CSCI 201 – Computer Science 1

Lab assignment 3: Implementing a simple accumulator
Due date: Friday January 26

Objective: *Become familiar with designing a solution for processing an input stream that contains two kinds of tokens. Learn how to translate such a design into code using the* `while` *loop in C++.*

IMPORTANT *There is sample code for this assignment. Please study that and use it as a template before you start coding.*

Description. An accumulator is a primitive kind of calculator that can evaluate arithmetic expressions. In fact, the Arithmetic-Logic Unit (ALU) of the first computers was just an accumulator. An arithmetic expression, as you know, consists of two kinds of **tokens: operands** and **operators**. All our operands will be (float) numbers and for a start, we shall use only two operators: **+ (plus)** and **- (minus)**. A sample run of the program would look like this.

```
Welcome to your friendly neighborhood accumulator! Please input
your expression, starting with an operand and
type in  '='  when completed.
3.0
 +
2.0
-
-5.2
 =
The answer is 10.2
Thank-you for using your friendly neighborhood accumulator!
```

Note that the **equals sign acts like a sentinel** to mark the end of the input. The problem is formally described as follows:
Input: An alternating sequence of operands and operators, starting and ending with an operand. The sequence is followed by an equals sign, to indicate termination. Each operand is a float value and each operator is a plus or a minus.
Output: The value of the expression defined by the sequence of operands and operators.

Assumption. We shall assume for now that the user will not make mistakes when using the accumulator, i.e., **there is no bad input** (e.g., two consecutive tokens being both operands or both operators, other unexpected characters etc.)
The Strategy. To initialize the process, the first operand (a number) is read and stored as the total. Next, an operator (a character) is read and we decide what action to take. The action can be one of the following:
(i) In case of termination (equals sign) the process is exited and the total is printed;
(ii) otherwise another operand is read and either added to or subtracted from the total depending on the operator.

Question 1: Create a suitable set of tests and list them in a testing table. Think of what tests

would be of particular interest, and put in comments in the testing table explaining why you chose them.

Question 2: Write the sequence of steps that would be carried out when evaluating an expression using the above strategy. Your answer will look something like this:

```
read num;
store num in total;
read operator;
...
(extend this sequence of steps,  following our strategy until the instructions repeat)
```

Question 3: From the answer to Question 2, identify a suitable block of repeating steps. The exit test should be performed at the start of the block.

Question 4: Create a flowchart for this on Raptor, and verify your design. (Note that Raptor does not have the `char` datatype. Operators are treated as strings: "+", "-", "=".)

Question 5: Implement your algorithm in C++. In a script session, display the program (using the `cat` command), compile it, and run your tests. Record the results in your testing table.

What to submit The script file, the .cpp file, and answers to the questions should be uploaded to .