

## CS 401/501 ASSIGNMENT #2

Due Tuesday, February 3, 2009

Consider the following extended BNF grammar for a subset of the Fortress programming language, called MicroFortress.

```
program ::= component component-identifier
         [ import MicroList.{ ... } ]
         export Executable
         { function-declaration } run-declaration
         end

function-declaration ::= function-identifier ( [ typed-var {, typed-var } ] ) : type =
                       function-do-expression

run-declaration ::= run ( args : String ... ) = do-expression

typed-var ::= variable-identifier : type

type ::= Int | List(\Int\)

function-do-expression ::= do block-elems ( add-expr ) end

do-expression ::= do block-elems end

block-elems ::= block-elem { block-elem }

block-elem ::= local-var-decl | expression

expression ::= do-expression
             | variable-identifier := ( add-expr )
             | if comparison then block-elems [else block-elems] end
             | while comparison do-expression
             | println ( add-expr )

local-var-decl ::= var typed-var

comparison ::= add-expr comp-operator add-expr | isEmpty ( primary )

comp-operator ::= < | > | = | >= | <= | !=

add-expr ::= mult-expr { add-operator mult-expr }

add-operator ::= + | -

mult-expr ::= primary { mult-operator primary }

mult-operator ::= ε | /

primary ::= variable-identifier | integer | ( add-expr )
          | function-identifier ( [ add-expr-list ] )
          | list-expr | cons ( add-expr , primary )
          | head ( primary ) | tail ( primary )

list-expr ::= < | [ add-expr-list ] | >

add-expr-list ::= add-expr {, add-expr }
```

**Syntactic and Semantic Conventions** The keywords and the token symbols in MicroFortress are in bold. Note that MicroFortress has symbols { and }, which are distinguished from grammar metasympols { and }, respectively, by underlining. MicroFortress also has symbol combinations [\, \], <, and > which should be treated as individual tokens. A MicroFortress identifier consists of 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 bare names, but 6gh, two\_bad, and no\_end\_ are not. integer is an unsigned integer. MicroFortress 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 MicroFortress programs with which to test your lexical analyzer. These will be located on the class WWW page and will be of the form Test1.fss, Test2.fss, 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.