Parser Assignment

program → *stmt-sequence*
stmt-sequence → *stmt-sequence* **;** *statement* | *statement*
statement → *if-stmt* | *repeat-stmt* | *assign-stmt* | *read-stmt* | *write-stmt*
if-stmt → **if** *exp* **then** *stmt-sequence* **end**
| **if** *exp* **then** *stmt-sequence* **else** *stmt-sequence* **end**
repeat-stmt → **repeat** *stmt-sequence* **until** *exp*
assign-stmt → **identifier** **:=** *exp*
read-stmt → **read identifier**
write-stmt → **write** *exp*
exp → *simple-exp comparison-op simple-exp* | *simple-exp*
comparison-op → **<** | **=**
simple-exp → *simple-exp addop term* | *term*
addop → **+** | **-**
term → *term mulop factor* | *factor*
mulop → ***** | **/**
factor → **(** *exp* **)** | **number** | **identifier**

Description

- Given the TINY grammar rules you should implement the TINY parser using ***recursive descent*** method.
- You will need to convert grammar into **EBNF** form as we did before.
- The parser expects the terminals in the input to be as in rules. For example, it would expect to statement like ***x:=5*** as ***identifier := number*** so this means that after scanner identifies the tokens it puts them into a file with the same order but in terms of the parser terminals.
- You can first implement the parser and test it using code written in what it expects. Then, you can try to integrate it with the scanner.
- Note that it is not the parser job to identify whether the input is number, identifier, … so I expect to see two programs scanner and parser and not one do everything.
- There is a complete C code for arithmetic grammar in Chapter 4 you can try to use it as a guide.
- You should recognize each type of structure withing the program as you proceed with parsing. (i.e. recognition will not be in the same order of code)

Implementation

- Scanner should be implemented in **C, C++, or Java** only.
- Your program should read the input from a file containing same sample TINY code used generated from the scanner and file is supposed to be generated in the same directory of the program and adjust the path in the code accordingly (make a relative path so that I can test without modifying your code).
- The output should be saved to a file named **"parser_output.txt"** or print it to console.

Sample Input File to Scanner

```
{sample program in TINY language- computes factorial}
read x;{input an integer}
if 0<x then {don't compute if x<=0}
        fact:=1;
        repeat
                fact:=fact*x;
                x:=x-1
        until x=0;
        write fact{output factorial of x}
end
```

Sample Input File to Parser

```
read identifier ;
if number < number then
        identifier := number ;
        repeat
        ...
```

Sample Output File

```
Assignment found
…

If statement found
…
Program found
```

Deadline

26 April