**CMPT 225 Assignment 1: Grouping Symbol Checker**

You are to write a C++ program that reads in a text file and checks that all occurrences of the grouping symbols (,),{,}, and [,] are correctly matched and nested. (The text file could be, for example, a C++ source code file, although actual source code files provide problems that you will not be solving, such as distinguishing content inside comments.) If there is any incorrect matching or nesting, the program prints out a message, as described below, identifying where the first error was identified and the nature of the error. If no errors are found, it prints a line containing only the string "No Errors Found".

**Solution Method**

Here is an algorithm, slightly adapted from the version presented in class, that is suitable to start from. You will need to adapt this so that you can keep track of line numbers.

S = a new stack of characters

while ( there are still un-processed input characters ){

c = next un-processed character

if ( c is an opening symbol ){

S.push( c )

}

else if ( c is a closing symbol ){

if ( S is empty ){

report error *too many* ***c*** and halt.

}

l = S.pop()

if ( l does not match c ){

report error *read* ***c****, expected* ***r*** and halt.

// here, **r** is the closing symbol matching **l**

}

}

}

if ( S is not empty ){

c = S.pop()

report error *too many* ***c*** and halt.

}

report *no errors found*

**Some Further Details**

1. Your program **must** read its input from "standard input". If you do not understand what this means, learn about it before you start designing your program. It is described on page 4 of our textbook, and a web search for "standard input linux C++", or something similar, should lead you to plenty of additional information.
2. You must use a stack class that you implement based on the following files: [char\_stack.h](http://www.cs.sfu.ca/~mitchell/cmpt-225/2018-Spring/assigns/1-grouping_check/char_stack.h), [char\_stack.cpp](http://www.cs.sfu.ca/~mitchell/cmpt-225/2018-Spring/assigns/1-grouping_check/char_stack.cpp). Start by downloading these and the provided [Makefile](http://www.cs.sfu.ca/~mitchell/cmpt-225/2018-Spring/assigns/1-grouping_check/Makefile), and saving them together in an appropriate directory. You will need to complete the contents of the char\_stack header file and implementation file. The header file for the stack class contains the full set of public functions you need. **Do not change any of the public function declarations.** You may create additional .h and .cpp files, if you really want to (provided you revise the makefile to handle them correclty) **... but you are strongly encouraged not to!** Call your main program gscheck.cpp (as suggested by the provided makefile).
3. When you encounter either of the error conditions identified inside the main while loop, print out a message consisting of:
   1. A line containing "Error on line line\_number: error\_description".
   2. The input line where the error condition was found, up to the point it was found.
   3. A line containing as many blanks as there are characters on the previous line, plus the remainder of the input line.

Items 2 and 3 mean: Print out the line of input where you spotted the error, but with a line break at the character where you spotted the error. For example, if line 37 of file mytext.txt is

if ( x < y )){ then z = 2 ;}

your run might look like this:

uname@hostname: ~$ ./gscheck < mytext.txt

Error on line 37: Too many )

if ( x < y ))

{ then z = 2 ;}

1. When you encounter the error condition identified after the main while loop, print out a line containing "Error at end of file: error\_description".
2. There are many ways to read the input for this program. In a previous exercise, you used a statement of the form **cin >> x** to read something from standard input (which by default is the keyboard). You can do this here, but in your program you have to keep track of line numbers, so you may find a different method more convenient. One alternative, which works well, is to use the **cin.getline()** function. Start by reading [Basic I/O tutorial at cplusplus.com](http://www.cplusplus.com/doc/tutorial/basic_io/), and then look at the [getline documentation](http://www.cplusplus.com/reference/iostream/istream/getline/). If you're new to C++, this may be hard to read, but there is a useful example there. You need to read enough to understand how to make use of that example. The example just deals with one line of input, so it has to be altered for out context, where the input files may contain many lines. Also, matching symbols may appear many lines apart, for example an opening { could appear on line 1, and its matching } only several hundred lines later. So, you need to read in as many lines as there are, and the sequence of characters your algorithm works on is equivalent to the concatenation of all these lines. But, you probably don't want to construct the concatenation - just process each line as you read it. These functions of **cin** may be useful:
   1. **cin.peek()**, which looks ahead one character. You can check if that character is the special character **EOF**.
   2. **cin.getline(l,n)**, which is illustrated in the example mentioned above.
   3. **cin.gcount()**, which returns the length of the line just read.
3. Do not try to find and report more than one error per file.
4. You may assume that files contain at most 1000 lines, and that there are at most 250 characters per line.

