## CPSC-131 Data Structures

## Project 5: Organizing Minnie's bows

Minnie has lots of bows because every days she wants to wear a different one. She likes to buy new bows and donate the ones that are not in trend anymore. Bows are of different color, shape, and texture, and they all have a different product number which is a 5 digit number. She used to keep all her bows on a table, handy, but she has now too many bows, and they all end up being piled up on that table.

Daisy, Minnie’s friend, decided to help Minnie organize her bows, so she asked Mickey, another friend, to build a tall shelf with multiple vertical cubbies. Minnie tried to keep all the bows of the same color in the same cubby, but because she had too many bows of the pink and purple color, then the cubby for pink and purple was too full while the cubbies for black and grey had very few bows, because those were for special occasions.

Daisy came up with an idea, of storing the bows based on a single digit of their product number. Based on the existing large set of bows, they tried to organize based on the first digit, second digit, .., fifth (and last digit). (The digits are read from left to right; e.g. for the number 89765, the first digit is 8, the second is 9, the third is 7, the fourth is 6, and the fifth is 5.)

They settled on the digit that give them the best “balancing” of storing the bows, i.e., the difference between the number of bows in the most populated cubby and the least populated cubby is minimized among all possible five options.

Your project is about reading a large number of distinct product numbers (5 digit each) and deciding which digit among the five gives the best balanced storage of the bows.

For example, let’s assume that we have the following product numbers:

12345, 23456, 34567, 45678, 56789, 67890, 78901, 89012, 90123,

54321, 65432, 76543, 87654, 98765, 10987, 21098, 32109, 43210

If we choose storing the bows based on the first digit that we have:

Cubby 0: no bow

Cubby 1: 12345, 10987

Cubby 2: 23456, 21098

….

Cubby 9: 90123, 98765

The difference in the cubbies’ load is 2.

If we choose storing based on the second digit:

Cubby 0: 90123, 10987

Cubby 1: 21098

Cubby 2: 12345,

…

Cubby 9: 89012,

The difference in the cubbies’ load is 1 so clearly this is a more balanced organization of the bows.

### The Code

The code to decide which digit leads to the most balanced hashtable is to be implemented in class BowCollection. Since you will be comparing 5 hashtables (which differ only in their hash function), class BowCollection only has 5 hashtables as its member variables.The other member functions are:

1. addBow(): Given information about one bow, create a Bow object and insert into each of the 5 hashtables. Note that each hashtable has the product number as the key and a Bow object as the value. **To be completed.**
2. removeBow(): Given product number, remove the corresponding bow from each of the 5 hashtables. **To be completed.**
3. bestHashing(): The logic to calculate the balance for each of the 5 hashtables, and then identify the hashtable with the best balance should go into this method. Here, balance is defined as the difference between the sizes of the largest bucket and smallest bucket.

Some hints on how to get the number of items in each bucket are included.  **To be completed.**

1. readTextfile(): The list of bows are in a text file. This method calls addBow() for each line. The code to read from the text file is already given[[1]](#footnote-0).

#### Hashing functions

The 5 hashtables will differ in only the hash function that they will use. You are to provide these hash functions. Each hash function will take a 5-digit number and return either the first, second, …, or last digit.

1. hashfct1(number): return the first digit of number. **This is already complete as a hint to complete the other hash functions.**
2. hashfct2(number): return the second digit of number. **To be completed.**
3. hashfct3(number): return the third digit of number. **To be completed.**
4. hashfct4(number): return the fourth digit of number. **To be completed.**
5. hashfct5(number): return the fifth digit of number. **To be completed.**

Main.cpp: This is provided for you to use to test your software as you are writing it. You may change this file to add helpful functions for your own testing. We will test your project with a different but similar file.

README.md: You must edit this file to include your name and CSUF email. This information will be used so that we can enter your grades into Titanium. **To be completed.**

#### Hint

Implement your methods in the same order in which they are tested in the main(): complete hashfct(), then addBow(), then bestHashing() ...

#### Obtaining and Submitting Code

Click the assignment link to fork your own copy of the skeleton code to your PC.

<https://classroom.github.com/a/4IWxjRys>

#### Testing

Unless otherwise directed, use the following command to compile your program:  
**clang++ -g -Wall -std=c++14 main.cpp BowCollection.cpp -o test**  
  
To attempt to run the compiled test program, use the following command:  
**./test**

#### Grading rubric

Your grade will have two parts, *Form* and *Function*.

* *Function* refers to whether your code works properly as tested by the main function (80%).
* *Form* refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

#### Deadline

The project deadline is  **May 9 at 11:59pm**.

1. You should study this code to learn how to read from text files. Most data input to real-world programs is in files. [↑](#footnote-ref-0)