CCAC

CIT245

Sam Schock

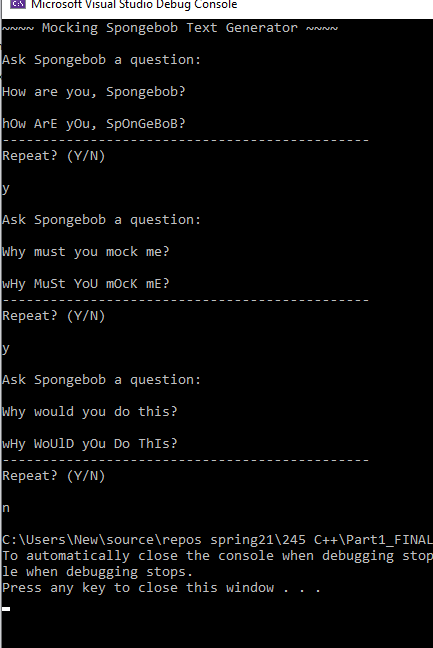
Due 5/13/21

Part 1, Strings: Mocking Spongebob Text Generator

This code takes an input line of text and converts it to ‘alternating caps’ – each successive character alternates between lowercase and uppercase, leaving non-letter characters unchanged. It is inspired by the ‘Mocking Spongebob’ meme.

The void function at the heart of the code is called sPoNgEbOb(). This function reads in a line of text and stores it in the string variable, Input. It then uses a for-loop to build the Output string, one character at a time. For each character in Input, two tests are run. First, the character is tested to see if it is a letter. If the character is not a letter (i.e. whitespace or punctuation), it is appended to Output unchanged. If it is a letter, the second test is run. In this test, a Boolean variable called makeLower (initialized to true) is checked. If makeLower is true, the character is made lowercase and appended to Output.; makeLower is then changed to false. If makeLower is false, the characteris made uppercase and appended to Output; makeLower is then changed to true. Once the for-loop terminates, the function outputs the string Output to the console.

The main function encloses the sPoNgEbOb() function in a do-while loop, which executes the function and then prompts the user to enter Y to repeat or N to exit. In fact, entering anything other than an ‘N’ or ‘n’ will cause the program to repeat, due to the makeup of the control structure. In addition, a piece of code from the textbook is used to clear the input line. The function ‘newLine()’ is taken from page 391 of the textbook.



Part 2, Recursion: Recursive Function Demonstrator

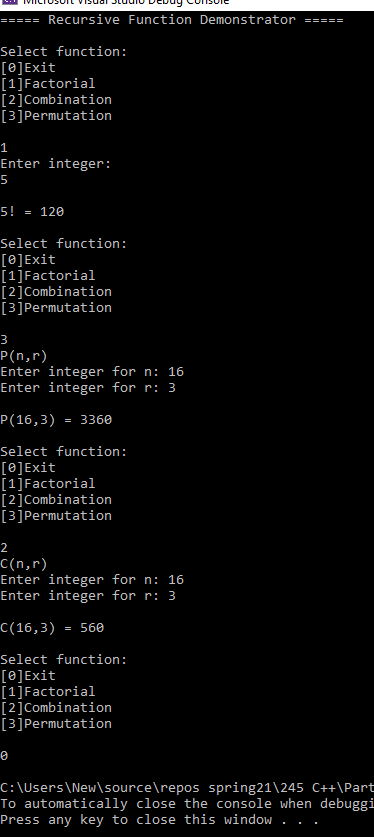
This code uses recursion to calculate factorials, and by extension, combinations & permutations. A factorial is a function, n!, equal to the product of all positive integers less than or equal to n. Combinations and permutations are math functions useful in the prediction of probabilities. Their equations are, respectively:

To start, I coded three functions. The first was the recursive factorial function. Very simply, for a given integer input, it is equal to the product of itself and the factorial of one minus itself.

n! = n \* (n-1)!

The function takes an integer input and tests it to make sure it is greater than 0. If it is not, it returns a value of 1. If it is, it returns [n \* factorial(n-1)]. I encountered problems getting this to work on my first attempt, as the upper limit of the standard integer data type in C++ is 2^31 (that is, 4 bytes, with one bit for the sign). In order to handle the calculations for combination and permutation functions, I declared all three functions as unsigned long long integers. This allows for an upper value of 2^64, which still causes problems for n or r values above 20, but at least allows for some lower level calculations to be performed. With that problem solved, coding the combination and permutation functions is as simple as writing out the equations.

