Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Electronics & Telecommunication Engineering
Data Structures & Algorithms Lab (DJ19ECSBL1)

Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

Experiment no: 5 Date:3-11-2022

Aim: Write a program to implement stack and queue using singly linked list.

Programming Language: C/C++/Java

Theory:

Basic Operations of Stack:

There are some basic operations that allow us to perform different actions on a stack.

- Push: Add an element to the top of a stack
    - IsFull: Check if the stack is full(overflow condition)
- Pop: Remove an element from the top of a stack
    - IsEmpty: Check if the stack is empty(underflow condition)
- Peek: Get the value of the top element without removing it

Algorithm for push operation: Adding a node to the stack is referred to as **push** operation. Pushing an element to a stack in linked list implementation is different from that of an array implementation. In order to push an element onto the stack, the following steps are involved.

1. Create a node first and allocate memory to it.
2. If the list is empty then the item is to be pushed as the start node of the list. This includes assigning value to the data part of the node and assign null to the address part of the node.
3. If there are some nodes in the list already, then we have to add the new element in the beginning of the list (to not violate the property of the stack). For this purpose, assign the address of the starting element to the address field of the new node and make the new node, the starting node of the list.

Algorithm for pop operation: Deleting a node from the top of stack is referred to as **pop** operation. Deleting a node from the linked list implementation of stack is different from that in the array implementation. In order to pop an element from the stack, we need to follow the following steps :

1. **Check for the underflow condition:** The underflow condition occurs when we try to pop from an already empty stack. The stack will be empty if the head pointer of the list points to null.
2. **Adjust the head pointer accordingly:** In stack, the elements are popped only from one end, therefore, the value stored in the top pointer must be deleted and the node must be freed. The next node of the top node now becomes the top node.

Algorithm for peek operation: Peek function displays the data value of the top where top pointer points.

1.Check for the underflow condition: The underflow condition occurs when we try to peek from an already empty stack. The stack will be empty if the head pointer of the list points to null.
2.else print the data value of the top where top pointer points.

Display the nodes (Traversing): Displaying all the nodes of a stack needs traversing all the nodes of the linked list organized in the form of stack. For this purpose, we need to follow the following steps.

1. Copy the head pointer into a temporary pointer.
2. Move the temporary pointer through all the nodes of the list and print the value field attached to every node.

Basic Operations of Queue:

Enqueue of an Element: We will add a new element to queue from the rear end

1. Create a new node with the value to be inserted.
2. If the queue is empty, then set both front and rear to point to newnode.
3. If the queue is not empty, then set next to the rear of the new node and the rear to point to the new node.

Dequeue of an Element : We will delete the existing element from the queue from the front end.

1. If the queue is empty, terminate the method.
2. If the queue is not empty, increment the front to point to the next node.
3. Finally, check if the front is null, then set rear to null also. This signifies an empty queue.

Display the nodes (Traversing): Displaying all the nodes of a queue needs traversing all the nodes of the linked list organized in the form of stack. For this purpose, we need to follow the following steps.

1.Copy the head pointer into a temporary pointer.
2.Move the temporary pointer through all the nodes of the list and print the value field attached to every node.

