Minilanguage specification

Alphabet:

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

b. Decimal digits(0-9)

c. Symbols: _ + - ! ? : . / * = & # < > = ;%

1. Lexic:

a. Special symbols, representing:

operators: + - * / < > <= >= != <- << >> % !

separators [] {} : ; space newline

reserverd words: program main declarations statements integer string boolean character array const and or in out if else then for while

b. indentifiers: a sequence of letters digits and underscores, such that the first character is a letter and the last character is not underscore; the rule is:

identifier::=letter{letter|digit} | (letter{letter|digit|_}letter|digit)

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

digit::="0"|non_zero_digit

non_zero_digit::="1"|"2"|...|"9"

c. constants:

1. integer - rule:

integer::=("+"|"-")?non_zero_number|0

non_zero_number::=non_zero_digit{digit}

2. character - rule:

character::=""letter""|""digit""

3. string - rule:

string::=""{char}""

char::=letter|digit|symbol

4. boolean - rule:

boolean="true"|"false"

2. Syntax:

The words - predefined tokens are specified between " and ", and they are case insensitive:

program::="program" "main" "->" "{" ("declarations" declarations_list ";")? ("statements" statements)? "}"

declarations_list::=declaration | declaration ";" declarations_list

declaration::=simple_declaration | array_declaration | const_declaration

simple_declaration::=symple_type ":" list_identifiers

simple_type::="character"|"integer"|"boolean"|"string"

list_identifiers::=(identifier|initialized_identifier) | (identifier|initialized_identifier) "," list_identifiers

initialized_identifier::=identifier "<-" constant

constant::=integer|character|boolean|string

const_declaration::=simple_type "const" ":" list_const_identifiers

list_const_identifiers::=initialized_const|initialized_const "," list_const_identifiers

initialized_const::=const_identifier "<-" constant

array_declaration::="array" "[" simple_type "]" ":" list_array_identifiers

list_array_identifiers::=array_identifier|array_identifier "," list_array_identifiers

array_identifier::=identifier "[" dimension "]"
dimension::=const_identifier|constant

statements::=statement|compound_statement

statement::=simple_statement|struct_statement

simple_statement::=assign_statement|read_statement|write_statement

assign_statement::=(identifier|array_element) "<-" expression
expression::=expression ("+"|"-") term | term
term::=term ("*" | "/") factor| factor
factor::="(" expression ")" | identifier | constant | const_identifier | array_element

read_statement::="in" ">>" list_identifiers | "in" "(" string ")" ">>" list_identifiers
list_identifiers::=identifier|array_element|identifier "," list_identifiers

write_statement::="out" "<<" list_outputs
list_outputs::=output | ouput "," list_outputs
output::=identifier | const_identifier | constant | expression | array_element

compound_statement::= "{" statement ";" { statement ";" } "}"

sturct_statemnt::=if_statement|while_statement|for_statement

if_statement::="if" "(" condition ")" (simple_statement|compound_statement) "else" (simple_statement|compound_statement)

while_statement::="while" "(" condition ")" "do" (simple_statement|compound_statement)

for_statement::="for" "(" assign_statement ";" condition ";" assign_statement ";" ")"

condition::=expression relation expression

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

###array indexing: begins from 0

array_element::=identifier "[" index "]"

index::=identifier|positive_number|const_identifier

positive_number::=non_zero_number|0

###EXAMPLE OF PROGRAMS

p1: compute the min of 3 numbers

PROGRAM

```
MAIN -> {
        DECLARATIONS
                INTEGER: nr_1, nr_2, nr_3, min;
                STRING: output_message <- "The minimum of the 3 numbers is: ";
        STATEMENTS
        {
                in>>nr_1, nr_2, nr_3;
                if (nr_1 < nr_2)
                        min <- nr_1;
                else
                        min <- nr_2;
                if (nr_3 < min)
                        min <- nr_3;
                out<<output_message, min;
        }
}
```

###

p1err: identifiers can not start with a number -> identifier 1_nr lexical error;

a string needs to be placed between "", output_message misses the closing "

###

```
PROGRAM
        MAIN -> {
                DECLARATIONS
                        INTEGER: 1_nr, nr_2, nr_3, min;
                        STRING: output_message <- "The minimum of the 3 numbers is: ;
```

```
        STATEMENTS
        {
            in>>nr_1, nr_2, nr_3;
            if (nr_1 < nr_2)
                min <- nr_1;
            else
                min <- nr_2;
            if (nr_3 < min)
                min <- nr_3;
            out<<output_message, min;
        }
    }
```

p2: verify if a number is prime

```
PROGRAM
    MAIN -> {
        DECLARATIONS
            INTEGER: input_number, i;
            BOOLEAN: prime <- true;
        STATEMENTS
        {
            in("Give a number: ")>>input_number;
            for(i <- 2; i <= input_number / 2; i <- i+1)
                if (input_number % i == 0)
                    prime <- false;
            if (prime == true)
```

```
                out<<"The number ", input_number, " is prime";
            else
                out<<"The number ", input_number, " is not prime";
        }
    }
```

p3: compute the average of the strictly positive integers from an array with n integers (n<=100);

```
PROGRAM
    MAIN -> {
        DECLARATIONS
            INTEGER: n, i, nr, sum <- 0, positive_integers <- 0;
            INTEGER CONST: MAX_ARRAY_SIZE <- 100;
            ARRAY[INTEGER]: a[MAX_ARRAY_SIZE];
        STATEMENTS
        {
            in("Give the size of the array:")>>n;
            for (i <- 0; i < n; i <- i+1)
            {
                in>>nr;
                a[i] <- nr;
            }
            for (i <- 0; i < n; i <- i+1)
            {
                if(a[i] > 0)
                {
                    sum <- sum + a[i];
```

```
                positive_integers <- positive_integers + 1;
            }
        }

        if (positive_integers == 0)
            out<<"There is no positive integer in the array";
        else
            out<<"The average of the positive integers is: ",
sum/positive_integers;
    }
}
```