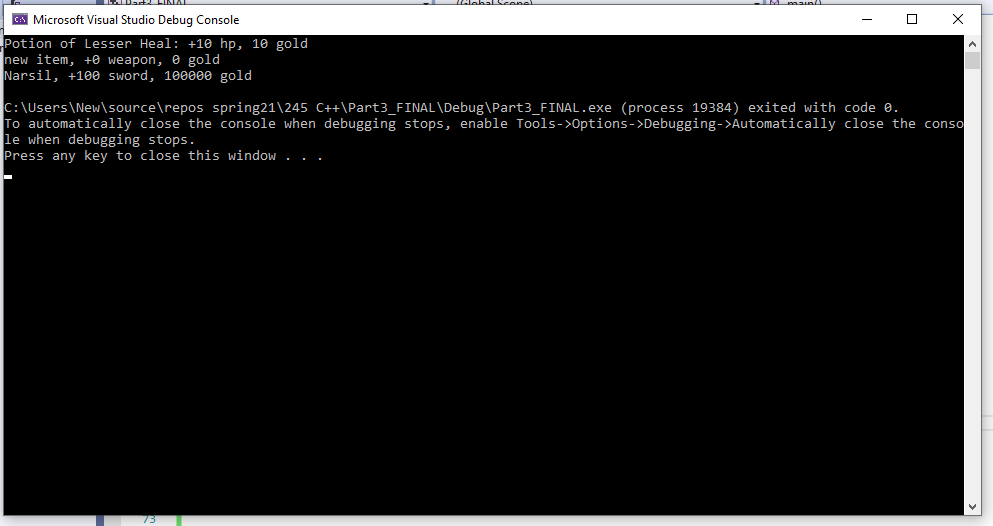
The main function is contained within a do-while loop. The user is prompted to enter a number that corresponds with one of the three functions. After selection, they are prompted to enter one or two integers. The function is then executed, and the output is formatted to make it easier to read. The function then loops over, and continues until the user enters 0 to exit the loop.



Part 3, Inheritance: Adventure Game Items

In this program, inheritance is used to build a prototype item system for a fantasy adventure game. The parent class ‘Item’ is rather generic, with its only member values being name and cost. This parent class is used to create two parallel child classes: ‘Potion’ and ‘Weapon.’ In fantasy role playing games, both potions and weapons may be held in a character’s pack, as they are both types of items – and so it is here! Potions get an integer representing healing strength, and Weapons get an attack modifier integer and a weapon type string. For all three classes, default & overloaded constructors, as well as setters & getters, are coded. In addition, Potion and Weapon have toString() functions that output their member values as formatted string.

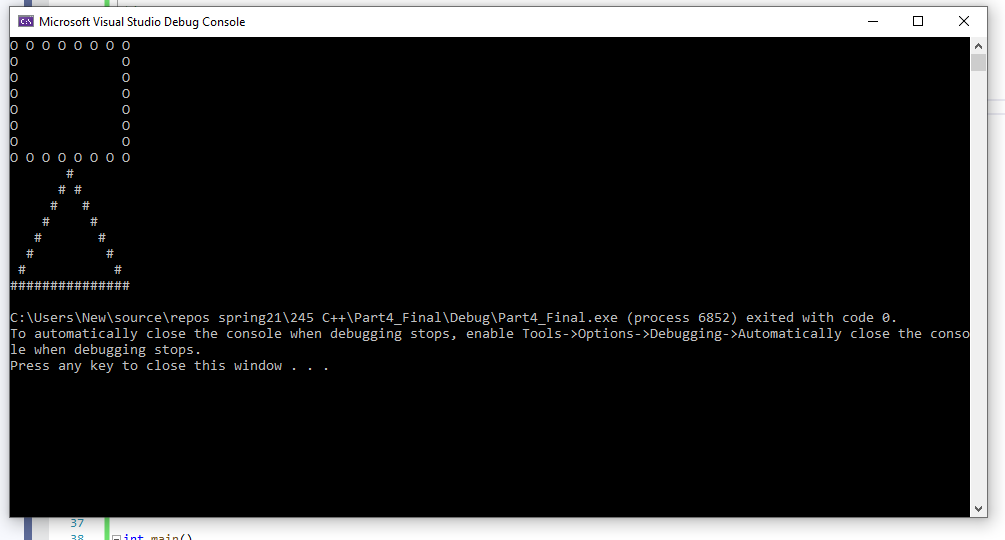
In the main method, a Potion object is initialized and its overloaded constructor called. A Weapon is also initialized, but with the default constructor. Both then have their toString() methods called and output to console. As can be seen, the data entered into the lesserHeal Potion’s constructor is output and nicely formatted; however, the weapon Narsil’s string is generic, as all values are still set to default. To remedy this, setter functions are called to fill in Narsil’s data. The toString() is then called again, showing the newly input data. Frankly, 100,000 gold pieces for the mighty Narsil is a steal!



Part 4, Polymorphism: Draw Shapes

This program uses overriding class methods to draw shapes using characters and whitespace. Three classes were made: a parent class called Shape, and two descendent classes called Square and Triangle. Shape has two member values – the integer size, and the char printChar, which determines which character is used to print the shape. Shape also has a pure virtual function for draw(). The only distinguishing feature of Square and Triangle is their distinct draw() functions, which override the virtual default. Each draw() uses nested for-loops and if-else control structures, along with some mathematical black magic, to determine whether to place a whitespace or character at each position. The specifics of each procedure are unique to each shape. In addition, extra white space was added to the Square.draw() function to make the image appear more square.

In the main function, an instance of Square and an instance of Triangle are each called and defined with unique size ad printChar values. Their respective draw() functions are then called. The output sent to the console shows the resulting shapes – a square drawn from O’s and a triangle drawn from #’s!



References

Savitch, W., & Mock, K. (2016). Absolute C++. Boston: Pearson.