CSE 474/574: Project Part 6
Due:

- Part A: 11:59 PM, April 17, 2014
- Part B: 11:59 PM, April 24, 2014

Overview: In this lab you will modify your lexer, parser, and code generator to add a Boolean type, Boolean operators, relations, and control-flow statements (**if** and **while**).

Detailed Description: In the following is a description of how we are augmenting the micro-language. You will need to augment the token definitions and language grammar, and modify your lexer, parser, and code generator to compile any program correctly written in this new, modified language. In addition to lexical and syntactic errors, semantic errors are now possible (probably caught during code-generation).

- Add a *bool* type.
  - *bool* literals should be called *True* and *False* (note capitalization).
  - This should allow for both *bool*-typed variables and *bool* literals. The built in *write* function should be able to print a Boolean value (printing True or False).
  - The *read* function does *not* have to be able to take Booleans as arguments.
- Add the following operators:
  - Numerical relational operators (equal, not-equal, less-than, less-than-equal, greater-than, greater-than-equal): operates on integers (and floats).
  - Boolean operators (and, or, not): operates on boolean.[1]

  The grammar should enforce standard precedence and associativity rules on the operators, consistent with all other languages and standard mathematical notation, and allow for parenthesized expressions. Hence your compiler should be able to correctly calculate both *x or y and z*, *(x or y) and z*, and *not x and y*.
- Add a conditional structure (**if**):
  - Operators on a Boolean (or an expression evaluating to a Boolean).
  - Can have a body of one or more statements.
  - Can support nested structures to an arbitrary depth (within the limits of memory).
  - Allows for an optional **else** statement (also allowing for a body of one or more statements and for nested structures to an arbitrary depth). *You are not required to provide any equivalent to the Python **elif** statement.*
- An indefinite looping structure (**while**):
  - Operates on a Boolean (or an expression evaluating to a Boolean).
  - Can have a body of one or more statements.
  - Can support nested structures to an arbitrary depth.
  - *You are not required to provide any equivalent to the Python while-associated **else**.*
- In addition to the semantic rules already in play, note:
  - You are not required to enable the **read** function to take *bool* types as arguments.
  - The *write* function should take *bool* types as arguments.
  - The relational operators should be able to compare floats to floats, and floats to ints (CSE 574 only).

[1] Expressions such as (5 + 6 and True) should trigger an error – but it is up to you whether you want to treat that as a syntactic error (catching it in the grammar) or a semantic error.

Note: It is expected that all test code from Part 5 should still compile; you cannot change your syntax in anyway that will break those tests. Further, your syntax for bool declarations, assignments, and expressions should be consistent. *S*yntax of **for** and **while** statements is up to you – but the syntax you employ *must be reasonable*. How you delineate blocks and allow for **else** statements is up to you, but the syntax should be something that an experienced programmer could understand and follow with a *minimum* of effort. There will be a grade penalty for syntax which is considered confusing, misleading, or weird.

Submission: Your group submission is the committed content of the **part5** directory of your group repository as of the deadline. All source code and files needed for compilation must be in (or under) that directory. Your submission should contain:

- Part A:
  - A modified version of the *token.txt* file reflecting your new tokens.
  - A modified version of the *grammar.txt* file reflecting your new grammar.
  - A single file **test.txt** that contains a *series* of micro-code programs separated by line consisting of ten or more # characters. The programs as a set should comprehensively test your new features, demonstrating that all aspects are correctly working. (You do *not* need to provide tests for any feature implemented in a previous assignment,.)
- Part B: Your submission should include:
  - All source code for your lexical analysis library.
  - Source code for a program we can use to run your compiler. The program should take two command-line parameters specifying the name of the code file, and the name of the output file.
  - A file README.txt explaining exactly how to compile (if necessary) and run your testing program.
  - Translation of distributed test cases in your syntax.
    - After the deadline for Part A, a set of test programs will be distributed in *my* syntax. You will be responsible to rewriting them in your syntax.
    - The re-writes should be *equivalent in structure*. That is, they should be a line-by-line translation (to the extent possible), not just a program that does the same thing. For example: if I give you a while loop that adds numbers from 1 to *n*, I want a program in your syntax that actually adds the numbers from 1 to *n* – not a program that just returns the value of $\frac{1}{2}(n^2+n)$.

Grading:

- Part A: 20% of the grade for the part. Points will be deducted for an excessively long test file.
- Part B: Grading of this project will be result-based. The grader will run your code through a number of tests, the success of which will determine your grade. Code will be only examined to ensure academic integrity.
  - You will be penalized if your code is *excessively* slow.
  - Programs that will not run will receive a 0.
  - You will be penalized for failing to return translations of the distributed test cases into the syntax of your micro-language.