|  |
| --- |
|  |
| Assignment 3 |
| CPTN278 |
|  |
| **Paul Fahey** |
| **2/12/2012** |

Table of Contents

[Introduction 3](#_Toc318285264)

[Application Summary 3](#_Toc318285265)

[Class Definition 3](#_Toc318285266)

[Class Variables 3](#_Toc318285267)

[Class Functions 3](#_Toc318285268)

[Code Analysis 4](#_Toc318285269)

[Assignment 3 Header File 4](#_Toc318285270)

[Assignment 3 Class Source File 4](#_Toc318285271)

[Application Variables 6](#_Toc318285272)

[Assignment 3 Application Source File 6](#_Toc318285273)

[Captured Program Output 8](#_Toc318285274)

[Application Conclusion 8](#_Toc318285275)

[Document Conclusion 9](#_Toc318285276)

[References 10](#_Toc318285277)

# Introduction

This document will provide a description on how to implement the stack data structure using an dynamically linked pointers in C++. A description will be provided on how to properly create a stack and pass its values to a completely separate stack. This document will as always demonstrate the proper way to create an application.

# Application Summary

The main purpose of this application is to input and create a stack of five integers and copy those integers onto another stack. This will be possible by using the permitted operations of stacks such as is full, is empty, push, and pop, to push items onto one stack then pop them off and push them onto a different stack. This program will create two stack objects to demonstrate the copying of integers from one stack to the other. The application will push five integers onto the first stack, pop and push onto the second stack and display the integers on the second stack.

# Class Definition

The section covers the variables and member functions of the stack class.

## Class Variables

|  |  |  |
| --- | --- | --- |
| **TYPE** | **NAME** | **Purpose** |
| node | top | Currently points to the newest memory block allocated |
| Integer | counter | Sets a fixed size for the stack |
| Integer | contents | Represents the data being pushed onto the stack |
| node | next | Represents the memory being allocated or de allocated |

## Class Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **NAME** | **Passes In** | **Purpose** |
| CONSTRUCTOR | Stack | Void | Instantiates and sets all values to 0 |
| DESTRUCTOR | Stack | Void | Just announces itself in this application |
| Void | Push | Integer | Pushes an Integer onto the stack |
| Integer | Pop | Void | Pops an Integer off of the stack |
| Boolean | is\_empty | Void | see if the stack is empty |
| Boolean | Is\_full | Void | see if the stack is full |

# Code Analysis

This section will cover the pseudo code for implementing a stack class using an array and passing it to another stack.

## Assignment 3 Header File

CREATE a class called node

PUBLIC MEMBERS

CREATE pointer called *next* as node

CREATE *contents* as integer

CREATE a class called stack

PRIVATE MEMBERS

CREATE pointer called *top* as node

CREATE *counter* as integer

PUBLIC MEMBERS

CREATE a constructor called **stack**

CREATE a destructor called **stack**

CREATE a function prototype called **push**

CREATE a function prototype called **pop**

CREATE a function prototype called **is\_empty**

CREATE a function prototype called **is\_full**

END Header

## Assignment 3 Class Source File

INCLUDE the Stack Header File

Constructor **Stack**

PASS IN Nothing

PRINT Announcement Message

INIT *top* to zero

INIT *counter* to zero

PASS OUT Nothing

END **Constructor**

Destructor **Stack**

PASS IN Nothing

PRINT Announcement Message

WHILE **is\_empty** is false

CALL **pop**

PASS OUT Nothing

END **Destructor**

Function **push**

PASS IN an integer called *entry*

CREATE *item* as node set to new node memory allocation

*item* -> *next* set to *top*

*item* -> *contents* set to *entry*

SET *top* to *item*

INCREMENT *counter*

PASS OUT Nothing

END **push**

Function **pop**

PASS IN nothing

CREATE *i* as integer

CREATE *item* as node

INIT *i* to *top* -> *contents*

SET *item* to *top*

SET *top* to *item* -> *next*

DELETE *item* memory allocation

DECREMENT *counter*

PASS OUT *i*

END **pop**

Function **is\_empty**

PASS IN Nothing

IF *top* == 0

PASS OUT true

ELSE

PASS OUT false

END **is\_empty**

Function **is\_full**

PASS IN Nothing

IF *counter* == *5*

PASS OUT true

ELSE

PASS OUT false

END **is\_full**

## Application Variables

|  |  |  |
| --- | --- | --- |
| **TYPE** | **NAME** | **USE** |
| Integer | entry | Stands as the value going onto the stacks |
| Stack | first | Holds the initial entries |
| Stack | second | Recieves the entries |

