

PROGRAM NO: 1.

DATE:

AIM: Write a c program to search the given element within the list of linear elements using the linear search.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Linear Search Algorithm.

Operates on a Linear Data Structure like arrays and Linked lists.

Accepts input to find the client's data in the available data.

Iterates over the entire linear data and return the index of the search element in the available data.

PROGRAM:

```
#include<stdio.h>

int LinearSearch(int arr[],int x,int k)
{
    for(int i=0;i<x;i++)
    {
        if(arr[i]==k)
        {
            return i;
        }
    }
    return -1;
}

void main()
{
    int n,a[100],i,key;
    printf("\n\tNo.of Elements : ");
    scanf("%d",&n);
    printf("\n\tInput the elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
}
```

```
}  
printf("\n\tSearch Element : ");  
scanf("%d",&key);  
printf("\n\tThe index in the array is : %d.",LinearSearch(a,n,key));  
}
```

EXPECTED OUTPUT:

Size of array:6

Elements of the array : 1 2 3 4 5 6

Input number to search:4

OBSERVED OUTPUT:

```
Size of array : 8
```

```
Elements of the array : 5 52 41 7 5 3 6 82
```

```
Input number to search : 41
```

```
Found at the index : 2
```

PROGRAM NO: 2.

DATE:

AIM: Write a c program to search the given element within the list of linear elements using the Binary search.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Binary Search Algorithm.

Operates on a Linear Data Structure like arrays and Linked lists which must be **pre-sorted**.

Accepts input to find a specific data in the data available.

Operates on the principle of divide and conquer technique and returns the index of search element if found.

PROGRAM:

```
#include<stdio.h>

int BinarySearch(int arr[],int n,int key)
{
    int s,e,mid;
    s=0;e=n;
    while(s<=e)
    {
        mid = ((s+e)/2);
        if(arr[mid]==key)
        {
            return mid;
        }
        else if(arr[mid]>key)
        {
            e = mid-1;
        }
        else
        {
            s=mid+1;
        }
    }
}
```

```
        return -1;
    }
    void main()
    {
        int n,a[100],key,i;
        printf("\n\tNo.of elements : ");
        scanf("%d",&n);
        printf("\n\tInput the elements : ");
        for(i=0;i<n;i++)
        {
            scanf("%d",&a[i]);
        }
        printf("\n\tSearch Element : ");
        scanf("%d",&key);
        printf("\n\tThe index in the array is : %d.",BinarySearch(a,n,key));
    }
```

EXPECTED OUTPUT:

No of elements:6

Input the elements :21 41 14 5 6 10

Search Elements :6

The index in the array is : 4

OBSERVED OUTPUT:

```
No.of elements : 8
```

```
Input the elements : 1 3 2 5 7 8 6 4
```

```
Search Element : 7
```

```
The index in the array is : 4.
```

PROGRAM NO: 3

DATE:

AIM: Write a c program to f=search the given elements within the list of linear elements using the fibonacci search.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Fibonacci Search Algorithm.

Operates on a Linear Data Structure like arrays and Linked lists which are pre-sorted.

Accepts input, to find the client's data in the available data.

Operates on the principle of comparison and returns the index of search element if found.

PROGRAM:

```
//FIBONACCI SERACH
//PRE-REQUISITES = ARRAY MUST BE SORTED IN ASCENDING ORDER BEFORE GIVING AS INPUT
#include<stdio.h>
void main()
{
    int i,j,a[100],offset,M,M1,M2,key,n;//declaration of necessary variables
    printf("\n\tNo.of Elements : ");
    scanf("%d",&n); //size of array
    printf("\n\tInput the elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]); // elements of array
    }
    M2=0; // Pre-assignment of fibonacci numbers
    M1=1;
    M=M1+M2;
    printf("\n\tSearch Element : ");
    scanf("%d",&key); // number to be searched
    while(M<n) // iterating for finding fibonacci just greater than or equal to 'n'- size of the array
    {
```

```

    M2=M1;

    M1=M;

    M=M1+M2;
}
offset=0; // useful variable for keeping notes of index
while(M>1)
{
    j=M2+offset; // for first iteration offset = 0
    if(a[j]==key)
    {
        printf("\n\t%d",j);

        break;
    }
    else if(key>a[j])
    {
        offset=j;

        M=M1;

        M1=M2;

        M2=M-M1;
    }
    else
    {
        offset=j;

        M=M2;

        M1=M1-M2;

        M2=M-M1;
    }
}
if(M<=1)
{
    printf("\nThe number is not found in the given array .");}

```


EXPECTED OUTPUT:

No of Elements:8

Input the elements: 5 9 4 3 2 0 7 6

Search Element :3

The index in the array is :3

OBSERVED OUTPUT:

```
No.of Elements : 10
```

```
Input the elements : 11 14 52 36 41 66 41 20 22 10
```

```
Search Element : 66
```

```
5
```

PROGRAM NO: 4.**DATE:**

AIM: Write C program to sort the given list of elements using the bubble sort Technique.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Bubble Sort Algorithm.

Operates on a Linear Data Structure like arrays.

Iterates over the entire array and sorts the data by comparison technique.

After all the iterations with nested looping concept, returns the sorted array.

PROGRAM:

```
#include<stdio.h>

int main()
{
    int a[50],i,j,n,t;

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

    scanf("%d",&n);

    printf("\nEnter the elements:\n ");

    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }

    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(a[i]>a[j])
            {
                t=a[i];
                a[i]=a[j];
                a[j]=t;
            }
        }
    }
}
```

```
printf("Sorted array in ascending order\n");  
for(i=0;i<n;i++)  
{  
    printf("%d \t",a[i]);  
}  
printf("\n");  
return 0;}
```

EXPECTED OUTPUT:

Enter number of elements :5

Enter the elements:7 9 100 10 4

Sorted array in the ascending order is : 4 7 9 10 100

OBSERVED OUTPUT:

```
Enter number of elements
8
Enter the elements:
19 20 4 53 26 2 8 28
Sorted array in ascending order
2      4      8      19      20      26      28      53

Process returned 0 (0x0)   execution time : 18.510 s
Press any key to continue.
```

PROGRAM NO: 5.

DATE:

AIM:Write a c program to sort the given list of elements using the insertion sort Technique.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Insertion Sort Algorithm.

Operates on a Linear Data Structure like arrays.

Iterates over the entire array and sorts the data by comparison and inserting the element at it's right position in the given array.

After all the iterations with nested looping concept, returns the sorted array.

PROGRAM:

```
#include <stdio.h>

void main()
{
    int a[100],n,i,count,temp;

    printf("Enter the no of elements to be entered:");

    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }

    count=1;

    while(count<n)
    {
        for(i=0;i<n-count;i++)
        {
            if(a[i]>a[i+1])
            {
                temp=a[i];
                a[i]=a[i+1];
                a[i+1]=temp;
            }
        }
        count++;
    }

    for(i=0;i<n;i++)
        printf("%d\t",a[i]);
    printf("\n");
}
```

```
        a[i+1]=temp;
    }
}
count++;
}
for(i=0;i<n;i++)
{
    printf("%d\t",a[i]);
}
}
```

EXPECTED OUTPUT:

Enter the no of elements to be entered :5

19 2 5 100 26

The sorted list is: 2 5 19 26 100

OBSERVED OUTPUT:

```
Enter the no of elements to be entered:10
110 25 36 817 28 36 21 26 12 82
12 21 25 26 28 36 36 82 110 817
Process returned 10 (0xA) execution time : 30.256 s
Press any key to continue.
```

PROGRAM NO: 6.

DATE:

AIM: Write a c program to sort the given list of elements using the selection sort Technique.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Selection Sort Algorithm.

Operates on a Linear Data Structure like arrays.

Iterates over the entire array and sorts the data by comparison and swapping the element at it's right position w r t compared element in the given array.

After all the iterations with nested looping concept, returns the sorted array.

