1 Grammar

$$
\begin{array}{rcl}
\mathit{stmt\_list} & \rightarrow & \mathit{stmt}\ \ ;\ \ \mathit{stmt\_list}\ \ \{\ \ \mathit{stmt\_list} \mathrel{+}= \mathit{stmt};\ \ \} \\
 & | & \mathit{stmt}\ \ \{\ \ \mathit{stmt\_list} = \mathit{stmt};\ \ \} \\
 & & \\
\mathit{stmt} & \rightarrow & \textbf{id}\ \ :=\ \ \mathit{expr}\ \ \{\ \ \mathit{stmt} = \text{new Assign}(\mathit{id}, \mathit{expr});\ \ \} \\
 & | & \textbf{if}\ \ \mathit{expr}\ \ \textbf{then}\ \ \mathit{stmt}\ \ \{\ \ \mathit{stmt} = \text{new If}(\mathit{expr}, \mathit{stmt});\ \ \} \\
 & | & \textbf{while}\ \ \mathit{expr}\ \ \textbf{do}\ \ \mathit{stmt}\ \ \{\ \ \mathit{stmt} = \text{new While}(\mathit{expr}, \mathit{stmt});\ \ \} \\
 & | & \textbf{begin}\ \ \mathit{opt\_stmts}\ \ \textbf{end}\ \ \{\ \ \mathit{stmt} = \mathit{opt\_stmts};\ \ \} \\
 & & \\
\mathit{opt\_stmts} & \rightarrow & \mathit{stmt\_list} \\
 & | & \varepsilon \\
 & & \\
\mathit{expr} & \rightarrow & \mathit{term}_1\ \ \mathit{moreterms} \\
\mathit{moreterms} & \rightarrow & +\ \ \mathit{term}_2\ \ \{\ \ \mathit{expr} = \text{new Op}(\text{`}+\text{'}, \mathit{term}_1, \mathit{term}_2);\ \ \}\ \ \mathit{moreterms} \\
 & | & -\ \ \mathit{term}_2\ \ \{\ \ \mathit{expr} = \text{new Op}(\text{`}-\text{'}, \mathit{term}_1, \mathit{term}_2);\ \ \}\ \ \mathit{moreterms} \\
 & | & \varepsilon \\
\mathit{term} & \rightarrow & \mathit{factor}_1\ \ \mathit{morefactors} \\
\mathit{morefactors} & \rightarrow & *\ \ \mathit{factor}_2\ \ \{\ \ \mathit{term} = \text{new Op}(\text{`}*\text{'}, \mathit{factor}_1, \mathit{factor}_2);\ \ \}\ \ \mathit{morefactors} \\
 & | & /\ \ \mathit{factor}_2\ \ \{\ \ \mathit{term} = \text{new Op}(\text{`/'}, \mathit{factor}_1, \mathit{factor}_2);\ \ \}\ \ \mathit{morefactors} \\
 & | & \textbf{div}\ \ \mathit{factor}_2\ \ \{\ \ \mathit{term} = \text{new Op}(\text{`DIV'}, \mathit{factor}_1, \mathit{factor}_2);\ \ \}\ \ \mathit{morefactors} \\
 & | & \textbf{mod}\ \ \mathit{factor}_2\ \ \{\ \ \mathit{term} = \text{new Op}(\text{`MOD'}, \mathit{factor}_1, \mathit{factor}_2);\ \ \}\ \ \mathit{morefactors} \\
 & | & \varepsilon \\
\mathit{factor} & \rightarrow & (\ \ \mathit{expr}\ \ )\ \ \{\ \ \mathit{factor} = \mathit{expr};\ \ \} \\
 & | & \textbf{id}\ \ \{\ \ \mathit{factor} = \text{new Id}(\textbf{id}.\mathit{lexeme});\ \ \} \\
 & | & \textbf{num}\ \ \{\ \ \mathit{factor} = \text{new Num}(\textbf{num}.\mathit{value});\ \ \}
\end{array}
$$

2 Requirement

With respect to the grammar above, write a syntax-directed-translation program[1]. The final outputs are to be a form of pseudo machine-code strings (i.e. stack-machine code), though. It is ultimately a part of our works in that understanding overall constructs and data flows is objective through implementation. Performing this assignment could be divided roughly into a few phases as followings:

1. implementing a scanner (i.e. taking care of lexcial matters)
2. writing a parser and a pretty-printer for each construct as its action
3. building AST and its pretty-printer as well (TBD)
4. generating pseudo machine-codes (TBD)

In order to conduct this task, there are some prerequisites.

1. It is required to understand the whole contents of the chapter two in advance. In a strict sense, it is, however, not required to have deep knowledge about any Compiler stuff. It's just the mechanism of a state machine. It could be enough to be capable of reading and understanding the grammar ROUGHLY.
2. all **bold** words are *terminal*, any symbol outside of curly braces, such as ':=', is also the *terminal* symbol, and every following curly braces contains actions (i.e. semantics). They, therefore, imply particular meanings individually in accordance with each production in the grammar defined above.
3. referring to source codes in the chapter two is helpful to write the program.

If you found some possible error(it's possible to be), then please let me know and correct it for you to proceed this task.

[1] it is the simple syntax directed translation in the chapter two.