PROGRAM-8

Objective: Write a Program to check balanced parenthesis using Stack.

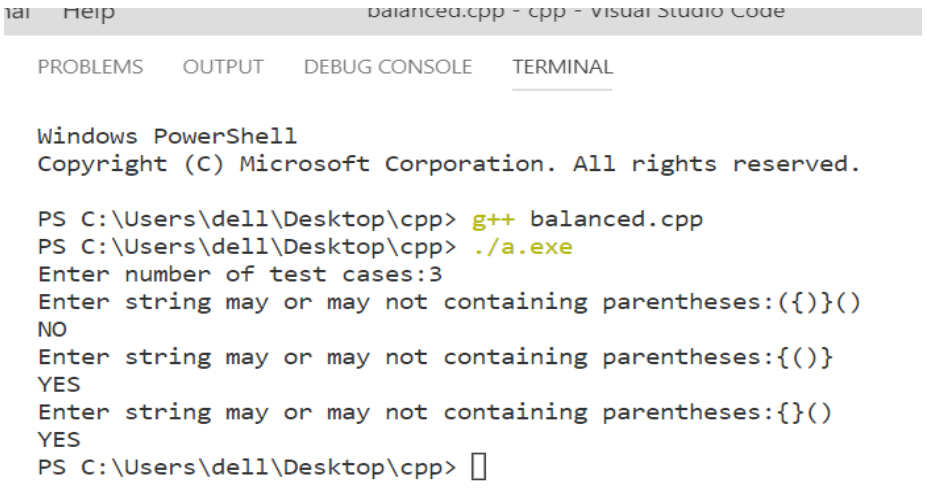
Code:

```
#include <iostream> //main header file
#include <stack>
using namespace std;

void balance_parentheses();

int main()
{
    int t;
    cout << "Enter number of test cases:";
    cin >> t;

    for (int i = 0; i < t; i++)
    {
        //calling of function for checking of brackets
        balance_parentheses();
    }

    return 0;
}
void balance_parentheses()
{
    stack<char> a;
    string s;
    cout << "Enter string may or may not containing parentheses:";
    cin >> s;
```

```
int flag = 0; //flag is an arbitrary variable

for (int i = 0; i < s.length(); i++)
//for length of the string calculated by number of letters
{
    if (s[i] == '{' || s[i] == '[' || s[i] == '(')
    {
        //push function to enter terms in a stack
        a.push(s[i]);
        flag = 1;
    }
    if (!a.empty())
    {
        if (s[i] == '}')
        {
            if (a.top() == '{')
            // top of the stack
            {
                a.pop();
                //pop function to delete terms from the top of array
                continue;
            }
            else
                break;
        }
        if (s[i] == ']')
        {
            if (a.top() == '[')
            {
                a.pop();
                continue;
            }
            else
```

```
                break;
            }
            if (s[i] == ')')
            {
                if (a.top() == '(')
                {
                    a.pop();
                    continue;
                }
                else
                    break;
            }
        }
        else
        {
            break;
        }
    }
    if ((a.empty()) && (flag == 1))
        cout << "YES" << endl;
    else
        cout << "NO" << endl;
}
```

Output:



PROGRAM-10

Objective: Write a Program to implement priority queue.

Code:

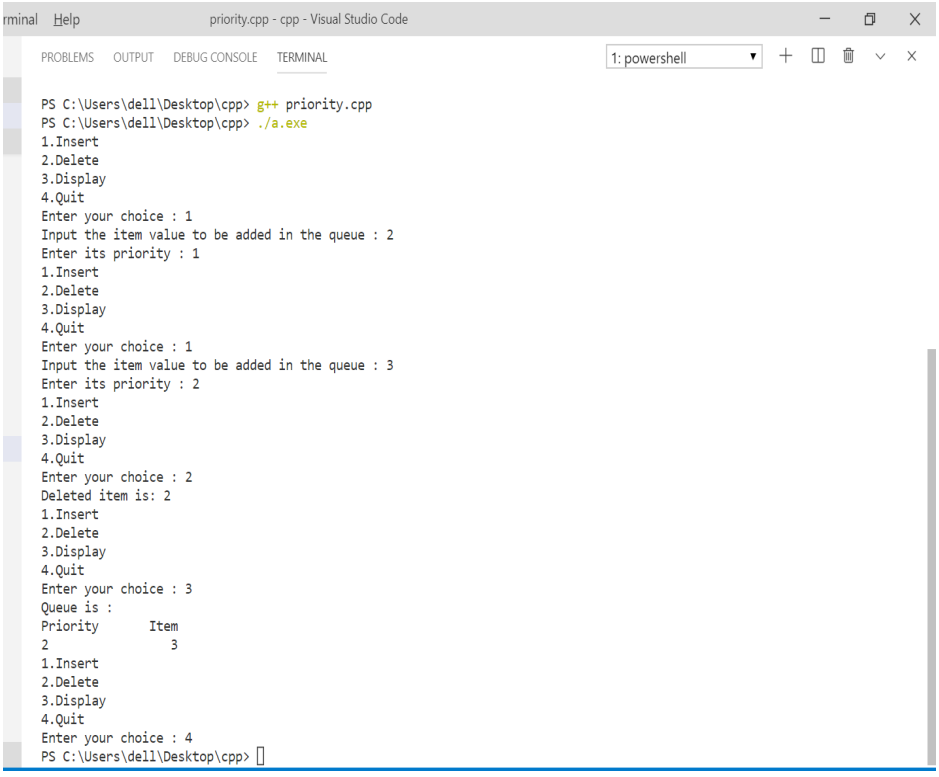
```
#include <iostream>
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
using namespace std;
struct node
{
   int priority;
   int info;
   struct node *link;
};
class Priority_Queue
{
private:
   node *front;

public:
   Priority_Queue()
   {
      front = NULL;
   }
   /*
        * Insert into Priority Queue
        */
   void insert(int item, int priority)
   {
```

```
    node *tmp, *q;
    tmp = new node;
    tmp->info = item;
    tmp->priority = priority;
    if (front == NULL || priority < front->priority)
    {
        tmp->link = front;
        front = tmp;
    }
    else
    {
        q = front;
        while (q->link != NULL && q->link->priority <= priority)
            q = q->link;
        tmp->link = q->link;
        q->link = tmp;
    }
}
/*
     * Delete from Priority Queue
     */
void del()
{
    node *tmp;
    if (front == NULL)
        cout << "Queue Underflow\n";
    else
    {
        tmp = front;
        cout << "Deleted item is: " << tmp->info << endl;
        front = front->link;
        free(tmp);
    }
```

```
    }
    void display()
    {
        node *ptr;
        ptr = front;
        if (front == NULL)
            cout << "Queue is empty\n";
        else
        {
            cout << "Queue is :\n";
            cout << "Priority       Item\n";
            while (ptr != NULL)
            {
                cout << ptr->priority << "                " << ptr->info << endl;
                ptr = ptr->link;
            }
        }
    }
};
int main()
{
    int choice, item, priority;
    Priority_Queue pq;
    do
    {
        cout << "1.Insert\n";
        cout << "2.Delete\n";
        cout << "3.Display\n";
        cout << "4.Quit\n";
        cout << "Enter your choice : ";
        cin >> choice;
        switch (choice)
        {
```

```
        case 1:
            cout << "Input the item value to be added in the queue : ";
            cin >> item;
            cout << "Enter its priority : ";
            cin >> priority;
            pq.insert(item, priority);
            break;
        case 2:
            pq.del();
            break;
        case 3:
            pq.display();
            break;
        case 4:
            break;
        default:
            cout << "Wrong choice\n";
        }
    } while (choice != 4);
    return 0;
}
```

