CP471, Winter 2020

ASSIGNMENT 1 (individual): Lexical analysis

DUE: Monday, January 27. (10% of the final grade)

Consider the grammar on page two. Your task in this assignment is to complete the definition of the grammar, and to implement lexical analysis phase of your future compiler.

Minimal implementation requirements:

- you implementation should provide lexical analysis functionality implemented as function (method) `getNextToken` (`main` in your implementation calls `getNextToken` in a loop until input is over);
- it should read source program from standard input character by character (DO NOT READ FROM A FILE!!!) and write the output to the standard output (DO NOT WRITE TO A FILE!!!);
- the output is a sequence of tokens recognized by you analyser and HTML encoding of source program with visually recognizable lexemes from the input (use color and font facilities at your discretion, but try to use different colors for different classes of lexemes);
- sequence of tokens is to be encoded in HTML as a comment;
- lexical error should not prevent your code from completing analysis (lexemes with errors must be displayed in red color using bold face font);
- test your implementation on test cases provided in the assignment entry on `MyLearningSpace` but also create your own tests.

What and how to submit?

Submission contains two parts: documentation of lexical analyser diagrams and implementation.

- Submit documentation as PDF file in A1 drop-box on MyLearningSpace.
- Submit implementation in A1 drop-box on MyLearningSpace.
- Submit source code only in a single Java or C file. It should be possible to compile your submission with `javac` or `gcc`.

```
<program> ::= <fdecls> <declarations> <statement_seq>.

<fdecls> ::= <fdec>; | <fdecls> <fdec>; |
<fdec> ::= def <type> <fname> ( <params> ) <declarations> <statement_seq> fed
<params> ::= <type> <var> | <type> <var> , <params> |
<fname> ::= <id>

<declarations> ::= <decl>; | <declarations> <decl>; |
<decl> := <type> <varlist>
<type> := int | double
<varlist> ::= <var>, <varlist> | <var>

<statement_seq> ::= <statement> | <statement>; <statement_seq>

<statement> ::= <var> = <expr> |
        if <bexpr> then <statement_seq> fi |
        if <bexpr> then <statement_seq> else <statement_seq> fi |
        while <bexpr> do <statement_seq> od |
        print <expr> |
        return <expr> |

<expr> ::= <expr> + <term> | <expr> - <term> | <term>
<term> ::= <term> * <factor> | <term> / <factor> | <term> % <factor> | <factor>
<factor> ::= <var> | <number> | (<expr>) | <fname>(<exprseq>)
<exprseq> ::= <expr>, <exprseq> | <expr> |

<bexpr> ::= <bexpr> or <bterm> | <bterm>
<bterm> ::= <bterm> and <bfactor> | <bfactor>
<bfactor> ::= (<bexpr>) | not <bfactor> | (<expr> <comp> <expr>)

<comp> ::= < | > | == | <= | >= | <>

<var> ::= <id> | <id>[<expr>]

<letter> ::= a | b | c | ... | z
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<id> ::= <letter> | <id><letter> | <id><digit>

<number> ::= <integer> | <double>
...
```