PROGRAM:

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

```
void main()
```

```
{
```

```
    int n,a[50],temp;
```

```
    printf("No.of Elements : ");
```

```
    scanf("%d",&n);
```

```
    printf("\nElements : ");
```

```
    for(int i=0;i<n;i++)
```

```
    {
```

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

```
    }
```

```
    for(int i=0;i<n-1;i++)
```

```
    {
```

```
        for(int j=i+1;j<n;j++)
```

```
        {
```

```
            if(a[j]<a[i])
```

```
            {
```

```
                temp = a[j];
```

```
                a[j] = a[i];
```

```
                a[i] = temp;
```



```
        }  
    }  
}  
printf("\n");  
printf("After Selection Sort : ");  
  
for(int i=0;i<n;i++)  
{  
    printf("%d ",a[i]);  
}  
  
}
```

EXPECTED OUTPUT:

No.of Elements : 7

Elements : 6 56 4 5 3 1 89

After Selection Sort : 1 3 4 5 6 56 89

OBSERVED OUTPUT:A screenshot of a Windows command prompt window. The title bar at the top reads "E:\6.exe". The command prompt shows the following text:
No.of Elements : 10
Elements : 9 8 7 6 5 4 3 2 1 10
After Selection Sort : 1 2 3 4 5 6 7 8 9 10
Process returned 10 (0xA) execution time : 10.888 s
Press any key to continue.

PROGRAM NO: 7.**DATE:**

AIM: Write a c program to sort the given list of elements using the quick sort Technique.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Quick Sort Algorithm.

Operates on a Linear Data Structure like arrays and linked lists.

Based on the Divide and Conquer Technique.

Picks an element as a pivot and partitions the given array around the picked pivot and returns the sorted data structure.

PROGRAM:

```
#include<stdio.h>

void swap(int arr[],int i,int j)
{
    int temp;
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

int partition(int arr[],int l,int r)
{
    int pivot = arr[r];
    int i = l-1;
    for(int j=l;j<r;j++)
    {
        if(arr[j]<pivot)
        {
            i++;
            swap(arr,i,j);
        }
    }
    swap(arr,i+1,r);
```

```

        return i+1;
    }
    void Quick_Sort(int arr[],int l,int r)
    {
        if(l<r)
        {
            int pi = partition(arr,l,r);
            Quick_Sort(arr,l,pi-1);
            Quick_Sort(arr,pi+1,r);
        }
    }
}

```

```

void main()
{
    int arr[50],n,i,l,r;
    printf("No.of Elements : ");
    scanf("%d",&n);
    printf("\nElements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    l=0;r=n-1;
    Quick_Sort(arr,l,r);
    printf("\nAfter Quick Sort : ");
    for(i=0;i<n;i++)
    { printf("%d ",arr[i]);
    }
}

```

EXPECTED OUTPUT:

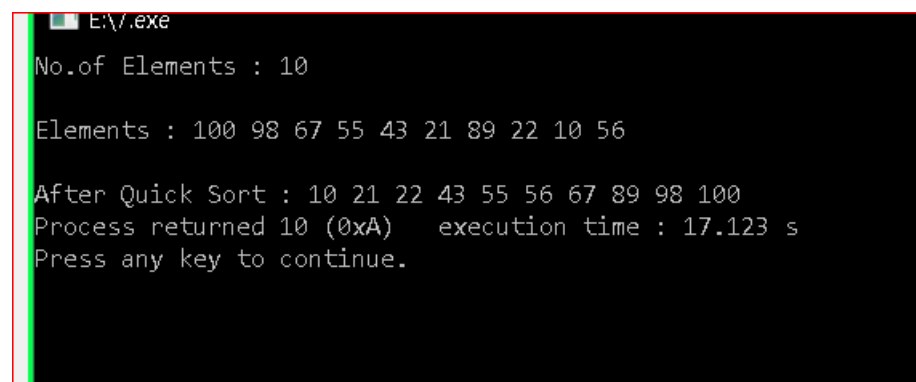
No of elements: 6

Elements: 6 5 4 3 2 1

After quick sort: 1 2 3 4 5 6

Process returned 6(0x6) execution time: 9.788s

Press any key to continue

OBSERVED OUTPUT:

```
E:\v.exe
No.of Elements : 10
Elements : 100 98 67 55 43 21 89 22 10 56
After Quick Sort : 10 21 22 43 55 56 67 89 98 100
Process returned 10 (0xA) execution time : 17.123 s
Press any key to continue.
```

PROGRAM NO: 8.**DATE:**

AIM: Write a c program to sort the given list of elements using the merge sort Technique.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Quick Sort Algorithm.

Operates on a Linear Data Structure like arrays and linked lists.

Based on the Divide and Conquer Technique.

Divides the array into two halves and then merges two sorted halves.

PROGRAM:

```
#include<stdio.h>

void merge(int arr[],int l,int mid,int r)
{
    int i,j,k;
    int n1 = mid-l+1;
    int n2 = r-mid;
    int a[n1],b[n2];
    for(i=0;i<n1;i++)
    {
        a[i]=arr[l+i];
    }
    for(i=0;i<n2;i++)
    {
        b[i]=arr[mid+1+i];
    }
    i=0,j=0,k=l;
    while(i<n1 && j<n2)
    {
        if(a[i] < b[j])
        {
            arr[k]=a[i];
            k++;i++;
        }
    }
}
```

```

    }
else
{
    arr[k]=b[j];
    k++;j++;
}
}
while(i<n1)
{
    arr[k]=a[i];
    k++;i++;
}
while(j<n2)
{
    arr[k]=b[j];
    k++;j++;
}
}
void MergeSort(int arr[],int l,int r)
{
    if(l<r)
    {
        int mid = (l+r)/2;
        MergeSort(arr,l,mid);
        MergeSort(arr,mid+1,r);
        merge(arr,l,mid,r);
    }
}
void main(){
    int arr[100],l,r,i,n;
    printf("No.of Elements : ");

```

```
scanf("%d",&n);  
printf("\nElements : ");  
for(i=0;i<n;i++)  
{  
    scanf("%d",&arr[i]);  
}  
l=0;r=n;  
MergeSort(arr,l,r);  
printf("\nAfter Merge Sort : ");  
for(i=0;i<n;i++)  
{  
    printf("%d ",arr[i]);  
}  
}
```


EXPECTED OUTPUT:

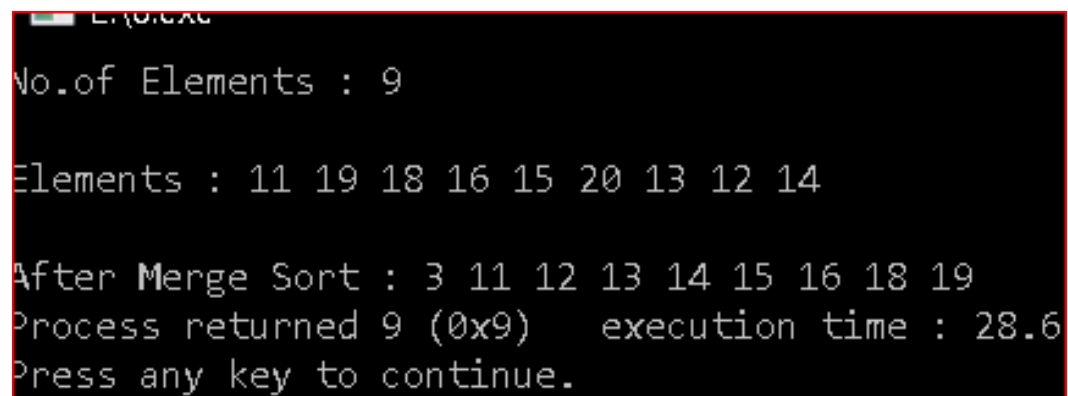
No of elements: 6

Elements: 8 90 67 56 43 10

After merge sort: 8 10 43 56 67 90

Process returned 6(0x6) execution time: 3.557s

Press any key to continue.

OBSERVED OUTPUT:A screenshot of a C++ program's output in a terminal window. The text is as follows:
No.of Elements : 9
Elements : 11 19 18 16 15 20 13 12 14
After Merge Sort : 3 11 12 13 14 15 16 18 19
Process returned 9 (0x9) execution time : 28.6
Press any key to continue.
The terminal window has a dark background and a red border. The title bar at the top shows a small icon and the text "C++".

PROGRAM NO: 9.

DATE:

AIM: Write a c program to sort the given list of elements using the radix sort Technique.