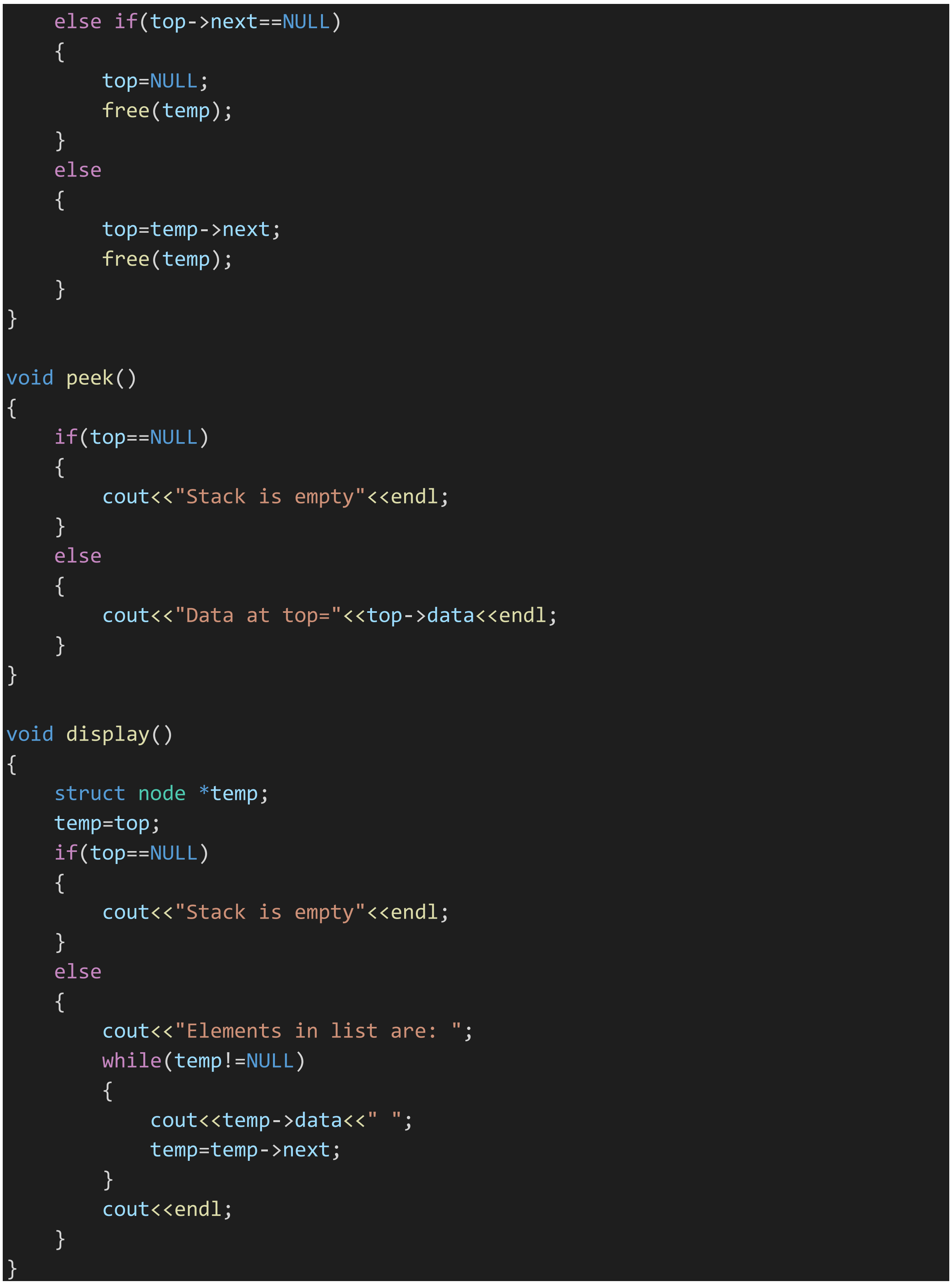
Department of Electronics & Telecommunication Engineering
Data Structures & Algorithms Lab (DJ19ECSBL1)

Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

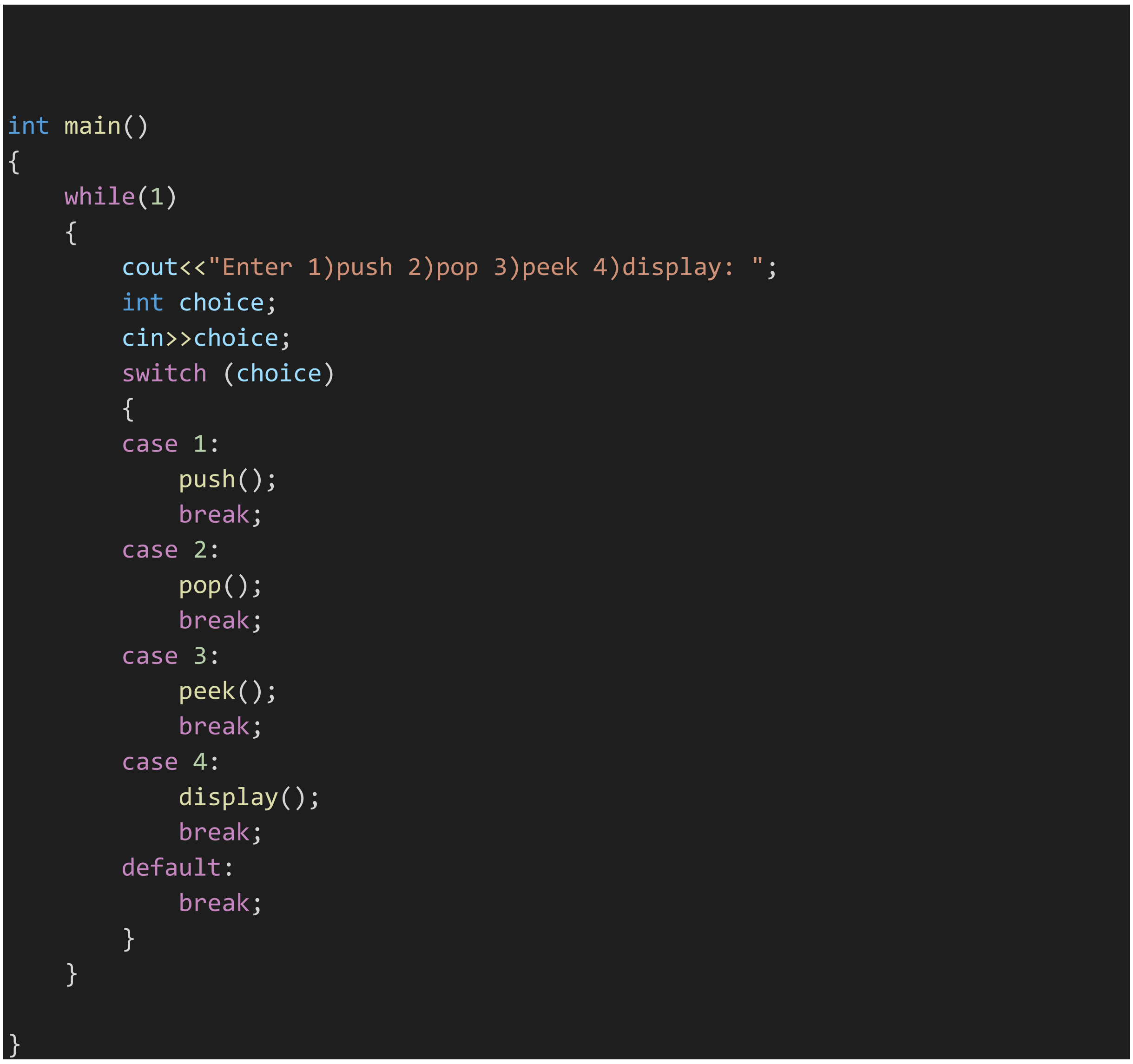
Code:

Basic Operations of Stack:

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

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

struct node *top=NULL;

void push()
{
    struct node *newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    cout<<"Enter data for node: ";
    int n;
    cin>>n;
    newnode->data=n;
    if(top==NULL)
    {
        top=newnode;
        newnode->next=NULL;

    }
    else
    {
        newnode->next=top;
        top=newnode;
    }
}

void pop()
{
    struct node *temp;
    temp=top;
    if(top==NULL)
    {
        cout<<"Stack is empty"<<endl;
    }
```

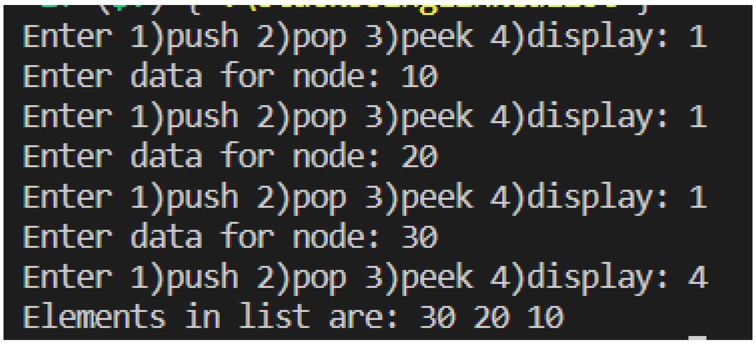


