[Top]))
    {
        push(ch);
    }
    else
    {

        printf("%c",pop());
        goto First;
    }
```

```
}

void special(char ch)
{
    char lim = Top;
    First:
    if (Top == lim)
    {
        push(ch);
    }
    else if (opre(ch) > opre(stack[Top]) && Top > lim)
    {
        push(ch);
    }
    else
    {

        if (lim > Top)
        {
            printf("%c",pop());
            goto First;
        }

    }
}
```

OUTPUT:

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

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

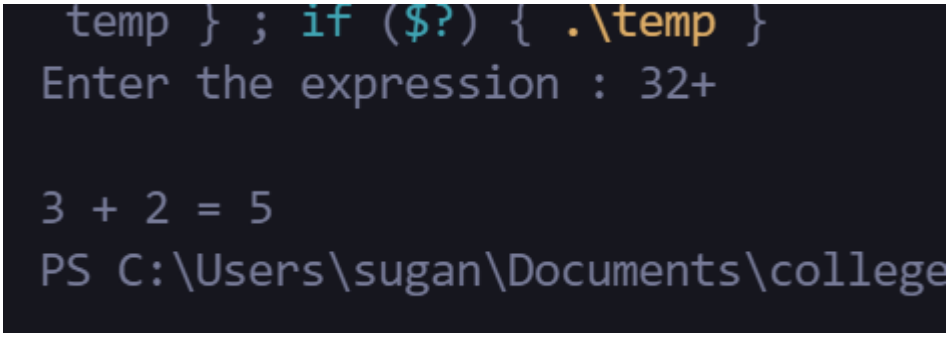
2. POSTFIX EXPRESSION EVALUATION

CODE:

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

int stack[20], top = -1;

void push(int);
int pop();

void main()
{
    char exp[20];
    char *e;
    int num0,num1,num2,num4;
    printf("\nEnter the expression: ");
```

```
    scanf("%s",exp);
    e = exp;
    while(*e != '\0')
    {
        if(isdigit(*e))
        {
            num0 = *e - 48;
            push(num0);
        }
        else
        {
            num1 = pop();
            num2 = pop();
            switch(*e)
            {
            case '+':
                num4 = num1 + num2;
                break;
            case '-':
                num4 = num2 - num1;
                break;
            case '*':
                num4 = num1 * num2;
                break;
            case '/':
                num4 = num2 / num1;
                break;
            }
            push(num4);

        }
        e++;

    }


    printf("\nThe result of expression %s  =  %d", exp, pop());
}

void push(int x)
{
    stack[++top] = x;
}

int pop()
{
    return stack[top--];
}
```

OUTPUT:



RESULT:

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

Exp 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

CODE :

```
#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);
    } while (boo);
```

```
    printf("----*----*----*----*----*----");
    printf("\nThanks for using this code :)\n");
    printf("----*----*----*----*----*----");
    //Done by sugan0tech

}

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 for enqueue
2 for printing
3 for dequeue

 Enter the option:1

Enter the num1

Enter opt :1

Enter the num2

Enter opt :1

Enter the num3

Enter opt :2
 1 2 3
Front : 1 Rear : 3
Enter opt :3

Enter opt :2
 1 2
Front : 1 Rear : 2
```

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.

2. ENQUEUE()

   Step 1-START

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

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

CODE:

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

// Node definition
struct Node
{
    int data ;
    struct Node *next;

};


void main()
{
    struct Node *Rear = NULL, *Front = NULL, *temp, *new;
    int n, option;
    // 1 for creation
    // 2 for enqueue
    // 3 for display
    // 4 for dequeue
    printf("1 for creation\n2 for enqueue\n3 for print\n4 for dequeue");
    printf("\nEnter your option :");
    scanf("%d", &option);
    do
    {
        switch (option)
        {
            case 1:
                printf("\n Enter the num :");
                scanf("%d", &n);
                do
                {
                    new = (struct Node *)malloc(sizeof(struct Node));
                    new->data = n;
                    new->next = NULL;
                    // case for if Front empty
                    if(Front == NULL)
                    {
                        Front = new;
                        Rear = new;
                        temp = Front;
                    }
                    // After test case
                    else
                    {
                        temp->next = new;
```