Output:



PROGRAM-9

Objective: Write a Program to implement Circular queue.

Code:

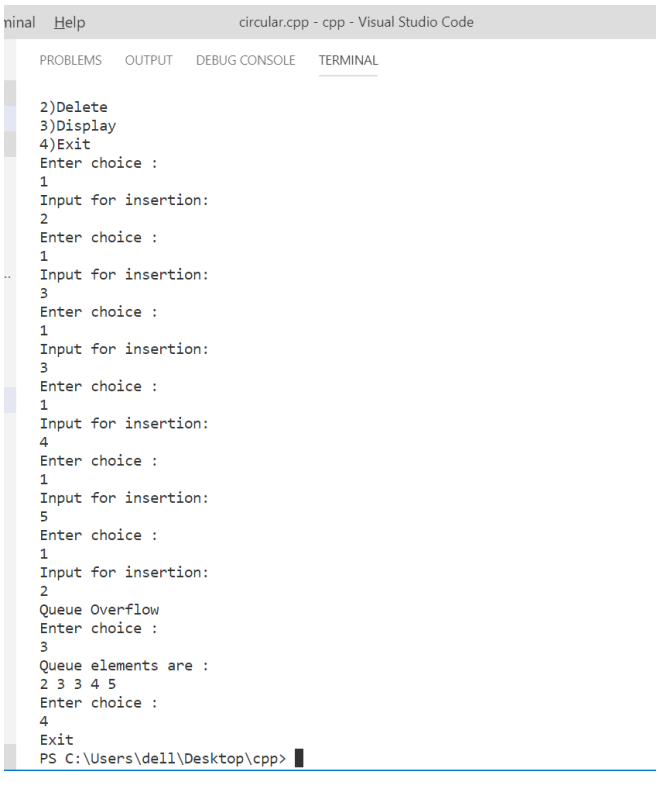
```
#include <iostream>
using namespace std;

int cqueue[5];
int front = -1, rear = -1, n = 5;

void insertCQ(int val)
{
   if ((front == 0 && rear == n - 1) || (front == rear + 1))
   {
      cout << "Queue Overflow \n";
      return;
   }
   if (front == -1)
   {
      front = 0;
      rear = 0;
   }
   else
   {
      if (rear == n - 1)
         rear = 0;
      else
         rear = rear + 1;
   }
   cqueue[rear] = val;
```

```
}
void deleteCQ()
{
    if (front == -1)
    {
        cout << "Queue Underflow\n";
        return;
    }
    cout << "Element deleted from queue is : " << cqueue[front] << endl;

    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else
    {
        if (front == n - 1)
            front = 0;
        else
            front = front + 1;
    }
}
void displayCQ()
{
    int f = front, r = rear;
    if (front == -1)
    {
        cout << "Queue is empty" << endl;
        return;
    }
    cout << "Queue elements are :\n";
    if (f <= r)
```

```
    {
        while (f <= r)
        {
            cout << cqueue[f] << " ";
            f++;
        }
    }
    else
    {
        while (f <= n - 1)
        {
            cout << cqueue[f] << " ";
            f++;
        }
        f = 0;
        while (f <= r)
        {
            cout << cqueue[f] << " ";
            f++;
        }
    }
    cout << endl;
}
int main()
{

    int ch, val;
    cout << "1)Insert\n";
    cout << "2)Delete\n";
    cout << "3)Display\n";
    cout << "4)Exit\n";
    do
    {
```

```
        cout << "Enter choice : " << endl;
        cin >> ch;
        switch (ch)
        {
        case 1:
            cout << "Input for insertion: " << endl;
            cin >> val;
            insertCQ(val);
            break;

        case 2:
            deleteCQ();
            break;

        case 3:
            displayCQ();
            break;

        case 4:
            cout << "Exit\n";
            break;
        default:
            cout << "Incorrect!\n";
        }
    } while (ch != 4);
    return 0;
}
```

Output:



PROGRAM-10(a)

Objective: : Write a Program to implement Singly linked list (searching, insertion, deletion).

Code:

```
#include<stdio.h>
#include<stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;
struct Node
{
    int e;
    Position next;
};
void Insert(int x, List l, Position p)
{
    Position TmpCell;
    TmpCell = (struct Node*) malloc(sizeof(struct Node));
    if(TmpCell == NULL)
        printf("Memory out of space\n");
    else
    {
        TmpCell->e = x;
        TmpCell->next = p->next;
        p->next = TmpCell;
    }
}
int isLast(Position p)
{
```

