Stack Project  
CS 2420 Section 01

## Instructor: Todd Peterson

## Due: Date shown on Canvas.

You may **not** usethe standard template library for this or any other project this semester unless specifically mentioned in the instructions -- Even though the book does.

**Disclaimer:**I <name> have not used any code other than my own (or that in the textbook) for this project. I also have not used any function or data-structure from the Standard-Template Library. I understand that any violation of this disclaimer will result in a 0 for the project.

Purpose: Gives students practice using stacks and in converting infix to postfix. Also reinforces the following concepts:

* File IO
* Converting pseudo-code to actual code
* Stacks

Instructions: Implement the pseudo-code on page 385 of Main and Savitch to convert an infix expression to a postfix expression. Lots of help on how to do this is in chapter 7 of the book.

***Your stack must use templates***.

Turn in: all code files

* calc.cpp <Put main and your function(s) to convert infix to postfix in here.>
* stack.h (you may use the stack in the book or write one of your own.)
* stack.cpp
* your .exe executable
* tests.txt

**Grading:** you will receive 5 points for each line processed correctly (assuming you wrote a correct algorithm.)

**Hints:**

You may wish to use recursion when encountering a left paren ‘(‘ computing this weight and returning it to the rest of the parser. But you do not have to.

**Help for reading input from a file: (Imported from the Main and Savitch website)**

You need to create a program that reads a file of arithmetic expressions as input and writes the resulting answers to cout. Input from a file is easy in C++. Here are the steps:

1. The program must #include <fstream.h>.
2. The program needs to declare an ifstream variable that will "attach" to the input file. The data type of this variable is ifstream. For example, you could make the declaration of an ifstream variable called input, as shown here:

ifstream input; // Will be attached to the input file

1. The ifstream must be attached to the input file by activating the open function. The argument to the open function is an ordinary C string constant or string variable. For example, if input is an ifstream, then we can attach input to a file named foo with the function call:

input.open("foo");  
The argument to the open function can also be a string variable-- i.e., an array of characters with the last character followed by the null character '\0'.

1. You must check that the file was successfully opened. This check can be made by activating the fail function. This function returns a true/false value to indicate whether the ifstream has failed to properly attach to the input file (true indicates failure). I'd suggest something like this:
2. if (input.fail( ))
3. {
4. cerr << "Could not open input file." << endl;
5. exit(0);
6. }

1. Once the ifstream input has been successfully opened, you may use the name input in the same way that you use other input devices (such as cin). For example, suppose that the next item in the input file is an integer, and i is an integer variable. Then you may execute the statement:

input >> i; // Read an integer from the input file

All the other familiar input functions (such as peek and ignore) can be used with the ifstream, in the same way that you have previously used these function with cin.

1. Eventually you will read the end of the input file. At the end of the file there is usually a special character called EOF (which is not actually part of the data of the file). You can also use the name of the ifstream (such as input in our example) as a boolean expression which is true so long as the file has not contained any bad input (such as an alphabetic character when a digit is expected). Thus, the complete boolean expression to test whether there is more input reads like this:

(input && input.peek( ) != EOF)

1. When you are done reading from the input file, then you should activate the close function, as shown here for our example:

input.close( );