```
                    temp = new;
                    Rear = new;
                }
                printf("\n Enter the num :");
                scanf("%d", &n);
            }while(n  != 0);
            break;


        case 2:
            new = (struct Node *)malloc(sizeof(struct Node));
            printf("\n Enter the num :");
            scanf("%d", &n);
            // enqueueing the data i.e pushing
            new->data = n;
            new->next = NULL;
            Rear->next = new;
            Rear = new;
            break;


        case 3:
            temp = Front;
            // printing using same linked list principle
            while (temp)
            {
                printf("%d ", temp->data);
                temp = temp->next;
            }
            printf("\nFront : %d Rear : %d", Front->data, Rear->data);
            break;


        case 4:
            // dequeueiug just removing the Front one value
            temp = Front;
            Front = temp->next;
            break;


    }
    printf("\nEnter your option :");
    scanf("%d", &option);
} while (option);


printf("----*----*----*----*----*----");
printf("\nThanks for using this code :)\n");
printf("----*----*----*----*----*----");
//Done by sugan0tech
```

```
}
```

OUTPUT:

```
1 for creation
2 for enqueue
3 for print
4 for dequeue
Enter your option :1

 Enter the num :1

 Enter the num :2

 Enter the num :3

 Enter the num :4

 Enter the num :0

Enter your option :2

 Enter the num :2

Enter your option :4

Enter your option :3
2 3 4 2
Front : 2 Rear : 2
```

RESULT:

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

Exp 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.

CODE:

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

// node formation here it have additional data of priority value
struct Node
{
    int data;
    int priority;
    struct Node *next;
};


void main()
{
    struct Node *Front=NULL,*Rear=NULL,*temp,*new;
    int choice,element,priority;

    do
    {
        printf("\n1 Insertion\n2 Deletion\n3 DIsplay");
        printf("\nEnter the choice :");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
                // getting
                printf("Enter the data:");
                scanf("%d",&element);
                printf("Enter the priority:");
                scanf("%d",&priority);
                // assignments and allocations
                new=(struct Node*)malloc(sizeof(struct Node));
                new->data=element;
                new->priority=priority;
                new->next=NULL;

                if(Front==NULL)
                {
```

```
                    Front  = new;
                    Rear = Front;
                }
                else
                {
                    if(new->priority > Front->priority)
                    {
                        Front->next = new;
                        Rear = new;
                    }
                    else if(new->priority < Rear->priority)
                    {
                        new->next = Front;
                        Front = new;
                    }
                    else{
                        temp=Front;
                        while(temp->next!=NULL)
                        {
                            if(new->priority>temp->priority && new->priority<temp->next-
>priority)
                            {
                                new->next  = temp->next;
                                temp->next = new;
                                break;
                            }
                        temp =temp->next;
                        }
                    }
                }
            break;


            case 2:
                    if(Front==NULL)
                        printf("\nQueue is empty");
                    else
                    {
                        temp=Front;
                        Front = Front->next;
                        free(temp);
                    }
                break;


            case 3:
                if(Front==NULL)
                    printf("\nQueue is empty");
                else{
                    temp=Front;
```

```
                    printf("\n --- Ascending order ---");
                    printf("\nDequeue region Front");
                    printf("\nFront |-");
                    while(temp->next!=NULL)
                    {
                        printf(" data :%d priority :%d <-",temp->data,temp->priority);
                        temp=temp->next;
                    }
                    printf(" data :%d priority :%d ",temp->data,temp->priority);
                    printf("-| Rear");
                    printf("\nEnqueue region Rear");
                }
                break;


        }

    }while(choice);


    printf("----*----*----*----*----*----");
    printf("\nThanks for using this code :)\n");
    printf("----*----*----*----*----*----");
    //Done by sugan0tech


}
```

