06/08/2012

Experiment No.:1a

STACK

# AIM:

# Implementation of a Stack

# ALGORITHM:

The structure of a stack is as following :

|  |  |  |
| --- | --- | --- |
| top | length | buffer |

# Here top is index of top of the stack which is initialized as -1, length is the length of the stack and buf is the array of data.

# The following function checks whether a stack is underflowing or not.

# Stack.underflow() {

# if(Stack.top<0)

# return true

# else

# return false

# }

# The following function checks whether a stack is overflowed or not.

# Stack.overflow() {

# if(Stack.top≥stack.length-1)

# return true

# else

# return false

# }

# The following function pushes an element at top of the stack.

# Stack.push(e) {

# if(NOT Stack.overflow()) {

# Stack.top=Stack.top+1

# Stack.buf[Stack.top]=e

# }

# }

# The following function pops and return the element at top of the stack

# stack.pop() {

# if(NOT Stack.underflow()) {

# temp=Stack.buf[Stack.top]

# stack.top=Stack.top-1

# return temp

# }

# }

# SOURCE CODE:

# /\*\*

# \* Author : Rakesh Malik

# \* Date : 24/07/2012

# \* Subject : Stack

# \* Assignment no. :

# \*/

# #include <iostream>

# #include "../myexception.h"

# #include <cstdlib>

# #include <iomanip>

# #include <cstdio>

# using namespace std;

# using namespace exception;

# template <class DT> class Stack {

# DT \*buf;

# int top;

# int length;

# public:

# Stack(const Stack<DT>&);

# Stack(int);

# ~Stack();

# bool isOverflowed();

# bool isUnderflowed();

# void push(DT);

# DT pop();

# int getLength();

# int getNumberOfElements();

# void clear();

# void display();

# };

# template <class DT> Stack<DT>::Stack(const Stack<DT>& s) {

# Stack::length=s.length;

# Stack::top=s.top;

# Stack::buf=new DT[s.length];

# for(int i=0;i<=s.top;i++)

# Stack::buf[i]=s.buf[i];

# }

# template <class DT> Stack<DT>::Stack(int length=0) {

# if(length>=0) {

# Stack::length=length;

# try {

# Stack::buf=new DT[length];

# }catch(bad\_alloc) {

# cerr<<"Error : 'bad\_alloc' exception during Stack allocation"<<endl;

# }

# }else {

# Stack::length=0;

# try {

# Stack::buf=new DT[0];

# }catch(bad\_alloc) {

# cerr<<"Error : 'bad\_alloc' exception during Stack allocation"<<endl;

# }

# }

# Stack::top=-1;

# }

# template <class DT> Stack<DT>::~Stack() {

# delete Stack::buf;

# Stack::length=0;

# Stack::top=-1;

# }

# template <class DT> bool Stack<DT>::isOverflowed() {

# return Stack::top>=Stack::length-1;

# }

# template <class DT> bool Stack<DT>::isUnderflowed() {

# return Stack::top<0;

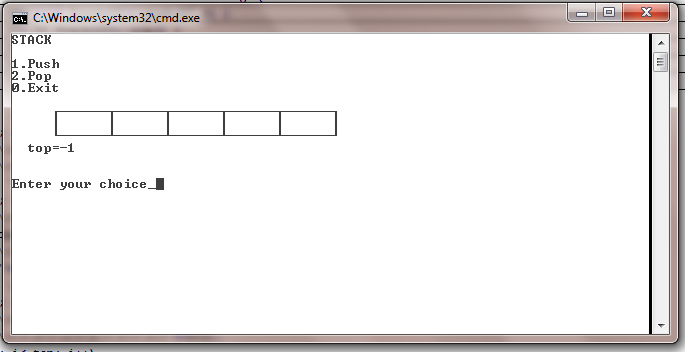


Fig 1: Empty Stack

# }

# template <class DT> void Stack<DT>::push(DT e) {

# if(!Stack::isOverflowed())

# Stack::buf[++Stack::top]=e;

# else

# throw StackOverflowException();

# }

# template <class DT> DT Stack<DT>::pop() {

# if(!Stack::isUnderflowed())

# return Stack::buf[Stack::top--];

# else

# throw StackUnderflowException();

# }

# template <class DT> int Stack<DT>::getLength() {

# return Stack::length;

# }

# template <class DT> int Stack<DT>::getNumberOfElements() {

# return Stack::top+1;

# }

# template <class DT> void Stack<DT>::clear() {

# top=-1;

# }

# template <class DT> void Stack<DT>::display() {

# cout<<" \xDA";

# for(int i=0; i<length-1; i++)

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xC2";

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xBF"<<endl

# <<" ";

# for(int i=0; i<=top; i++)

# cout<<"\xB3"<<setw(5)<<buf[i]<<" ";

# for(int i=top+1; i<length; i++)

# cout<<"\xB3 ";

# cout<<"\xB3"<<endl

# <<" \xC0";

# for(int i=0; i<length-1; i++)

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xC1";

