Chapter 5

Linked Stacks and Queues

5.1 Linked Stacks

- Let us first define a structure for an element of the stack

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

node *temp, *top;
```

- As with the entries of contiguous stacks, we shall push and pop nodes from one end of a linked stack, called its top.

- If we have an item that we wish to push onto a linked stack, we must first create a new node and put this item into the node, and then push the node onto the stack.

- The first question to settle is to determine whether the beginning or the end of the linked structure will be the top of the stack.

- At first glance it may appear that (as for contiguous stacks) it may be easier to add a node at the end, but this method makes popping the stack difficult.

- There is no quick way to find the node immediately before a given one in a linked structure, since the pointers stored in the structure give only one-way directions. Thus after we remove the last element, finding the element at the end of the linked structure might require tracing all the way from its head.

- To pop out linked stack, it is much better to make all addition and deletion at the **beginning** of the structure. Hence the top of the stack will always be the first node of the linked structure.

- Let us now consider how to push an element into a linked stack. Initially top needs to be initialised to NULL. The method below considers two situations: the case where the linked stack is empty and the case where it already has some elements.

```
void push(int num)
{
	node *temp;
	temp = new node;
	temp->data = num;

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

- Now let us consider popping elements from the linked stack.

```
int empty()
{
	if(top == NULL)
		return 1;
	else
		return 0;
}
int pop()
{
	int num;
	node *temp;

	if(!empty())
```

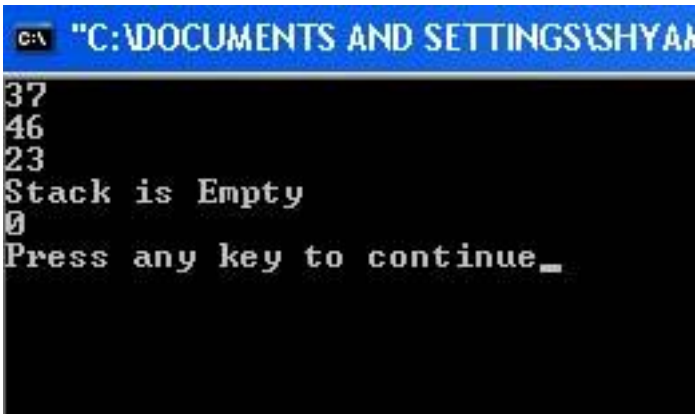
```
        {
                num = top->data;
                temp = top;
                top = top->next;
                delete temp;
                return num;
        }
        else
        {
                cout<<"Stack is Empty"<<endl;
                return 0;
        }
}
```

- The full program to handle a linked stack is as follows:

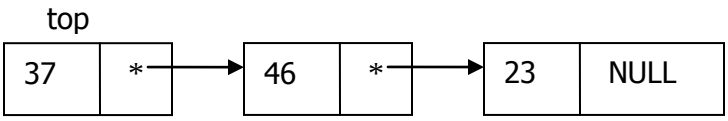
```
#include <iostream.h>
#include <conio.h>

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

class ADTstack
{
private:
        node *top;

public:
        ADTstack()
        {
                top = NULL;
        }

        int empty()
        {
```

```
            if(top == NULL)
                  return 1;
            else
                  return 0;
      }

      void push(int num)
      {
            node *temp;
            temp = new node;
            temp->data = num;

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

      int pop()
      {
            int num;
            node *temp;

            if(!empty())
            {
                  num = top->data;
                  temp = top;
                  top = top->next;
                  delete temp;
                  return num;
            }
            else
            {
```

```
                    cout<<"Stack is Empty"<<endl;
                    return 0;
                }
        }
};

void main()
{
        ADTstack st;
        st.push(23);
        st.push(46);
        st.push(37);
        cout<<st.pop()<<endl;
        cout<<st.pop()<<endl;
        cout<<st.pop()<<endl;
        cout<<st.pop()<<endl;
}
```



- Below is a picture of what the linked stack looks like after the first three push operations.



5.2 Linked Queues

- Let us first define a structure for an element of the queue

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

node *temp, *front, *rear;
```

- As with the entries of contiguous queue, we shall append from one end (tail) and serve from another end (front).

- Consider how to append an element into the linked queue. Initially front and tail needs to be initializes to NULL. Since we can only add to the end of the list, we just have to check if it's the last node or not.

```
void append(int num)
{
      if(rear != NULL)
      {
            rear->next = new node;
            rear = read->next;
            rear->data = num;
            rear->next = NULL;
      }
}
```

- If the queue is empty, then you need to use the following code.

```
else
{
      front = rear = new node;
      front->data = num;
      front->next = NULL;
}
```

- Checking if the queue is empty

```
int empty()
{
      if(front == NULL)
            return 1;
      else
            return 0;
}
```

- Let's consider serving elements from the front of the linked queue

```
int serve()
{
      int num;
      if (!empty())
      {
            num = front->data;
            node *temp = front;
            front = front->next;
            delete temp;

            if(front == NULL)
                  rear = NULL;
            return num;
      }
      else
      {
            cout<<"Queue is Empty";
            return 0;
      }
}
```

- The full program to handle a linked queue is as follows:

```
#include <iostream.h>
#include <conio.h>
```

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

class ADTqueue
{
private:
      node *front, *rear;

public:
      ADTqueue()
      {
            front = NULL;
            rear = NULL;
      }

      int empty()
      {
            if(front == NULL)
                  return 1;
            else
                  return 0;
      }
```

```
      void append(int num)
      {
            if(rear != NULL)
            {
                  rear->next = new node;
                  rear = rear->next;
                  rear->data = num;
                  rear->next=NULL;
            }
            else
            {
                  front = rear = new node;
                  front->data=num;
```

```
                front->next=NULL;
            }
        }

        int serve()
        {
            int num;
            if(!empty())
            {
                num = front->data;
                node *temp = front;
                front = front->next;
                delete temp;

                if (front == NULL)
                    rear = NULL;

                return num;
            }
            else
            {
                cout<<"Queue is Empty";
                return 0;
            }
        }
};

void main()
{
        ADTqueue q;
        q.append(23);
        q.append(46);
        q.append(37);
        cout<<q.serve()<<endl;
        cout<<q.serve()<<endl;
        cout<<q.serve()<<endl;
        cout<<q.serve()<<endl;
}
```


"C:\DOCUMENTS AND SETTINGS\SHYAMA
23
46
37
Queue is Empty0
Press any key to continue