**COVER PAGE**

Name: Maitra Patel (00984222)

Class: CIS 200 (Computer Science II)

Assignment: Project 2 -Part 1 - Stacks – Section 01 (Assigned – 3/7/2018)

Date: 3/18/2018

Semester: Winter 2018

1. **Problem Statement for Stacks**
   1. Problem 1
      1. Implement a Class Stack of elements with integer type. The stack must have makeEmpty, Push, Pop, Print, Length, IsEmpty, and IsFull functions along with the copy constructor. Max size of this stack should be 100 elements. The stack must be implemented using linked lists.
2. **Lab Status**
   1. Problem 1: The program works for any type of integer input into a stack. There are no issues with this program.
3. **Source Code:**

Compiler used for coding: Microsoft C/C++ -Visual Studio.

* 1. **Stacks:**
     1. **Source Code for header file of Stacks:**

/\*Author: Maitra Patel

\* Date Created: 3/10/2018

\* Date Modified: 3/18/2018

\* Purpose: Create a Stack of type int using linked list, and also have a copy constructer to copy one stack to another.

\*/

#ifndef STACK\_H

#define STACK\_H

struct NodeType

{

int info;

NodeType\* next;

};

class Stack

{

private:

int top;

NodeType \*head; // It points to the head of a singly-linked list

public:

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

Stack(Stack &x); // copy constructor

void MakeEmpty(); // Stack is made empty

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

bool IsFull(); // test if the stack is full; assume MAXITEM=100

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(int x); // insert x onto the stack

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

// Precondition: the stack is not empty

void AppendList(int data); //Function used solely for copy constructor..

int getTop(); //In order to print the top value..

~Stack(); // Destructor

};

#endif !STACK\_H

* + 1. **Source Code for implementation file of Stacks:**

/\*Author: Maitra Patel

\* Date Created: 3/10/2018

\* Date Modified: 3/18/2018

\* Purpose: Create a Stack of type int using linked list, and also have a copy constructer to copy one stack to another.

\*/

#include <iostream>

#include "Stack.h"

using namespace std;

//Constructor

Stack::Stack()

{

head = NULL;

top = 0;

}

//Copy constructor

//Pre-Condition: A stack to copy from

//Post-Condition: Returns the copied stack.

Stack::Stack(Stack &x)

{

//Initializes current stack first..

head = NULL;

top = 0;

//If the head of new stack is NULL, then make this stack NULL as well..

if (x.head == NULL)

{

this->head = NULL;

}

//Else, copy each element of new stack to this stack..

else

{

//Deep Copy

NodeType \*currPtr = x.head;

while (currPtr != NULL)

{

AppendList(currPtr->info);

top++;

currPtr = currPtr->next;

}

}

}

//Append (FUNCTION ONLY TO BE USED FOR COPY CONSTRUCTOR)

//Pre-Condition: A number of type int

//Post-Condition: Creates a new node with the value passed & links it with the end of list.

void Stack::AppendList(int data)

{

//Use temp pointer to iterate

NodeType \*tempPtr;

//If first node does not exist..

if (head == NULL)

{

head = new NodeType;

head->info = data;

head->next = NULL;

}

//else, find last node and create a new node after it..

else

{

//Find last node..

tempPtr = head;

while (tempPtr->next != NULL)

{

tempPtr = tempPtr->next;

}

//Create newNode..

tempPtr->next = new NodeType;

tempPtr->next->info = data;

tempPtr->next->next = NULL;

}

}

//MakeEmpty Function

//Post-Condition: Clears the whole stack

void Stack::MakeEmpty()

{

while (head != NULL)

{

Pop();

}

}

//IsEmpty Function

//Post-Condition: Returns if stack is empty or not

bool Stack::IsEmpty()

{

return (head == NULL);

}

//IsFull Function

//Post-Condition: Returns if stack is full or not.

bool Stack::IsFull()

{

return (length() == 100);

}

//Length of Stack Function

//Returns the number of elements in stack

int Stack::length()

{

NodeType \*tempPtr = head;

int countElements = 0;

while (tempPtr != NULL)

{

countElements++;

tempPtr = tempPtr->next;

}

return countElements;

}

//Print Function

//Post-Condition: Prints the stack

void Stack::Print()

{

if (IsEmpty())

{

cout << "Stack is empty!" << endl;

}

else

{

NodeType \*tempPtr = head;

while (tempPtr != NULL)

{

cout << tempPtr->info << endl;

tempPtr = tempPtr->next;

}

}

}

//Push Function

//Pre-Condition: An integer or a data point to be inserted into stack

//Post-Condition: Adds an element to top of stack.

void Stack::Push(int x)

{

if (IsFull())

{

cout << "Error: Cannot Push, stack is full" << endl;

