Project 7: `PostfixCalculator`

For this project you will implement a simple calculator. Your calculator is going to parse infix algebraic expressions, create the corresponding postfix expressions and then evaluate the postfix expressions. The operators it recognizes are: +, -, * and /. The operands are integers.

Your program will either evaluate individual expressions or read from an input file that contains a sequence of infix expressions (one expression per line). When reading from an input file, the output will consist of two files: `postfix.txt` that contains the corresponding postfix expressions one per line and `results.txt` that contains their evaluation, also one per line.

An example of an infix expression is the following:
(12 + 3) * (9 – 74) + 34 / ((85 – 93) + (3 + 5) * 3) - 5

It is guaranteed to:

- Contain only integers
- Contain only round parentheses
- All operands and operators are separated by a single space

A malformed expression may however:

- Have unbalanced parentheses – e.g. (((3 + 2)
- Have operators other than the expected ones – e.g 2 ^ 3

Your program will parse the infix expression from left to right, identify the operands, the operators and the parentheses and ignore blanks. You will **implement the infix to postfix conversion algorithm using a stack of operators and parentheses**. You can use the STL stack (`#include <stack>`)
Your program should handle syntactically malformed infix expressions using exceptions (you can use the `PrecondViolatedExcep` class used with the List class – see Project4). When an exception occurs, the exception message should be written to the postfix file and results file instead of the respective postfix expression and result.
For unbalanced parenthesis the message must be:
`"Precondition Violated Exception: Unbalanced parenthesis"`
For unknown operator the message must be:
`"Precondition Violated Exception: Unknown operator"`

Implementation:

You will implement the calculator as a class named `PostfixCalculator`.
You may design this class as you wish, but it MUST have the following **public** methods:

- A default constructor
- `std::string convertToPostfix(std::string infix_expression);`
  Takes a string representation of the infix expression, parses it into the corresponding postfix expression and returns the postfix expression as a string. In the string representation of the postfix expression, operands and operators are also separated by a single space. For example a valid postfix expression string corresponding to infix "3 - (137 + 76)" would be: "3 134 76 + -"
  Note that there are no parentheses in the output postfix expression.
  If the input expression is malformed (i.e. unbalanced parenthesis or unknown operator) this function will throw a `PrecondViolatedExcept` exception with the corresponding error message as described above.
  When we described the algorithm in lecture, we did not consider the fact that, without parenthesis, * and / have higher precedence than + and - and your calculator will need to take that into account.
  On Blackboard under Course Materials/Project7 you will find pseudocode for parsing infix to postfix from your textbook that takes precedence into account. Your textbook uses peek() instead of top() for retrieving the element at the top of the stack.
- `double calculatePostfix(std::string postfix_expression);`
  takes a string representation of the postfix expression (in the format described

above), calculates the result and returns it. It assumes that the postfix expression is well formed (don't forget to specify all pre and post conditions as well s inputs and outputs in your comment preambles)

- `void testCalculator(std::string input_file_name);`
  will implement the file input/output behavior described above:
  Read every infix expression from the input file and:
    - Convert it to postfix (don't forget to do this in a try-catch block, `convertToPostfix` may throw an exception!)
    - Write the corresponding postfix expression into `postfix.txt`, but if `convertToPostfix` throws and exception, write the error message instead
    - Evaluate the postfix expression if wellformed and write the result in `results.txt`, otherwise write the error message there too

All data is on a new line, so if the input file is:

input.txt

```
( 2 + 3 )
( ( 3 + 4
(5 + 3 ) * 2
```

postfix.txt is:

```
2 3 +
Precondition Violated Exception: Unbalanced parenthesis
5 3 + 2 *
```

results.txt is

```
5
Precondition Violated Exception: Unbalanced parenthesis
16
```

You may add as many private functions and data members as necessary.

Review:

Writing output:

Writing to an output file is similar to reading from one and is done using `<fstream>` Similarly to what we have done with `ifstream`, you can declare an `ofstream` object, you <u>open</u> it and <u>close</u> it like you do for a `ifstream`, and you can write to it as you do with the standard output stream using the << operator.

Thus, if you declare an `ofstream` object `out_file:`

```
out_file.open(“results.txt”);
ou_file << 4.5 << “/n”;
```

writes the number 4.5 to results.txt followed by a new line
`out_file.close();` will close the stream.

Standard Template Library (STL) :

You can use the Standard Template Library for this project, in particular `<stack>`.
C++ Interlude 8 in your textbook gives an overview of the STL. You should start familiarizing with containers/adapters and algorithms in the STL as you learn about them. There are also many references and tutorials online.
You don’t need to overwhelm yourself, you can look-for-and-learn as you need, but it is good to know what is there so you’ll know to look for it.
Also, you should familiarize with the STL equivalent of what you are learning

- Vector
- List (in STL list is doubly-linked, forward-list is singly-linked)
- Stack
- Queue

Be aware of other containers so you will know to look for them when you need them.
Be aware of what is available to you in <algorithm>
Understand iterators
Explore and have fun!!!

Testing:

Come up with infix expressions and test them individually. Some correct, some with unbalanced parenthesis (both opening-unbalanced and closing-unbalanced) and with unknown operators. Make sure your output is as specified above.

Once you have tested the functions individually, create an input file with all the examples you came up with, and make sure you are correctly writing the output files `postfix.txt` and `results.txt`

Submission:

There are no starter files for this project, but we will use GitHub classroom anyways. You will find the link on Blackboard under Course Materials/Project7. Please accept the assignment and work using the repo. I strongly encourage you to get into the habit of working with version control (which here translates to: it would be nice to see many more commits!!!).
Your project must be submitted to Gradescope through GitHub. The due date is Tuesday November 19 by 5pm. No late submissions will be accepted.

Have Fun!!!!!