DESCRIPTION:

The program is developed in C language.

Uses the concept of Radix Sort Algorithm.

Operates on a Linear Data Structure like arrays and linked lists.

Also know as bucket sort or digit by digit sort technique.

Uses counting sort as a subroutine to sort returns the sorted data structure.

PROGRAM:

```
#include<stdio.h>

int get_max(int a[], int n){
    int max = a[0];
    for (int i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];
    return max;
}

void radix_sort (int a[], int n){
    int bucket[10][10], bucket_cnt[10];
    int i, j, k, r, NOP = 0, divisor = 1, lar, pass;
    lar = get_max (a, n);
    while (lar > 0){
        NOP++;
        lar /= 10;
    }
    for (pass = 0; pass < NOP; pass++){
        for (i = 0; i < 10; i++){
            bucket_cnt[i] = 0;
        }
        for (i = 0; i < n; i++){
            r = (a[i] / divisor) % 10;
```

```

        bucket[r][bucket_cnt[r]] = a[i];
        bucket_cnt[r] += 1;
    }
    i = 0;
    for (k = 0; k < 10; k++){
        for (j = 0; j < bucket_cnt[k]; j++){
            a[i] = bucket[k][j];
            i++;
        }
    }
    divisor *= 10;
    printf ("After pass %d : ", pass + 1);
    for (i = 0; i < n; i++)
        printf ("%d ", a[i]);
    printf ("\n");
}
}

int main (){
    int i, n, a[10];
    printf ("Enter the number of items to be sorted: ");
    scanf ("%d", &n);
    printf ("Enter items: ");
    for (i = 0; i < n; i++){
        scanf ("%d", &a[i]);
    }
    radix_sort (a, n);
    printf ("Sorted items : ");
    for (i = 0; i < n; i++)
        printf ("%d ", a[i]);
    printf ("\n"); return 0; }

```

EXPECTED OUTPUT:

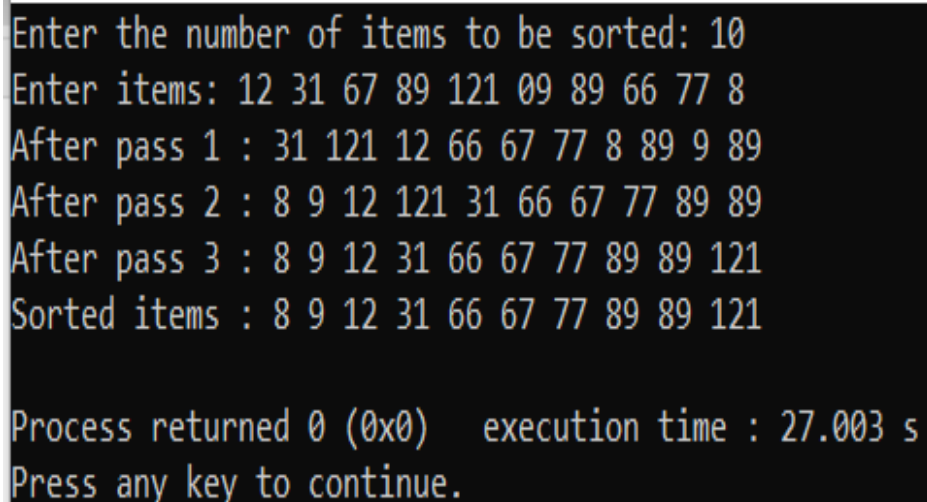
Enter the number of items to be sorted :8

Enter the items:90 78 56 34 25 27 88 99

After pass 1 : 90 34 25 56 27 78 88 99

After pass 2 : 25 27 34 56 78 88 90 99

After pass 3 : 25 27 34 56 78 88 90 99

OBSERVED OUTPUT:

```
Enter the number of items to be sorted: 10
Enter items: 12 31 67 89 121 09 89 66 77 8
After pass 1 : 31 121 12 66 67 77 8 89 9 89
After pass 2 : 8 9 12 121 31 66 67 77 89 89
After pass 3 : 8 9 12 31 66 67 77 89 89 121
Sorted items : 8 9 12 31 66 67 77 89 89 121

Process returned 0 (0x0)   execution time : 27.003 s
Press any key to continue.
```

PROGRAM NO: 10.**DATE:**

AIM: Write a c program to create a single linked list with set of five elements using the static representation(arrays).

DESCRIPTION:

The program is developed in C language.

Using the concepts of structures and pointers.

Passes the linear data to a method defined for creating and linking each node with others respectively.

Uses loops for assigning the data to the data field of the node and

Next node's address in the address field designated to a pointer as a data member to the structure of node.

PROGRAM:

```
#include <stdio.h>

#define N 5

// A Linked List Node
struct Node
{
    int data;
    struct Node* next;
};

struct Node node[N];

// Helper function to print a given linked list
void printList(struct Node* head)
{
    struct Node* curr = head;
    while (curr)
    {
        printf("%d -> ", curr->data);
        curr = curr->next;
    }
    printf("NULL");
}
```

```
struct Node* createStaticList(int arr[])
{
    for (int i = 0; i < N; i++)
    {
        node[i].data = arr[i];
        node[i].next = NULL;

        if (i > 0) {
            node[i - 1].next = &node[i];
        }
    }
    return node;
}

int main(void)
{
    int arr[N] = { 6,7,8,9,10};
    struct Node* root = createStaticList(arr);
    printList(root);
    return 0;
}
```

EXPECTED OUTPUT:

1->2->3->4->5->NULL

OBSERVED OUTPUT:

```
6 -> 7 -> 8 -> 9 -> 10 -> NULL
Process returned 0 (0x0)   execution time : 0.372 s
Press any key to continue.
```

PROGRAM NO: 11.

DATE:

AIM: Write a c program to create a single linked list with the set of five elements using Dynamic representation (free pool storage).

DESCRIPTION:

The program is developed in C language.

Uses the concept of dynamic memory allocation by including stdlib.h library.

Pointers and Structures concepts are used in creation and implementation of the single linked list.

Malloc method is used for assigning the memory in the heap of the computer.

Data and address of nodes are assigned using member access operator (->) as in the case of instance is a pointer.

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

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

typedef struct node node;

void print_list(node *n)
{
    while(n!=NULL)
    {
        printf("%d->",n->data);
        n=n->next;
    }
    printf("NULL");
}

void main()
{
    node *h,*sec,*thi,*fou,*fif;

    h = (node*)malloc(sizeof(node));
```



```
sec = (node*)malloc(sizeof(node));
thi = (node*)malloc(sizeof(node));
fou = (node*)malloc(sizeof(node));
fif = (node*)malloc(sizeof(node));
h->data = 9;h->next=sec;
sec->data = 8;sec->next=thi;
thi->data = 7;thi->next = fou;
fou->data = 6;fou->next = fif;
fif->data = 5;fif->next = NULL;
printf("\nThe Single Linked List using Dynamic Representation is : ");
print_list(h);
}
```

EXPECTED OUTPUT:

The single link list using dynamic representation is: 10->20->30->40->50->NULL

Process return (0x40) execution time: 0.031 s

Press any key to continue.

OBSERVED OUTPUT:

```
The Single Linked List using Dynamic Representation is : 9->8->7->6->5->NULL
Process returned 4 (0x4)   execution time : 0.031 s
Press any key to continue.
```

PROGRAM NO:12.**DATE:****AIM:** Write a c program to exhibit the operation on single linked list.

Operations on single linked list.

1. Traverse the list -> code for traversal
2. Display elements of the list. ->code to Display
- 3.Insert a node
 - a. Insert at First ->code to insert at First
 - b. Insert at last ->code to insert at Last
 - c. Insert at Middle ->code to insert at Middle
- 4.Delete a node
 - a.Delete at First ->code to Delete at First
 - b.Delete at last ->code to Delete at last
 - c.Delete at Middle ->code to Delete at Middle

DESCRIPTION:

The program is developed in C language.

Uses the concept of structures and nested structures.

Dynamic memory allocation is implemented to assign the memory for the nodes.

Specific methods are defined for each operation as requested by the user.

Using switch case statements specific functions are called based on the choice of the user.

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

struct node
{
    int data;

    struct node *next;
};

typedef struct node node;

void print_list(node *n)
{
    printf("\n");
```

```

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

void insert_at_first(node **head_ref)
{
    node *new_node = (node*)malloc(sizeof(node));
    printf("\nData : ");
    scanf("%d",&new_node->data);
    new_node->next = *head_ref;
    *head_ref = new_node;
}

void insert_at_last(node *prev_node)
{
    node *new_node = (node*)malloc(sizeof(node));
    printf("\nData : ");
    scanf("%d",&new_node->data);
    new_node->next = NULL;
    prev_node->next = new_node;
}

void insert_at_middle(node *prev_node)
{
    node *new_node = (node*)malloc(sizeof(node));
    printf("\nData : ");
    scanf("%d",&new_node->data);
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}