## Assignment 3 Application Source File

INCLUDE stack header file

PRINT welcome message

CREATE a stack called *first*

CREATE a stack called *second*

PRINT checker message for first stack

CALL **is\_empty** for first stack

PRINT checker message for first stack

CALL **is\_full** for first stack

PRINT checker message for second stack

CALL **is\_empty** for second stack

PRINT checker message for second stack

CALL **is\_full** for second stack

PRINT prompt integer message

DECLARE *entry* integer

WHILE CALL **is\_full** for first stack is false

INPUT five integers as *entry*

CALL **push** for *first* stack

PASS IN *entry*

END WHILE

PRINT checker message for first stack

CALL **is\_empty** for first stack

PRINT checker message for first stack

CALL **is\_full** for first stack

WHILE

CALL is\_empty from first stack is false

SET *entry* as *first* stack calling **pop**

CALL **push** for *second* stack

PASS IN *entry* for *second* stack

PRINT *entry*

END WHILE

PRINT checker message for second stack

CALL **is\_empty** for second stack

PRINT checker message for second stack

CALL **is\_full** for second stack

PRINT closing message

# Captured Program Output

Welcome to assignment 3

Making a stack

Making a stack

First Stack empty? 1

First Stack Full? 0

Second Stack empty? 1

Second Stack Full? 0

Enter five integers (w/ a space after each entry): 1 2 3 4 5

First Stack empty? 0

First Stack Full? 1

The output stack consists of: 5 4 3 2 1

Second Stack empty? 0

Second Stack Full? 1

Thank you for using assignment 3

Removing Stack

Removing Stack

Press any key to continue . .

# Application Conclusion

Creating this program was about learning how to implement a stack data structure using linked pointers. The first step to really understand this concept was looking at pictures in the text book and the pictures from the in class demo. Once the pictures made sense to me I gave the coding a go. As I suspected, I hit many roadblocks and had trouble figuring out how to actually allocate and de allocate memory. Once I figured out how it was done it wasn't too bad from there. The push and pop functions were the only ones that played with memory usage so once the logic came together the rest of the program should work just like the array implementation with minimal changes. One issue I did realize was that I didn't have a state of fullness with this program anymore so I would just keep pushing data onto the stack since in this case a state of fullness would be running out of memory on my computer. I realized that I needed to implement a counter to give a state of fullness and allow the program to finish its operations. If there was a second version of this program I would probably want it to have more explicit error checking features for allocating and de allocating memory.

# Document Conclusion

|  |  |
| --- | --- |
| Introduction | Gave a summary of the document content |
| Application Overview | The application was implement a stack with linked pointers; add five integers to a stack and copy in reverse order to another stack |
| Class Variables | Summary of the variable data members used in the stack class |
| Class Functions | Summary of the member functions used in the stack class |
| Assignment 3 Header File | Pseudo code for the assignment 3 header file (function prototypes, data member creations) |
| Assignment 3 Class source File | Pseudo code for the assignment 3 class source file that initialized data members and defined member functions |
| Application Variables | Summary of the variables local to the application source file or main function |
| Assignment 3 Application source File | Pseudo code for the assignment 3 applications source file; pushing integers onto first stack, popping and pushing onto second stack; printed results |
| Captured Program Output | Showed the output of the program proving that the program compiled, linked, ran as intended and that all deliverables were met |
| Application Conclusion | Initial logic came out mostly well; memory allocation and de allocation for push and pop had some issues but were resolved, realized a counter was needed for a sense of fullness, and error checking for the memory allocation and de allocation process. |

# References

Bettle, H. (n.d.). Assignment 3 Description for CPTN278. 4.

Bettle, H. (n.d.). In Class Demo.

Kruse, R. (1999). *Data Structures And Program Design in C++.* Prentice Hall.

*PseudoCode Standard.* (n.d.). Retrieved February 8, 2012, from http://users.csc.calpoly