# cout<<"\xC4\xC4\xC4\xC4\xC4\xC4\xD9"<<endl;

# for(int i=0; i<=top; i++)

# cout<<" ";

# cout<<" top="<<top<<endl;

# }

# main() {

# Stack<int> s(5);

# int choice=1,e;

# char msg[50]="";

# while(choice) {

# system("cls");

# cout<<"STACK"<<endl

# <<endl

# <<"1.Push"<<endl

# <<"2.Pop"<<endl

# <<"0.Exit"<<endl

# <<endl;

# s.display();

# cout<<msg<<endl

# <<endl

# <<"Enter your choice\_";

# cin>>choice;

# cout<<endl;

# switch(choice) {

# case 0:

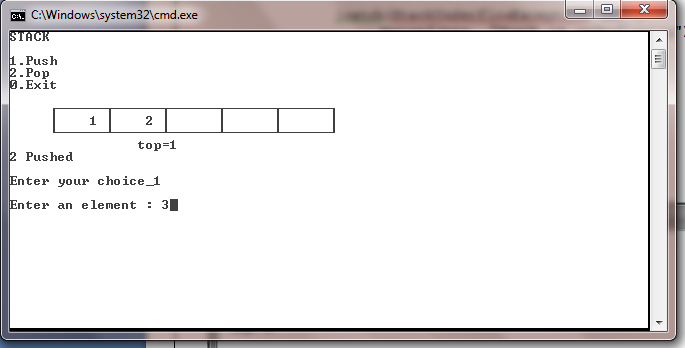


Fig 2: Push

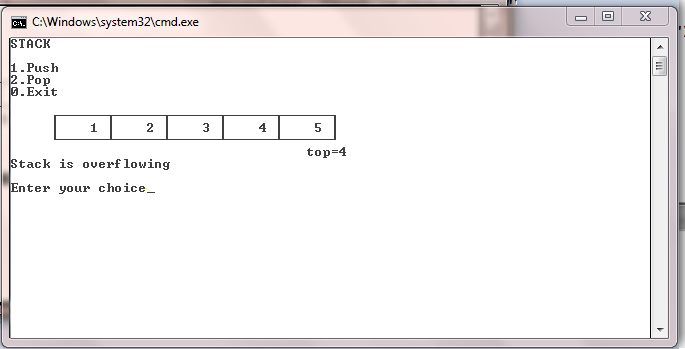


Fig 3: Overflow Message while Pushing

# return 0;

# case 1:

# if(s.isOverflowed())

# sprintf(msg, "Stack is overflowing");

# else {

# cout<<"Enter an element : ";

# cin>>e;

# s.push(e);

# sprintf(msg, "%d Pushed", e);

# }

# break;

# case 2:

# try {

# sprintf(msg, "%d Popped", s.pop());

# }catch(StackUnderflowException ex) {

# sprintf(msg, "Stack is underflowing");

# }

# break;

# default:

# sprintf(msg, "Wrong choice");

# }

# }

# }

**myexception.h**

#include <iostream>

using namespace std;

namespace exception {

class Exception {

int err\_no;

string msg;

public:

Exception(int,string);

int getErrorNumber();

string getMessage();

};

Exception::Exception(int err\_no,string msg) {

Exception::err\_no=err\_no;

Exception::msg=msg;

}

int Exception::getErrorNumber() {

return Exception::err\_no;

}

string Exception::getMessage() {

return Exception::msg;

}

/\* Queue \*/

class QueueEmptyException:public Exception {

public:

QueueEmptyException():Exception(100,"Queue Empty Exception"){}

};

class QueueFullException:public Exception {

public:

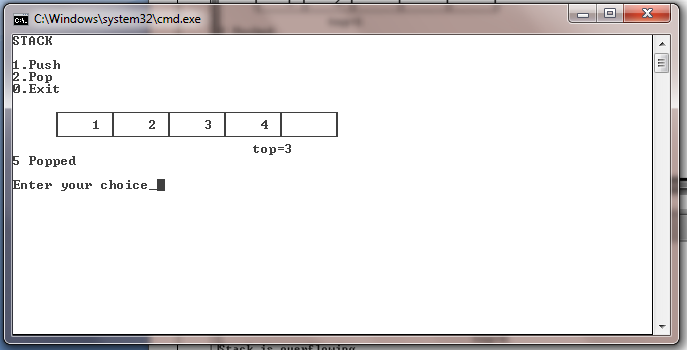


Fig 4: Pop

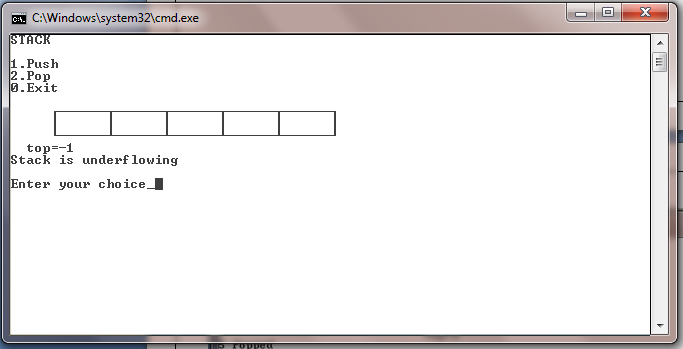


Fig 5: Underflow message while popping

QueueFullException():Exception(101,"Queue Full Exception"){}

};

/\* Stack \*/

class StackUnderflowException:public Exception {

public:

StackUnderflowException():Exception(200,"Stack Underflow Exception"){}

};

class StackOverflowException:public Exception {

public:

StackOverflowException():Exception(201,"Stack Overflow Exception"){}

};

/\*List \*/

class IndexOutOfBoundsException:public Exception {

public :

IndexOutOfBoundsException():Exception(300, "Linked list index out of bounds"){}

};

class LinkedListEmptyException:public Exception {

public :

LinkedListEmptyException():Exception(301, "Linked list is empty"){}

};

/\* Tree \*/

class NodeNotFoundException:public Exception {

public :

NodeNotFoundException():Exception(401, "Node not found"){}

};

class NodeAlreadyExistsException:public Exception {

public :

NodeAlreadyExistsException():Exception(402, "Node already exists"){}

};

}

# RESULT:

The program has run successfully and given result as expected.