Consider the following extended BNF grammar for a subset of the Scala programming language, called MicroScala.

```
CompilationUnit ::= object id { {Def} MainDef }
MainDef ::= def main ( args : Array [String ] ) { {VarDef} Statement {Statement} }
Def ::= def id ( [id : Type {, id : Type}] ) : Type =
   { {VarDef} {Statement} return ListExpr ; }
   | VarDef
VarDef ::= var id : Type = Literal ;
Type ::= Int | List [ Int ]
Statement ::= if ( Expr ) Statement [else Statement]
   | while ( Expr ) Statement
   | id = ListExpr ;
   | println ( ListExpr ) ;
   | { Statement {Statement} }
Expr ::= AndExpr {|| AndExpr}
AndExpr ::= RelExpr {&& RelExpr}
RelExpr ::= ![ ] ListExpr [RelOper ListExpr]
RelOper ::= < | <= | > | >= | == | !=
ListExpr ::= AddExpr | AddExpr :: ListExpr
AddExpr ::= MulExpr {AddOper MulExpr}
AddOper ::= + | −
MulExpr ::= PrefixExpr {MulOper PrefixExpr}
MulOper ::= * | /
PrefixExpr ::= [AddOper] SimpleExpr {ListMethodCall}
ListMethodCall ::= . head | . tail | . isEmpty
SimpleExpr ::= Literal ( Expr ) | id [ ( [ListExpr {, ListExpr}] ) ]
Literal ::= integer | Nil
```

**Syntactic and Semantic Conventions** The keywords and the token symbols in MicroScala are in bold. Note that MicroScala, like C, C++ and Java, has symbols {, }, [, and ] which are distinguished from grammar metasymbols {, }, [, and ], respectively, by underlining.

Assume that an id can only contain letters (only alphabetic characters), digits, and underscores (_) with the restrictions that it must begin with a letter, cannot end with an underscore and cannot have two consecutive underscores. For example, give_2_Joe, tell_me and A45Asm3 are valid identifiers, but 6gh, two_bad, and no_end_ are not. integer is an unsigned integer. Assume that comments are indicated by being preceded by //.

1. Determine the set of tokens which a lexical analyzer would need to recognize.

2. Design and implement a lexical analyzer procedure to read a source program in the above language and print the next token in the input stream. If the token detected is a valueless token, such as a keyword, then it is sufficient to print only the keyword. If it has a value, then both the token type and lexeme should be printed.
3. You will be given several MicroScala programs with which to test your lexical analyzer. These will be located on the class WWW page and will be of the form Test1.scala, Test2.scala, etc.

**Suggestions:**

1. Construct a token class to represent the token data structure, including a method to print a token.

2. Use the lex, flex, or JFlex tool to automatically construct a lexical analyzer in Scala from a set of regular expressions specifying tokens. This can interface with your token class to return a token to the main program which then prints the token.