CS 4110 - Compilers and Interpreters
Project #2
(Due: 11:59 PM, 12/03/2020)

You are to implement a syntax analyzer for the programming language Toy, as defined in project #1. You should first design a CFG G for Toy based on the following Backus Normal Form (BNF) description, and then write a program to (1) create a parsing table for G, and (2) perform a one-symbol lookahead parsing on various input Toy programs and print appropriate parsing actions.

The grammar of Toy is given in a variant of extended BNF. The meta-notation used:

x	means that x is a terminal i.e. a token. All terminal names are lowercase.
_X_	(in italic) means _X_ is a nonterminal. All nonterminal names are capitalized.
<x>	means zero or one occurrence of x, i.e., x is optional
x*	means zero or more occurrences of x
x+	means one or more occurrences of x
x+,	a comma-separated list of one or more x's (commas appear only between x's)
|	separates production alternatives

For readability, we represent operators by the lexeme that denotes them, such as + or != as opposed to the token (_plus, _notequal) returned by the scanner.

1. _Program_ ::= _Decl_+
2. _Decl_ ::= _VariableDecl_ | _FunctionDecl_ | _ClassDecl_ | _InterfaceDecl_
3. _VariableDecl_ ::= _Variable_;
4. _Variable_ ::= _Type_ id
5. _Type_ ::= int | double | boolean | string | _Type_ [] | id
6. _FunctionDecl_ ::= _Type_ id (_Formals_) _StmtBlock_ | void id (_Formals_) _StmtBlock_
7. _Formals_ ::= _Variable_+, | ε
8. _ClassDecl_ ::= class id <extends id> < implements id+, > { _Field_* }
9. _Field_ ::= _VariableDecl_ | _FunctionDecl_
10. _InterfaceDecl_ ::= interface id { _Prototype_* }
11. _Prototype_ ::= _Type_ id (_Formals_) ; | void id (_Formals_) ;
12. _StmtBlock_ ::= { _VariableDecl_* _Stmt_* }
13. _Stmt_ ::= <_Expr_> ; | _IfStmt_ | _WhileStmt_ | _ForStmt_ | _BreakStmt_ | _ReturnStmt_ | _PrintStmt_ | _StmtBlock_
14. _IfStmt_ ::= if (_Expr_) _Stmt_ <else _Stmt_>
15. _WhileStmt_ ::= while (_Expr_) _Stmt_
16. _ForStmt_ ::= for (<_Expr_> ; _Expr_ ; <_Expr_>) _Stmt_
17. _BreakStmt_ ::= break ;
18. _ReturnStmt_ ::= return <_Expr_> ;
19. _PrintStmt_ ::= println (_Expr_+,) ;
20. _Expr_ ::= _Lvalue_ = _Expr_ | _Constant_ | _Lvalue_ | this | _Call_ | (_Expr_) |
    Expr + _Expr_ | _Expr_ – _Expr_ | _Expr_ * _Expr_ | _Expr_ / _Expr_ | _Expr_ % _Expr_ | -_Expr_ |
    Expr &lt; _Expr_ | _Expr_ &lt;= _Expr_ | _Expr_ &gt; _Expr_ | _Expr_ &gt;= _Expr_ |
    Expr == _Expr_ | _Expr_ != _Expr_ | _Expr_ && _Expr_ | _Expr_ || _Expr_ | ! _Expr_ |
    readln () | new (id) | newarray (intconstant , _Type_)
21. _Lvalue_ ::= id | _Lvalue_ [_Expr_] | _Lvalue_ . id
22. _Call_ ::= id (_Actuals_) | id . id (_Actuals_)
23. _Actuals_ ::= _Expr_+, | ε
24. _Constant_ ::= intconstant | doubleconstant | stringconstant | booleanconstant | null

Operator precedence from highest to lowest:

[.	(array indexing and field selection)
! -	(logical not, unary minus)
*/ %	(multiply, divide, mod)
+ -	(addition, subtraction)
<<= > >=	(relational)
== !=	(equality)
&&	(logical and)
||	(logical or)
=	(assignment)

- All binary arithmetic and logical operators are left-associative.
- The relational operators are not associate, which means we cannot chain a sequence of operators that are on the same precedence level. For example, a < b >= c should not parse, but a < b == c is allowed.
- Parentheses may override precedence and associativity.