**Remarks**

1. Read all the instructions before you start designing your program (and review them from time to time).
2. Code in very small bits: work out how to read the file line-by-line first, then how to find the important characters, then how to keep track of them with the stack, etc.
3. The exact form of the error messages matters: we may use a program to grade your program.
4. You can do initial testing of your program just typing input at the keyboard, but this will soon become tedious. The idea is to use the operating system "redirection" facility to send a file to your program's standard input. This is easily done at the command line, as follows, (assuming your test text file is called "mytext.txt"):
5. uname@hostname: ~$ ./gscheck < mytext.txt
6. No Errors Found
7. uname@hostname: ~$

**Make sure you understand this** - it is how we will test your program. A web search for "linux input redirection", for example, should lead you to explanations and examples.

1. We talk about "finding errors", but in fact you cannot find the error, you can only figure out that there is one somewhere. For example, given the text "((()))]", you don't know if the mistake is that there is supposed to be an opening [ - which could belong in either of two spots - or if the mistake is that there should be no closing ]. You do know, when you read the ], that there is *some* mistake, and that is what you are to report.
2. In principle, a program like your solution for this assignment program should be able to check its own source code for correctness. In practice, this requires a more complex program, because because to correctly check C++ programs, you have to ignore the grouping symbols that occur inside comments. Do not address this problem for the assignment. (In my own sample solution, I carefully wrote my comments so that the grouping symbols inside the comments were correctly matched and nested. That way, I could use my solution code as a test case for my solution code. You don't need to do this.)

**Testing**

You will be provided with some example of inputs and correct corresponding outputs for testing. You should also run other test cases that you choose yourself.

**Grading**

This assignment will be graded out of 18.

* 8 marks for correctness based on the provided set of [sample test files](http://www.cs.sfu.ca/~mitchell/cmpt-225/2018-Spring/assigns/1-grouping_check/tests.zip);
* 8 marks for correctness based on additional files not provided to you;

*The sample test files contain no tab characters. Three marks for additional test files will be based on correct handling of the tab character ('\t'). For this, you need to understand the tab character and how it prints on a terminal. (In particular, you should understand that it is not translated into a fixed number of spaces - how much space it takes depends on the terminal settings.) For example, if we wrote the example 2a.in using tabs for indentation instead of spaces, then we would have the following*[*input file*](http://www.cs.sfu.ca/~mitchell/cmpt-225/2018-Spring/assigns/1-grouping_check/tabEx.in)*and corresponding*[*output file*](http://www.cs.sfu.ca/~mitchell/cmpt-225/2018-Spring/assigns/1-grouping_check/tabEx.out)*. (To see what is happening, you have to use an editor that displays non-printing characters, or use the Linux cat or more command to output them to your terminal.)*

* 2 marks for coding style: reability of code, etc.

Failure to follow key instructions may result in deductions. For example, if you modify the public functions of the char\_stack class, you may get zero even if your program works perfectly. Programs that do not compile and run on the CSIL linux workstations will get zero. Programs that read from a named file, instead of standard input, will get zero.

**Submission**

You should submit your assignment online to [CourSys](http://courses.cs.sfu.ca/) .  Make a single .zip file by running zip on the directory containing all your source code files (which, by default, will be, char\_stack.h, char\_stack.cpp, gscheck.cpp, and Makefile). Submit this .zip file to Coursys. The assignment is due by 8:00 pm on Tuesday Feb. 20.