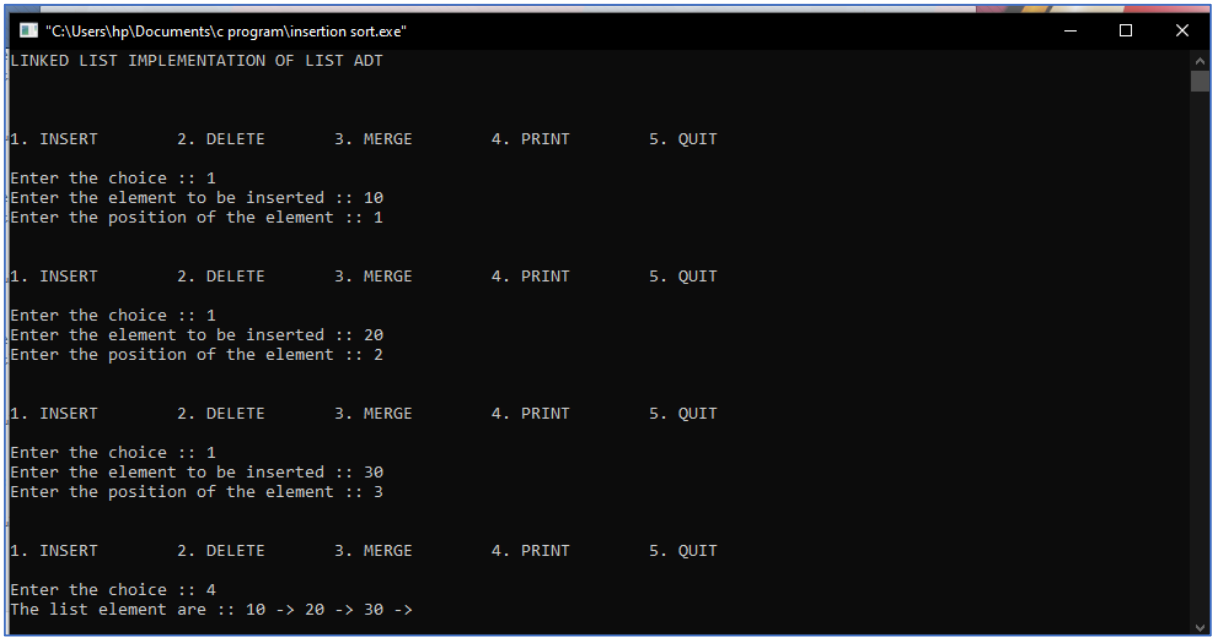
```
    return (p->next == NULL);
}
Position FindPrevious(int x, List l)
{
    Position p = l;
    while(p->next != NULL && p->next->e != x)
        p = p->next;
    return p;
}
void Delete(int x, List l)
{
    Position p, TmpCell;
    p = FindPrevious(x, l);
    if(!isLast(p))
    {
        TmpCell = p->next;
        p->next = TmpCell->next;
        free(TmpCell);
    }
    else
        printf("Element does not exist!!!\n");
}
void Display(List l)
{
    printf("The list element are :: ");
    Position p = l->next;
    while(p != NULL)
    {
        printf("%d -> ", p->e);
        p = p->next;
    }
}
void Merge(List l, List l1)
```

```
{
    int i, n, x, j;
    Position p;
    printf("Enter the number of elements to be merged :: ");
    scanf("%d",&n);
    for(i = 1; i <= n; i++)
    {
        p = l1;
        scanf("%d", &x);
        for(j = 1; j < i; j++)
            p = p->next;
        Insert(x, l1, p);
    }
    printf("The new List :: ");
    Display(l1);
    printf("The merged List ::");
    p = l;
    while(p->next != NULL)
    {
        p = p->next;
    }
    p->next = l1->next;
    Display(l);
}
int main()
{
    int x, pos, ch, i;
    List l, l1;
    l = (struct Node *) malloc(sizeof(struct Node));
    l->next = NULL;
    List p = l;
    printf("LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");
    do
```

```
   {
      printf("\n\n1. INSERT\t 2. DELETE\t 3. MERGE\t 4. PRINT\t 5. QUIT\n\nEnter
the choice :: ");
      scanf("%d", &ch);
      switch(ch)
      {
      case 1:
         p = l;
         printf("Enter the element to be inserted :: ");
         scanf("%d",&x);
         printf("Enter the position of the element :: ");
         scanf("%d",&pos);
         for(i = 1; i < pos; i++)
         {
            p = p->next;
         }
         Insert(x,l,p);
         break;
      case 2:
         p = l;
         printf("Enter the element to be deleted :: ");
         scanf("%d",&x);
         Delete(x,p);
         break;
      case 3:
         l1 = (struct Node *) malloc(sizeof(struct Node));
         l1->next = NULL;
         Merge(l, l1);
         break;
      case 4:
         Display(l);
         break;
      }
```

```
    }
    while(ch<5);
    return 0;
}
```

Output:



PROGRAM-10(b)

Objective: : Write a Program to implement doubly linked list (searching, insertion, deletion).

Code:

```
#include<stdio.h>
#include<stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;
struct Node
{
    int e;
    Position previous;
    Position next;
};
void Insert(int x, List l, Position p)
{
    Position TmpCell;
    TmpCell = (struct Node*) malloc(sizeof(struct Node));
    if(TmpCell == NULL)
        printf("Memory out of space\n");
    else
    {
        TmpCell->e = x;
        TmpCell->previous = p;
        TmpCell->next = p->next;
        p->next = TmpCell;
    }
}
```

```
int isLast(Position p)
{
   return (p->next == NULL);
}
Position Find(int x, List l)
{
   Position p = l->next;
   while(p != NULL && p->e != x)
      p = p->next;
   return p;
}
void Delete(int x, List l)
{
   Position p, p1, p2;
   p = Find(x, l);
   if(p != NULL)
   {
      p1 = p -> previous;
      p2 = p -> next;
      p1 -> next = p -> next;
      if(p2 != NULL)                // if the node is not the last node
         p2 -> previous = p -> previous;
   }
   else
      printf("Element does not exist!!!\n");
}
void Display(List l)
{
   printf("The list element are :: ");
   Position p = l->next;
   while(p != NULL)
   {
      printf("%d -> ", p->e);
```

```
        p = p->next;
    }
}
void main()
{
    int x, pos, ch, i;
    List l, l1;
    l = (struct Node *) malloc(sizeof(struct Node));
    l->previous = NULL;
    l->next = NULL;
    List p = l;
    printf("DOUBLY LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");
    do
    {
        printf("\n\n1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5. QUIT\n\nEnter the
choice :: ");
        scanf("%d", &ch);
        switch(ch)
        {
        case 1:
            p = l;
            printf("Enter the element to be inserted :: ");
            scanf("%d",&x);
            printf("Enter the position of the element :: ");
            scanf("%d",&pos);
            for(i = 1; i < pos; i++)
            {
                p = p->next;
            }
            Insert(x,l,p);
            break;
        case 2:
            p = l;
```

```
            printf("Enter the element to be deleted :: ");
            scanf("%d",&x);
            Delete(x,p);
            break;
        case 3:
            p = l;
            printf("Enter the element to be searched :: ");
            scanf("%d",&x);
            p = Find(x,p);
            if(p == NULL)
                printf("Element does not exist!!!\n");
            else
                printf("Element exist!!!\n");
            break;
        case 4:
            Display(l);
            break;
        }
    }
    while(ch<5);
}
```

Output:

```
DOUBLY LINKED LIST IMPLEMENTATION OF LIST ADT


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 1
Enter the element to be inserted :: 10
Enter the position of the element :: 1


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 1
Enter the element to be inserted :: 20
Enter the position of the element :: 2


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 1
Enter the element to be inserted :: 30
Enter the position of the element :: 3


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 4
The list element are :: 10 -> 20 -> 30 ->
```

PROGRAM-10(c)

Objective: Write a Program to implement circular linked list (searching, insertion, deletion).

