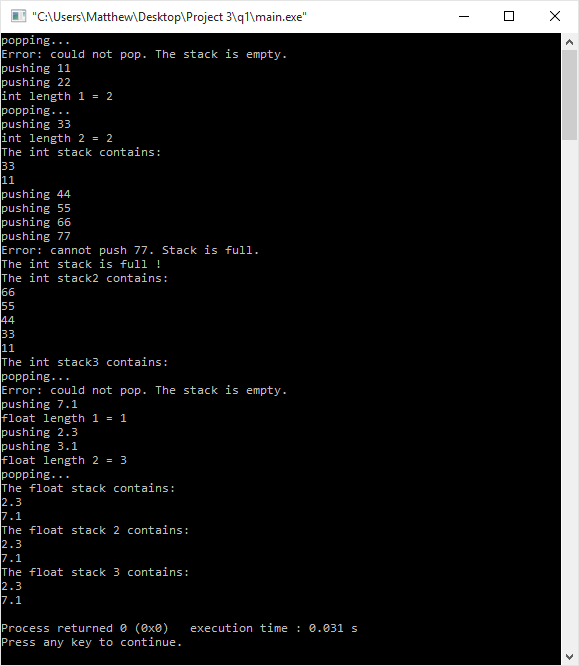
Matthew Wills

Machine Type: AMD Phenom II X4 970 Processor 3.50 GHz

Compiler used: mingw

**Question 1**



**Source Code:**

**main.cpp**

#include "stack.h"

int main()

{

Stack <int> IntStack;

int x;

IntStack.Pop(x);

IntStack.Push(11);

IntStack.Push(22);

cout << "int length 1 = ";

cout << IntStack.length() << endl;

IntStack.Pop(x);

IntStack.Push(33);

cout << "int length 2 = " << IntStack.length() << endl;

cout << "The int stack contains: " << endl;

IntStack.Print();

IntStack.Push(44);

IntStack.Push(55);

IntStack.Push(66);

IntStack.Push(77);

if(IntStack.IsFull() == false)

cout << "The int stack is not full !" << endl;

else

cout << "The int stack is full !" << endl;

Stack <int> IntStack2(IntStack);

cout << "The int stack2 contains: " << endl;

IntStack2.Print();

IntStack2.MakeEmpty();

cout << "The int stack3 contains: " << endl;

IntStack2.Print();

Stack <float> FloatStack;

float y;

FloatStack.Pop(y);

FloatStack.Push(7.1);

cout << "float length 1 = " << FloatStack.length() << endl;

FloatStack.Push(2.3);

FloatStack.Push(3.1);

cout << "float length 2 = " << FloatStack.length() << endl;

FloatStack.Pop(y);

cout << "The float stack contains: " << endl;

FloatStack.Print();

Stack <float> FloatStack2 = FloatStack;

cout << "The float stack 2 contains: " << endl;

FloatStack2.Print();

FloatStack.MakeEmpty();

cout << "The float stack 3 contains: " << endl;

FloatStack2.Print();

return 0;

}

**stack.h**

#include <cstddef>

#include <iostream>

#define MAX\_ITEM 5

using namespace std;

template<class ItemType>

struct NodeType

{

ItemType info;

NodeType\* next;

};

template<class ItemType>

class Stack

{

private:

NodeType<ItemType>\* topPtr = NULL; // It points to a singly-linked list

public:

Stack( ); // default constructor: Stack is created and empty

Stack(const Stack<ItemType> &x); // copy constructor: implicitly called for a

// deep copy

void MakeEmpty(); // Stack is made empty; you should deallocate all the

// the nodes of the linked list

bool IsEmpty( ); // test if the stack is empty

bool IsFull( ); // test if the stack is full; assume MAXITEM=5

int length( ); // return the number of elements in the stack

void Print( ); // print the value of all elements in the stack in the sequence

// from the top to bottom

void Push(ItemType x); // insert x onto the stack

void Pop(ItemType &x); // delete the top element from the stack

// Precondition: the stack is not empty

~Stack(); // Destructor: memory for nodes needs to be deallocated

};

template <class ItemType> Stack<ItemType>::Stack()

{

topPtr = NULL;

}

template <class ItemType> Stack<ItemType>::Stack(const Stack<ItemType> &x)

{

//if the stack being passed is empty, use the default constructor instead

if(x.topPtr == NULL)

{

Stack();

return;

}

topPtr = new NodeType<ItemType>;

NodeType<ItemType>\* newNode = topPtr;

for(NodeType<ItemType>\* iter = x.topPtr; iter != NULL; iter = iter->next)

{

newNode->info = iter->info;

if (iter->next != NULL)

{

newNode->next = new NodeType<ItemType>;

newNode = newNode->next;

}

}

newNode->next = NULL;

}

template <class ItemType> void Stack<ItemType>::Push(ItemType x)

{

cout << "pushing " << x << endl;

//display an error message if the stack is already full

if(this->IsFull())

{

cout << "Error: cannot push " << x << ". Stack is full." << endl;

return;

}

//create a new node

NodeType<ItemType>\* newNode = new NodeType<ItemType>;

//set the info in the new node to the value being pushed

newNode->info = x;

//link the new node to the previous node

newNode->next = topPtr;

//set the new node as the top of the stack

topPtr = newNode;

}

template <class ItemType> void Stack<ItemType>::Pop(ItemType &x)

{

cout << "popping..." << endl;

//display an error message if the stack is empty

if(topPtr == NULL)

{

cout << "Error: could not pop. The stack is empty." << endl;

return;

}

//set x equal to the data at the top of the stack

x = topPtr->info;

//create temporary variable to save the location of the item beneath while you delete the top of the stack

NodeType<ItemType>\* temp = topPtr->next;

//deallocate the top item on the stack

delete topPtr;

//set the new top of the stack equal to the location of the new highest item.

topPtr = temp;

}

template <class ItemType> int Stack<ItemType>::length()

{

int length = 0;

//iterate through the stack object until it points to NULL

for(NodeType<ItemType>\* iter = topPtr; iter != NULL; iter = iter->next)

{

length++;

}

return length;

}

template <class ItemType> void Stack<ItemType>::Print()

{

//iterate through the stack object until it points to NULL

for(NodeType<ItemType>\* iter = topPtr; iter != NULL; iter = iter->next)

{

cout << iter->info << endl;

}

}

template <class ItemType> bool Stack<ItemType>::IsEmpty()

{

//returns true if empty, false if not

return topPtr == NULL;

}

template <class ItemType> bool Stack<ItemType>::IsFull()

{

if(this->length() == MAX\_ITEM)

return true;

return false;

}

template <class ItemType> void Stack<ItemType>::MakeEmpty()

{

for(NodeType<ItemType>\* temp = topPtr; topPtr != NULL; topPtr = temp)

{

temp = topPtr->next;

delete topPtr;

}

}

template <class ItemType> Stack<ItemType>::~Stack()

{

this->MakeEmpty();

}