|  |
| --- |
|  |
| Assignment 2 |
| CPTN278 |
|  |
| **Paul Fahey** |
| **2/8/2012** |

Table of Contents

[Introduction 3](#_Toc316852019)

[System Information 3](#_Toc316852020)

[Hardware Model 3](#_Toc316852021)

[Operating System 3](#_Toc316852022)

[Software Development Environment 3](#_Toc316852023)

[Creating a Program 3](#_Toc316852024)

[Starting Visual Studio 3](#_Toc316852025)

[Entering Code From Scratch 3](#_Toc316852026)

[Compiling, Linking, and Running the Application 4](#_Toc316852027)

[Saving and Closing the Application 4](#_Toc316852028)

[Application Summary 4](#_Toc316852029)

[Class Definition 5](#_Toc316852030)

[Class Variables 5](#_Toc316852031)

[Class Functions 5](#_Toc316852032)

[Code Analysis 5](#_Toc316852033)

[Assignment 2 Header File 5](#_Toc316852034)

[Assignment 2 Class Source File 6](#_Toc316852035)

[Application Variables 8](#_Toc316852036)

[Assignment 2 Application Source File 8](#_Toc316852037)

[Captured Program Output 9](#_Toc316852038)

[Application Conclusion 10](#_Toc316852039)

[Document Conclusion 10](#_Toc316852040)

[References 11](#_Toc316852041)

# Introduction

This document will provide a description on how use an IDE to create a program and implement the stack data structure using an array 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.

# System Information

This section will give the hardware and software used to create this program.

## Hardware Model

Acer Extensa 5620-6419

## Operating System

Window 7 Ultimate

## Software Development Environment

Microsoft Visual Studio 2010

# Creating a Program

This section will cover the necessary steps to creating, compiling, linking, running, saving, and closing a program in Microsoft Visual Studio.

## Starting Visual Studio

1. First find an appropriate place to save your program (computer, jump drive, etc.).
2. Open up the Microsoft Visual Studio 2010 application.

## Entering Code From Scratch

1. Once the application starts up, on the upper left side of the window it will say "New Project" you will want to click this.
2. A window will pop up, and on the top left of the window it will say "Installed Templates" .
3. Underneath it click the section that says "Visual C++".
4. Once this is done, the section to the right will list the types of applications you can create.
5. Click the very first one at the top that says "Win32 Console Application".
6. After you clicked that section, look at the bottom of that window .
7. You will see three different sections entitled Name, Location, and Solution name.
8. Go to where Name is, and inside the box type the name of your application.
9. The name will automatically be added to the solution name section.
10. Once you have done this, go to location to enter in where the application will be saved.
11. Click "browse" to the right of the location input box to browse out to the location of where you want your application to be saved and hit "Select Folder".
12. Once you chose an appropriate location, hit "Ok" in the bottom right corner of the window.
13. A new window will pop up that will be blank this is supposed to happen.
14. Look to the right of the screen, and there will be a section entitled “Solution Explorer”.
15. You will see a list of folders.
16. Right click the folder that says “source files” then under “add” hit “new item”.
17. A new window will pop up and give you a list of file extensions you can choose for your application.
18. These extensions will be in the middle of the window, and you want to click the one that says “C++ File(.cpp)”.
19. Once you complete this, go to the bottom of the window where it says “name”.
20. In the box to the left, type in the name you want to call your application source file and then hit “add”.
21. You should now see the file under source files in the Solution Explorer.
22. You now have white space on your screen to start typing code.

## Compiling, Linking, and Running the Application

In Microsoft Visual Studio 2010, compiling, linking, and running is bundled in the ide’s “build" feature.

* To execute this “build" feature you hit “F5” and the ide will compile, link, and run the program and then it exits
* Hold Ctrl and F5 at the same time to not allow the program to exit after execution

## Saving and Closing the Application

1. To save your work, go to file and hit “save all” to update the files or do Ctrl+Shift+S.
2. Once doing this, you can hit the “x” at the top right of the screen to exit visual studio or you can go to file and click “exit”.

# 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, pop, and top 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** |
| Integer | top | Currently points to the top of the stack |
| Integer Array | array\_of\_ints | Contains the contents of the stack, indexed by *top* |
| Constant Integer | MAX\_SIZE | Sets a fixed size forthe stack |

## 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 | Checks to see if the stack is empty |
| Boolean | Is\_full | Void | Checks to 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 2 Header File

DEFINE a constant called MAX\_SIZE with Five

CREATE a class called stack

PRIVATE MEMBERS

CREATE an integer called *top*

CREATE an integer array called *array\_of\_ints*

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 2 Class Source File

INCLUDE the Stack Header File

Constructor **Stack**

PASS IN Nothing

PRINT Announcement Message

INIT *top* to zero

FOR

CREATE integer called *value*;

SET to zero;

WHILE *value* is < *MAX\_SIZE*;

INCREMENT *value*

INIT *array\_of\_ints* to zero

PASS OUT Nothing

END **Constructor**

Destructor **Stack**

PASS IN Nothing

PRINT Announcement Message

PASS OUT Nothing

END **Destructor**

Function **push**

PASS IN an integer called *number*

INCREMENT *top* in *array\_of\_ints*

SET *array\_of\_ints* to *number*

END **push**

Function **pop**

PASS IN nothing

CREATE integer *index* as *array\_of\_ints[top]*

DECREMENT *top*

PASS OUT *index*

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 *top* == *MAX\_SIZE*

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 2 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 2

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 2

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 an array. Initially what went well was the logic I created for the implementation of the program before beginning the coding stage. I started to encounter problems when I began writing the program. I had some trouble with copying the integers from the first stack to the second stack. At first, I was trying to pop onto the second stack without really calling a way to push the first stack on so what I realized I had to do was call the push method of the second stack and passing in a call to the first stack's pop method. In that sense what's popping off the first stack is directly being pushed right onto the second stack. There was nothing that ended up not working unless my implementation was not really the proper representation. I believe the techniques I used were as simple as I could make them to understand concepts the best, but I'm sure once I gain more knowledge of stacks I will find better techniques to use. If there was a second version of this program I would probably add some over and under flow error checking to make sure the size stays within the limits set for this application.

# Document Conclusion

|  |  |
| --- | --- |
| Introduction | Gave a summary of the document content |
| Application Overview | The application was implement a stack with an array; add five integers to a stack and copy in reverse order to another stack |
| Application Variables & Objects | Summary of the variables and objects used with their name, type, initial value, and purpose |
| Assignment 2 Header File | Pseudo code for the assignment 2 header file (function prototypes, data member creations) |
| Assignment 2 Class source File | Pseudo code for the assignment 2 class source file that initialized data members and defined member functions |
| Assignment 2 Application source File | Pseudo code for the assignment 2 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; copying from one stack to the other had some issues, and over and under flow error checking could be useful in second version |

# References

Bettle, H. (n.d.). Assignment 2 Description for CPTN278. 10.

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