Code:

```
#include<stdio.h>
#include<stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;

struct Node
{
    int e;
    Position next;
};

void Insert(int x, List l, Position p)
{
    Position TmpCell;
    TmpCell = (struct Node*) malloc(sizeof(struct Node));
    if(TmpCell == NULL)
        printf("Memory out of space\n");
    else
    {
        TmpCell->e = x;
        TmpCell->next = p->next;
        p->next = TmpCell;
    }
}
```

```
int isLast(Position p, List l)
{
    return (p->next == l);
}

Position FindPrevious(int x, List l)
{
    Position p = l;
    while(p->next != l && p->next->e != x)
        p = p->next;
    return p;
}

Position Find(int x, List l)
{
    Position p = l->next;
    while(p != l && p->e != x)
        p = p->next;
    return p;
}

void Delete(int x, List l)
{
    Position p, TmpCell;
    p = FindPrevious(x, l);
    if(!isLast(p, l))
    {
        TmpCell = p->next;
        p->next = TmpCell->next;
        free(TmpCell);
    }
```

```
    else
        printf("Element does not exist!!!\n");
}

void Display(List l)
{
    printf("The list element are :: ");
    Position p = l->next;
    while(p != l)
    {
        printf("%d -> ", p->e);
        p = p->next;
    }
}

void main()
{
    int x, pos, ch, i;
    List l, l1;
    l = (struct Node *) malloc(sizeof(struct Node));
    l->next = l;
    List p = l;
    printf("CIRCULAR LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");
    do
    {
        printf("\n\n1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5. QUIT\n\nEnter the
choice :: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                p = l;
                printf("Enter the element to be inserted :: ");
```

```
            scanf("%d",&x);
            printf("Enter the position of the element :: ");
            scanf("%d",&pos);
            for(i = 1; i < pos; i++)
            {
               p = p->next;
            }
            Insert(x,l,p);
            break;

         case 2:
            p = l;
            printf("Enter the element to be deleted :: ");
            scanf("%d",&x);
            Delete(x,p);
            break;

         case 3:
            p = l;
            printf("Enter the element to be searched :: ");
            scanf("%d",&x);
            p = Find(x,p);
            if(p == l)
               printf("Element does not exist!!!\n");
            else
               printf("Element exist!!!\n");
            break;

         case 4:
            Display(l);
            break;
      }
   }while(ch<5);
```

```
    return 0;
}
```

Output:

```
CIRCULAR LINKED LIST IMPLEMENTATION OF LIST ADT


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 1
Enter the element to be inserted :: 10
Enter the position of the element :: 1


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 1
Enter the element to be inserted :: 20
Enter the position of the element :: 2


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 1
Enter the element to be inserted :: 30
Enter the position of the element :: 3


1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 4
The list element are :: 10 -> 20 -> 30 ->

1. INSERT        2. DELETE       3. FIND         4. PRINT        5. QUIT

Enter the choice :: 5
```

PROGRAM-11

Objective: To implement stack using linked list.

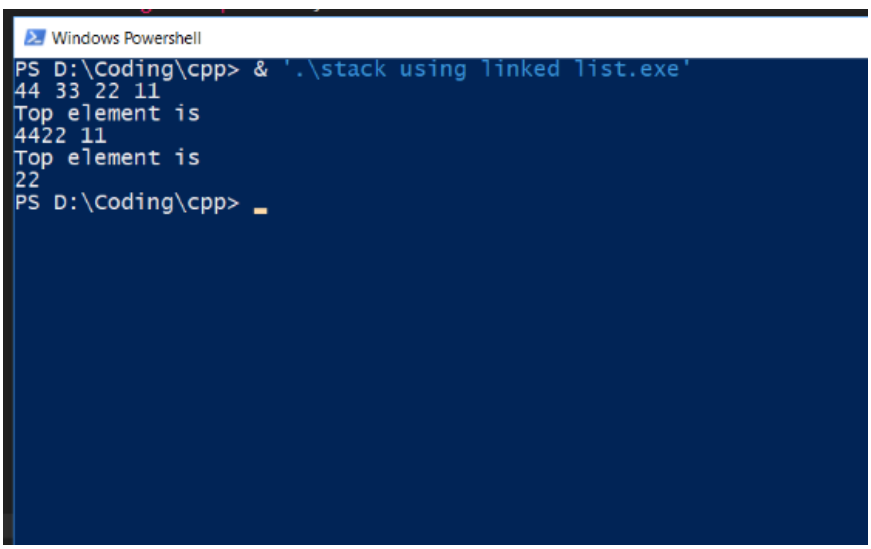
Code:

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int data;
    struct Node* link;
};
struct Node* top;

void push(int data)
{
    struct Node* temp;
    temp = new Node();

    if (!temp) {
        cout << "\nHeap Overflow";
        exit(1);
    }

    temp->data = data;

    temp->link = top;
    top = temp;
}
int isEmpty()
{
    return top == NULL;
```

```
}
int peek()
{

    if (!isEmpty())
        return top->data;
    else
        exit(1);
}
void pop()
{
    struct Node* temp;
    if (top == NULL) {
        cout << "\nStack Underflow" << endl;
        exit(1);
    }
    else {
        temp = top;
        top = top->link;
        temp->link = NULL;
        free(temp);
    }
}
void display()
{
    struct Node* temp;

    if (top == NULL) {
        cout << "\nStack Underflow";
        exit(1);
    }
    else {
        temp = top;
```

```
        while (temp != NULL) {
            cout <<  temp->data << " ";
            temp = temp->link;
        }
    }
}
int main()
{

    push(11);
    push(22);
    push(33);
    push(44);

    display();

    cout << "\nTop element is \n" <<  peek();

    pop();
    pop();

    display();

    cout << "\nTop element is \n" << peek();
    return 0;

