|  |
| --- |
|  |
| Assignment 1 |
| CPTN230 |
|  |
| **Paul Fahey** |
| **9/16/2011** |

Table of Contents

[Introduction 3](#_Toc303803047)

[Operating System and IDE Information 3](#_Toc303803048)

[Software Development Environment 3](#_Toc303803050)

[Entering Code From Scratch 3](#_Toc303803051)

[Importing Code From Another Source 4](#_Toc303803052)

[Compiling, Linking, and Running the Application 4](#_Toc303803053)

[Saving the Application 4](#_Toc303803054)

[Application Summary 4](#_Toc303803055)

[Variable and Object Functionality 4](#_Toc303803056)

[Code Analysis 5](#_Toc303803057)

[Assignment 1 Header File: 5](#_Toc303803058)

[Assignment 1 Application Source File: 6](#_Toc303803059)

[Conclusion 9](#_Toc303803060)

[Works Cited 10](#_Toc303803061)

# Introduction

This Document will give a complete analysis of a C++ program. I will be using the Assignment 1 description document as my primary source to completing this assignment. The assignment document will successfully allow me to reverse engineer a C++ application where the code is already written out. (Bettle)

# Operating System and IDE Information

# Windows 7 Ultimate Service Pack 1

* Microsoft Visual Studio 2010

# Software Development Environment

## Entering Code From Scratch

* First find an appropriate place to save your program (computer, jump drive, etc.)
* Open up the Microsoft Visual Studio 2010 application
* Once the application starts up, on the upper left side of the window it will say "New Project" you will want to click this
* A window will pop up, and on the top left of the window it will say "Installed Templates"
* Underneath it click the section that says "Visual C++"
* Once this is done, the section to the right will list the types of applications you can create
* Click the very first one at the top that says "Win32 Console Application"
* After you clicked that section, look at the bottom of that window
* You will see three different sections entitled Name, Location, and Solution name
* Go to where Name is, and inside the box type the name of your application
* The name will automatically be added to the solution name section
* Once you have done this, go to location to enter in where the application will be saved
* 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"
* Once you chose an appropriate location, hit "Ok" in the bottom right corner of the window
* A new window will pop up that will be blank this is supposed to happen
* Look to the right of the screen, and there will be a section entitled “Solution Explorer”
* You will see a list of folders
* Right click the folder that says “source files” then under “add” hit “new item”
* A new window will pop up and give you a list of file extensions you can choose for your application
* These extensions will be in the middle of the window, and you want to click the one that says “C++ File(.cpp)”
* Once you complete this, go to the bottom of the window where it says “name”
* In the box to the left, type in the name you want to call your application source file and then hit “add”
* You should now see the file under source files in the Solution Explorer
* You now have white space on your screen to start typing code

## Importing Code From Another Source

* Follow the previous section's steps all the way up to locating the Solution Explorer
* You will see folders listed and you want to right click the one that says “source files” then under “add” hit “existing item”
* A new window will pop up, and you want to navigate out to where you keep your source files
* Click the source file (.cpp), and then hit “add” (Do this for your class and application source files)
* This whole procedure can be used for adding the header file (.h), but instead you need to right click “header file" in the Solution Explorer
* You should now see the files you imported under source header files folders respectively in the Solution Explorer
* Click on each file to open them to access the 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 the Application

* To save your work, go to file and hit “save all” to update the files or do Ctrl+Shift+S
* 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 application starts by welcoming you to the program, and prompting the user for a number. This process is repeated four more times for a total of five numeric entries. The program will then list the five values you just entered. Furthermore, it will tell you what the third entry was, and give a sum and average of all the numbers. Depending on how large the sum was would determine the number of digits displayed for the average. If there was very large numbers entered, the program would show a weird error or number back at you. Lastly, a message will appear telling you that you successfully used the program, and there is no more number entering.

