CS-102 Design Principles: Project 4

Due date: Monday, February 20 at 11:00pm

This project is for everyone who has been wondering what the remainder is when you divide the number of ways that ten Hamilton students can be chosen from the student body, that is,

```
126290941955384738291274515
```

into the number of ways that a deck of 52 cards can be arranged, that is,

```
80658175170943878571660636856403766975289505
440883277824000000000000
```

Please start by accepting the assignment at:

https://classroom.github.com/a/ANRLBfyI

which gives you a GitHub repository for your project.

How to submit:

- ☐ Add/commit/push your final code to your GitHub repository; this is the code that will be graded.
- ☐ On Gradescope, click on the assignment for this project, and submit your GitHub repository. (Select *master* as your branch.)

Big Numbers

The bigger of the two numbers above, written as a binary number, has 226 bits, so it cannot be directly stored as even an `unsigned long long`. But a deck of cards, which can fit easily into your hands, can be arranged that many ways, making that *big number* an entirely reasonable number to consider.

In this project, you will write a program that will read a mathematical operator along with a pair of numbers from a text file (using file i/o, not standard input); using this input, your program will produce the desired result. Even if a number is not technically big, you should treat all numbers as if they are, and use dynamic arrays to store them. The only library that you should use is `iostream`.

Example input

The input will be in this form:

```
+
240614678640326224736921497279 91
126765060022822940149670320537 6
```

The symbol + in the first row is the mathematical operation that you should carry out on the two big numbers[1] that will appear in rows two and three. You can assume that each big number is positive, and begins with a non-zero digit.

Example output

The example output should simply be a big number that shows the result of applying the operation to the two given big numbers. The output for the example input above should be the sum of the two big numbers, namely:

```
253291184642608518751888529333 67
```

Unless you really dig doing math[2] by hand, you can use wolframalpha.com to create your own tests and check solutions.

Operation: + (addition)

If the symbol on line one of the input is + then your output should be the sum of the two big numbers, with no leading zeros. For example, if the input is

```
+
123459999
1111
```

then the output should be

```
123461110
```

[1] As an aside, the top number (in the example input) is the number of ways that 1000 can be written as the sum of smaller numbers, and the bottom number is the maximum number of genome sequences of length 30.

[2] Navigators on sailing ships traveling across oceans had to multiply big numbers by hand very quickly, which prompted the discovery of logarithms.

Operation: − (subtraction)

If the symbol on line one of the input is - then your output should be the difference of the two big numbers, with no leading zeros (unless the two big numbers are exactly equal, in which case the output should be a single `0`). You can assume that, if we use −, then the big number in row two will be at least as large as the big number in row three. That is, the result of subtraction will never be negative.

```
-
78000203
87566
```

then the output should be

```
77912637
```

Operation: % (modulo)

If the symbol on line one of the input is `%` then your output should be the remainder when the second number is divided into the first number. You can assume that, if we use `%`, then the big number in row two will be strictly larger than the big number in row three.

```
%
892384619
1729839
```

then the output should be

```
1517534
```

Your `%` operator should be able to handle any two big numbers. Solutions only handling "small" numbers (that fit in `int`, `long`, or `long long` for example) won't earn points for the `%` operator, even if tests are passed.

Grading

Your grade will come from:

75%	correctness + passing the tests
25%	style

There will be 12 tests:

- ☐ 7 tests will use `+`
- ☐ 3 tests will use `-`
- ☐ 2 tests will use `%`

Projects passing zero tests will receive at most 70% of the project grade (assuming perfect style). Passing tests increases the grade proportionally up from 70%; however, this assumes that your code isn't written to circumvent testing (for example, by "hard coding" answers to the tests), which is a violation of the honor code.

The elements of style that we will assess on this project are:

- ☐ Write functions that are no more than about 20 lines long; each function should have one purpose, and be named according to that purpose.
- ☐ Use appropriate programming constructs; e.g., don't use a loop when an if statement is sufficient, a while loop when a for loop is better, and so on.
- ☐ Prune your dead code: remove commented out code, unused variables, other code not contributing to the overall project.

  We will give you one aggregate grade for the rest of these style elements:

- ☐ Include all file and function header comments, using the required format.
- ☐ Declare variables near their initial use.
- ☐ Indent properly and use correct placement of curly braces and parentheses.
- ☐ Use correct spacing around operators.
- ☐ Use reasonable variable names in proper naming style: using `lower_case` naming in local scope, and `UPPER_CASE` naming for global constants.