```

```

void delete_at_first(node **head_ref)
{
    node *temp;
    temp = *head_ref;
    *head_ref = temp->next;
    free(temp);
}

void delete_at_end(node *n)
{
    node *prev;
    while(n->next != NULL)
    {
        prev = n;
        n = n->next;
    }
    prev->next=NULL;
    free(n);
}

void delete_at_any(node *end_prev)
{
    node *temp;
    temp = end_prev->next;
    end_prev->next=temp->next;
    free(temp);
}

void main()
{
    node *h,*fir,*sec,*thi,*fou,*fif;
    int choice;
    h = (node*)malloc(sizeof(node));
    sec = (node*)malloc(sizeof(node));

```

```

thi = (node*)malloc(sizeof(node));
fou = (node*)malloc(sizeof(node));
fif = (node*)malloc(sizeof(node));
h->data=99;h->next=sec;
sec->data=88;sec->next=thi;
thi->data = 77;thi->next=fou;
fou->data = 66;fou->next=fif;
fif->data = 55;fif->next = NULL;
printf("\nSelect :");
printf("\n\t 1 - Traverse/Print \n\t 2- Insertion \n\t 3 - Deletion : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
    print_list(h);
    break;
case 2:
    printf("\n\t 1 - first \n\t 2 - middle \n\t 3 - last : ");
    scanf("%d",&choice);
    switch(choice)
    {
case 1:
        insert_at_first(&h);
        print_list(h);
        break;
case 2:
        printf("\n\t Position of Insertion : ");
        scanf("%d",&choice);
        switch(choice)
        {
case 1:

```

```

        insert_at_middle(h);

        print_list(h);

        break;
case 2:
    insert_at_middle(sec);

    print_list(h);

    break;
case 3:
    insert_at_middle(thi);

    print_list(h);

    break;
case 4:
    insert_at_middle(fou);

    print_list(h);

    break;
}
break;

```

```

case 3:
    insert_at_last(fif);

    print_list(h);

    break;
}
break;

```

```

case 3:

printf("\n\t 1 - first \n\t 2 - middle \n\t 3 - last : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

```

```

        delete_at_first(&h);

        print_list(h);

        break;
case 2:
    printf("\n\t Position of Deletion : ");
    scanf("%d",&choice);
    switch(choice)
    {
    case 1:
        delete_at_any(h);
        print_list(h);
        break;
    case 2:
        delete_at_any(sec);
        print_list(h);
        break;
    case 3:
        delete_at_any(thi);
        print_list(h);
        break;
    case 4:
        delete_at_any(fou);
        print_list(h);
        break;
    }
    break;
case 3:
    delete_at_end(fou);
    print_list(h);
    break;
}

```



```
        break;  
    }  
}
```

EXPECTED OUTPUT:

Select:

1- Traverse/print

2- Insertion

3- Deletion: 1

99->88->77->66->55->NULL

Process return 4 (0x4) execution time :5

Press any key to continue,

OBSERVED OUTPUT:

```
Select :
    1 - Traverse/Print
    2- Insertion
    3 - Deletion : 3

    1 - first
    2 - middle
    3 - last : 2

    Position of Deletion : 3

99->88->77->55->NULL
Process returned 4 (0x4)   execution time : 7.1
Press any key to continue.
```

PROGRAM NO: 13.**DATE:**

AIM: Write a c program to create a circular single linked list with a set of five elements.

DESCRIPTION:

The program is developed in C language.

Uses the concept of pointers, structures and dynamic memory allocation.

Any method either malloc and calloc can be implemented by using stdlib.h library in the source code.

PROGRAM:

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

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

typedef struct Node node;

void print_list(node *head_ref)
{
    if(head_ref==NULL)
    {
        printf("\nThe list does not exist \n");
        return;
    }
    printf("The Circular Double Linked list is : ");
    node *temp = head_ref;
    do
    {
        printf("<-%d->",temp->data);
        temp = temp->next;
    }
```

```
    }while(temp != head_ref);  
}  
  
void main()  
{  
    node *head,*second,*third,*fourth,*fifth;  
    head = (node*)malloc(sizeof(node));  
    second = (node*)malloc(sizeof(node));  
    third = (node*)malloc(sizeof(node));  
    fourth = (node*)malloc(sizeof(node));  
    fifth = (node*)malloc(sizeof(node));  
    head->data = 20;  
    second->data = 30;  
    third->data = 40;  
    fourth->data = 60;  
    fifth->data = 80;  
    head->next = second;head->prev = fifth;  
    second->next = third;second->prev=third;  
    third->next = fourth;third->prev=second;  
    fourth->next = fifth;fourth->prev=third;  
    fifth->next = head;fifth->prev=fourth;  
    print_list(head);  
}
```

EXPECTED OUTPUT:

1->2->3->4->5->

OBSERVED OUTPUT:

```
6->7->8->9->10->
Process returned 1512528 (0x171450)   execution time : 0.370 s
Press any key to continue.
```

PROGRAM NO: 14.

DATE:

AIM: Write a c program to create a Double linked list with set of five elements using the static representation(arrays).

DESCRIPTION:

The program is developed in C language.

Uses pointers, structures and arrays

Takes input of the elements into an static array and passed to a method defined for creating of nodes.

Structure arrays are implemented for creating the nodes of the double linked list.

Links are established and connected on either sides of each nodes and traversed using dedicated methods.

PROGRAM:

```
#include<stdio.h>

struct Node
{
    struct Node* p;
    int d;
    struct Node* n;
};

typedef struct Node node;

void printlist(node* head_ref)
{
    node* temp = head_ref;
    printf("\n\t The Double Linked List created using static method is :");
    printf("\n\n\tNULL<->");
    while(temp)
    {
        printf("%d<->",temp->d);
        temp = temp->n;
    }
    printf("NULL\n\n\t");
}
```

```

void static_list_creation(int arr[],int n)
{
    node A[n];
    for(int i = 0;i<n;i++)
    {
        A[i].d = arr[i];

        if(i==0)
        {
            A[i].p = NULL;
            A[i].n = &A[1];
        }
        else if(i==n-1)
        {
            A[i].p = &A[i-1];
            A[i].n = NULL;
        }
        else
        {
            A[i].n = &A[i+1];
            A[i].p = &A[i-1];
        }
    }
    printlist(A);
}

void main()
{
    int arr[5] = {1,2,3,4,5};
    static_list_creation(arr,5);
}

```

EXPECTED OUTPUT:

The Double Linked List created using static method is :

NULL<->1<->2<->3<->4<->5<->NULL

OBSERVED OUTPUT:

```
The Double Linked List created using static method is :  
NULL<->10<->9<->8<->7<->6<->NULL  
  
Process returned 7 (0x7)   execution time : 0.352 s  
Press any key to continue.
```


PROGRAM NO:15.**DATE:**

AIM: Write a c program to create a Double linked list with the set of five elements using Dynamic representation (free pool storage).

DESCRIPTION:

The program is developed in C language.

Uses the concept of POINTERS, STRUCTURES AND DYNAMIC MEMORY ALLOCATION.

Malloc method is implemented with typecasting of the pointer it return type as of node.

Each node is allocated its memory in the heap.

Every node is connected with the previous and next node's using the pointer field specified within the nodes as previous and next.

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

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

typedef struct Node node;

void printlist(node *head)
{
    node *temp;
    temp = head;
    do
    {
        printf("%d->",temp->data);
        temp = temp->next;
    }while(temp != head);
}

void main()
{
```