}
```

Output:



PROGRAM-12

Objective: To implement queue using linked list.

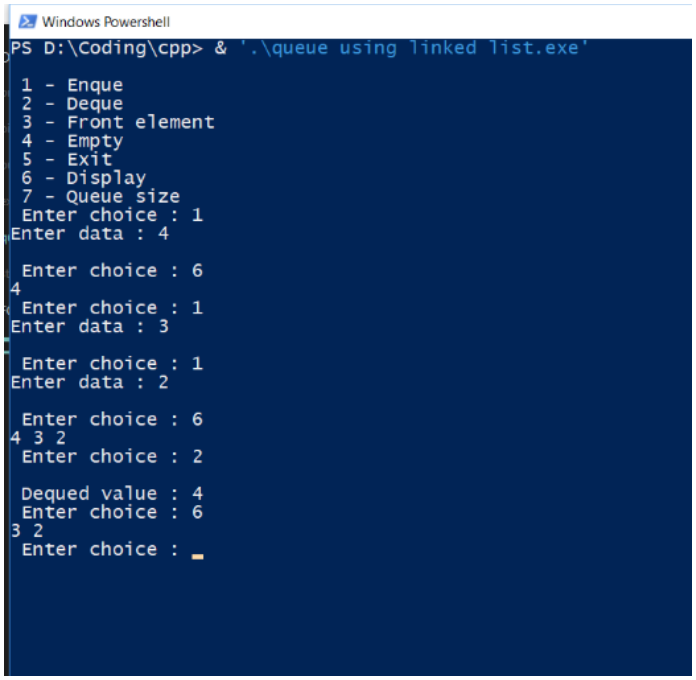
Code:

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    struct Node* link;
};
struct Node* top;

void push(int data)
{
    struct Node* temp;
    temp = new Node();


    if (!temp) {
        cout << "\nHeap Overflow";
        exit(1);
    }

    temp->data = data;
    temp->link = top;
    top = temp;
}

int isEmpty()
{
    return top == NULL;
```

```
}
int peek()
{

    if (!isEmpty())
        return top->data;
    else
        exit(1);
}
  void pop()
{
    struct Node* temp;

    if (top == NULL) {
        cout << "\nStack Underflow" << endl;
        exit(1);
    }
    else {
        temp = top;
        top = top->link;
        temp->link = NULL;
        free(temp);
    }
}
void display()
{
    struct Node* temp;

    if (top == NULL) {
        cout << "\nStack Underflow";
        exit(1);
    }
    else {
```

```
        temp = top;
        while (temp != NULL) {

            cout <<  temp->data << " ";
            temp = temp->link;
        }
    }
}
  int main()
{
    push(11);
    push(22);
    push(33);
    push(44);
    display();

    cout << "\nTop element is \n" <<  peek();

    pop();
    pop();
    display();

    cout << "\nTop element is \n" << peek();
    return 0;
}
```

Output:



PROGRAM-13

Objective: To implement queue using linked list.

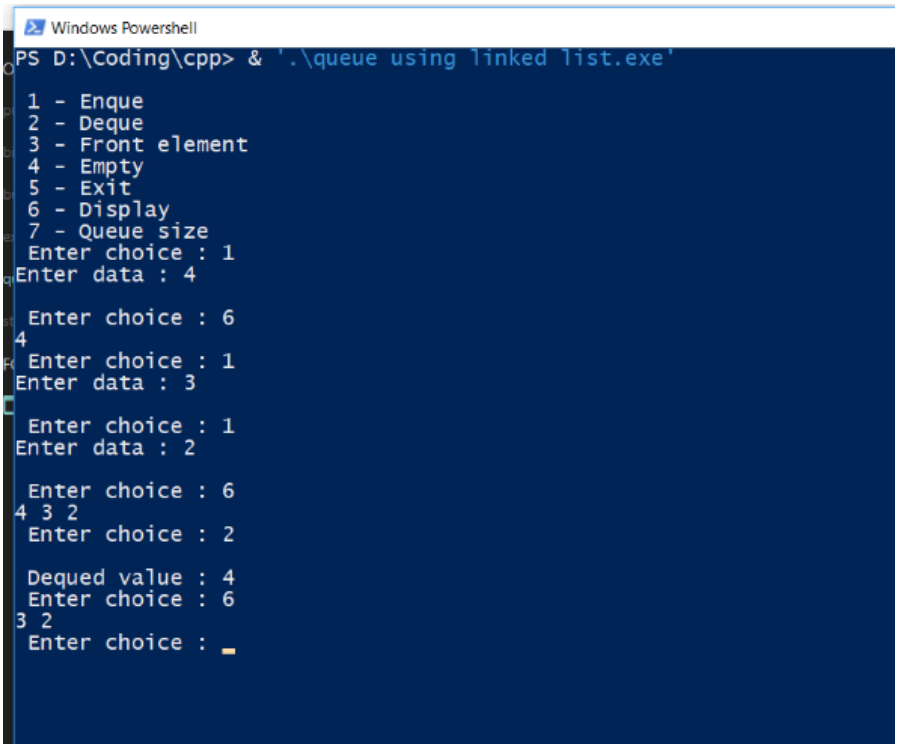
Code:

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    struct Node* link;
};
struct Node* top;

void push(int data)
{
    struct Node* temp;
    temp = new Node();


    if (!temp) {
        cout << "\nHeap Overflow";
        exit(1);
    }
    temp->data = data;
    temp->link = top;
    top = temp;
}

int isEmpty()
{
    return top == NULL;
}
```

```
int peek()
{

    if (!isEmpty())
        return top->data;
    else
        exit(1);
}
  void pop()
{
    struct Node* temp;

    if (top == NULL) {
        cout << "\nStack Underflow" << endl;
        exit(1);
    }
    else {
        temp = top;
        top = top->link;
        temp->link = NULL;
        free(temp);
    }
}
void display()
{
    struct Node* temp;

    if (top == NULL) {
        cout << "\nStack Underflow";
        exit(1);
    }
    else {
        temp = top;
```

```
        while (temp != NULL) {

            cout <<  temp->data << " ";
            temp = temp->link;
        }
    }
}
  int main()
{
    push(11);
    push(22);
    push(33);
    push(44);
    display();

    cout << "\nTop element is \n" <<  peek();

    pop();
    pop();
    display();

    cout << "\nTop element is \n" << peek();
    return 0;

