Homework #10 – The Postal Service

DUE: December 3 by 11:59:59 PM **Assigned**: November 19

Background

IMPORTANT NOTE: There are NO late submissions for this assignment.

From 1965 – 2013, the US Postal Service used a bar code on every envelope representing the zip code (The ZIP code was introduced in 1963) using a format called POSTNET. A bar code is still used, but POSTNET was retired completely in 2013. We will be doing the same with 5-digit zip codes. POSTNET consists of long and short lines or bars, as seen below:

Figure 1 - The POSTNET representation of 67260, WSU's zip code

In the program, the zip code will be placed into a class. See the included Zipcode.hpp to see how the class should be defined & implemented. For the bar code representation, a string will be used where the character '1' represents a long bar and the character '0' represents the short bar.

A POSTNET code consists of 32 bars/digits. The first and last bars are always 1. Stripping these leaves 30 digits, which can be split into groups of 5.

```
10110010001001010110011000101001
01100 10001 00101 01100 11000 10100
```

Now, we look at the first five groups of 5. There will always be two 1's. Depending on its location within the group, each 1 represents a number. When the numbers that the 1's represent are added together, you get that digit of the zip code. The table below translates the underlined group, which represents the number 6.

POSTNET Digits	0	1	1	0	0
Value	7	4	2	1	0

We see that the 1's correspond to the values of 4 and 2, respectively. Adding them up gives us 6, which is the first digit of the zip code (and also the fourth since the same group appears again, due to the zip code having two 6's).

In order to represent the number 0, the 1's will add up to a value of 11. This is done because of the requirement that every group of five must have two 1's in it.

The sixth group is known as the "check digit." This is included so that sorting machines can catch reading errors. The check digit is calculated by summing the 5 digits of the zip code and adding the check digit such that the total sum is a multiple of 10. In the example above:

```
6 + 7 + 2 + 6 + 0 = 21
  10 - (21 % 10) = 9
```

The check digit for the zip code 67260 is 9, meaning all six digits sum to 30. If the sum of the 5 digits is already a multiple of 10, the check digit is 0.

Assignment Requirements

- You will only submit the file `Zipcode.hpp` (Fill out the file overview comment)
- Download the files: `Zipcode.hpp`, `doctest.h`, and `main.cpp` that are attached to this assignment on Blackboard
- The file `Zipcode.hpp` contains a class declaration. You must write the implementations in this file
    - See Hints for extra information
- You may declare and use additional functions if needed
- You may assume that any 5 digit number is a valid zip code

Sample Run

```
[doctest] doctest version is "2.4.1"
[doctest] run with "--help" for options
===============================================================================
=======
[doctest] test cases:  8 |  8 passed | 0 failed | 0 skipped
[doctest] assertions: 15 | 15 passed | 0 failed |
[doctest] Status: SUCCESS!
```

Hints

- Ignore the contents of doctest.h, it is the testing framework; there is no information in it that will aid in the completion of the assignment
    - Naturally, if you find yourself with the time, it might be worth checking out the repository at https://github.com/onqtam/doctest
- You do not need to write the entire class before testing
    - You can either stub functions you have not written yet, or comment out those tests
- Look at the tests to know what is expected of the class
    - Corollary: The comments in Zipcode.h are also there to help you
    - The tests for print_barcode() will prove very helpful for that function

- After successfully compiling the first time, you'll see that the program comes with options (Second line of the Sample Run). A couple of note:
  - `-s` Shows verbose output for successful tests
  - `-nc` Will not print using color
- The required functions must be implemented, but there is nothing stopping you from adding private helper functions and data to the class
  - As an example, my implementation contains one other data member and four private functions on top of what's required

Reminders

- Be sure to include a comment block at the top of every file with the required information
  - Refer to the General Homework Requirements handout on Blackboard
- Provide meaningful comments
  - If you think a comment is redundant, it probably is
  - If you think a comment is helpful, it probably is
  - Remember that you are writing comments for other programmers, not people who know nothing (obligatory Jon Snow) about coding
  - Comments are more helpful when they explain why, not what or how
- There will be no extensions

Preparing and Submitting

- Your code must be able to compile and run on the EECS lab machines
  - You are responsible for testing your code
  - "But it runs fine on my machine!" will **not** earn you any points
- Submit **ONLY** your source code file
- Homework submission will be handled exclusively through Blackboard