```
    else if(top->next==NULL)
    {
        top=NULL;
        free(temp);
    }
    else
    {
        top=temp->next;
        free(temp);
    }
}

void peek()
{
    if(top==NULL)
    {
        cout<<"Stack is empty"<<endl;
    }
    else
    {
        cout<<"Data at top="<<top->data<<endl;
    }
}

void display()
{
    struct node *temp;
    temp=top;
    if(top==NULL)
    {
        cout<<"Stack is empty"<<endl;
    }
    else
    {
        cout<<"Elements in list are: ";
        while(temp!=NULL)
        {
            cout<<temp->data<<" ";
            temp=temp->next;
        }
        cout<<endl;
    }
}
```



```
int main()
{
    while(1)
    {
        cout<<"Enter 1)push 2)pop 3)peek 4)display: ";
        int choice;
        cin>>choice;
        switch (choice)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            peek();
            break;
        case 4:
            display();
            break;
        default:
            break;
        }
    }

}
```

Push and Display Operation:

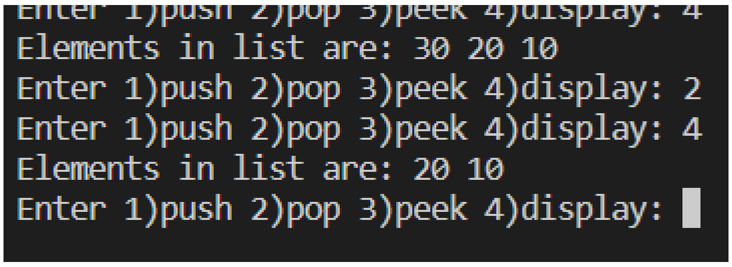


```
Enter 1)push 2)pop 3)peek 4)display: 1
Enter data for node: 10
Enter 1)push 2)pop 3)peek 4)display: 1
Enter data for node: 20
Enter 1)push 2)pop 3)peek 4)display: 1
Enter data for node: 30
Enter 1)push 2)pop 3)peek 4)display: 4
Elements in list are: 30 20 10
```

Department of Electronics & Telecommunication Engineering
Data Structures & Algorithms Lab (DJ19ECSBL1)

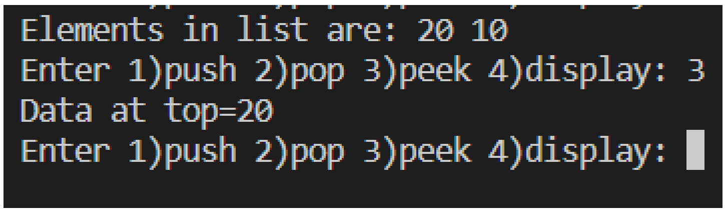
Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

Pop and Display operation:



```
Enter 1)push 2)pop 3)peek 4)display: 4
Elements in list are: 30 20 10
Enter 1)push 2)pop 3)peek 4)display: 2
Enter 1)push 2)pop 3)peek 4)display: 4
Elements in list are: 20 10
Enter 1)push 2)pop 3)peek 4)display: 
```

Peek Operation:



```
Elements in list are: 20 10
Enter 1)push 2)pop 3)peek 4)display: 3
Data at top=20
Enter 1)push 2)pop 3)peek 4)display: 
```

Department of Electronics & Telecommunication Engineering
Data Structures & Algorithms Lab (DJ19ECSBL1)

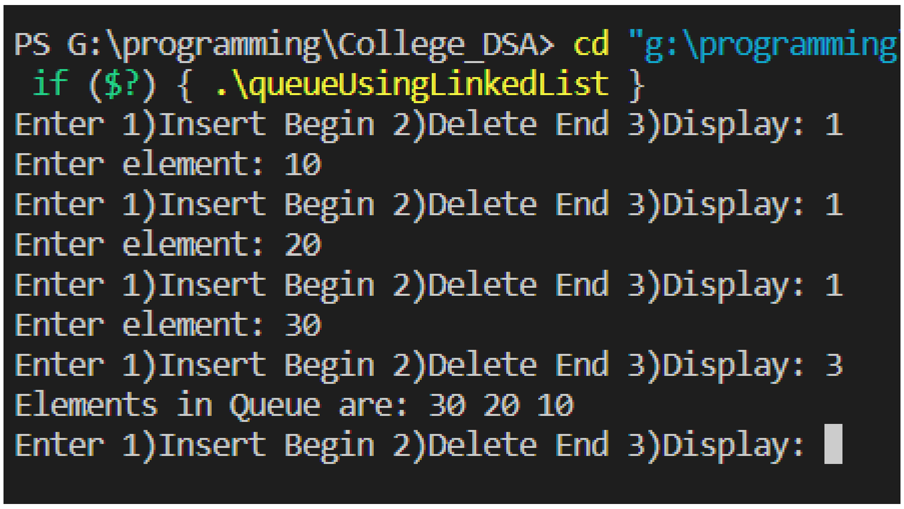
Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

Basic Operations of Queue:

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

struct node
{
    int data;
    struct node *next;
};
struct node *front=NULL;
struct node *rear=NULL;
void insertFront()
{
    struct node *newnode;
    newnode=(struct node *)malloc(sizeof(struct node));
    int n;
    cout<<"Enter element: ";
    cin>>n;
    newnode->data=n;
    if(front==NULL)
    {
        front=newnode;
        rear=newnode;
        newnode->next=NULL;
    }
    else
    {
        newnode->next=front;
        front=newnode;
    }
}

void deleteEnd()
{
    struct node *temp;
    struct node *temp1;
    temp=front;
    if(temp==NULL)
    {
        cout<<"No element in linked list"<<endl;
    }
    else
    {
        while(temp->next!=NULL)
        {
            temp1=temp;
            temp=temp->next;
        }
        rear=temp1;
        rear->next=NULL;
         free(temp);
    }
}

void display()
{
    struct node *temp;
    temp=front;
    if(front==NULL)
    {
        cout<<"Stack is empty"<<endl;
    }
    else
    {
        cout<<"Elements in Queue are: ";
        while(temp!=NULL)
        {
            cout<<temp->data<<" ";
            temp=temp->next;
        }
        cout<<endl;
    }
}

int main()
{
    while(1)
    {
        cout<<"Enter 1)Insert Begin 2)Delete End 3)Display: ";
        int choice;
        cin>>choice;
        switch (choice)
        {
        case 1:
            insertFront();
            break;
        case 2:
            deleteEnd();
```

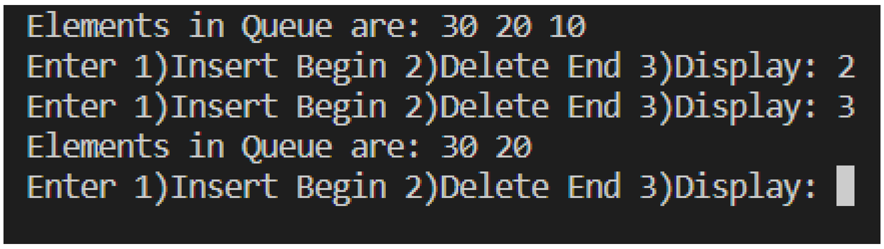
```
                break;
            case 3:
                display();
                break;
            default:
                break;
            }
    }
}
```

Insert Begin Operation on queue created using linked list:



```
PS G:\programming\College_DSA> cd "g:\programming
 if ($?) { .\queueUsingLinkedList }
Enter 1)Insert Begin 2)Delete End 3)Display: 1
Enter element: 10
Enter 1)Insert Begin 2)Delete End 3)Display: 1
Enter element: 20
Enter 1)Insert Begin 2)Delete End 3)Display: 1
Enter element: 30
Enter 1)Insert Begin 2)Delete End 3)Display: 3
Elements in Queue are: 30 20 10
Enter 1)Insert Begin 2)Delete End 3)Display: 
```

Delete end operation in queue created by linked list:



```
Elements in Queue are: 30 20 10
Enter 1)Insert Begin 2)Delete End 3)Display: 2
Enter 1)Insert Begin 2)Delete End 3)Display: 3
Elements in Queue are: 30 20
Enter 1)Insert Begin 2)Delete End 3)Display: 
```

Name: Nihal Nisar Shaikh **SAP ID: 60002200155** **Batch:E1-4**

Result and Conclusion:

1) Implemented stacks using linked list.
    a) Here push operation became the insert begin operation considering head as top pointer.
    b) Pop operation is done by delete begin as stack is LIFO.
2) Implemented queue using linked list.
    a) As queue is FIFO we implemented enqueue by insert begin operation.
    b) For implementation of dequeue we have used delete end.