CSCE2014 Programming Foundations II
Homework Six
By Wing Ning Li

1 Problem Description

In this homework, we will combine all that we have learned so far (programming techniques, algorithms, data structures) to build a expression evaluation program. The two classes that we have developed in **Homework Three** should be used and enhanced to make our task easier.

This program is an interactive program, that is the program prompts the user for input and action, and displays the answer as output. The input to the program is **a sequence of expressions**. Here are a few examples of valid expressions: 2+3, (2+3)*4, 2+3*4, ((2+3)*4), a+b-c*d/e, (a+(b-d)*d)/e, a=2, and so on. An expression consists of operands and operators. For this homework, operands are integers or identifiers (see the definition in **Homework Three**). The operators include -,+,*,/,=. Here are **two** examples of **a sequence of expressions(a series of expressions separated by ";" and ended with ";')**: 2+3;((2+3)*4);a1+b2-cc;a1=1;b2=2;cc=3;a1 +(b2- cc);, and 2+3;. A sequence of expressions can be defined recursively as:

Defining: A Sequence of Expressions is

- Basis: An empty sequence (nothing or just spaces).
- Recursion (induction): an expression followed by ";" followed by **a sequence of expressions** (recursive).

The above definition assumes we know what is an expression.

In plain English, **a sequence of expressions** is either empty or a series of expressions separated by ";" and ended with ";'.

After the program has read in a sequence of expressions, the user may specify the following actions: =, >, <, f, q, c, s.

The meaning of each action is as follows.

- = : evaluate each expression in the sequence of expressions.
- > : convert each expression in the sequence of expressions to the equivalent prefix expression.
- < : convert each expression in the sequence of expressions to the equivalent postfix expression.
- f : convert each expression in the sequence of expressions to the equivalent fully parenthesized expression.
- q : quit the program.

- c : continue inputting a sequence of expressions (the current sequence appends to the the previous sequence)
- s : start over for a new sequence of expressions.

Unless the actions are q, or c, or s, the program remains in the state of taking user action.

A sample session:

```
=== expression evaluation program starts ===
input:2+3;
action:=
2+3 = 5
action:>
prefix of 2+3 is: + 2 3
action:<
postfix of 2+3 is: 2 3 +
action:c
input:(2+3)*4;2+3*4;
action:=
2+3 = 5
(2+3)*4 = 20
2+3*4 = 14
action:>
prefix of 2+3 is: + 2 3
prefix of (2+3)*4 is: * + 2 3 4
prefix of 2+3*4 is: + 2 * 3 4
action:<
postfix of 2+3 is: 2 3 +
postfix of (2+3)*4 is: 2 3 + 4 *
postfix of 2+3*4 is: 2 3 4 * +
action:s
input:a+b-c*d/e;a=1;b=2;c=3;d=4;e=5;
action:=
a+b-c*d/e = 1
cannot evaluate a=1 which is not an arithmetic expression, but assignment.
cannot evaluate b=2 which is not an arithmetic expression, but assignment.
cannot evaluate c=3 which is not an arithmetic expression, but assignment.
cannot evaluate d=4 which is not an arithmetic expression, but assignment.
cannot evaluate e=5 which is not an arithmetic expression, but assignment.
action:>
prefix of a+b-c*d/e is - + a b / * c d e
no prefix of a=1 which is not an arithmetic expression, but assignment.
no prefix of b=2 which is not an arithmetic expression, but assignment.
no prefix of c=3 which is not an arithmetic expression, but assignment.
no prefix of d=5 which is not an arithmetic expression, but assignment.
no prefix of e=5 which is not an arithmetic expression, but assignment.
```

```
action:<
postfix of a+b-c*d/e is a b + c d * e / -
no postfix of a=1 which is not an arithmetic expression, but assignment.
no postfix of b=2 which is not an arithmetic expression, but assignment.
no postfix of c=3 which is not an arithmetic expression, but assignment.
no postfix of d=5 which is not an arithmetic expression, but assignment.
no postfix of e=5 which is not an arithmetic expression, but assignment.
action:f
fully parenthesizing a+b-c*d/e results: ((a+b)-((c*d)/e))
no fully parenthesizing of a=1 which is not an arithmetic expression, but assignment.
no fully parenthesizing of b=2 which is not an arithmetic expression, but assignment.
no fully parenthesizing of c=3 which is not an arithmetic expression, but assignment.
no fully parenthesizing of d=5 which is not an arithmetic expression, but assignment.
no fully parenthesizing of e=5 which is not an arithmetic expression, but assignment.
action:q
=== expression evaluation program ends ===
```

Since our program needs to handle user errors, it would be a good idea to consider some possible errors and their classifications. First, the expression could be in error. For instance, 3-, 3+ + 4,)2+3, (2+3)), a b + c, a=, are incorrect expressions, to list just a few. Second, a sequence of expressions could be in error. For instance, 2+3 (missing ;), ;2+3; (starts with ;), 2+3;; (two ; in a row at the end), are incorrect sequences of expressions. Third, wrong choice is used for the action. For instance, "y", "ok", are incorrect action input.

2 Purpose

Give us the opportunity to synthesize our programming techniques, algorithm development skills, knowledge of data structures, problem decomposition and refinement ideas, and input error handling in solving a somewhat complicated problem.

3 Design

The design of the program should be logical and hence easy to understand. To achieve it, we should select our classes (two of classes had been designed for us in **Homework Three** and you may want to enhance the expression class with more methods) or functions that address a subproblem or a particular logical task. In addition to being easy to understand, the design should select efficient data structures and algorithms whenever we have a choice to make. User input error should be handled gracefully, meaning that the error should not crash the program and feedback to the user is provided. We should handle the errors in the same way as that of the sample executable program. Please run the sample program on your turing account (after you have copied the executable file) and use the run to answer the questions that you may have about how you program

should work. **A sample executable file that runs on Turing is provided to you and you should mimic the behavior of that in your program.**

4 Implementation

We are free to choose a single file implementation or multiple file implementation. Source code file `homework6.cpp` should contain the main logic of the program (this is the only file for a single file implementation). In general, it would be wise to add functionality to our program piece by piece. And making sure each functionality is tested and working before we move to the next task.

5 Test and evaluation

Since it is a interactive program and user is often creative in typing something that the program does not expect, we should test the program rather thoroughly.

6 Report and documentation

A description of your design and a short report about things observed and things learned and understood. The report should also describe the test cases used to interact with the program and the reasons for such test cases. Properly document and indent the source code. The source code must include the author name and as well as a synopsis of the files.

7 Project submission

Use Blackboard to submit the source file(s) and the report. Note that due to the complexity of the project, there are three check points before the due date. You need to complete the task of each check point and upload the source code that completes the check point for each check point.