Project#2 Language

Description:

PROJECT#2 is a case sensitive objects oriented Language. A program in Project#2 consists of class declaration which includes variables declaration and sequence of Methods declarations. Each method consists in turn of variable declarations and statements. The types in Project#2 are very restricted as in table 1.

Review the token in scanner:

Keywords	Meaning	Return Token
Type	is the blueprint from which individual objects are created.	Class
DerivedFrom	Inheritance in oop	Inheritance
TrueFor—Else	conditional statements	Condition
Ity	Integer type	Integer
Sity	Signed Integer type	SInteger
Cwq	Character Type	Character
CwqSequence	Group of characters	String
Ifity	Float type	Float
Sifity	Signed Float type	SFloat
Valueless	Void Type	Void
Logical	Boolean type	Boolean
Endthis	Break immediately from a loop	Break

However/When	**repeatedly execute code as long as condition is true**	**Loop**
Respondwith	**Return a value from a function**	**Return**
Srap	**grouped list of variables placed under one name**	**Struct**
Scan –Conditionof	**To switch between many cases**	**Switch**
@| ^	**Program Starting Symbols**	**Stat Symbol**
$|#	**Program Ending Symbols**	**End Symbol**
(+, -, *, /,)	**Used to add, subtract, multiply and divide respectively**	**Arithmetic Operation**
(&&, ||, ~)	**Used to and, or and not repectively**	**Logic operators**
(==, <, >, !=, <=, >=)	**Used to describe relations**	**relational operators**
=	**Used to describe Assignment operation**	**Assignment operator**
->	**Used in Seop to access Seop elements**	**Access Operator**
{,},[,]	**Used to group class statements, statements or array index respectively**	**Braces**
[0-9] and any combination	**Used to describe numbers**	**Constant**
“,’	**Used in defining strings and single character reprctively**	**Quotation Mark**
Require	**Used to include one file in another**	**Inclusion**
/*	**Used to Comment some portion of code (Single Line)**	**Comment**
/<	**Used to Comment some portion of code (Multiple Lines)**	**Comment**
>/	**Used to a matcher to Comment left side (Multiple Lines)**	**Comment**

Comments in Project#2:

Project#2 includes multiple line comment is written between /< and >/ and single line comment is written as /* .Your parser must ignore all comments and white spaces.

Requiring file command:

In order to facilitate the using of multiple files, your Project #2 scanner/parser is also responsible for directly handling the using file command. When encountering the using command placing at the first column of a given line, the scanner/parser opens the file indicated by the file name in the command and start processing its contents. Once the included file has been processed the scanner/parser must return to processing the original file. An included file may also include another file and so forth. If the file name does not exist in the local directory you should simply ignore the using command and proceed with the tokens in the current file.

Tokens and return values:

You must build a dictionary to save Keywords that are defined in Project #2 language.

Project#2 Language Delimiters (words and lines):

The words are delimited by Space and tab. The line delimiter is semicolon (;) and newline.

Output format:

Scanner:

In case of correct token: Line #: (Number of line) Token Text: ---------- Token Type: ----------

In case of Error tokens: Line #: (Number of line) Error in Token Text: ---------- ----------

Total NO of errors: (NO of errors found)

Parser:

Firstly you must sate Scanner phase output as above then state Parser Phase output

In case of correct Statement: Line #: (Number of line) Matched Rule Used:----------

In case of Error: Line #: (Number of line) Not Matched

Total NO of errors: (NO of errors found)

Parser grammar rules:

1. Program -> Start_Symbols ClassDeclaration End_Symbols .
2. Start_Symbols -> @ | ^
3. End_Symbols -> $ | #
4. ClassDeclaration -> Type ID ClassBody
   | Type ID DerivedFrom ClassBody
5. ClassBody -> { ClassMembers }
6. ClassMembers -> ClassMember ClassMembers | ε
7. ClassMember -> VariableDecl
   | MethodDecl
   | FuncCall
   | Comment
   | RequireCommand
8. MethodDecl -> FuncDecl ;
   | FuncDecl { VariableDecls Statements }
9. FuncDecl -> Type ID (ParameterList)
10. ParameterList -> ε | Parameters
11. Parameters -> Parameter | Parameters , Parameter
12. Parameter -> Type ID
13. VariableDecl -> Type IDList ;
    | Type IDList [ID] ;
14. VariableDecls -> VariableDecl VariableDecls | ε
15. IDList -> ID | IDList , ID
16. Statements -> Statement Statements | ε
17. Statement -> Assignment
    | TrueForStmt
    | HoweverStmt
    | WhenStmt
    | RespondwithStmt

| EndthisStmt
| ScanStmt
| SrapStmt
| FuncCallStmt

18. Assignment -> ID = Expression ;
19. FuncCall -> ID (ArgumentList) ;
20. FuncCallStmt -> FuncCall ;
21. ArgumentList -> ε | ArgumentSequence
22. ArgumentSequence -> Expression | ArgumentSequence , Expression
23. TrueForStmt -> TrueFor (ConditionExpression) Block
| TrueFor (ConditionExpression) Block TrueForElse Block

24. TrueForElse -> Else
25. HoweverStmt -> However (ConditionExpression) Block
26. WhenStmt -> When (Expression ; Expression ; Expression) Block
27. RespondwithStmt -> Respondwith Expression ;
| Respondwith ID ;
28. EndthisStmt -> Endthis ;
29. ScanStmt -> Scan(Conditionof ID) ;
30. SrapStmt -> Srap (Expression) ;
31. Block -> { Statements }
32. ConditionExpression -> Condition
| Condition LogicalOp Condition
33. LogicalOp -> && | || | ~
34. Condition -> Expression ComparisonOp Expression
35. ComparisonOp -> == | != | > | >= | < | <=
36. Expression -> Term
| Expression AddOp Term
37. AddOp -> + | -
38. Term -> Factor

| Term MulOp Factor

39. MulOp -> * | /
40. Factor -> ID | Number | (Expression)
41. Comment -> /< STR >/ | /* STR
42. RequireCommand -> Require (F_name.txt) ;
43. F_name -> STR
44. Type -> Ity | Sity | Cwq | CwqSequence | Ifity | Sifity | Valueless | Logical

Sample Input and Output

Input:

```
1- @ Type Person{
2- Logical G() {
3- int frt=5;
4- when (in counter<num){
5- int reg3=reg3-1;
6- }
7- }
8- }
9- $
```

Scanner Output:

Line : 1 Token Text: @ Token Type: Start Symbol

Line : 1 Token Text: Type Token Type: Class

Line : 1 Token Text: Person Token Type: Identifier

Line : 1 Token Text: { Token Type: Braces

Line : 2 Token Text: Logical Token Type: Boolean

Line : 2 Token Text: G Token Type: Identifier

Line : 2 Token Text: (Token Type: Braces

Line : 2 Token Text:) Token Type: Braces

Line : 2 Token Text: { Token Type: Braces

----------Etc.

Total NO of errors: 0

Scanner and Parser Output:

Firstly you must sate Scanner phase output as in scanner sample input and output then state parser output based on scanner output

Line : 1 Matched Rule used: Program and ClassDeclaration

Line : 2 Matched Rule used: Func Decl

Line : 3 Not Matched Error: Invalid Type (int)

----------Etc.

Total NO of errors: 1