}

else

{

NodeType \*location = new NodeType;

location->info = x;

location->next = head;

head = location;

top = head->info;

}

}

//Pop Function

//Post-Condition: Removes top element from stack

void Stack::Pop()

{

if (IsEmpty())

{

cout << "Error: Cannot Pop, stack is already empty" << endl;

}

else

{

//Delete Top Node

NodeType \*location = head;

if (head->next != NULL)

{

head = head->next;

top = head->info;

}

else

{

//Assign head to NULL

head = NULL;

top = NULL;

}

delete location;

}

}

//To print the top value

int Stack::getTop()

{

return top;

}

//Destructor

//Post-Condition: Deallocates the stack

Stack::~Stack()

{

if (head != NULL)

{

MakeEmpty();

}

delete head;

}

* + 1. **Source Code for main file of Stacks:**

/\*Author: Maitra Patel

\* Date Created: 3/10/2018

\* Date Modified: 3/18/2018

\* Purpose: Create a Stack of type int using linked list, and also have a copy constructer to copy one stack to another.

\*/

#include <iostream>

#include "Stack.h"

using namespace std;

int main()

{

Stack IntStack;

IntStack.Pop();

IntStack.Push(11);

IntStack.Push(22);

cout << "int length 1 = " << IntStack.length() << endl;

IntStack.Pop();

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);

if (IntStack.IsFull() == false)

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

else

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

Stack IntStack2(IntStack);

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

IntStack2.Print();

IntStack2.MakeEmpty();

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

IntStack2.Print();

system("pause");

return 0;

}

1. **Executable Module Instructions:**
   1. There are no specific module instructions required to run the code.
2. **Test Cases:**
3. **Stacks:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 1 | None | Message: Stack is empty | Message: Stack is empty | Pass |
| **Description:** To test the functionality of the pop function with an empty stack | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 2 | None  Note: Stack was pushed twice prior to printing length 1. Next, it was popped once & pushed once for length 2 | Message: Length 1 = 2;  Length 2 = 2; | Message: Length 1 = 2;  Length 2 = 2; | Pass |
| **Description:** To test the functionality of the length function for a filled stack | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 3 | None  Note: Prior to printing,  Push(11);  Push(12);  Pop();  Push(33); | Current Stack:  33  11 | Message:  33  11 | Pass |
| **Description:** To test the functionality of the push & pop functions for a filled stack | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 4 | None  Note: Function IsFull () was called. If Isfull is false, print Stack is not full. Else Stack is full | Message: Stack is not full | Message:  Stack is not full | Pass |
| **Description:** To test the functionality of the IsFull function for a partially filled stack. | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 5 | None  Note: stack 1 is passed into stack 2.  Stack 1 is:  66, 55, 44, 33, 11 | Stack 2: 66  55  44  33  11 | Stack 2:  66  55  44  33  11 | Pass |
| **Description:** To test the functionality of the copy constructor | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 6 | None  Note: Stack1 is made empty. | Stack is Empty! | Stack is Empty! | Pass |
| **Description:** To test the functionality of the MakeEmpty function | | | | |

1. **Running Logs/Outputs:**
2. **Running Log of Stacks in Screenshot format:**
   * **Test Case 1:**



The image above shows the Pop function’s result when called with an empty stack.

* + **Test Case 2:**



The image above shows the length function’s result after popping & pushing twice.

* + **Test Case 3:**



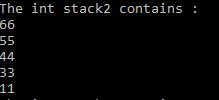
The image above shows the functionality of Pop and push functions in which 11, 12 were pushed, then the top element (12) is popped. Next 33 is pushed.

* + **Test Case 4:**



The above image shows the functionality of the IsFull function in which the condition is set to IsFull() == False, then print “Stack is not full”.

* + **Test Case 5:**



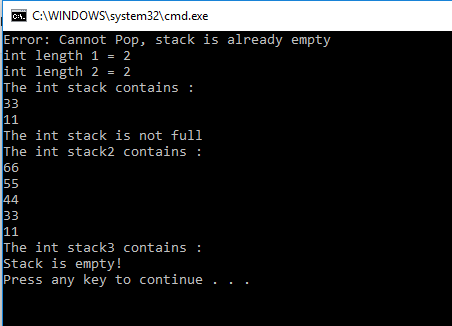
The above image shows the functionality of copy constructer in which stack 1 (elements: 66, 55, 44, 33, 11) is copied to stack 2. Next, stack 2 is printed.

* + **Test Case 6:**



The above image shows the functionality of the makeEmpty function in which stack 1 is made empty and printed as stack 3.

* + **Overall Program Printed based on main file:**



The above image shows the result of overall Stacks program based on the arguments in the main file.