For every input Toy program, you should print out a sequence of tokens generated by your first project (the lexical analyzer), and print out the action (either "shift" or "reduce") decided by your parser for each token. If the action is "reduce", also print out the production number. For instance, given the following CFG:

1. $S \rightarrow a X c$
2. $X \rightarrow b X$
3. $X \rightarrow b$
4. $X \rightarrow Y d$
5. $Y \rightarrow Y d$
6. $Y \rightarrow d$

The sample output for string ***abbbc*** should be

```
a [shift]
b [shift]
b [shift]
b [shift]
c [reduce 3][reduce 2][reduce 2][shift]
[reduce 1]
[accept]
```

and the output for ***abcd*** should be

```
a [shift]
b [shift]
c [reduce 3][shift]
d [error]
[reject]
```

Now write a report to include the following:

(1) Your context-free grammar for Toy. Please sequentially number all production rules.

(2) Comment on the number of S/R and R/R conflicts you get initially, discuss your attempts and strategies to solve each of these conflicts, and the number of conflicts remain unsolved at the end.

(3) Describe your test case design. You should run your program with at least the following test cases and fill the table below. For each test case, indicate the expected output based on the syntax of Toy and compare that with your program output. If a test case is syntactically correct, give the sequence of productions for the rightmost derivation of that input string. Otherwise, explain the syntax error.

(4) A discussion of the strength and constraints of your work.

(5) A team-member effort report signed by all members.

	Input	Expected outcome	Program result
1	`void f(double x, double y) { }`		
2	`int i = 1;`		

3	`// in function`
	`result = 5.times(4);`		

4	`// in function`
	`front = in.nextLine();`		

5	`// in function`
	`front = in.nextLine;`		

6	`int[][][] super;`		
7	`a[3][4.5][b] = result = x + y + z;`		

8	`// in function`
	`for ( ; ; ) x = 1;`		

9	`// in function`
	`if (h>w) g=h;`
	`else h=g;`
	`double a;`		

10	`// in class`
	`boolean b;`
	`userDefinedClassType  f(){}`
	`double d;`
	`string g(){}`		

11	`int f() {`
	`    class A`
	`    {`
	`      string lastName;`
	`      string firstName;}}`		

12	`class One {`
	`    int oneInt;`
	`    double f() { this.oneInt=3;}`
	`}`		

13	the sample Toy program given in project #1		
14	the sample Toy program given in project #1 but replace "`funny = 123456E+7;`" with "`funny = 123456.0E+7;`"		
15	design your own test cases including at least 5 other syntax rules not covered in the above		

Turn in materials:

Each group should make one Bb submission and turn in a hard copy of all the following files.

- A readme file that indicates the software version you downloaded and step-by-step instructions to compile and run your project
- A report as described above. Also include a Github URL of all source code
- Your program (both .l and .y files) with proper comments.
- Your program in y.tab.c containing code to print the [shift] actions. Please highlight the modified code.
- Input files: you must test your parser with the above 15 test cases in order to cover most of the language features of Toy.
- Output of executions, one for each input file, should include a sequence of tokens and parsing actions as shown in the example on page 2 of this document.

Grading criteria:

This project will be graded based on program correctness (60%), quality of the report and test case analysis and design (40%).

- All hard copies have the same due date as your Bb submission. No late submission/hardcopy will be accepted. If you cannot complete the project by the due date, then submit whatever you have completed. Partial credit will be given for reasonable partial solutions.

- Your project will not be graded if I cannot run your program based on the instructions you provide or if the turn-in material is not complete. The highest grade you can get is 80% of the earned score if your project requires to be graded the second time.

- Each person in a project team is expected to contribute a fair amount of the work involved. It is assumed that all documentation turned in for the project represents the best-combined effort of the team members and has been reviewed by all members of the team. Normally all members of a team receive the same grade on the project. If any team member does not do sufficient work to deserve equal credit, the project grade may be adjusted. Please see the instructor if you feel your project's grade should not be shared equally.

Presentation:

Each team should use 8-10 minutes to demonstrate the highlights of your .l and .y files with well-defined scenarios to show most of the functionality of your program. Discuss your test case design, your attempts and strategies to solve the S/R and R/R conflicts, and conclude your presentation with the strength and constraints of your work.