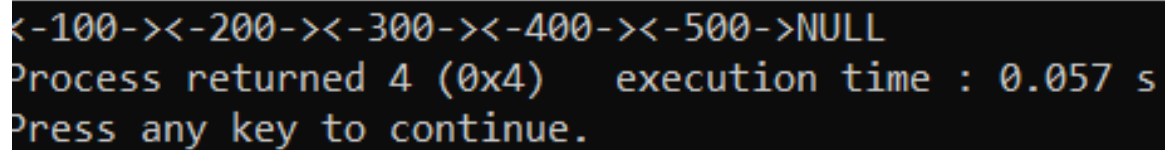
```
node *head,*second,*third,*fourth,*fifth;
head = (node*)malloc(sizeof(node));
second = (node*)malloc(sizeof(node));
third = (node*)malloc(sizeof(node));
fourth = (node*)malloc(sizeof(node));
fifth = (node*)malloc(sizeof(node));
head->data = 1;
second->data = 2;
third->data = 3;
fourth->data = 4;
fifth->data = 5;
head->prev = fifth;
head->next = second;
second->prev = head;
second->next = third;
third->next = fourth;
third->prev = second;
fourth->next = fifth;
fourth->prev = third;
fifth->next = head;
fifth->prev = fourth;
printlist(head);
}
```

EXPECTED OUTPUT:

<-1-><-2-><-3-><-4-><-5->NULL

Process returned 4(0x4) execution time: 0.335s

Press any key to continue.

OBSERVED OUTPUT:A screenshot of a terminal window with a black background and yellow text. The text displays the expected output of a program, including a sequence of pointers, the return value, execution time, and a prompt to press a key.

```
<-100-><-200-><-300-><-400-><-500->NULL  
Process returned 4 (0x4)   execution time : 0.057 s  
Press any key to continue.
```

AIM: Write a C program to create a circular double linked list with set of five elements.

DESCRIPTION:

The program is written in C language.

Uses the concept of pointers, structures and dynamic memory allocation concepts.

Nodes are created using structure data type with two pointers for pointing previous and next nodes.

Data field is assigned data using member access operator.

Malloc method is implemented for allocating memory in the heap.

PROGRAM:

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

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

```
struct Node
```

```
{
```

```
    struct Node *prev;
```

```
    int data;
```

```
    struct Node *next;
```

```
};
```

```
typedef struct Node node;
```

```
void print_list(node *head_ref)
```

```
{
```

```
    if(head_ref==NULL)
```

```
    {
```

```
        printf("\nThe list does not exist \n");
```

```
        return;
```

```
    }
```

```
    printf("The Circular Double Linked list is : ");
```

```
    node *temp = head_ref;
```

```
    do
```

```
    {
```

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

```

        temp = temp->next;
    }while(temp != head_ref);
}

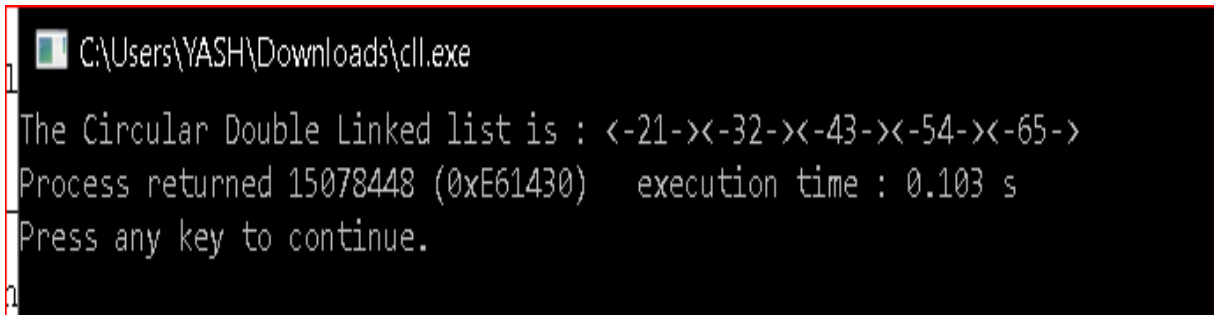
void main()
{
    node *head,*second,*third,*fourth,*fifth;
    head = (node*)malloc(sizeof(node));
    second = (node*)malloc(sizeof(node));
    third = (node*)malloc(sizeof(node));
    fourth = (node*)malloc(sizeof(node));
    fifth = (node*)malloc(sizeof(node));
    head->data = 20;
    second->data = 30;
    third->data = 40;
    fourth->data = 60;
    fifth->data = 80;
    head->next = second;head->prev = fifth;
    second->next = third;second->prev=third;
    third->next = fourth;third->prev=second;
    fourth->next = fifth;fourth->prev=third;
    fifth->next = head;fifth->prev=fourth;
    print_list(head);

}

```

EXPECTED OUTPUT:

The circular Double Linked List is : <-20-><-30-><-40-><-50-><-80->

OBSERVED OUTPUT:

```
C:\Users\YASH\Downloads\dll.exe
The Circular Double Linked list is : <-21-><-32-><-43-><-54-><-65->
Process returned 15078448 (0xE61430) execution time : 0.103 s
Press any key to continue.
```

AIM: Write a c program to exhibit the operation on double linked list.

Hint:Using the switch case create a Menu:

Operations on Double linked list.

1. Traverse the list -> code for traversal
2. Display elements of the list. ->code to Display
- 3.Insert a node
 - a. Insert at First ->code to insert at First
 - b. Insert at last ->code to insert at Last
 - c. Insert at Middle ->code to insert at Middle
- 4.Delete a node
 - a.Delete at First ->code to Delete at First
 - b.Delete at last ->code to Delete at last
 - c.Delete at Middle ->code to Delete at Middle

DESCRIPTION:

The program is developed in C language.

Uses the concept of POINTERS, STRUCTURES AND DYNAMIC MEMORY ALLOCATION.

Malloc method is implemented with typecasting of the pointer it return type as of node.

Each node is allocated its memory in the heap.

Every node is connected with the previous and next node's using the pointer field specified within the nodes as previous and next.

Respective functions are created to perform each operation individually in the program using switch case, user is provided with choice to take a particular action on the double linked list.

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

struct Node
{
    struct Node *prev;

    int data;

    struct Node *next;
};
```

```

typedef struct Node node;

//code for traversing the linked list
void traverse(node *iptr)
{
    printf("\n\t");
    while(iptr != NULL)
    {
        printf("%d<->",iptr->data);
        iptr = iptr->next;
    }
    printf("\n\tTraversion completed\n\t");
}

// code for traversing the list and printing the data
void print_list(node *iptr)
{
    printf("NULL<->");
    while(iptr != NULL)
    {
        printf("%d<->",iptr->data);
        iptr = iptr->next;
    }
    printf("NULL");
}

// code for insertion at node
void in_at_first(node **head_ref )// for inserting a new node at the starting of linked list
{
    node *new_node = (node*)malloc(sizeof(node));
    node *temp = *head_ref;
    printf("\nInput the data of the new node to insert : ");
    scanf("%d",&new_node->data);
    new_node->next = *head_ref;

```



```

    new_node->prev = NULL;
    temp->prev = new_node;
    *head_ref = new_node;
}

void in_at_any(node *prev_node)
{
    node *new_node = (node*)malloc(sizeof(node));
    printf("\nInput the data of the new node to insert : ");
    scanf("%d",&new_node->data);
    new_node->next = prev_node->next;
    prev_node->next->prev = new_node;
    new_node->prev = prev_node;
    prev_node->next = new_node;
}

void in_at_last(node *prev_node)
{
    node *new_node = (node*)malloc(sizeof(node));
    printf("\nInput the data of the new node to insert : ");
    scanf("%d",&new_node->data);
    new_node->next = NULL;
    prev_node->next = new_node;
    new_node->prev = prev_node;
}

// codes for deletion of a node
void del_at_first(node **head_ref)
{
    node *temp,*temp1;
    temp = *head_ref;
    *head_ref = temp->next;
    temp1 = temp->next;
    temp1->prev = NULL;

```

