Compiler project tasks — part 1, due Friday 7 September

Below you will find a semiformal description of the lexical structure of our source language. You are to write a scanner. The scanner will be invoked from the command line as follows: `scan <filename>`; for example, `scan primes.m`.

The input is presented as a file and it is a sequence of 1-byte characters (as opposed to Unicode characters, for instance). The output should be written to a listing file, for example, `primes.tokens`.

The listing should have a line for each lexical element (token, comment, or white space) giving its category, its position in the source file (line and column index), its literal text, and, for tokens with meaning, its value as it will be presented to the parser.

In addition to this printed output, you should construct tokens internally, in a form suitable to be passed to the parser (which you will write in the next part of the project). Details will depend on your choice of implementation language and strategy.

Lexical structure of the source language

Tokens with meaning

- *identifier* **of string** = *alpha* (*alpha*|*digit*|**_**)*
- *text* **of string** = **"** *character*+ **"**
- *integer* **of int** = *digit*+
- *real* **of real** = (*digit*+ **.** *digit*+) (ε | (**e**|**E**)(ε|**+**|**-**)*integer*)

Real numbers in our language are IEEE 754 double-precision numbers.

Keywords (exceptions from *identifier*)

`array begin by const do else elsif end eval exit for if loop module new nil of procedure read record ref repeat return then to type typerec until value var while write or and not div mod`

Keywords can also be written in all-uppercase.

Punctuation

colon = **:**	*less* = **<**	*equal* = **=**	*greater* = **>**
nequal = **#**	*le* = **<=**	*ge* = **>=**	*plus* = **+**
minus = **-**	*star* = *****	*lpar* = **(**	*rpar* = **)**
lbrack = **[**	*rbrack* = **]**	*dot* = **.**	*comma* = **,**
semi = **;**	*caret* = **^**	*dotdot* = **..**	*coloneq* = **:=**

Comments

Comments are enclosed in **(*** ⋯ ***)** pairs, and may be nested.

White space

White space between tokens consists of blanks, tabs, and new lines.

Example

```
1   MODULE Primes;
2
3   PROCEDURE IsPrime(Number: INTEGER): BOOLEAN =
4   BEGIN
5     FOR Count := 3 TO Number DIV 2 BY 2 DO
6       IF (Number MOD Count) = 0 THEN
7         RETURN(FALSE);
8       END;
9     END;
10
11    RETURN(TRUE);
12  END IsPrime;
13
14  PROCEDURE PrintPrimes(Max: INTEGER) =
15  VAR
16    Count : INTEGER;
17  BEGIN
18    WRITE("2");
19    FOR Count := 3 TO Max BY 2 DO
20      IF (IsPrime(Count)) THEN
21        WRITE(", %d",Count);
22      END;
23    END;
24    WRITE ("\n");
25  END PrintPrimes;
26
27  VAR
28    MaxPrime: INTEGER;
29  BEGIN
30    WRITE("Enter max: ");
31    READ("%d\n",MaxPrime);
32
33    PrintPrimes(MaxPrime);
34  END Primes.
```