

CS 405/505 ASSIGNMENT #1

Due Wednesday, February 6, 2008

Consider the following extended BNF grammar for a subset of the Ruby programming language, called MicroRuby.

```
program      ::= require 'list' { funcdef } function-identifier ( )
funcdef      ::= def function-identifier ( [ variable-identifier { , variable-identifier } ] )
               suite return add-expr end
suite        ::= statement { statement }
statement    ::= variable-identifier = add-expr
               | if or-test suite [else suite] end
               | while or-test suite end
               | puts add-expr
or-test      ::= and-test { || and-test }
and-test     ::= not-test { && not-test }
not-test     ::= comparison | ! ( not-test )
comparison   ::= add-expr comp-operator add-expr
comp-operator ::= < | > | == | >= | <= | !=
add-expr     ::= mult-expr { add-operator mult-expr }
add-operator ::= + | -
mult-expr    ::= unary-expr { mult-operator unary-expr }
mult-operator ::= * | /
unary-expr   ::= [add-operator] primary
primary      ::= variable-identifier | integer | ( add-expr ) | function-identifier ( [ add-expr-list ] )
               | list-expr | cons ( add-expr , add-expr )
               | head ( add-expr ) | tail ( add-expr )
list-expr    ::= [ [ add-expr-list ] ]
add-expr-list ::= add-expr { , add-expr }
```

Syntactic and Semantic Conventions The keywords and the token symbols in MicroRuby are in bold. Note that MicroRuby has symbols [and], which are distinguished from grammar metasymbols [and], respectively, by bold face type. A MicroRuby identifier (called a function-identifier or variable-identifier) is a name, which consists of letters (only alphabetic characters), digits, and underscores (`_`) with the restrictions that it must begin with a lowercase letter, cannot end with an underscore and cannot have two consecutive underscores. For example, `give_2_Joe`, `tell_me` and `a45Asm3` are valid bare names, but `Fgh`, `two__bad`, and `no_end_` are not. `integer` is an unsigned integer. MicroRuby comments are indicated by being preceded by `#` and terminated by the end of the line.

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 MicroRuby programs with which to test your lexical analyzer. These will be located on the class WWW page and will be of the form `test1.rb`, `test2.rb`, etc.

Suggestions:

1. Construct a token class to represent the token data structure, including a method to print a token.
2. Use the JFlex tool to automatically construct a lexical analyzer in Java 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.