```

    free(temp);
}
void del_at_end(node *n)
{
    node *prev = n->prev;
    prev->next = NULL;
    free(n);
}
void del_at_any(node *prev_n)
{
    prev_n->next->prev = prev_n->prev;
    prev_n->prev->next = prev_n->next;
    free(prev_n);
}
void main()
{
    node *head,*second,*third,*fourth,*fifth,*sixth;
    int choice;
    head = (node*)malloc(sizeof(node));
    second = (node*)malloc(sizeof(node));
    third = (node*)malloc(sizeof(node));
    fourth = (node*)malloc(sizeof(node));
    fifth = (node*)malloc(sizeof(node));
    sixth = (node*)malloc(sizeof(node));
    head->data = 6;
    second->data = 7;
    third->data = 8;
    fourth->data = 8;
    fifth->data = 9;
    sixth->data = 10;
    head->next = second;head->prev = NULL;

```

```

second->next = third;second->prev = head;

third->next = fourth;third->prev = second;

fourth->next = fifth;fourth->prev = third;

fifth->next = sixth;fifth->prev = fourth;

sixth->next = NULL;sixth->prev = fifth;

printf("\nPlease select any one of four choices : \n");

printf("\nSelect \n\t1 for Traversing the list \n\t2 for Printing the linked list \n\t3 for performing
Insertion \n\t4 for performing deletion : ");

scanf("%d",&choice);

switch (choice)
{
    case 1:
        traverse(head);
        break;
    case 2:
        print_list(head);
        break;
    case 3:
        printf("\nSelect 1 for Inserting at the beginning of list,\nSelect 2 for performing Insertion at
Middle or Any position,\nSelect 3 for performing Insertion at end : ");
        scanf("%d",&choice);
        switch (choice)
        {
            case 1:
                in_at_first(&head);
                printf("\n\n\tThe new list formed after the operation is : ");
                print_list(head);
                break;
            case 2:
                printf("\n\tSpecify the position after which insertion should take place : ");
                scanf("%d",&choice);
                switch (choice)

```

```

{
    case 1:
        in_at_any(head);
        printf("\n\n\tThe new list formed after the operation is : ");
        print_list(head);
        break;
    case 2:
        in_at_any(second);
        printf("\n\n\tThe new list formed after the operation is : ");
        print_list(head);
        break;
    case 3:
        in_at_any(third);
        printf("\n\n\tThe new list formed after the operation is : ");
        print_list(head);
        break;
    case 4:
        in_at_any(fourth);
        printf("\n\n\tThe new list formed after the operation is : ");
        print_list(head);
        break;
    case 5:
        in_at_any(fifth);
        printf("\n\n\tThe new list formed after the operation is : ");
        print_list(head);
        break;
}
break;

```

```

case 3:
    in_at_last(sixth);

```

```

        printf("\n\n\tThe new list formed after the operation is : ");
        print_list(head);

        break;
    }

    break;

case 4:

    printf("\nSelect 1 for Deleting at the beginning of list,\nSelect 2 for performing Deletion at
Middle or Any position,\nSelect 3 for performing Deletion at end : ");

    scanf("%d",&choice);

    switch (choice)
    {
        case 1:

            del_at_first(&head);

            printf("\n\n\tThe new list formed after the operation is : ");

            print_list(head);

            break;

        case 2:

            printf("\n\tSpecify the position after which Deletion should take place : ");

            scanf("%d",&choice);

            switch (choice)
            {
                case 1:

                    del_at_any(head);

                    printf("\n\n\tThe new list formed after the operation is : ");

                    print_list(head);

                    break;

                case 2:

                    del_at_any(second);

                    printf("\n\n\tThe new list formed after the operation is : ");

                    print_list(head);

                    break;
            }
        }
    }

```

```

        case 3:
            del_at_any(third);
            printf("\n\n\tThe new list formed after the operation is : ");
            print_list(head);
            break;
        case 4:
            del_at_any(fourth);
            printf("\n\n\tThe new list formed after the operation is : ");
            print_list(head);
            break;
        case 5:
            del_at_any(fifth);
            printf("\n\n\tThe new list formed after the operation is : ");
            print_list(head);
            break;
    }
    break;
case 3:
    del_at_end(sixth);
    printf("\n\n\tThe new list formed after the operation is : ");
    print_list(head);
    break;
}
break;
}
}

```

EXPECTED OUTPUT:

Please select any one of the four choices :

Select

- 1 For Traversing the list
- 2 For printing the Linked list
- 3 For performing Insertion
- 4 For performing deletion : 1

NULL<-> 1<->2<->3<->4<->5<->6<->NULL

Traversion completed

OBSERVED OUTPUT:

```
Please select any one of four choices :
```

```
Select
```

- ```
 1 for Traversing the list
 2 for Printing the linked list
 3 for performing Insertion
 4 for performing deletion : 2
```

```
NULL<->6<->7<->8<->8<->9<->10<->NULL
```

```
Process returned 4 (0x4) execution time : 4.194 s
```

```
Press any key to continue.
```

**AIM:** Write a c program to exhibit operations on the stack using arrays

**DESCRIPTION:**

The program is developed in C language.

Uses the concept of arrays and global declaration.

Switch case statements are implemented with specific case labels .

User is provided multiple choices to perform operations on the stack data structure which follows LIFO – LAST IN FIRST OUT .

operations : pop, push, isfull, isempty and printing the elements are implemented using respective methods to perform the task.

**PROGRAM:**

```
#include<stdio.h>

define SIZE 10

void push();void pop();void print();

int top = -1;int stack[SIZE];

void main()
{
 int ch;
 do
 {
 printf("\n\n Select :\n1. To Push\n2. To Pop\n3.To Print\n4.To Exit\n::");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:
 push();
 break;
 case 2:
 pop();
 break;
 case 3:
 print();
```



```

 break;
 case 4:
 printf("\n\tExited .");
 break;
 default:
 printf("\nNot a valid choice");

 }
}while(ch!=4);
}
void push()
{
 int data;
 if(top==SIZE-1)
 {
 printf("\nStack is Full !! No push possible .");
 }
 else
 {
 printf("\nPlease input the data to be pushed : ");
 scanf("%d",&data);
 top++;
 stack[top] = data;
 }
}
void pop()
{
 if(top==--1)
 {
 printf("\nStack is Empty !! No further pop possible .");
 }
}

```

```
 else
 {
 printf("\nThe element popped is %d.",stack[top]);
 top--;
 }
}
void print()
{
 if(top== -1)
 {
 printf("\nThe stack is in Underflow State .");
 }
 else
 {
 printf("\nThe stack is : ");
 for(int i = top;i>=0;i--)
 {
 printf("\n\t%d",stack[i]);
 }
 }
}
```

**EXPECTED OUTPUT:**

Select :

1.To push

2.To pop

3.To print

4.To Exit

::1

Please input the data to be pushed : 10

Select :

1.To push

2.To pop

3.To print

4.To Exit

::2

The element popped is :10

Select :

1.To push

2.To pop

3.To print

4.To Exit

::4

Existed

## OBSERVED OUTPUT:

```
Select :
1. To Push
2. To Pop
3.To Print
4.To Exit
::1

Please input the data to be pushed : 5

Select :
1. To Push
2. To Pop
3.To Print
4.To Exit
::1

Please input the data to be pushed : 8

Select :
1. To Push
2. To Pop
3.To Print
4.To Exit
::4

Exited .
Process returned 4 (0x4) execution time : 34.852 s
Press any key to continue.
```

**AIM:** Write a c program to exhibit operations on the queues using arrays

**DESCRIPTION:**

The program is developed in C language.

Uses the concept of arrays and global declaration.

Switch case statements are implemented with specific case labels .

User is provided multiple choices to perform operations on the stack data structure which follows FIFO – FIRST IN FIRST OUT .

Operations : peek, enqueue, dequeue, isfull, isempty and printing the elements are implemented using respective methods to perform the task.

**PROGRAM:**

```
#include<stdio.h>

#define SIZE 10

void EnQueue();void DeQueue();void Print();void isFull();void isEmpty();void peek();

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

void main()
{
 int ch;
 do
 {
 printf("\n\n Select :\n1. ENQUEUE\n2. DEQUEUE\n3. PRINT\n4. IsFull\n5. IsEmpty\n6. peek\n7. EXIT\n::");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:
 EnQueue();
 break;
 case 2:
 DeQueue();
 break;
 case 3:
```

