1. Lexic

- Alphabet: A-Z, a-z, 0-9, _
- Lexic:
  - operators: +, -, *, /, <<, +<, -<, *<, /<, <, <=, >, >=, =
  - separators: …, ( ), [ ], space, newline, comma, “”
  - reserved words: givenThat, from, to, step, end, print, repeat, else
  - identifier = [“_”] letter { letter | digit }
  - letter = “A” | “B” | ... | “Z” | “a” | “b” | … | “z”
  - digit = “0” | “1” | … | “9”
  - int_const = “0” | [“+” | “-”] nz_digit { digit }
  - nz_digit = “1” | “2” | … | “9”
  - char_const = letter | digit
  - string_const = “ “ ” { letter | digit | “ ” | “_” } “ ” ”

2. Tokens

+, -, *, /, <<, +<, -<, *<, /<, <, <=, >, >=, =, …, (,), [,], space, newline, comma, “, ”, givenThat, from, to, step, end, print, repeat, else

3. Syntax

- program = “statement_list”
- statement_list = statement { statement }
- statement = if_stmt | while_stmt | for_stmt | assign_stmt | decl_stmt | array_stmt
- if_stmt = “givenThat” (expression) “then…” statement_list [“…else…” statement_list_] “…end”
- while_stmt = “givenThat” (expression) “then…” statement_list “…repeat”
- for_stmt = “from” identifier “to” identifier “, step” int “…” statement_list “…end”
- assign_stmt = identifier (“(” type “)”) “<<” expression
- decl_stmt = identifier “(” type “)”
- array_stmt = identifier “(” type “list)”
- type = int | char | string | bool
- expression = int_expression | string_expression
- int_expression = int | identifier | int_expression (“+” | “-” | “*” | “/”) int_expression
- string_expression = string | identifier | string_expression + string_expression