}
```

Output:



PROGRAM-14(a)

Objective: Write a Program for insertion, deletion and traversal for binary tree.

Code:

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
struct treeNode
{
   int data;
   treeNode *left;
   treeNode *right;
};
treeNode* FindMin(treeNode *node)
{
   if(node==NULL)
   {
      /* There is no element in the tree */
      return NULL;
   }
   if(node->left) /* Go to the left sub tree to find the min element */
      return FindMin(node->left);
   else
      return node;
}
treeNode* FindMax(treeNode *node)
{
   if(node==NULL)
   {
      /* There is no element in the tree */
      return NULL;
```

```
    }
    if(node->right) /* Go to the left sub tree to find the min element */
        return(FindMax(node->right));
    else
        return node;
}
treeNode *Insert(treeNode *node,int data)
{
    if(node==NULL)
    {
        treeNode *temp;
        temp=new treeNode;
//temp = (treeNode *)malloc(sizeof(treeNode));
        temp -> data = data;
        temp -> left = temp -> right = NULL;
        return temp;
    }
    if(data >(node->data))
    {
        node->right = Insert(node->right,data);
    }
    else if(data < (node->data))
    {
        node->left = Insert(node->left,data);
    }
    /* Else there is nothing to do as the data is already in the tree. */
    return node;
}
treeNode * Delet(treeNode *node, int data)
{
    treeNode *temp;
    if(node==NULL)
    {
```

```
    cout<<"Element Not Found";
}
else if(data < node->data)
{
    node->left = Delet(node->left, data);
}
else if(data > node->data)
{
    node->right = Delet(node->right, data);
}
else
{
    /* Now We can delete this node and replace with either minimum element
    in the right sub tree or maximum element in the left subtree */
    if(node->right && node->left)
    {
        /* Here we will replace with minimum element in the right sub tree */
        temp = FindMin(node->right);
        node -> data = temp->data;
        /* As we replaced it with some other node, we have to delete that node */
        node -> right = Delet(node->right,temp->data);
    }
    else
    {
        /* If there is only one or zero children then we can directly
        remove it from the tree and connect its parent to its child */
        temp = node;
        if(node->left == NULL)
            node = node->right;
        else if(node->right == NULL)
            node = node->left;
        free(temp); /* temp is longer required */
    }
```

```
    }
    return node;
}
treeNode * Find(treeNode *node, int data)
{
    if(node==NULL)
    {
        /* Element is not found */
        return NULL;
    }
    if(data > node->data)
    {
        /* Search in the right sub tree. */
        return Find(node->right,data);
    }
    else if(data < node->data)
    {
        /* Search in the left sub tree. */
        return Find(node->left,data);
    }
    else
    {
        /* Element Found */
        return node;
    }
}
void Inorder(treeNode *node)
{
    if(node==NULL)
    {
        return;
    }
    Inorder(node->left);
```

```
    cout<<node->data<<" ";
    Inorder(node->right);
}
void Preorder(treeNode *node)
{
    if(node==NULL)
    {
        return;
    }
    cout<<node->data<<" ";
    Preorder(node->left);
    Preorder(node->right);
}
void Postorder(treeNode *node)
{
    if(node==NULL)
    {
        return;
    }
    Postorder(node->left);
    Postorder(node->right);
    cout<<node->data<<" ";
}
int main()
{
    treeNode *root = NULL,*temp;
    int ch;
    //clrscr();
    while(1)
    {
        cout<<"\n1.Insert\n2.Delete\n3.Inorder\n4.Preorder\n5.Postorder\n6.FindMin\n7
.FindMax\n8.Search\n9.Exit\n";
        cout<<"Enter ur choice:";
```

```
cin>>ch;
switch(ch)
{
case 1:
    cout<<"\nEnter element to be insert:";
    cin>>ch;
    root = Insert(root, ch);
    cout<<"\nElements in BST are:";
    Inorder(root);
    break;
case 2:
    cout<<"\nEnter element to be deleted:";
    cin>>ch;
    root = Delet(root,ch);
    cout<<"\nAfter deletion elements in BST are:";
    Inorder(root);
    break;
case 3:
    cout<<"\nInorder Travesals is:";
    Inorder(root);
    break;
case 4:
    cout<<"\nPreorder Traversals is:";
    Preorder(root);
    break;
case 5:
    cout<<"\nPostorder Traversals is:";
    Postorder(root);
    break;
case 6:
    temp = FindMin(root);
    cout<<"\nMinimum element is :"<<temp->data;
    break;
```

```
        case 7:
            temp = FindMax(root);
            cout<<"\nMaximum element is :"<<temp->data;
            break;
        case 8:
            cout<<"\nEnter element to be searched:";
            cin>>ch;
            temp = Find(root,ch);
            if(temp==NULL)
            {
                cout<<"Element is not foundn";
            }
            else
            {
                cout<<"Element "<<temp->data<<" is Found\n";
            }
            break;
        case 9:
            exit(0);
            break;
        default:
            cout<<"\nEnter correct choice:";
            break;
        }
    }
    return 0;
}
```

Output:

```
1.Insert
2.Delete
3.Inorder
4.Preorder
5.Postorder
6.FindMin
7.FindMax
8.Search
9.Exit
Enter ur choice:1

Enter element to be insert:2

Elements in BST are:2
1.Insert
2.Delete
3.Inorder
4.Preorder
5.Postorder
6.FindMin
7.FindMax
8.Search
9.Exit
Enter ur choice:1

Enter element to be insert:5

Elements in BST are:2 5
1.Insert
2.Delete
3.Inorder
4.Preorder
5.Postorder
6.FindMin
7.FindMax
8.Search
9.Exit
Enter ur choice:8
```

PROGRAM-14(b)

Objective: Write a Program for insertion, deletion and traversal for threaded binary tree.

Code:

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

enum marker {
    CHILD,
    THREAD
};

struct tbstNode {
    int data;
    struct tbstNode *link[2];
    int marker[2];
};

struct tbstNode *root = NULL;

struct tbstNode * createNode (int data) {
    struct tbstNode *newNode;
    newNode = (struct tbstNode *)malloc(sizeof (struct tbstNode));
    newNode->data = data;
    newNode->link[0] = newNode->link[1] = NULL;
    newNode->marker[0] = newNode->marker[1] = THREAD;
    return newNode;
}

void insertion(int data) {
```

```
    struct tbstNode *parent, *newNode, *temp;
    int path;

    if (!root) {
        root = createNode(data);
        return;
    }

    parent = root;
    /* find the location to insert the new node */
    while (1) {
        if (data == parent->data) {
            printf("Duplicates Not Allowed\n");
            return;
        }
        path = (data > parent->data) ? 1 : 0;
        if (parent->marker[path] == THREAD)
            break;
        else
            parent = parent->link[path];
    }
    /*
     * newnode's left points to predecessor and
     * right to successor
     */
    newNode = createNode(data);
    newNode->link[path] = parent->link[path];
    parent->marker[path] = CHILD;
    newNode->link[!path] = parent;
    parent->link[path] = newNode;
    return;
}
```

