Alphabet:

    a. Upper (A-Z) and lower case letters (a-z) of the English alphabet

    b. Underline character "\_";

    c. Decimal digits (0-9);

Lexic:

     a. operator:

        arithmetic: +, -, \*, /, %;

        assignment: =;

        boolean logic: and, or, not;

        comparison: ==, !=, >=, <=, >, <;

    b. separators:

        [], {}, :, ;, space

    c. reserved words:

        break continue pass do if else while return start print read

        Integer String Float Boolean

    d. identifiers:

a sequence of letters and  digits, such that the first character is a letter;

the rule is:

            identifier ::= letter | letter{letter}{digit}

            letter ::= "A" | "B" | . ..| "Z" | "a" ... | "z"

            digit ::= "0" | "1" |...| "9"

    e. constants:

        integer:

            integer ::= "0" | ["+"|"-"] ("1"|"2"|...|"9") {"0"|"1"|"2"|...|"9"}

        char:

            char ::= """ alphanumeric """

        float:

            float ::=

                ["+" | "-"] ("1"|"2"|...|"9") rest\_of\_the\_float |

                ("1"|"2"|...|"9") rest\_of\_the\_float |

                "0"

            rest\_of\_the\_float ::= {digit} ["." digit {digit}]

            alphanumeric = letter | digit

        string:

            string ::= {("""alphanumeric""" | " ")}

Syntax:

    The words - predefined tokens are specified between " and ":

    program ::= "start" declaration\_list ";" compound\_statement "."

    declaration\_list ::= declaration | declaration ";" declaration\_list

    declaration ::= type identifier

    compound\_type ::= "Integer" | "Char" | "Float" | "String"

    array\_declaration ::= type1 "[" number\_of\_elements "]"

    number\_of\_elements ::= integer

    array\_access ::= identifier "[" number | identifier "]"

    type  ::= type1 | array\_declaration

    compound\_statement ::= "{" statement\_list "}"

    statement\_list ::= statement | statement statement\_list

    statement ::= simple\_statement ";" | structured\_statement

    simple\_statement ::= assignment\_statement | IO\_statement

    assignment\_statement ::= identifier "=" expression

    expression ::= expression "+" term | term

    term ::= term operand [factor | array\_access]  | factor

    factor ::= "(" expression ")" | identifier

    IO\_statement ::= ["read" | "write"] "(" identifier ")"

    structured\_statement ::= compound\_statement | if\_statement | while\_statement

    if\_statement ::= "if" condition compound\_statement "else" compound\_statement

    while\_statement ::= "while" condition compound\_statement

    condition ::= "(" expression relation expression ")"

    relation ::= "<" | "<=" | "==" | "!=" | ">=" | ">"

    operand ::= "+" | "-" | "\*" | "/"

Tokens:

|  |  |
| --- | --- |
| Token type | code |
| identifier | 0 |
| constant | 1 |
| [ | 2 |
| ] | 3 |
| { | 4 |
| } | 5 |
| ; | 6 |
| : | 7 |
| , | 8 |
| < | 9 |
| <= | 10 |
| > | 11 |
| >= | 12 |
| = | 13 |
| ! | 14 |
| not | 15 |
| and | 16 |
| or | 17 |
| read | 18 |
| write | 19 |
| + | 20 |
| - | 21 |
| \* | 22 |
| % | 23 |
| / | 24 |
| break | 25 |
| start | 26 |
| Char | 27 |
| continue | 28 |
| do | 29 |
| else | 30 |
| float | 31 |
| for | 32 |
| while | 33 |
| if | 34 |
| pass | 35 |
| return | 36 |
| Integer | 37 |
| String | 38 |
| Float | 39 |
| Boolean | 40 |