```

 Print();

 break;

 case 4:

 isFull();

 break;

 case 5:

 isEmpty();

 break;

 case 6:

 peek();

 break;

 case 7:

 printf("\n\tExited .");

 break;

 default:

 printf("\nNot a valid choice");

 }

}while(ch!=7);

}

void EnQueue()

{

 int data;

 if(rear==SIZE-1)

 {

 printf("\nQueue is Full !! No Enqueueing possible .");

 }

 else

 {

 if(front==-1)

 front++;

```

```

 rear++;

 printf("\nPlease input the data to be added : ");

 scanf("%d",&data);

 queue[rear] = data;

 printf("\n\tEnqueued %d to the queue",data);

 }
}

void DeQueue()
{
 if(front== -1)
 {
 printf("\nQueue is Empty !! No further dequeuing possible .");
 }
 else
 {
 printf("\nThe element removed is %d.",queue[front]);

 front++;

 if(front > rear)
 {
 front = rear = -1;
 }
 }
}

void Print()
{
 if(rear== -1)
 {
 printf("\nThe Queue is Underflow !! Nothing to print.");
 }
 else

```

```

{
 printf("\nThe Queue is :");
 for(int i=front;i<=rear;i++)
 {
 printf(" %d ",queue[i]);
 }
 printf("\n");
}
}

void isFull()
{
 if(rear==SIZE-1)
 {
 printf("\n The Queue is FULL !");
 }
 else
 {
 if(front== -1)
 {
 printf("\nThe Queue is Not Full !");
 printf("\nContains %d elements.",0);
 return;
 }
 printf("\nThe Queue is Not Full !");
 printf("\nContains %d elements.",rear-front+1);
 }
}

void isEmpty()
{
 if(rear== -1)
 {

```



```
 printf("\nThe Queue is Empty !!");
 }
 else
 {
 printf("\nThe Queue is Not Empty !");
 printf("\nContains %d elements.",rear-front+1);
 }
}

void peek()
{
 if(front== -1)
 {
 printf("\nThe Queue is Empty .");
 }
 else
 {
 printf("\nThe element at the first of the Queue is :");
 printf(" %d",queue[front]);
 }
}
```

### EXPECTED OUTPUT:

Select :

1.ENQUEUE

2.DEQUEUE

3.PRINT

4.ISFULL

5.ISEMPTY

6.PEEK

7.EXIT ::1

Please input the data to be added:17

Enqueued 17 to the queue

Select :

1.ENQUEUE

2.DEQUEUE

3.PRINT

4.ISFULL

5.ISEMPTY

6.PEEK

7.EXIT ::1

Please input the data to be added:19

Enqueued 19 to the queue

Select :

1.ENQUEUE

2.DEQUEUE

3.PRINT

4.ISFULL

5.ISEMPTY

6.PEEK

7.EXIT ::7

Exited.

OBSERVED OUTPUT:

```
Select :
1. ENQUEUE
2. DEQUEUE
3. PRINT
4. IsFull
5. IsEmpty
6. peek
7. EXIT
::1

Please input the data to be added : 13

 Enqueued 13 to the queue

Select :
1. ENQUEUE
2. DEQUEUE
3. PRINT
4. IsFull
5. IsEmpty
6. peek
7. EXIT
::1

Please input the data to be added : 11

 Enqueued 11 to the queue

Select :
1. ENQUEUE
2. DEQUEUE
3. PRINT
4. IsFull
5. IsEmpty
6. peek
7. EXIT
::3

The Queue is : 13 11

Select :
1. ENQUEUE
2. DEQUEUE
3. PRINT
4. IsFull
5. IsEmpty
6. peek
7. EXIT
::7

 Exited .█
```

**PROGRAM NO:20.****DATE:**

**AIM:** Write a c program to accept the list of elements of a tree 'T' and perform tree traversal operations.

**DESCRIPTION:**

The program is developed in the C language.

Uses structures to declare the structure of the node in the tree.

A custom method is implemented for inserting the nodes in the tree.

Pre order, post order , in order traversals are implemented using specific methods that follow the respective principle of showing the data in the tree.

Uses structure pointers, dynamic memory allocation concepts.

Switch case statements are used to perform any one If the three traversals Possible.

**PROGRAM:**

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

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

struct node *root = NULL;

void insert(int data)
{
 struct node *tempN = (struct node*)malloc(sizeof(struct node));
 struct node *current,*parent;
 tempN->data = data;
 tempN->left = NULL;
 tempN->right = NULL;
 if(root == NULL)
 {
 root = tempN;
 }
}
```

```

else
{
 current = root;
 parent = NULL;

 while(1)
 {
 parent = current;
 if(data < parent->data)
 {
 current = current->left;

 if(current == NULL)
 {
 parent->left = tempN;
 return;
 }
 }
 else{

 current = current->right;

 if(current==NULL)
 {
 parent->right = tempN;
 return;
 }
 }
 }
}

```

```

}

void pre_order(struct node *root)
{
 if(root!=NULL)
 {
 printf("%d ",root->data);
 pre_order(root->left);
 pre_order(root->right);
 }
}

void in_order(struct node *root)
{
 if(root!=NULL)
 {
 in_order(root->left);
 printf("%d ",root->data);
 in_order(root->right);
 }
}

void post_order(struct node *root)
{
 if(root!=NULL)
 {
 post_order(root->left);
 post_order(root->right);
 printf("%d ",root->data);
 }
}

void main()
{
 int data,n;

```

```

printf("No.of elements : ");
scanf("%d",&n);
printf("\nElements for the tree:\n");
for(int i=0;i<n;i++)
{
 scanf("%d",&data);
 insert(data);
}
printf("\nselect 1 - Pre-Order\t2 - In-Order\t3 - Post-Order : ");
scanf("%d",&n);
switch(n)
{
 case 1:
 printf("\nPre-Order Traversal :\n");
 pre_order(root);
 break;
 case 2:
 printf("\nIn-Order Traversal :\n");
 in_order(root);
 break;
 case 3:
 printf("\nPost-Order Traversal :\n");
 post_order(root);
 break;
 default:
 printf("\nInvalid input !!");
 break;
}
}

```

**EXPECTED OUTPUT:**

No of elements: 10

Elements for the tree:

9 8 4 5 6 3 7 2 1 3

Select 1 – pre order 2-In order 3 – post order:1

Pre – Order Traversal:

9 8 4 3 2 1 3 5 5 6 7

Process returned 0(0x0) execution time: 31.949s

Press any key to continue

**OBSERVED OUTPUT:**

```
No.of elements : 6

Elements for the tree:
5 6 7 2 19 9

select 1 - Pre-Order 2 - In-Order 3 - Post-Order : 3

Post-Order Traversal :
2 9 19 7 6 5
Process returned 2 (0x2) execution time : 20.380 s
Press any key to continue.
```



**PROGRAM NO:21.****DATE:**

**AIM:** Write a c program to convert the given infix expression to its equivalent postfix expression.

**DESCRIPTION:**

The program is developed in C language.

Uses the concept of stacks , strings , arrays and pointers.

Pop , push , isempty , isfull of stack are used to do specific operations by iterating over the infix expression given as string by the user.

Switch case statements are used to return the precedence level of the operators encountered in the infix expression.

Postfix expression is obtained from the infix and returned to the screen of the user.

