BNF

```
<program> ::= program <identifier> <compound statement>
<compound statement>::= begin <statement'> end
<statement'> ::= <statement><statement'> | <statement>
<statement> ::= <conditional statement> | <simple statement> ;
<conditional statement>::= if <expression> <compound statement> <else_if statement> |
                           if <expression> <compound statement> <else_if statement> else <compound statement>
<else_if statement>::= else_if <expression> <compound statement> <else_if statement> | ε
<Simple statement>::= <assigument statement> | <print statement> | <declaration statement>
<assigument statement> ::= <identifier> = <expression>
<print statement>::= print_line ( <string literal> )
<declaration statement>::= <type> <variable declaration'>
<variable declaration'>::= , <variable declaration> <variable declaration'> | <variable declaration>
<Variable declaration> ::= <identifier> | <identifier> = <expression>
<identifier> ::= identifier
<expression> ::= <simple expression> | <simple expression> <relational operator> <simple expression>
<simple expression> ::= <simple expression> <adding operator> <term> | <term>
<term> ::= <term> <multiplying operator> <factor> | <factor>
<factor> ::= <identifier> | <number literal> | ( <expression> )
<relational operator> ::= < | > | =
<adding operator> ::= + | -
<multiplying operator> ::= * | /
<string literal> ::= string_literal
<number literal> ::= number_literal
<type> ::= int
```

# EBNF

```
        <program> ::= program <identifier> <compound statement>
<compound statement> ::= begin {<statement>} <statement> end
      <statement> ::= <conditional statement> | <simple statement> ;
<conditional statement> ::= if <expression> <compound statement>
                       {else_if <expression> <compound statement>}
                       [ else <compound statement> ]
<simple statement> ::= <assignment statement> | <print statement> | <declaration statement>
<assignment statement> ::= <identifier> = <expression>
<print statement> ::= print_line ( <string literal> )
<declaration statement> ::= <type> {<variable declaration> ,} <variable declaration>
<variable declaration> ::= <identifier> [ = <expression> ]
         <identifier> ::= identifier
       <expression> ::= <simple expression> [ <relational operator> <simple expression> ]
<simple expression> ::= {<term> <adding operator>} <term>
          <term> ::= {<factor> <multiplying operator>} <factor>
        <factor> ::= <identifier> | <number literal> | ( <expression> )
<relational operator> ::= ≤ | ≥ | =
<adding operator> ::= + | -
<multiplying operator> ::= * | /
<string literal> ::= string_literal
<number literal> ::= number_literal
        <type> ::= int
```