Description of Project 1

Consider the grammar provided in file `grammar.txt`.

Implement a lexical analyzer (a.k.a. scanner, lexer or tokenizer) for this mini-language. The scanner reads a file containing a sequence of characters and outputs the corresponding sequence of tokens (i.e., representations of terminal symbols), while omitting whitespace and comments. If it reads any character that is not allowed by the grammar, it should generate an appropriate error message (with the position of the character) and stop the computation. For instance, the scanner should reject “$um”, as “$” is not included in the alphabet of identifiers.

For simplicity, you can assume that all integers are within range, i.e., you do not have to check for overflow.

Each token should include the following information:

1. The kind of the current lexeme. To keep things simple, use strings for representing the kind of lexemes. For identifiers and integers, the kinds are “ID” and “NUM”, respectively. For keywords, the keyword itself is the kind. For other symbols, the kind is a string corresponding to the symbol. Upon encountering the end of the input file, the scanner must generate a token whose kind is “end-of-text”. (For example, for identifier `speed`, the kind is “ID”. For the integer 3400, the kind is “NUM”. For the keyword `false`, the kind is “false” and for the symbol `:=` the kind is “:=“.)

2. The value of the lexeme, if applicable. Only identifiers and integers have values. For example the value of lexeme `19` is 19 and the value of lexeme `speed` is “speed”. However, the lexeme `(` does not have any value.

3. The position of the lexeme. The position is a pair consisting of the line number of the lexeme and the position of the first character of the lexeme in that line.

Your program should provide four procedures (that will be called by the parser in the next project):

next() : reads the next lexeme in the input file.

kind() : returns the kind of the lexeme that was just read.

value() : returns the value of the lexeme (if it is an “ID” or a “NUM”).

position() : returns the position of the lexeme that was just read.

The main program should have a loop for reading the input and printing the tokens:

```
        next();
        while ( kind() != “end-of-text” )
        {
                next();
                print( position(), kind(), value() );
        }
```

Note: You are not allowed to use any library or module that supports regular expressions.

Examples of such libraries/modules are re of Python and regex of Java or C++.