LAB WORK - 2

Aawishkar Tiwari

Computer Engineering – 3rd Semester

Roll no – 59

GitHub : https://github.com/Aawishkar/DSA_LAB.git

Stack:

stack.h

```
#ifndef stack_h
#define stack_h


class Stack{
    public:
        virtual void push(int )=0;
        virtual int pop()=0;
        virtual bool isEmpty()=0;
        virtual bool isFull()=0;
        virtual int top()=0;
};


#endif
```

Linkedlist.h

```
#ifndef LinkedList_h
#define LinkedList_h

class Node{
    public:
        int data;
        Node *next;
        Node(){}
        Node(int x){
            data =x;
            next=nullptr;
        }
        Node(int x, Node *y){
            data =x;
            next=y;
        }
```

```
};
class LinkedList{
    public:
        Node *HEAD;
        Node *TAIL;

        LinkedList(Node *HEAD =nullptr, Node *TAIL=nullptr){
            this->HEAD=HEAD;
            this->TAIL=TAIL;
        }

        bool Empty();
        void addToHead(int );
        void removeFromHead();
};

#endif
```

stack_array.h

```
#ifndef ArrayStack_h
#define ArrayStack_h
#define MAX_STACKSIZE 50
#include "stack.h"

class ArrayStack: public Stack{
    public:
        int array[MAX_STACKSIZE];
        int topindex;
        ArrayStack();
        void push(int );
        int pop();
        bool isEmpty();
        bool isFull();
        int top();
};


#endif
```

stack_linkedlist.h

```
#ifndef LinkedListStack_h
#define LinkedListStack_h
```

```
#include "stack.h"
#include "linkedlist.h"
class LinkedListStack : public Stack
{
    public:
        LinkedList *list;
        LinkedListStack();
        void push(int);
        int pop();
        bool isEmpty();
        bool isFull();
        int top();
};



#endif
```

Linkedlist.cpp

```
#include<iostream>
#include "linkedlist.h"
#include "stack_linkedlist.h"


using namespace std;


//Checking List is empty or not
bool LinkedList::Empty(){
    if(HEAD==nullptr){
        return true;
    }
    else{
        return false;
    }
}


//Adding data to head of list
void LinkedList::addToHead(int data){
    Node* newNode= new Node(data);
    if(Empty()){
        HEAD=newNode;
        TAIL=newNode;
```

```
    }
    else{
        newNode->next=HEAD;
        HEAD=newNode;
    }
}



//Removing data from the head of list
void LinkedList::removeFromHead(){
    Node *nodeToDelete =HEAD;
    HEAD=HEAD->next;
    delete nodeToDelete;

}
```

Stack_array.cpp

```
#include "stack_array.h"

#include <iostream>
using namespace std;

ArrayStack::ArrayStack(){
    topindex=-1;
}


bool ArrayStack::isEmpty(){
    if(topindex==-1){
        return true;
    }
    else{
        return false;
    }
}


bool ArrayStack::isFull(){
    if(topindex==MAX_STACKSIZE){
        return true;
    }
```

```
    else{
        return false;
    }
}


void ArrayStack::push(int num){
    if(!isFull()){
        topindex++;
        *(array+topindex)=num;

    }
    else{
        cout<<"Stack Overflow"<<endl;
    }
}


int ArrayStack::pop(){
    int result;
    if(!isEmpty()){
        result=*(array+topindex);
        topindex--;
    }
    else{
        cout<<"Stack Underflow"<<endl;

    }
    return result;
}

int ArrayStack::top(){
    return *(array+topindex);
}
```

Stack_linkedlist.cpp

```
#include "stack_linkedlist.h"

LinkedListStack::LinkedListStack(){
    list=new LinkedList();
}
```

```
bool LinkedListStack::isEmpty(){
    if(list->Empty()){
        return true;
    }
    else{
        return false;
    }
}

bool LinkedListStack::isFull()
{
    return false;
}
void LinkedListStack::push(int num)
{
    list->addToHead(num);
}

int LinkedListStack::pop()
{
    int result;
    result= list->HEAD->data;
    list->removeFromHead();
    return result;
}
int LinkedListStack::top()
{
    return list->HEAD->data;
}
```

Main.cpp

```
#include <iostream>
using namespace std;
#include "stack_array.h"
#include "stack_linkedlist.h"


int main()
{
    cout<<"\n\tStack Implementation Using Array";


    Stack *A1 = new ArrayStack();
```

```
cout<<"\n\nEmpty check";
if(A1->isEmpty()){
    cout<<"\nStack is Empty";
}
else{
    cout<<"\nStack is Not Empty";
}

cout<<"\n\nFull check";
if(A1->isFull()){
    cout<<"\nStack is Full";
}
else
{
    cout<<"\nStack is Not Full";
}




cout<<endl<<"\nPushing into stack";
A1->push(19);
A1->push(12);
A1->push(14);


if(A1->isFull())
{
    cout<<"\nStack is Full";
}
else
{
    cout<<"\nStack is Not Full";
}


cout<<endl<<"\nPop implementation";
cout<<"\nPopped "<<A1->pop();
cout<<"\nPopped "<<A1->pop();

if(A1->isEmpty())
{
    cout<<"\nStack is Empty";
}
```

```
    else
    {
        cout<<"\nStack is Not Empty";
    }

    delete A1;


    cout<<"\n\n\tStack Implementation Using Linked List";
    Stack *A2=new LinkedListStack();

    cout<<"\n\nEmpty check";
    if(A2->isEmpty())
    {

        cout<<"\nStack is Empty";
    }
    else
    {
        cout<<"\nStack is Not Empty";
    }


    cout<<"\n\nFull check";
    if(A2->isFull())
    {
        cout<<"\nStack is Full";
    }
    else
    {
        cout<<"\nStack is Not Full";
    }

    cout<<endl<<"\nPush Implementation";
    A2->push(13);
    A2->push(45);
    A2->push(9);
    if(A2->isFull())
    {
        cout<<"\nStack is Full";
    }
    else
    {
        cout<<"\nStack is Not Full";
    }
```

```
    cout<<endl<<"\nPop implementation";
    cout<<"\nPopped "<<A2->pop();
    cout<<"\nPopped "<<A2->pop();
    if(A2->isEmpty())
    {
        cout<<"\nStack is Empty";
    }
    else
    {
        cout<<"\nStack is Not Empty\n";
    }
    delete A2;
}
```

Output screen:

```
        Stack Implementation Using Array

Empty check
Stack is Empty

Full check
Stack is Not Full

Pushing into stack
Stack is Not Full

Pop implementation
Popped 14
Popped 12
Stack is Not Empty

        Stack Implementation Using Linked List

Empty check
Stack is Empty

Full check
Stack is Not Full

Push Implementation
Stack is Not Full

Pop implementation
Popped 9
Popped 45
Stack is Not Empty
PS C:\DSA_LAB\Stack>
```