**PROGRAM:**

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;
void push(char x)
{
 stack[++top] = x;
}
char pop()
{
 if(top == -1)
 return -1;
 else
 return stack[top--];
}
int priority(char x)
{
 if(x == '(')
 return 0;
 if(x == '+' || x == '-')
 return 1;
```

```

 if(x == '*' || x == '/')
 return 2;
 return 0;
}
int main()
{
 char exp[100];
 char *e, x;
 printf("Enter the expression : ");
 scanf("%s", exp);
 printf("\n");
 e = exp;
 while(*e != '\0')
 {
 if(isalnum(*e))
 printf("%c ", *e);
 else if(*e == '(')
 push(*e);
 else if(*e == ')')
 {
 while((x = pop()) != '(')
 printf("%c ", x);
 }
 else
 {
 while(priority(stack[top]) >= priority(*e))
 printf("%c ", pop());
 push(*e);
 }
 e++;
 }
}

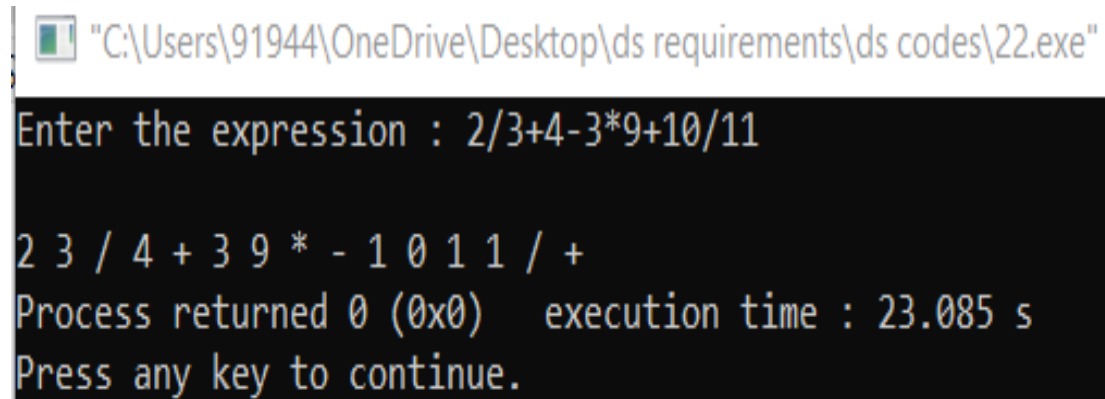
```

```
while(top != -1)
{
 printf("%c ",pop());
}return 0;
}
```

**EXPECTED OUTPUT:**

Enter the expression :a-b\*c+d/e

a b c \*- d e/+

**OBSERVED OUTPUT:**

```
"C:\Users\91944\OneDrive\Desktop\ds requirements\ds codes\22.exe"
Enter the expression : 2/3+4-3*9+10/11
2 3 / 4 + 3 9 * - 1 0 1 1 / +
Process returned 0 (0x0) execution time : 23.085 s
Press any key to continue.
```

**PROGRAM NO: 22.****DATE:**

**AIM:** Write a c program to convert the given infix expression to its equivalent prefix expression.

**DESCRIPTION:**

The program is developed in C language.

Uses the concept of stacks , strings , arrays and pointers.

Dynamic memory allocation is implemented using malloc method type casted into type of stack.

Pop , push , isempty , isfull of stack are used to do specific operations by iterating over the infix expression given as string by the user.

Switch case statements are used to return the precedence level of the operators encountered in the infix expression.

Prefix expression is obtained from the infix and returned to the screen of the user.

**PROGRAM:**

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

struct Stack {
 int top;
 int maxSize;
 int* array;
};

struct Stack* create(int max)
{
 struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
 stack->maxSize = max;
 stack->top = -1;
 stack->array = (int*)malloc(stack->maxSize * sizeof(int));
 return stack;
}

int isFull(struct Stack* stack)
{
 if(stack->top == stack->maxSize - 1){
```

```

 printf("Will not be able to push maxSize reached\n");
 }
 return stack->top == stack->maxSize - 1;
}
int isEmpty(struct Stack* stack)
{
 return stack->top == -1;
}
void push(struct Stack* stack, int item)
{
 if (isFull(stack))
 return;
 stack->array[++stack->top] = item;
}
int pop(struct Stack* stack)
{
 if (isEmpty(stack))
 return INT_MIN;
 return stack->array[stack->top--];
}
int peek(struct Stack* stack)
{
 if (isEmpty(stack))
 return INT_MIN;
 return stack->array[stack->top];
}
int checkIfOperand(char ch)
{
 return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}
int precedence(char ch)

```

```

{
 switch (ch)
 {
 case '+':
 case '-':
 return 1;

 case '*':
 case '/':
 return 2;

 case '^':
 return 3;
 }
 return -1;
}

int getPostfix(char* expression)
{
 int i, j;
 struct Stack* stack = create(strlen(expression));
 if(!stack)
 return -1 ;
 for (i = 0, j = -1; expression[i]; ++i)
 {
 if (checkIfOperand(expression[i]))
 expression[++j] = expression[i];
 else if (expression[i] == '(')
 push(stack, expression[i]);
 else if (expression[i] == ')')
 {
 while (!isEmpty(stack) && peek(stack) != '(')

```

```

 expression[++j] = pop(stack);
 if (!isEmpty(stack) && peek(stack) != '(')
 return -1;
 else
 pop(stack);
}
else
{
 while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peek(stack)))
 expression[++j] = pop(stack);
 push(stack, expression[i]);
}
}
while (!isEmpty(stack))
 expression[++j] = pop(stack);
expression[++j] = '\0';
}

void reverse(char *exp)
{
 int size = strlen(exp);
 int j = size, i=0;
 char temp[size];

 temp[j--]='\0';
 while(exp[i]!='\0')
 {
 temp[j] = exp[i];
 j--;
 i++;
 }
 strcpy(exp,temp);
}

```



```

}

void brackets(char* exp){
 int i = 0;
 while(exp[i]!='\0')
 {
 if(exp[i]=='(')
 exp[i]=')';
 else if(exp[i]==')')
 exp[i]='(';
 i++;
 }
}

void InfixtoPrefix(char *exp){
 int size = strlen(exp);
 reverse(exp);
 brackets(exp);
 getPostfix(exp);
 reverse(exp);
}

int main()
{
 char expression[100];
 printf("Input the Expression : ");
 gets(expression);
 printf("The infix is: ");
 printf("%s\n",expression);
 InfixtoPrefix(expression);
 printf("The prefix is: ");
 printf("%s\n",expression);
 return 0;
}

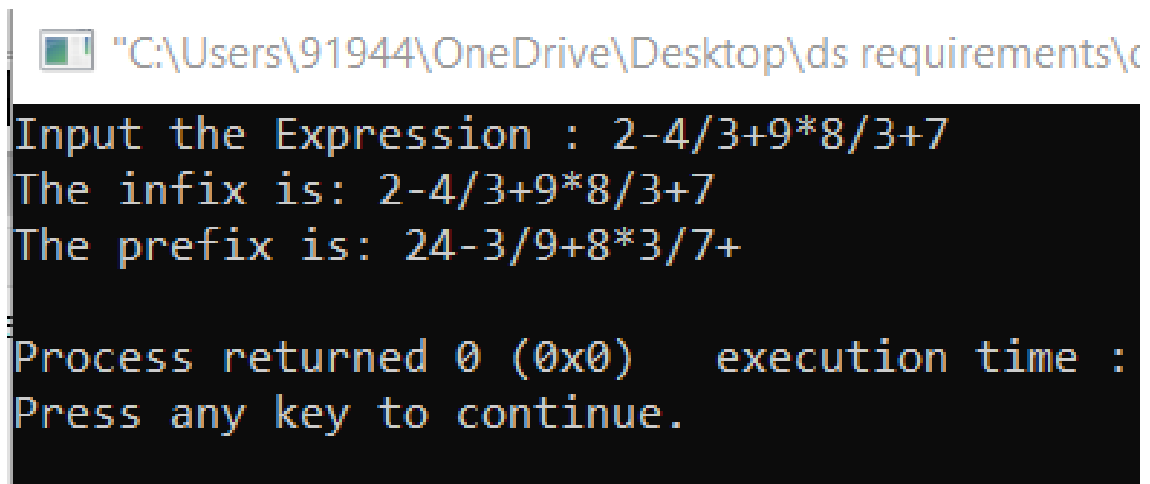
```

**EXPECTED OUTPUT:**

Input the expression :  $((a/b)+c)-(d+(e*f))$

The infix is :  $((a/b)+c)-(d+(e*f))$

The prefix is :  $-+ / abc + d * ef$

**OBSERVED OUTPUT:**

```
"C:\Users\91944\OneDrive\Desktop\ds requirements\c
Input the Expression : 2-4/3+9*8/3+7
The infix is: 2-4/3+9*8/3+7
The prefix is: 24-3/9+8*3/7+
Process returned 0 (0x0) execution time :
Press any key to continue.
```