ECE 3220 Lab 8 Report

John Walter

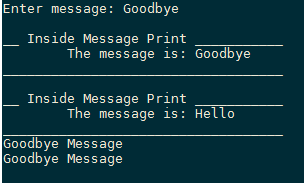
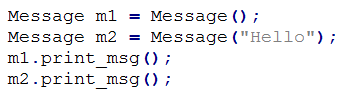
November 1, 2016

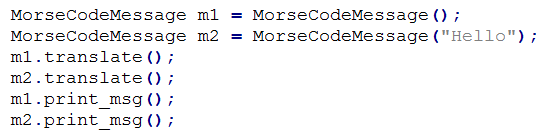
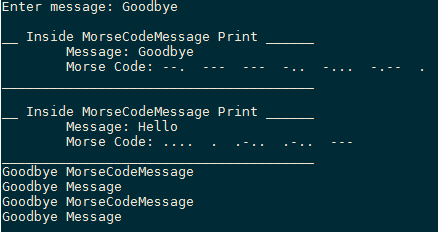
**Objective**

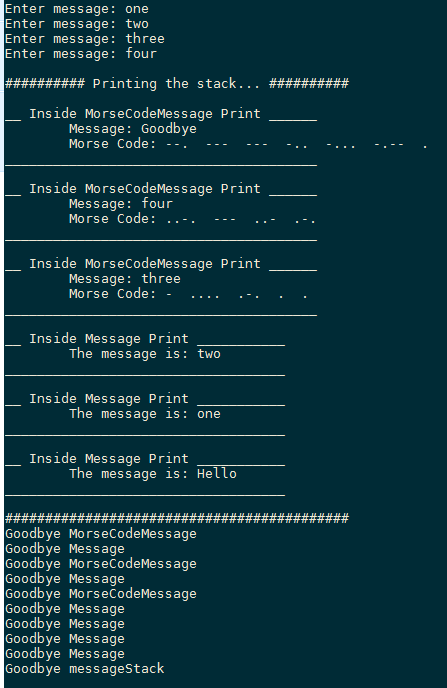
The objected of this lab was to create a program that would convert a word into its equivalent Morse Code Representation. This process involved the use of classes for Message, MorseCodeMessage, and a messageStack. This lab also dealt with the concept behind inheritance in object oriented programming in the form of having a base class and then derived classes built upon it. Another interesting topic reviewed in this lab was dynamic binding in which objects of the base class are applicable to objects of the derived classes.

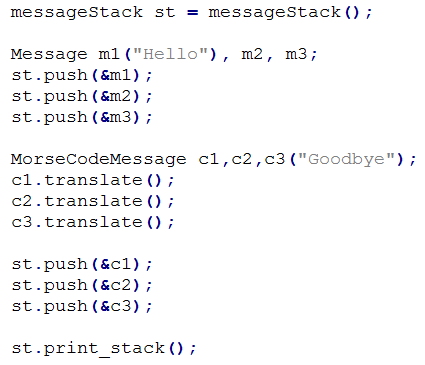
**Results**

The first part of this lab involved constructing a Message class that would act as the base class and would take in a message or prompt the user to enter a message. This class also defined a virtual print\_msg() method that would be redefined in other classes.

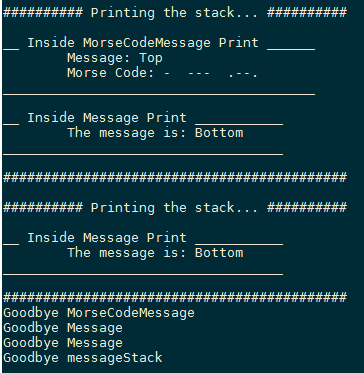
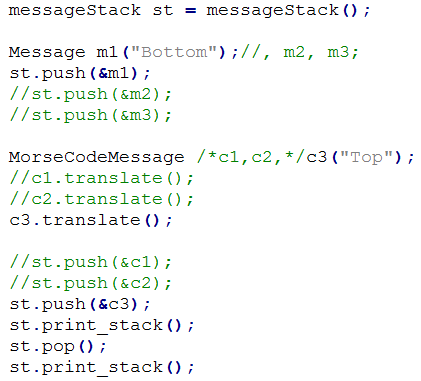


 The next part of this lab involved constructing a MoreCodeMessage class which is a derived class of the Message class. This class can take in either northing or a string. This class contains a public method called translate() that converts the message into its Morse Code representation, and a redefined print\_msg() to print out the string and its Morse Code form.

The final part of this lab was to construct a messageStack class that would allow Message objects and MorseCodeObjects object to be pushed onto. This class takes in nothing and defines an array of pointers (the stack) with each pointer being NULL. This class contained a push() method to push the object onto the “stack” and the pop() method to pop from the stack. This class also contained a print\_stack() method to print out the objects on the stack based on dynamic binding. If the object is of class Message then the Message print\_msg() method is called, or If the object is of class MorseCodeMessage then the MorseCodeMessage print\_msg() is called.



If we pop() from the stack then the top object in the stack is removed.



Notice that when an object is popped from the stack, it is still not deleted until the end of the program. This is due to the objects never being physically allocated memory in the code, and are therefore still managed by the compiler.

**Discussion**

One of the issues that was encountered was the dynamic binding. Previously, I tried passing a Message object to the push() method and then putting the address of that into the stack. While this worked, it would only call the Message print\_msg() to display the message. In order to address this issue, dynamic binding had to me implemented in main(). To do this, instead of passing the object itself, the pointer to that object had to be passed to the push() method in order for the compiler to keep track of what print\_msg() to use. Once that was done, then the correct print\_msg() functions were called correctly.

Another interesting thing that was encountered was during the conversion of the message to Morse Code. Instead of using arrays to achieve this, I implemented the string class. As each character was being converted to its Morse Code representation, I would add that converted character to a string called “transformed.” Because we are dealing with strings, this was simply done by using the operator (+), and then a (#) was added right after each character to represent a space. Once the whole message was converted, I added (+) on a ($) to indicated the end of the message. Then, when transformed was to be printed, I would look for each (#) replacing them with a space until the end of the message ($) was reached.