```
void delete(int data) {
    struct tbstNode *current, *parent, *temp;
    int path;

    parent = root;
    current = root;

    /* search the node to delete */
    while (1) {
        if (data == current->data)
            break;

        path = (data > current->data) ? 1 : 0;

        if (current->marker[path] == THREAD) {
            printf("Given data is not available!!\n");
            return;
        }

        parent = current;
        current = current->link[path];
    }

    if (current->marker[1] == THREAD) {
        if (current->marker[0] == CHILD) {
            /* node with single child */
            temp = current->link[0];
            while (temp->marker[1] == CHILD) {
                temp = temp->link[1];
            }
            temp->link[1] = current->link[1];
            if (current == root) {
                root = current->link[0];
```

```
            } else {
                parent->link[path] = current->link[0];
            }
        } else {
            /* deleting leaf node */
            if (current == root) {
                root = NULL;
            } else {
                parent->link[path] = current->link[path];
                parent->marker[path] = THREAD;
            }
        }
    } else {
        temp = current->link[1];
        /*
         * node with two child - whose right child has
         * no left child
         */
        if (temp->marker[0] == THREAD) {
            temp->link[0] = current->link[0];
            temp->marker[0] = current->marker[0];
            if (temp->marker[0] == CHILD) {
                struct tbstNode *x = temp->link[0];
                while (x->marker[1] == CHILD) {
                    x = x->link[1];
                }
                x->link[1] = temp;
            }

            if (current == root) {
                root = temp;
            } else {
                printf("path: %d data:%d\n", path, parent->data);
```

```
            parent->link[path] = temp;
        }

    } else {
        /* node with two child */
        struct tbstNode *child;
        while (1) {
            child  = temp->link[0];
            if (child->marker[0] == THREAD)
                break;
            temp = child;
        }

        if (child->marker[1] == CHILD)
            temp->link[0] = child->link[1];
        else {
            temp->link[0] = child;
            temp->marker[0] = THREAD;
        }

        child->link[0] = current->link[0];
        /* update the links */
        if (current->marker[0] == CHILD) {
            struct tbstNode *x = current->link[0];
            while(x->marker[1] == CHILD)
                x = x->link[1];
            x->link[1] = child;
            child->marker[0] = CHILD;
        }
        child->link[1] = current->link[1];
        child->marker[1] = CHILD;

        if (current == root)
```

```
                root = child;
            else
                parent->link[path] = child;
        }
    }
    /* deallocation */
    free(current);
    return;
}

void traversal() {
    struct tbstNode *myNode;
    if (!root) {
        printf("Threaded Binary Tree Not Exists!!\n");
        return;
    }

    myNode = root;
    while (1) {
        while(myNode->marker[0] == CHILD) {
            myNode = myNode->link[0];
        }
        printf("%d ", myNode->data);
        myNode = myNode->link[1];
        if (myNode) {
            printf("%d ", myNode->data);
            myNode = myNode->link[1];
        }
        if (!myNode)
            break;
    }
    printf("\n");
    return;
```

```
}

void search(int data) {
    struct tbstNode *myNode;
    int path;

    if (!root) {
        printf("Tree Not Available!!\n");
        return;
    }

    myNode = root;
    while (1) {
        if (myNode->data == data) {
            printf("Given data present in TBST!!\n");
            return;
        }

        path = (data > myNode->data) ? 1 : 0;
        if (myNode->marker[path] == THREAD)
            break;
        else
            myNode = myNode->link[path];
    }
    printf("Given data is not present in TBST!!\n");
    return;
}

int main () {
    int data, ch;
    while (1) {
        printf("1. Insertion\t2. Deletion\n");
        printf("3. Searching\t4. Traversal\n");
```

```
            printf("5. Exit\nEnter your choice:");
            scanf("%d", &ch);
            switch (ch) {
                case 1:
                    printf("Enter your input data:");
                    scanf("%d", &data);
                    insertion(data);
                    break;
                case 2:
                    printf("Enter your input data:");
                    scanf("%d", &data);
                    delete(data);
                    break;
                case 3:
                    printf("Enter your input data:");
                    scanf("%d", &data);
                    search(data);
                    break;
                case 4:
                    traversal();
                    break;
                case 5:
                    exit(0);
                default:
                    printf("You have entered wrong option!!\n");
                    break;
            }
            printf("\n");
        }
    }
```

Output:

```
1. Insertion 2. Deletion
  3. Searching 4. Traversal
  5. Exit
  Enter your choice:1
  Enter your input data:50

  1. Insertion 2. Deletion
  3. Searching 4. Traversal
  5. Exit
  Enter your choice:1
  Enter your input data:40

  1. Insertion 2. Deletion
  3. Searching 4. Traversal
  5. Exit
  Enter your choice:1
  Enter your input data:60

  1. Insertion 2. Deletion
  3. Searching 4. Traversal
  5. Exit
  Enter your choice:1
  Enter your input data:30

  1. Insertion 2. Deletion
  3. Searching 4. Traversal
  5. Exit
  Enter your choice:1
  Enter your input data:45

  1. Insertion 2. Deletion
  3. Searching 4. Traversal
  5. Exit
  Enter your choice:1
  Enter your input data:55

  1. Insertion 2. Deletion
  3. Searching 4. Traversal
  5. Exit
```

```
Enter your choice:1
Enter your input data:70

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:4
30 40 45 50 55 60 70

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:2
Enter your input data:50

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:4
30 40 45 55 60 70

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:3
Enter your input data:70
Given data present in TBST!!

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:5
```

PROGRAM-15(a)

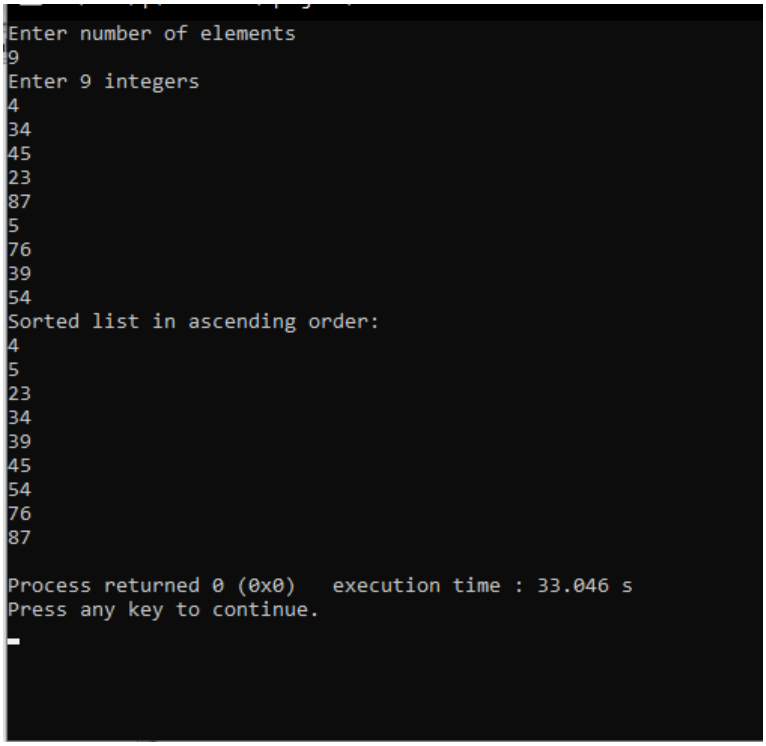
Objective: Write a Program to implement insertion sort.

Code:

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

