HOMEWORK #2:

Recursive Descent Parser

Due Date: Saturday, December the 13th, 11:59.59pm

Write a Recursive Descent Parser using the techniques used in class to recognize valid program of the **AL3** Programming Language.

Submission Guidelines:

Submit via '`cssubmit`'. Your main file shall be called "`al3parser`" regardless of extension. (e.g. if you are programming in C++, your main file should be called "`al3parser.cpp`". If you are programmin in Java your main file should be called "`al3parser.java`").
Your main file should include your name.
Include any other necessary files in your submission.

Description:

Programs of the **AL3** programming language are build from the following tokens:

Tokens:

- Integers are non-empty sequences of digits optionally preceded with either a '+' or '-' sign.
- Decimal numbers are Integers followed by a '.', followed by a non-empty sequence of digits.
- Strings are any non-space sequences of characters enclosed in "". e.g. `"hello" "abc123"`.
- Keywords are the following strings: `is`, `+`, `-`, `*`, `/`, `or`, `and`, `not`, `(`, `)`, `<`, `>`, `=`, `$`, `!`, `print`, `if`, `else`, `endif`, `while`, `endw`, `routine`, `endr`, `run`.
- Identifiers are a sequences of digits or letters. The first character must be a letter, and an identifier cannot be a Keyword.

In **AL3** tokens are always separated by white-spaces.

Grammar:

Programs in the **AL3** programming language conform to the following EBNF grammar:
" RoutineDeclaration " is the start symbol
Terminal symbols are in **bold**. Collections of terminal symbols are in **blue**
Brackets, '[' and ']' , denote an *optional* section of a rule.
Braces, '{' and '}', denote *repetition* of a rule section (possibly 0 times).

Expression := SimpleExpression [Relation SimpleExpression]
SimpleExpression := Term { AddOperator Term }
Term := Factor { MulOperator Factor }
Factor := integer | decimal | string | identifier | **(** Expression **)** | **not** Factor

Relation := **<** | **>** | **=**
AddOperator := **+** | **-** | **or**
MulOperator := ***** | **/** | **and**

Assignment := identifier **is** Expression
RoutineCall := **run** identifier
IfStatement = **if** Expression **$** StatementSequence [**else** StatementSequence] **endif**
WhileStatement = **while** Expression **$** StatementSequence **endw**
PrintStatement = **print** identifier

Statement := [Assignment | RoutineCall | IfStatement | WhileStatement | PrintStatement]
StatementSequence = Statement { **!** Statement }

RoutineDeclaration := **routine** identifier **$** StatementSequence **endr**

Input:

Your Parser should accept input from "standard input" and output to "standard output".

Output:

If the input program is valid, output “`CORRECT`”, otherwize output “`INVALID!`”.

Sample 1:

Input	Output

`routine main $`
`  x is 2 + 2`
`endr`
	`CORRECT`

Sample 2:

Input	Output

`routine fibo $`
`  x is 1 !`
`  y is 2 !`
`  c is 3 !`
`  while c < 10 $`
`    x is x + y !`
`    y is x - y !`
`    c is c + 1`
`  endw !`
`  print c !`
`  print x`
`endr`
	`CORRECT`

Sample 3:

Input	Output
	`INVALID!`

Sample 4:

Input	Output
	`INVALID!`