Assignment #4 (Due 11/13/2017 @ 11:59 P.M.; e-mail must be sent by the due date at 11:59 P.M.)

Soreland, a software company, is planning on releasing its first C++ compiler with the hopes of putting MiniSoft's Visual C++ out of business (that reminds me of another story). Consequently, Soreland has contracted with the Fruugle Corporation to design and build part of their compiler. Of course, since Fruugle gets all of its labor by exploiting El Camino undergraduate students, you'll be doing all of the programming.

For Fruugle, you will write a C++ program to test the "balance" of certain features of C++ programs. This is a task that is done by C++ compilers. The technique you will use is similar to the technique used by a typical C++ compiler.

Requirements

There are many features in C++ programs that must be ``balanced'' to be syntactically correct. For example, every (must be balanced by a corresponding). Similarly for {} and []. There are other (somewhat more complex) situations.

- Block comments must be balanced. Each /* must be balanced by a */ The complication here is that the tokens being balanced are not composed of single characters. Moreover, the ``closing comment'' */ starts with a character that is used in other ways (pointers, multiplication).
- Quotes must be balanced. This applies to single quotes and double quotes. The complication here is that all the material between the matching quotes can contain any characters. The characters between the quotes do not require balancing. For example, the string "This is a /* ' * quote" is legal, and the comment and single-quote need not be balanced.
- Inline comments are not balanced. Anything that comes after a comment is ignored until you get to the end of the line that's read in.

There are some balanced C++ programs you need not test. For example, some balanced C++ programs contain character constants such as '\'' or '\'"'. These are inherently unbalanced and would require a bit more work to catch. You need not worry about such problems for the compiler (unless you like the modest challenge).

The Basic Algorithm

We will go over the basic algorithm in class. But note that simply counting the number of characters will not work. For example, the expression }abc{ has the correct number of balancing braces (one opening brace and one closing brace), but it is not valid C++ code. A stack provides the mechanism needed for the algorithm. Basically, whenever a closing symbol (such as }) is seen, it must match the nearest opening symbol of the same type (such as {). One important point: **do not write a large main file**. Your main

file should simply get a file name, call a function to print the file with line numbers, and then call a balance-checking function.

Your Program

Your program prompts the user for the name of the C++ source file to test. If unable to open the file, the program issues an appropriate ``file not found'' error message, and exits. Before beginning the check, your program prints the file with line numbers. Every time your program pops the stack (to check for balance), it issues a message such as

```
pair matching [  and  ]
```

As soon as a balance-error condition is found, the program issues a message such as

```
unmatched { symbol at line 16
```

and exits.

Hints

1. You don't have to implement the stack, you can use the Standard Template Library's implementation .
2. Here is a write up on files that might be useful to you.
3. You can read a file character-by-character using the get method from fstream.
4. You can read a file line-by-line by using getline.
5. A file that you have read all the way to the end can be read again by doing
6. infile.clear();
7. infile.seekg(0, inFile.beg);

8. If you read up on vectors you won't need to read through the file twice.
9. When you read the / character, it may be the opening of a comment (as in /* or //), or it may be the division operator. You will have to read the character after it to decide. If it's not the opening comment, you may want to put back the following character. You can use putback() to do this.
10. When you read the * character, it may be the closing of a comment (as in */), or it may be the multiplication operator or a pointer operation. Once again, you may want to use putback().
11. The opening-comment ``symbol'' /* and the closing-comment ``symbol'' */ are composed of two characters. One choice for a single character to push for a comment symbol might be an otherwise unused symbol character (for example, the character c).
12. You may find it useful to define a lot of utility functions, here are some examples although depending on your implementation you may not need these particular ones:

    bool isOpeningBrace(int c)
    returns true if c is an opening brace, parenthesis, or bracket, false otherwise.

bool opensBlockComment(...)
returns true if c and the character following it in infile form the opening of a comment (i.e., slash-star). If so, the character following the slash (i.e., *) is eaten. Returns false otherwise and puts the character following the slash back in the ifstream.

Sample Runs

Here are some samples of the output generated by the program. Notice that the C++ source code does not have to be legal C++, it just has to meet balance conditions. The program begins by printing the file to be tested with line numbers. It then checks the balance and reports on the outcome.

A Balanced Test Case

```
Please enter filename for C++ code: good_balance.cpp
  1 /*
  2   good_balance.cpp
  3   Test data for CS2 Project 3
  4
  5   Created: 24 October 1997
  6   Modified: 6 October 2015
  7   */
  8
  9 #define FOO   '"* /* */ foo'
 10 #define BAR   " /* ' "
 11
 12 int main(int argc, char * argv[])
 13 {
 14   char c = {'a';}
 15   y = 3;
 16   z = 4;
 17   x = y * z;
 18   y = x / z;
 19   cout << "Hello world\n"; // :)
 20 }
 21
pair matching /* and */
pair matching ' and '
pair matching " and "
pair matching [ and ]
pair matching ( and )
pair matching ' and '
pair matching { and }
pair matching " and "
pair matching { and }
Balance is ok
```

An Unbalanced Test Case

```
Please enter filename for C++ code: bad_balance.cpp
  1 /*
  2   bad_balance.cpp
```

```
  3   Test data for CS2 Project 3
  4   This program has bad balance conditions.
  5
  6   Created: 24 October 1997
  7   Modified: 6 October 2015
  8   */
  9
 10 #define FOO   '"* /* */ foo'
 11 #define BAR   " /* ' "
 12
 13 int main(int argc, char * argv[])
 14 {
 15   char c = {'a';}
 16   y = 3;
 17   z = 4;
 18   x = y * z;
 19   y = x / z;
 20   {
 21     cout << "Hello world\n"; // :)
 22   }
 23
pair matching /* and */
pair matching ' and '
pair matching " and "
pair matching [ and ]
pair matching ( and )
pair matching ' and '
pair matching { and }
pair matching " and "
pair matching { and }
unbalanced { symbol on line 14
```

Another Unbalanced Test Case

```
Please enter filename for C++ code: bad_balance2.cpp
  1 /*
  2   bad_balance2.cpp
  3   Test data for CS2 Project 3
  4   This program has bad balance conditions.
  5
  6   Created: 24 October 1997
  7   Modified: 6 October 2015
  8   */
  9
 10 #define FOO   '"* /* */ foo'
 11 #define BAR   " /* ' "
 12
 13 int main(int argc, char * argv[])
 14 {
 15   char c = {'a';}
 16   y = 3; '
 17   z = 4;
 18   x = y * z;
 19   y = x / z;
 20   {
 21     cout << "Hello world\n"; // :)
```

```
 22   }
 23 }
 24
pair matching /* and */
pair matching ' and '
pair matching " and "
pair matching [ and ]
pair matching ( and )
pair matching ' and '
pair matching { and }
unbalanced quote ' on line 16
```

Please submit a zip file to Canvas in the format FirstInitialLastName-CS2-149-A4.zip. For example, go ahead and zip the file as EAmbrosio-CS2-149-A4.zip, containing these two files:

1. A file named **balance.cpp** that contains the source code for the completed C++ program. This program must build successfully, and its correctness must not depend on undefined program behavior.
2. A file named **report.doc** or **report.docx** (in Microsoft Word format) or **report.txt** (an ordinary text file) that contains
    a. A brief description of notable obstacles you overcame.