```
    }
    return 0;
}
```

Output:



PROGRAM-15(b)

Objective: Write a Program to implement Quick sort.

Code:

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

void swap(int *m,int *n)
{
   int temp;
   temp = *m;
   *m = *n;
   *n = temp;
}
int get_key_position(int x,int y )
{
   return((x+y) /2);
}

// Function for Quick Sort
void quicksort(int list[],int m,int n)
{
   int key,i,j,k;
   if( m < n)
   {
      k = get_key_position(m,n);
      swap(&list[m],&list[k]);
      key = list[m];
      i = m+1;
      j = n;
```

```
      while(i <= j)
      {
         while((i <= n) && (list[i] <= key))
               i++;
         while((j >= m) && (list[j] > key))
                  j--;
           if( i < j)
                  swap(&list[i],&list[j]);
      }
      swap(&list[m],&list[j]);
      quicksort(list,m,j-1);
      quicksort(list,j+1,n);
   }
}

// Function to read the data
void read_data(int list[],int n)
{
   int j;
   printf("\n\nEnter the elements:\n");
   for(j=0;j<n;j++)
       scanf("%d",&list[j]);
}

// Function to print the data
void print_data(int list[],int n)
{
   int j;
   for(j=0;j<n;j++)
       printf("%d\t",list[j]);
}

main()
```

```
{
   int list[MAX], num;
   //clrscr();
   printf("\n***** Enter the number of elements Maximum [10] *****\n");
   scanf("%d",&num);
   read_data(list,num);
   printf("\n\nElements in the list before sorting are:\n");
   print_data(list,num);
   quicksort(list,0,num-1);
   printf("\n\nElements in the list after sorting are:\n");
   print_data(list,num);
   //getch();
}
```

Output:

```
***** Enter the number of elements Maximum [10] *****
6


Enter the elements:
56
26
16
66
06
36


Elements in the list before sorting are:
56      26      16      66      6       36

Elements in the list after sorting are:
6       16      26      36      56      66
```

PROGRAM-15(c)

Objective: Write a Program to implement Merge sort.

Code:

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

void Merge(int a[], int tmp[], int lpos, int rpos, int rend)
{
    int i, lend, n, tmppos;
    lend = rpos - 1;
    tmppos = lpos;
    n = rend - lpos + 1;

    while(lpos <= lend && rpos <= rend)
    {
        if(a[lpos] <= a[rpos])
            tmp[tmppos++] = a[lpos++];
        else
            tmp[tmppos++] = a[rpos++];
    }

    while(lpos <= lend)
        tmp[tmppos++] = a[lpos++];
    while(rpos <= rend)
        tmp[tmppos++] = a[rpos++];

    for(i = 0; i < n; i++, rend--)
        a[rend] = tmp[rend];
}
```

```
void MSort(int a[], int tmp[], int left, int right)
{
    int center;
    if(left < right)
    {
        center = (left + right) / 2;
        MSort(a, tmp, left, center);
        MSort(a, tmp, center + 1, right);
        Merge(a, tmp, left, center + 1, right);
    }
}
void MergeSort(int a[], int n)
{
    int *tmparray;
    tmparray = malloc(sizeof(int) * n);
    MSort(a, tmparray, 0, n-1);
    free(tmparray);
}
main()
{
    int i, n, a[10];
    printf("Enter the number of elements :: ");
    scanf("%d",&n);
    printf("Enter the elements :: ");
    for(i = 0; i < n; i++)
    {
        scanf("%d",&a[i]);
    }
    MergeSort(a,n);
    printf("The sorted elements are ::  ");
    for(i = 0; i < n; i++)
        printf("%d  ",a[i]);
```

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

Output:

```
Enter the number of elements :: 7
Enter the elements :: 70 60 50 40 10 20 30
The sorted elements are ::  10  20  30  40  50  60  70
```

PROGRAM-16

Objective: To implement Depth First Traversal and Breadth First Traversal in Graph.

Code:

Depth First Traversal:

```
#include<iostream>
#include<list>
using namespace std;
class Graph
{
    int V;    // No. of vertices

    list<int> *adj;
      void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);
    void addEdge(int v, int w);

    void DFS(int v);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
```

```
}

void Graph::DFSUtil(int v, bool visited[])
{
    visited[v] = true;
    cout << v << " ";

    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}

void Graph::DFS(int v)
{
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    DFSUtil(v, visited);
}

int main()
{
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal"
```

```
        " (starting from vertex 2) \n";
    g.DFS(2);

    return 0;
}
```

Breadth First Traversal:

```
#include<iostream>
#include <list>

using namespace std;

class Graph
{
        int V;
        list<int> *adj;
public:
        Graph(int V); // Constructor

        void addEdge(int v, int w);
        void BFS(int s);
};

Graph::Graph(int V)
{
        this->V = V;
        adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
        adj[v].push_back(w); // Add w to v's list.
```

```
}

void Graph::BFS(int s)
{
	bool *visited = new bool[V];
	for(int i = 0; i < V; i++)
		visited[i] = false;

	list<int> queue;
	visited[s] = true;
	queue.push_back(s);

	list<int>::iterator i;

	while(!queue.empty())
	{
		s = queue.front();
		cout << s << " ";
		queue.pop_front();
		for (i = adj[s].begin(); i != adj[s].end(); ++i)
		{
			if (!visited[*i])
			{
				visited[*i] = true;
				queue.push_back(*i);
			}
		}
	}
}
int main()
{
	Graph g(4);
	g.addEdge(0, 1);
```

```
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        cout << "Following is Breadth First Traversal "
             << "(starting from vertex 2) \n";
        g.BFS(2);

        return 0;
}
```

Output: