3. (45 points) Language **L** is simplified propositional logic for the purpose of writing logical formulas as programs. In this question you will design an LL parser for it.

   It has two connectives ⇒, ¬ (implication and negation), to build complex logical formulas. Both connectives are right-associative.

   Proposition names are single letters (proposition letters).

   `1` (true) and `0` (false) are constant formulas of **L**.

   Negation has higher precedence than implication. Formulas can be parenthesized, which has the highest precedence.

   An **L** program is a sequence of definitions in the form of

   ```
    x = formula
   ```

   followed by a single `formula`, which is executed.

   `x` is a name standing for the entire formula on the right hand side, and `formula` is a propositional formula.

   The use of a propositional letter on the right side of a definition is meant to be substituted with its definition during execution. For example

   ```
    x = 0 => 1
    y = x => 1
    ¬y => x => 1
   ```

   executes the propositional formula `¬((0 => 1) => 1) => ((0 => 1) => 1)` as a program.

   i) Show your FIRST and FOLLOW sets for the grammar you designed for **L**.

   ii) Design an LL parsing table for **L**, with one-symbol lookahead. Assume that a tokenizer returns atomic tokens.

   [hint: 4 grammar variables (yes, four) are enough to describe a grammar of **L**.]

   iii) Show the sequential derivation of the example above using your LL table.