A Simple Calculator

The goal of this assignment is to implement a simple calculator that reads its operations from a file. This project uses concepts from the slides on loops and on reading data from files.

Documentation and Style

This project **will have a portion of its grade devoted to documentation and code style.** Refer to the style guidelines on the course website for examples of what the TAs will be looking for.

Project 2 is worth **100 points**. When you submit Project 2, Web-CAT will test that your program works correctly and creates the proper output, **if your program works correctly** you will receive **80 points** from the automated testing provided by Web-CAT. The remaining **20 points** are for **style and documentation**, which will be **manually graded by the TAs**, using the **rubric** posted on the course website.

Program Logic

In this project, you will implement a simple calculator that supports the addition, subtraction, multiplication, and division *operations*. **Each of these operations will use will integer arithmetic when performing calculations.** Further, the operations can be combined together to create larger computations, so your calculator will be able to compute: ((5 + 4) + 4) * 2.

Instead of reading information from the keyboard, in this project you will read all of the necessary information from a file that **will be specified by the user at runtime**. The operations will then be read from the file and the results calculated.

Much like a standard calculator, your program will be able to use the result from the immediately previous computation. Only the last the result from the previous computation needs to be stored. For the remainder of the specification we refer to this as the `PreviousResult`. Initially, the `PreviousResult` should be zero.

Calculator Operations

Operations have a name, which should be represented by a string, and then have either one or two parameters, depending on the name (or type) of the operation. For example, you might encounter `add 3 2` in the input file. This is the "add" operation and it takes two parameters (in this case `3` and `2`). Operations that end with `_prev` use the `PreviousResult` and take a single parameter. Below are the operations your calculator must support.

Adding

`add X Y` - Same as `X + Y`, stores the newly computed sum in `PreviousResult`.
`add_prev Y` - same as `PreviousResult + Y`, stores the newly computed sum in `PreviousResult`.

Subtracting

`sub X Y` - Same as `X - Y`, stores the newly computed difference in `PreviousResult`.
`sub_prev Y` - same as `Previous Result - Y`, stores the newly computed difference in `PreviousResult`.

Multiplying
`mul X Y` - Same as `X * Y`, stores the newly computed result in `PreviousResult`.
`mul_prev Y` - same as `PreviousResult * Y`, stores the newly computed result in `PreviousResult`.

Dividing
`div X Y` - Same as `X / Y`, stores the newly computed result in `PreviousResult`. You may assume `Y` is never 0.
`div_prev Y` - Same as `PreviousResult / Y`, stores the newly computed result in `Previous Result`. Like above you may assume `Y` is never 0.

These operations can be combined together to create larger computations.

Example Input and Output
For ease of understanding, a *calculation or a computation* is a **grouping** of *operations* (add, sub, etc.) inside of the input file.

There can multiple computations in a single file, each comprised of one or more basic operations. The example below contains 3 computations, but the input files used to test your code may have more or less (an arbitrary number of) computations and operations.

The first line of each calculation starts with a number telling you how many operations are in that particular calculation. After the number, each line contains a single operation. After all of the operations are listed, each calculation is separated by a blank line.

Sample File
In the sample file below the first operation is equivalent to `((4 + 5) + 4) * 2`, the second `(3 - 1)`, and the third `(0 / 2)`.

```
3
add 5 4
add_prev 4
mul_prev 2

1
sub 3 1

1
div_prev 2
```

While testing your program you can use the above example, it should also be posted on the course website. On Windows you should store the input file in the same directory the as the rest of the project, so if your project is stored in Desktop/Project2 you should be able to drop the input file into that directory. On Mac I've had luck putting the input file in my home directory (/Users/yourid/input.txt) regardless of where the project is stored.

Note: that the order of the operations in the file, are the order they should be computed in, so you can assume there are implied parentheses around each operation.

Also note: The `PreviousResult` should be initialized to zero before every group of calculations.

Sample Execution
You'll first prompt the user for a file name, open the file, then proceed to execute each calculation in the file printing the result as you go:

```
Please enter a filename: calc.txt
The result of calculation 1 is: 26
The result of calculation 2 is: 2
The result of calculation 3 is: 0
```

What to Submit
For this assignment you should submit your `main.cpp` file from inside of your project directory.

This assignment will be graded automatically. Test your programs thoroughly before submitting them. Make sure that your programs produce correct results for every logically valid test case you can think of. Do not waste submissions on untested code, or on code that does not compile with the supplied code from the course website.

Web-CAT will assign a score based on runtime testing of your submission; your best score will be counted; the TAs will later verify that your best submission meets the stated restrictions, and assess penalties if not.

To submit this assignment:

1. Visit http://web-cat.cs.vt.edu in your web browser.
2. Enter your Virginia Tech PID and password in the appropriate fields on the log-in screen, and make sure that **Virginia Tech** is selected as the institution. Click **Login**.
3. The Web-CAT home screen will display useful announcements and assignments that are currently accepting submissions. Find the assignment that you want to submit in the table, and click the "Submit" button next to it.
4. Click the **Browse...** button and select the file you want to upload. The homework assignments and programming projects for this course should be self-contained in a single **main.cpp** file, so you can simply select that one file.
5. Click the **Upload Submission** button. The next page will ask you to review your selection to ensure that you have chosen the right file. If everything looks correct, click **Confirm**.

The next page will show that your assignment is currently queued for grading, with an estimated wait time. This page will refresh itself automatically, and when grading is complete you will be taken to a page with your results.

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the submitted file:

```
//    On my honor:
//
//    - I have not discussed the C++ language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used C++ language code obtained from another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any C++ language code or documentation used in my program
//      was obtained from an allowed source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    <Student Name>
```

Failure to include this pledge in a submission will result in the submission being disallowed during code review.