Project Phase2: Scanning and Parsing using Lex and Yacc
Due Date: Monday Novmber 16th, 2020 (11:59 pm)

---

1 MyLang Language

For this phase, you will use Lex tool for writing the scanner and Yacc tool to generat a parser for a programming language called MyLang.

1. The Scanner and the Lexemes:

   - **Keywords**: int, double, if, read, print, while,START,END.
   - **Operators**:
     - Arithmetic: `+ , - , * , /`
     - Logic: `&&,||,!`
     - Relational: `==, <>, <, <=, >,>=`
     - Assignment: `=`
     - Punctuation elements: `(),,,{}`
   - **Precedence and associativity**
     The table below shows all MyLang operators from highest to lowest precedence, along with their associativity.

Table 1: precedence and associativity table

Operator	Description	Associativity
!	Unary Logical NOT	right to left
*/	Multiplication/Division	left to right
+−	Addition/ Subtraction	left to right
<, <=, >,>=	Relational	left to right
==<>	Equality	left to right
&&	Logical AND	left to right
||	Logical OR	left to right
=	Assignment	right to left

   - **Identifiers**: should start and end with capital letter and can include small, capital letters, underscores, or digits.
   - **Comment**: is sequence of any characters between two @ `@.....@`
   - **Literals**: INT_LITERAL e.g. `12345`, STRING_LITERAL e.g. `"Hello"`and DBL_LITERAL e.g. `3.14`.

2. The Parser and the Grammar:

When YACC finds input that doesn't match the grammar, it automatically terminates with the message 'Syntax error'. You will need to implement an error routine (yyerror) that also prints out the line number before this termination, and a main method that read Mylang code from a file. Print an appropriate message in case of no syntax error. Build your parser using the grammar below:

- Program→ START Statements END
- Statements → Statements Statement| Statement
- Statement→ Dec_stmt | Assignment_stmt | Print_stmt | Read_stmt | Condition_stmt | While_stmt
- Dec_stmt→ Type ID |Type ID,IDList
- IDList → ID | ID,IDList
- Type→ int | double
- Assignment_stmt→ID = Expression
- Expression → exp == exp | exp <> exp | exp < exp | exp <= exp| exp >= exp | exp >= exp | exp
- exp→ exp + exp | exp - exp| exp * exp | exp /exp | exp || exp | exp && exp | !exp | Factor
- Factor→ (exp)| INT_LITERAL | DBL_LITERALL | ID
- Print_stmt→ print (ID) | print (STRING_LITERAL)
- Read_stmt→ ID = read ( )
- Condition_stmt→ if (Expression){ statements }
- While_stmt→ while (Expression) { statements}