2 The parser

Write a parser for the following context free grammar based on the tokens (e.g., `id`, `lparen` etc.) given in project 1.

```
<program> → <stmt_list> $$
<stmt_list> → <stmt> <stmt_list> | ϵ
<stmt> → id assign <expr> | read id | write <expr>
<expr> → <term> <term_tail>
<term_tail> → <add_op> <term> <term_tail> | ϵ
<term> → <factor> <fact_tail>
<fact_tail> → <mult_op> <factor> <fact_tail> | ϵ
<factor> → lparen <expr> rparen | id | number
<add_op> → plus | minus
<mult_op> → times | div
```

Note that `$$` is a pseudo-token (i.e., not a real token in the input program) which is returned by the scan function when it hits the end of the input file and there is no lexical error.

For the parser,

For the parser,

1. its input is a program (a text file).
2. it prints to the console an XML-like tree structure indicating the parse tree of the input program. If the parser meets an unexpected token, your parser should output to the console the following information only and stops: `Error`. Error recovery and handling multiple errors are beyond the scope of this project.
3. you MUST use the recursive descent approach.
4. `scan` function developed for the scanner in project 1 must be used. **Note** the output of `scan` is one token instead of a sequence of tokens.

One way to generate the XML-like tree is as follows. At the beginning of each function for a non-terminal symbol, print to a string buffer a new line containing `[indent]<[Non-Terminal]>`. Before ending the function, print to the buffer a new line of `[indent]</[Non-Terminal]>`. The `indent` is the spaces one should keep before the `Non-Terminal` symbol. When a token is recognized you will print to the buffer `[indent]<[Token-Type]>[Token]</[Token-Type]>`.

Commandline: `parser <input file name>`

Assume the input file has the following program

```
read A
```

Example Output:

```
<Program>
  <stmt_list>
    <stmt>
        <read>
            read
        </read>
        <id>
            A
        </id>
    </stmt>
```

```
    <stmt_list>
    </stmt_list>
  </stmt_list>
</Program>
```

Note. Here, you output both token type and its content. For example, the `id` token type here has a content `A`.