# Variable and Object Functionality

Line 20:

* **j** is an integer variable that is assigned the initial value of the constant **MAX\_SIZE** that tells which number entry you are on

Line 21:

* **k** is also an integer that is assigned the initial value of 0 which represents the actual number entry itself

Line 23:

* An object is created called **my\_numbers** for the **Numbers** class that has data members of lines 17-30 of the class header file analysis as well as the initial values of each data member

Line 25:

* A pointer object called **the\_numbers** is created that is assigned the address using **&** of the **my\_numbers** object for means of giving the address of **my\_numbers** object when needed

Line 16:

* A function prototype is declared called **calculate\_results** as an integer that will take a pointer of the **Numbers** class for its usage in the main function

# Code Analysis

## Assignment 1 Header File:

Line 9:

* This line introduces the **iostream**  "is an object-oriented library that provides input and output functionality using streams" (Iostream Library)
* The **#include**  "tells the preprocessor to treat the contents of a specified file as if those contents had appeared in the source program at the point where the directive appears" (The #include Directive)

Line 10:

* This line introduces the **iomanip** library, which contains manipulator functions to change the state of data used in the program in conjunction with the iostream library

Line 11:

* On this line, it is defining the usage of the standard library of functions, objects, and classes in C++ so you don't have to define **using namespace** **std::** before each function, object, or class you want to use in the **iostream** library (Deitel & Deitel, 2010)

Line 13:

* The **#define** allows the identifier **MAX\_SIZE** be assigned a constant value of 5 (Deitel & Deitel, 2010)
* This allows **MAX\_SIZE** to be used in the program instead of 5

Line 15:

* A **class** tells the compiler what data members and member functions will be used to complete a similar task (Deitel & Deitel, 2010)
* The **class** in this program is called **Numbers**

Line 17:

* The keyword **private** will list all of the variables and data members that cant only be used by the member functions of the **Numbers** class (Deitel & Deitel, 2010)

Line 19:

* An integer array is declared that has a size of the constant **MAX\_SIZE**

Line 20:

* An integer variable named **total** is declared that hasn't been assigned a value yet

Line 21:

* A double variable is declared called **average** that also hasn't been assigned a value yet

Line 23:

* The keyword **public** will list all the data members and functions that can be used by all other functions or classes in the program (Deitel & Deitel, 2010)

Line 25:

* A constructor for the program is declared called **Numbers** that doesn't have any parameters

Line 26:

* A function called **Display\_the\_numbers** is declared that will not return any values or take in any values
* At this stage it is referred to as a **function prototype** along with any following functions listed since it doesn't tell you what the functions do yet

Line 27:

* A function called **Set\_an\_int** is declared that will not return any values and will take in two integer values

Line 28:

* A function called **Get\_an\_int** is declared that will return a value and taking in an integer value

Line 29:

* A function called **Do\_the\_math** is declared that will not return a value and not take in any values

Line 30:

* A function called **Display\_the\_results** is declared that will not return a value and will not take in any values

Line 31:

* A closing brace is placed in with a semi colon after it signaling the end of the class definition

## Assignment 1 Application Source File:

Lines 10-12:

* Refer to lines 9-11 on the Assignment 1 Header File

Line 14:

* This line allows the header and class file to be used in the application file by typing the header file in quotes after the **#include** (Deitel & Deitel, 2010)

Line 18:

* Main function starts

Line 27:

* A welcome message is outputted to the screen for the program with a **\n** at the end of the message followed by **endl**
* The **endl** is an object that flushes the rest of the stream buffer and starts a new line (Iostream Library)
* **\n** is a character that doesn't flush the stream and just goes to a new line (Deitel & Deitel, 2010)
* **<<** operator inserts the data that follows it into the stream preceding it (Deitel & Deitel, 2010)

Line 29-32:

* A message prompts you of how many integers you will be entering using the **j** variable

Line 33:

* The rest of the message that accompanies lines 29-32
* The **\** character is used to allow quotations be printed to the screen with a string of characters (Deitel & Deitel, 2010)

Line 36:

* The value of **j** is changed to zero

Lines 37:

* A loop is created to allow five number entries then terminates
* A while loop is used that iterates **j** up to **MAX\_SIZE**

Line 39:

* A message prompts the current number entry
* **flush** "synchronizes the buffer so all unwritten characters in the buffer are written to its controlled output sequence as soon as possible" (Iostream Library)

Line 40:

* **k** is entered

Line 41:

* **my\_numbers** activates function **Set\_an\_int**
* The **.** is used to access data members and member functions of a class using an object (Deitel & Deitel, 2010)
* **Set\_an\_int** was activated by the application source file on line 41 and is returned from the class source file on line 39
* Local variable **Location** is created as a parameter of the function that is passed by reference
* Location refers to **MAX\_SIZE** in private data member **nums** in its sizing operator
* Local variable **value** is created as a parameter of the function that is passed by reference
* **Value** has the same value as **Location**, but refers to the number being entered instead of the current number entry
* The function returns and **Location** and **Value** are erased
* **return** terminates the execution of a function and returns control to the calling function or operating system (Deitel & Deitel, 2010)

