# Lexical Analyzer for Toy program
# Instructions

You are to implement a lexical analyzer for a simple object-oriented programming language called Toy. Your program should be able to (1) translate any input Toy program into a sequence of tokens, and (2) create a symbol table using the trie structure for all keywords and user-defined identifiers.

Lexical Conventions of Toy:

1. Keywords which are also reserved words. They cannot be used as identifiers or redefined:

```
boolean break class double else extends false for if
implements int interface new newarray null println readln
return string true void while
```

2. An identifier is a sequence of letters, digits, and underscores, starting with a letter. Toy is case-sensitive, e.g., `if` is a keyword, but `IF` is an identifier; `hello` and `Hello` are two distinct identifiers.

3. White space consists of blanks, newlines, and tabs. White space serves to separate tokens, but is otherwise ignored. Keywords and identifiers must be separated by white space or a token that is neither a keyword nor an identifier. For instance, `ifintvoid` is a single identifier, not three keywords; while `if(23void` scans as four tokens.

4. An integer constant can either be specified in decimal (base 10) or hexadecimal (base 16). A decimal integer is a sequence of at least one decimal digits (0-9). A hexadecimal integer must begin with 0X or 0x (that is a zero, not the letter oh) and is followed by a sequence of at least one hexadecimal digits. Hexadecimal digits include the decimal digits and the letters a through f (either upper or lowercase). Examples of valid integers: 8, 012, 0x0, 0X12aE.

5. A double constant is a sequence of at least one digit, a period, followed by any sequence of digits, may be none. Thus, .12 is not a valid double but both 0.12 and 12. are valid. A double can also have an optional exponent, e.g., 12.2E+2 is a double constant. For a double in this sort of scientific notation, the decimal point is required, the sign of the exponent is optional (if not specified, + is assumed), the E can be lower or upper case, and at least one digit after E or e. As above, .12E+2 and 1.2E are invalid, but 12.E+2 is valid. Leading zeroes on the mantissa and exponent are allowed.

6. A string constant is a sequence of characters enclosed in double quotes. Strings can contain any character except a newline or double quote. A string must start and end on a single line, it cannot be split over multiple lines.

7. A Boolean constant is either `true` or `false`.

8. Operators and punctuation characters include:

```
+ - * / % < <= > >= == != && || ! = ; , . ( ) [ ] { }
```

9. A single-line comment is started by // and extends to the end of the line. Multi-line comments start with /* and end with the first subsequent */. Any symbol is allowed in a comment except the sequence */ which ends the current comment. Multi-line comments do not nest.

How to define tokens?

A Token is a simple syntactic unit defined based on the lexical rules of the language. The set of tokens should consist of grammar elements of the languages including keywords, operators, special symbols, identifiers, and constants like integer literals, double literals, boolean literals, and string literals. Below is the list of tokens for Toy:

_boolean, _break, _class, _double,
_else, _extends, _for, _if,
_implements, _int, _interface, _new,
_newarray, _null, _println, _readln,
_return, _string, _void, _while,
_plus, _minus, _multiplication, _division,
_mod, _less, _lessequal, _greater,
_greaterequal, _equal, _notequal, _and,
_or, _not, _assignop, _semicolon,
_comma, _period, _leftparen, _rightparen,
_leftbracket, _rightbracket, _leftbrace, _rightbrace,
_intconstant, _doubleconstant, _stringconstant, _booleanconstant,
_id

Note that tokens must be represented as integers in your lexer, which will be passed as a parameter to the parser of your 2nd project. However, ordinal numbers are not readable. Therefore, your 1st project ought to print the corresponding string names of those ordinal numbers for verification.

For instance, a sample Toy program is as follows:

```
int fact (int x) {
// recursive factorial function
    if (x>1) return x * fact(x-1);
    else return 1;
}
void main () {
/* Fall Semester 2018
CS 4110 project #1
A lexical analyzer */
    int x;
    int total;
    println (“factorial of 10 is ”, fact (10), “ from the recursive function”);
    total = 1; x = 1;
    for ( ; x<=10; ) { total = total * x; x = x + 1; }
    println (“iterative result of 10! is ”, total);
}
class cs4110 {
    int Funny;
    double funny;
    boolean flag;
    string s;
    int [] a;
    flag = true;
    Funny = 0X89aB; funny = 123456E+7;
    s = “hello world”;
    while (x = (Funny/10) <0) println (s, “ have fun !”);
    a = newarray (20, int);
}
```

The output of your project should be:

```
int id leftparen int id rightparen leftbrace
if leftparen id greater intconstant rightparen return id multiplication id leftparen id minus
intconstant rightparen semicolon
else return intconstant semicolon
rightbrace
void id leftparen rightparen leftbrace
int id semicolon
int id semicolon
println leftparen stringconstant comma id leftparen intconstant rightparen comma stringconstant
rightparen semicolon
……….
class id leftbrace
int id semicolon
double id semicolon
boolean id semicolon
string id semicolon
……….
```

You must use the following data structure – trie (retrieval) to store keywords and identifiers, as part of your symbol table. (The complete symbol table will be explained later.)

```
switch  : array [0..51] of integer;
symbol  : array [0..maxtransition] of char;
next    : array [0..maxtransition] of integer;
```

Algorithm: Search and Create identifiers in Trie

```
valueOfSymbol = getNextSymbol();
ptr = switch [valueOfSymbol];
if ptr is undefined then Create() // new identifier
else {
    valueOfSymbol = getNextSymbol();
    exit = false;
    while not exit {
        if (symbol [ptr] == valueOfSymbol)
        then if valueOfSymbol is not the endmarker
             then { ptr = ptr + 1;
                    valueOfSymbol = getNextSymbol(); }
             else { exit = true; }
        else if next [ptr] is defined
            then ptr = next [ptr]
            else { Create(); exit = true; } // new identifier
    } //while
} //if
```

```
          A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T
switch:  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1

          U   V   W   X   Y   Z   a   b   c   d   e   f   g   h   i   j   k   l   m   n
switch:  -1  -1  -1  -1  -1  -1  -1   0  12  17  23  34  -1  -1  42  -1  -1  -1  -1  -1

          o   p   q   r   s   t   u   v   w   x   y   z
switch:  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1


          0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
symbol:   o   o   l   e   a   n   *   r   e   a   k   *   l   a   s   s   *   o   u   b
next:     7

         20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39
symbol:   l   e   *   l   s   e   *   x   t   e   n   d   s   *   a   l   s   e   *   o
next:                27                                          39


         40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
symbol:   r   *   f   *   m   p   l   e   m   e   n   t   s   *   n   t   *   e   r   f
next:                44      54                                              57

...
```