# Project #1 Language

## *Description:*

The language Project #1 is a case sensitive object-oriented language with support for inheritance and encapsulation. A program in Project #1 consists of a class definition which contains a sequence of function definitions. Each function consists in turn of variable declarations, type declarations, function declarations, and statements. The types in Project #1 are very restricted look at table 1.In addition ,there are arrays of Integers Ilap id[ n ] there are arrays of Booleans : Logical id‘[’ n ‘]’ where n is an integer. The array index value can only be simple unary expression such as an identifier, a constant or another simple array access expression. Access to structure types can be done using the “object dereferencing” operator (“.”).

## *Scanner:*

## *Lexical Analysis:*

Project #1 Scanner must recognize the following keywords and returns Return Token in table 1:

| Keywords | Meaning | Return Token |
|---|---|---|
| **Division** | **is the blueprint from which individual objects are created.** | **Class** |
| **InferedFrom** | **Inheritance in oop** | **Inheritance** |
| **WhetherDo-Else** | **conditional statements** | **Condition** |
| **Ire** | **Integer type** | **Integer** |
| **Sire** | **Signed Integer type** | **SInteger** |

| Clo | Character Type | Character |
|---|---|---|
| SetOfClo | Group of characters | String |
| FBU | Float type | Float |
| SFBU | Signed Float type | SFloat |
| None | Void Type | Void |
| Logical | Boolean type | Boolean |
| terminatethis | Break immediately from a loop | |
| Rotatewhen/Continuewhen | repeatedly execute code as long as condition is true | Loop |
| Replywith | Return a value from a function | Return |
| Seop | grouped list of variables placed under one name | Struct |
| Check –situationof | To switch between many cases | Switch |
| Program | Program Starting Statement | Stat Statement |
| End | Program Ending Statement | End Statement |
| (+, -, *, /,) | 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 |
| Using | 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 |

***Table 1: Tokens Description***

The Scanner also recognizes identifiers. An identifier is a sequence of letters and digits, starting with a letter. The underscore '_' counts as a letter. For each identifier, the Project #1 Scanner returns the token IDENTIFIER.

### *Comments in Project #1:*

Project #1 includes two types of comments single line comments are prefixed by /- and multiple line comment are written between /## and ##/ .Your scanner must ignore all comments and white spaces.

### *Using file command:*

In order to facilitate the using of multiple files, your Project #1 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 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 #1 language.

## *Project#1 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 → ClassDeclarationList End.*
2. ClassDeclarationList → ClassDeclaration ClassDeclarationList | ε
3. ClassDeclaration → Division ID { ClassImplementation }
   | Division ID InferredFrom { ClassImplementation }
4. ClassImplementation → ClassItem ClassImplementation | ε
5. ClassItem → VarDeclaration
   | MethodDeclaration
   | Comment
   | UsingCommand
   | FuncCall
6. MethodDeclaration → FuncDecl ;
   | FuncDecl { VarDeclaration Statements }
7. FuncDecl → Type ID ( ParameterList )
8. Type → Ire | Sire | Clo | SetOfClo | FBU | SFBU | None | Logical
9. ParameterList → ε | None | NonEmptyParameterList
10. NonEmptyParameterList → Type ID | NonEmptyParameterList , Type ID
11. VarDeclaration → ε | Type IDList ; VarDeclaration
12. IDList → ID | IDList , ID
13. Statements → ε | Statement Statements
14. Statement → Assignment
    | WhetherDoStatement
    | RotateWhenStatement
    | ContinueWhenStatement
    | ReplyWithStatement
    | TerminateThisStatement
    | read ( ID ) ;
    | write ( Expression ) ;
15. Assignment → VarDeclaration = Expression ;
16. FuncCall → ID ( ArgumentList ) ;
17. ArgumentList → ε | NonEmptyArgumentList
18. NonEmptyArgumentList → Expression | NonEmptyArgumentList , Expression
19. BlockStatements → { Statements }
20. WhetherDoStatement → WhetherDo ( ConditionExpression ) BlockStatements

21. ConditionExpression → Condition
| Condition ConditionOp Condition
22. ConditionOp → and | or
23. Condition → Expression ComparisonOp Expression
24. ComparisonOp → == | != | > | >= | < | <=
25. RotateWhenStatement → RotateWhen ( ConditionExpression ) BlockStatements
26. ContinueWhenStatement → ContinueWhen ( Expression ; Expression ; Expression ) BlockStatements
27. ReplyWithStatement → ReplyWith Expression ;
| return ID ;
28. TerminateThisStatement → TerminateThis ;
29. Expression → Term | Expression AddOp Term
30. AddOp → + | -
31. Term → Factor | Term MulOp Factor
32. MulOp → * | /
33. Factor → ID | Number
34. Comment → /## STR ##/ | /- STR
35. UsingCommand → using ( FName.txt ) ;
36. FName → STR

## *Sample Input and Output*

**Input:**

```
1- Program
2- Division x {
3-   W decrease() {
4-     Ire reg3 = 5;
5-     Continuewhen (counter < num ; reg3 ; 1) {
6-       reg3 = reg3 - 1;
7-     }
8-   }
9- }
10- End
```

**Scanner Output:**

Line : 1 Token Text: Program Token Type: Start Statement

Line : 2 Token Text: Division Token Type: Class

Line : 2 Token Text: x Token Type: Identifier

Line : 2 Token Text: { Token Type: Braces

Line : 3 Token Text: W Token Type: Identifier

Line : 3 Token Text: decrease Token Type: Identifier

Line : 3 Token Text: ( Token Type: Braces

Line : 3 Token Text: ) Token Type: Braces

Line : 3 Token Text: { Token Type: Braces

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

Total NO of errors: 0

**Scanner and Parser Output:**

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

Line : 1 Matched Rule used: Program
Line : 2 Matched Rule used: ClassDeclaration
Line : 3 Not Matched Error: 'W' is not a valid Type
Line : 4 Matched Rule used: VarDeclaration
Line : 5 Matched Rule used: ContinueWhenStatement

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

Total NO of errors: 1