4) FIND MIDDLE ELEMENT OF THE STACK

METHOD:
Deleting an element from the middle is not O(1) for an array. Also, we may need to move the middle pointer up when we push an element and move down when we pop(). In a singly linked list, moving the middle pointer in both directions is not possible.
The idea is to use a Doubly Linked List (DLL). We can delete the middle element in O(1) time by maintaining a mid pointer. We can move the mid pointer in both directions using previous and next pointers.

CODE OF THE PROGRAM:

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

class node{
   public:
   int data;
   node *next=NULL;
   node *previous=NULL;
};

class stack{
   private:
   int size=0;
   int midPosition=0;
   node *head=NULL;
   node *tail=NULL;
   node *middle=NULL;

   public:
   void adjustMiddle(){
      if(size==1){
         midPosition=1;
         middle=head;
      }
      else if(size%2==1){
         int position=size/2+1;
         while(midPosition<position){
            midPosition+=1;
            middle=middle->next;
         }
         while(midPosition>position){
            midPosition-=1;
            middle=middle->previous;
         }
      }
      else{
         int position=size/2;
```

```
        while(midPosition<position){
            midPosition+=1;
            middle=middle->next;
        }
        while(midPosition>position){
            midPosition-=1;
            middle=middle->previous;
        }
    }
}
void push(int data){
    node *t=new node;
    t->data=data;
    size++;
    if(head==NULL){
        head={tail=t};
        middle=head;
    }
    else{
        tail->next=t;
        t->previous=tail;
        tail=t;
    }
    adjustMiddle();
}
void pop(){
    size--;
    if(head==tail){
        node *t=head;
        head={tail=NULL};
        delete(t);
    }
    else{
        node *t=tail->previous;
        delete(tail);
        t->next=NULL;
        tail=t;
    }
    adjustMiddle();
}
void deleteMid(){
    if(size==1){
        head={tail={middle=NULL}};
    }
    else if(size%2==0){
        node *t=middle->next;
        if(middle->previous!=NULL){
            middle->previous->next=middle->next;
        }
        else{
```

```
            head=middle->next;
        }
        middle->next->previous=middle->previous;
        delete(middle);
        middle=t;
    }
    else{
        node *t=middle->previous;
        if(middle->previous!=NULL){
            middle->previous->next=middle->next;
        }
        else{
            head=middle->next;
        }
        middle->next->previous=middle->previous;
        delete(middle);
        middle=t;
        midPosition-=1;
    }
    size--;
  }
  void peek(){
    cout<<"\n The stack elements : ";
    node *traverse=head;
    while(traverse!=NULL){
        cout<<traverse->data<<" ";
        traverse=traverse->next;
    }
    cout<<"\n head : "<<head->data;
    cout<<"\n tail : "<<tail->data;
    cout<<"\n Size : "<<size;
    cout<<"\n midP : "<<midPosition;
    cout<<"\n The mid element : "<<middle->data;
  }
};

int main(){
  stack s;
  s.push(10);
  s.peek();
  s.push(20);
  s.peek();
  s.push(30);
  s.peek();
  s.push(40);
  s.peek();
  s.push(50);
  s.peek();
  s.push(60);
  s.peek();
```

```
    s.push(70);
    s.peek();
    s.pop();
    s.peek();
    s.pop();
    s.peek();
    s.pop();
    s.peek();
    s.deleteMid();
    s.peek();
    s.deleteMid();
    s.peek();
    s.deleteMid();
    s.peek();
    s.push(10);
    s.peek();
    return 0;
}
```