OUTPUT :

```
1 Insertion
2 Deletion
3 DIsplay
Enter the choice :1
Enter the data:1
Enter the priority:1

1 Insertion
2 Deletion
3 DIsplay
Enter the choice :1
Enter the data:2
Enter the priority:3

1 Insertion
2 Deletion
3 DIsplay
Enter the choice :3

 --- Ascending order ---
Dequeue region Front
Front |- data :1 priority :1 <- data :2 priority :3 -| Rear
Enqueue region Rear
1 Insertion
2 Deletion
3 DIsplay
```

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

CODE:

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

// node formation here it have additional data of priority value
struct Node
{
    int data;
    int priority;
    struct Node *next;
};


void main()
```

```
{
    struct Node *Front=NULL,*Rear=NULL,*temp,*new;
    int choice,element,priority;

    do
    {
        printf("\n1 Insertion\n2 Deletion\n3 DIsplay");
        printf("\nEnter the choice :");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
                // getting
                printf("Enter the data:");
                scanf("%d",&element);
                printf("Enter the priority:");
                scanf("%d",&priority);
                // assignments and allocations
                new=(struct Node*)malloc(sizeof(struct Node));
                new->data=element;
                new->priority=priority;
                new->next=NULL;

                if(Front==NULL)
                {
                    Front  = new;
                    Rear = Front;
                }
                else
                {
                    if(new->priority < Front->priority)
                    {
                        Front->next = new;
                        Rear = new;
                    }
                    else if(new->priority > Rear->priority)
                    {
                        new->next = Front;
                        Front = new;
                    }
                    else{
                        temp=Front;
                        while(temp->next!=NULL)
                        {
                            if(new->priority>temp->priority && new->priority<temp->next-
>priority)
                            {
                                new->next  = temp->next;
```

```
                            temp->next = new;
                            break;
                        }
                    temp =temp->next;
                    }
                    }
            }
        break;


        case 2:
                if(Front==NULL)
                    printf("\nQueue is empty");
                else
                {
                    temp=Front;
                    Front = Front->next;
                    free(temp);
                }
        break;


        case 3:
            if(Front==NULL)
                printf("\nQueue is empty");
            else{
                temp=Front;
                printf("\n --- Desending order ---");
                printf("\nDequeue region Front");
                printf("\nFront |-");
                while(temp->next!=NULL)
                {
                    printf(" data :%d priority :%d <-",temp->data,temp->priority);
                    temp=temp->next;
                }
                printf(" data :%d priority :%d ",temp->data,temp->priority);
                printf("-| Rear");
                printf("\nEnqueue region Rear");
            }
        break;


    }

}while(choice);


printf("----*----*----*----*----*----");
printf("\nThanks for using this code :)\n");
printf("----*----*----*----*----*----");
```

```
    //Done by sugan0tech


}
```

OUTPUT :

```
1 Insertion
2 Deletion
3 DIsplay
Enter the choice :1
Enter the data:1
Enter the priority:1

1 Insertion
2 Deletion
3 DIsplay
Enter the choice :1
Enter the data:2
Enter the priority:2

1 Insertion
2 Deletion
3 DIsplay
Enter the choice :3

 --- Desending order ---
Dequeue region Front
Front |- data :2 priority :2 <- data :1 priority :1 -| Rear
Enqueue region Rear
1 Insertion
2 Deletion
3 DIsplay
```

RESULT:

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

Exp 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

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_Q_SIZE 500


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


struct node** createQueue(int *, int *);
void enQueue(struct node **, int *, struct node *);
struct node *deQueue(struct node **, int *);


void printLevelOrder(struct node* root)
{
    int rear, front;
    struct node **queue = createQueue(&front, &rear);
    struct node *temp_node = root;
```

```
    while (temp_node)
    {
        printf("%d ", temp_node->data);


        if (temp_node->left)
            enQueue(queue, &rear, temp_node->left);

        if (temp_node->right)
            enQueue(queue, &rear, temp_node->right);

        temp_node = deQueue(queue, &front);
    }
}


struct node** createQueue(int *front, int *rear)
{
    struct node **queue =
        (struct node **)malloc(sizeof(struct node*)
                                        *MAX_Q_SIZE);

    *front = *rear = 0;
    return queue;
}

void enQueue(struct node **queue, int *rear,
                            struct node *new_node)
{
    queue[*rear] = new_node;
    (*rear)++;
}

struct node *deQueue(struct node **queue, int *front)
{
    (*front)++;
    return queue[*front - 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);
    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
PS C:\Users\sugan\Documents\college files\sem 2
```

RESULT:

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

Exp no :14	BINARY SEARCH TREE

AIM :

To write a C program to implement 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.

CODE:

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

struct Node
{
    int data;
    struct Node *left;
    struct Node *right;

};


struct Node* constructTreeUtil(int pre[], int* preIndex, int low, int high, int size)
{
    if (*preIndex >= size || low > high)
        return NULL;

    struct Node* root = (struct Node *)malloc(sizeof(struct Node));
    root->left = NULL;
    root->right = NULL;
    root->data = pre[*preIndex];
    *preIndex = *preIndex + 1;

    if (low == high)
        return root;

    int i;
    for (i = low; i <= high; ++i)
        if (pre[i] > root->data)
            break;

    root->left = constructTreeUtil(pre, preIndex, *preIndex, i - 1, size);
    root->right = constructTreeUtil(pre, preIndex, i, high, size);
    return root;
```

```
}

struct Node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, 0, size - 1, size);
}

void inorder(struct Node *t)
{
    if (t!=NULL)
    {
        inorder(t->left);
        printf("%d ", t->data);
        inorder(t->right);
    }

}

void preorder(struct Node *t)
{
    if (t!=NULL)
    {
        printf("%d ", t->data);
        preorder(t->left);
        preorder(t->right);
    }

}


int FindMin(struct Node *tree)
{
    int min;
    min = tree->data;
    if (tree->left == NULL)
    {
        return min;
    }
    FindMin(tree->left);

}

int FindMax(struct Node *tree)
{
    int max;
    max = tree->data;
    if (tree->right == NULL)
    {
        return max;
```

```
    }
    FindMax(tree->right);

}
int search(struct Node *tree, int n)
{
    if (n == tree->data)
    {
        printf("Found it ");
        return 0;
    }
    else if (n < tree->data)
    {
        search(tree->left, n);
    }
    else if (n > tree->data)
    {
        search(tree->right, n);
    }

}
void main()
{
    int opt;
    printf("1 for default values\n2 for custom values\n");
    printf(" Enter the option\n");
    scanf("%d", &opt);
    if (opt == 1)
    {
        int pre[] = { 27 ,19 ,14, 10, 35, 31, 42 };
        int size = sizeof(pre) / sizeof(pre[0]);
        struct Node *head;
        head = constructTree(pre, size);
        printf("Display in Inorder ");
        inorder(head);
        printf("\nThe min value %d \n", FindMin(head));
        printf("The max value %d ", FindMax(head));
        int n;
        printf("Enter the num to be searched ");
        scanf("%d", &n);
        search(head, n);

    }
    else if(opt == 2)
    {
        int pre[100], i = 0, data, size;
        struct Node *Tree;
        printf("\nEnte the pre order values, enter 0 to exit ");
        do
```

```
        {
            printf("\nEnter the data :");
            scanf("%d", &data);
            pre[i] = data;
            i++;
        }while(data);
        size = i - 1;
        Tree = constructTree(pre, size);
        printf("\nDisplay in Inorder ");
        inorder(Tree);
        printf("\nThe min value %d", FindMin(Tree));
        printf("\nThe max value %d ", FindMax(Tree));

        int n;
        printf("Enter the num to be searched ");
        scanf("%d", &n);
        search(Tree, n);

    }
    else
    {
        printf("Thank you ");
    }

}
```

OUTPUT:

```
1 for default values
2 for custom values
 Enter the option
1
Display in Inorder 10 14 19 27 31 35 42
The min value 10
The max value 42 Enter the num to be searched 27
Found it
PS C:\Users\sugan\Documents\college files\sem 2\C and
```

RESULT:

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

successfully.

Exp no :15	HASHING TECNIQUES

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 6: attach the newnode as first node

step 7: else

step 8 : 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 7: if i< TABLE_SIZE then go to step 4

1. Separate chaining.

CODE:

```c
#include<stdio.h>
#include<stdlib.h>
#define size 10

struct Node
{
    int data;
    struct Node *next;

};


void print(struct Node *node)
{
    if (node == NULL)
    {
        printf(" NULL");
        exit;
    }

    printf("%d", node->data);
    node = node->next;
    // usual linked list print loop
    while (node != NULL)
    {
        printf(" -> %d", node->data);
        node = node->next;
    }
    printf(" - > NULL");
}


int hash(int a)
{
    return a%10;
}


void insert(struct Node *arr[], int val)
{
    int key = hash(val);
    struct Node *node, *temp;
```

```
    node = (struct Node *)malloc(sizeof(struct Node));
    node->data = val;
    node->next = NULL;
    if (arr[key] == NULL)
    {
        arr[key] = node;
    }
    else if (arr[key] != NULL)
    {
        temp = arr[key];
        while (temp->next)
        {
            temp = temp->next;
        }
        temp->next = node;
    }


}


void main()
{
    int val, key;
    struct Node *hash_table[size], *temp = NULL;
    for (int i = 0; i < size; i++)
    {
        hash_table[i] = NULL;
    }

    printf("Enter the val :");
    scanf("%d", &val);
    insert(hash_table, val);
    while(val != -1)
    {
        printf("\nEnter the val :");
        scanf("%d", &val);
        if (val == -1)
        {
            break;
        }

        insert(hash_table, val);
    }


    for (int i = 0; i < size; i++)
    {
        if (hash_table[i] == NULL)
        {
```

```
            printf("\nkey :%d | val :NULL", i);
            continue;
        }

        printf("\nkey :%d | val :",i);
        print(hash_table[i]);
    }

}
```

OUTPUT:

```
Enter the val :27

Enter the val :72

Enter the val :65

Enter the val :36

Enter the val :42

Enter the val :18

Enter the val :37

Enter the val :19

Enter the val :46

Enter the val :0
```

```
Enter the val :0

Enter the val :-1

key :0 | val :0 - > NULL
key :1 | val :NULL
key :2 | val :72 -> 42 - > NULL
key :3 | val :NULL
key :4 | val :NULL
key :5 | val :65 - > NULL
key :6 | val :36 -> 46 - > NULL
key :7 | val :27 -> 37 - > NULL
key :8 | val :18 - > NULL
key :9 | val :19 - > NULL
```

2. Linear probing

CODE:

```
#include<stdio.h>
#define size 10

int hash(int a)
{
    return a%10;
}

void insert(int arr[], int val)
{
    int key = hash(val);
    if (arr[key] == -1)
    {
        arr[key] = val;
    }
    else if (arr[key] != -1)
    {
        int temp = 0;
        while (arr[key] != -1)
        {
            if(temp == size)
            {
                break;
            }
            temp++;
            key = hash(key + 1);
            if (arr[key] == -1)
            {
                arr[key] = val;
                break;
            }

        }
```

```
    }


}
void main()
{
    int val, key, hash_table[size];
    for (int i = 0; i < size; i++)
    {
        hash_table[i] = -1;
    }

    printf("Enter the val :");
    scanf("%d", &val);
    insert(hash_table, val);
    while(val != -1)
    {
        printf("\nEnter the val :");
        scanf("%d", &val);
        insert(hash_table, val);
    }

    for (int i = 0; i < size; i++)
    {
        if (hash_table[i] == -1)
        {
            printf("\nkey :%d -> val :Empty", i);
            continue;
        }

        printf("\nkey :%d -> val :%d",i , hash_table[i]);
    }

}
```

OUTPUT:

```
Enter the val :27

Enter the val :14

Enter the val :38

Enter the val :72

Enter the val :42

Enter the val :63

Enter the val :-1

key :0 -> val :Empty
key :1 -> val :Empty
key :2 -> val :72
key :3 -> val :42
key :4 -> val :14
key :5 -> val :63
key :6 -> val :Empty
key :7 -> val :27
key :8 -> val :38
key :9 -> val :Empty
PS C:\Users\sugan\Documents\college files\sem 2\C and Data structures\C programs\lab 16>
```

RESULT:

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