Line 42:

* **j** is incremented and the loop iterates four more entries then the loop exits

Line 47:

* **my\_numbers** activates function **Display\_the\_numbers**
* **Display\_the\_numbers** was activated by the application source file on line 47 and is returned from the class source file on line 28
* Local Variable **k** is created in a for loop
* The loop will display the current entry number and entry itself
* The for loop is used that iterates **k** up to **MAX\_SIZE**
* **k** acts as the current entry number
* Private data member **nums[k]** is used as the entry itself using **k** in the case to represent the entry being displayed
* **'.'** displays that character to the screen which **" "** also does
* The loop iterates four more entries then exits
* **k** is erased on the return

Line 49:

* A message prompts that will display the third entry

Line 50:

* **my\_numbers** activates function **Get\_an\_int**
* **Get\_an\_int** was activated by the application source file on line 50 and is returned from the class source file on line 33
* Local variable **Location** is passed from **Set\_an\_int** and is passed as a reference
* **Location** refers to **MAX\_SIZE** in private data member **nums** in its sizing operator
* **nums** is then returned to the application source file
* Third entry is specified and displayed

Line 55:

* A conditional If statement is created that uses function **calculate\_results**
* **calculate\_results** was activated by the application's source file on line 55 and is returned from the application source file on line 75
* **the\_numbers** pointer is used as a parameter of the function and is passed by reference
* Local variable and pointer **generate\_results** is created as a parameter of the function and is passed by reference
* **generate\_results** then points to the **Do\_the\_math** member function using **->**
* one argument is returned

Line 75:

* **Do\_the\_math** was activated by the application source file on line 75 and is returned from the class source file on line 50
* Data member **total** is initialized to zero
* A for loop is created that will generate a total of all the entries
* Local variable **k** is created in the loop that is iterated to **MAX\_SIZE**
* **total** is populated by adding up the previous entry to the current one using **nums[k]**
* Loop starts from the top again and adds on four more entries to **total** then exits
* Data member **average** is initialized and coverts **total** divided by **MAX\_SIZE** into a double since **average** is a double
* **k** is erased on the return

Line 57:

* **my\_numbers** activates **display\_the\_results**
* **display\_the\_results** was activated by the application source file on line 57 and is returned from the class source file on line 73
* A message is printed about the total along with the variable **total**
* Two If followed by two else statements are created that set the number **precision** of the average
* The precision ranges from 7 numbers to 3 numbers
* **showpoint** "shows  the decimal point written for floating point values inserted into the stream, even for whole numbers and following the decimal point, as many digits as necessary are written to match the **precision**internal setting for the stream" (Iostream Library)
* The **average** variable is displayed with **precision**
* The function returns and then exits
* If conditional ends

Line 59:

* A conditional else statement is created in conjunction with the If conditional statement from line 55

Line 61-62:

* An error message is displayed if the If conditional statement could not be executed
* The conditional is then terminated and goes onto the next line of code
* **Exit** terminates the process

Line 67-69:

* A message is displayed that signals the end of the program and no more arguments are returned signaling the end of the main function

# Conclusion

After documenting this code, I realized that it was hard to get a good grasp on terminology used in each file, which made it a little more difficult to document the code. First off, I thought that many of the variable names were too vague, and should have been more specific to understand the code better. Also, pointers seemed to jump around too much, which made it hard to follow their path, and get a good grasp on their main function. Return statements in some void functions I thought were unnecessary especially since void functions don't return values anyways. I think the function that was created in main was unnecessary, and that main should have just made calls to the member functions of the class. I think the application could have added some more error checking statements for when the user doesn't follow program directions. I thought the code to generate precision was confusing at first because it looked like it was giving precision to one variable, but it really was giving it to a different variable. Despite all of these issues I addressed, I thought this program was a good example of how to pin point terminology used, errors, and see how the code is structured. A simpler structure probably could have been developed to execute the program's main purpose, but the structure documented was a good example of concluding that there can always be something improved in a set of code to make it cleaner and clearer.

# Works Cited

Bettle, H. (n.d.). Assignment 1 for CPTN230 Description Document. 14.

Deitel, P., & Deitel, H. (2010). *C++ How To Program.* Upper Saddle River: Pearson Education Inc.

*Iostream Library*. (n.d.). Retrieved September 14, 2011, from cplusplus.com: http://www.cplusplus.com/

*The #include Directive*. (n.d.). Retrieved September 14, 2011, from msdn: http://msdn.microsoft.com