Number: any sequence of digits and maybe floats (e.g. 123 | 554 | 205 | 0.23 | …)   
  
=> Number = [0-9]+ (\. [0-9]+)?

// token

String: starts with double quotes followed by any combination of characters and digits then ends with double quotes (e.g. “Hello” | “2nd + 3rd” | …)   
  
=> String = ^\" . \"$

//token

Reserved\_Keywords: int | float | string | read | write | repeat | until | if | elseif | else | then | return | endl   
  
=> Reserved\_Keywords= int | float | string | read | write | repeat | until | if | elseif | else | then | return | endl

//token

Comment\_Statement: starts with /\* followed by any combination of characters and digits then ends with \*/ (e.g. /\*this is a comment\*/ | …)   
  
=> Comment\_Statement= ^(/ \\*) (Number|[A-Z]|[a-z])\* (\\* /)$  
  
//token

Identifiers: starts with letter then any combination of letters and digits. (e.g. x | val | counter1 | str1 | s2 | …)   
  
=> Identifiers= ([A-Z] | [a-z]) ([A-Z | a-z] | [0-9])\*  
  
//token

Function\_Call: starts with Identifier then left bracket “(“ followed by zero or more Identifier separated by “,” and ends with right bracket “)”. (e.g. sum(a,b) | factorial(c) | rand() | … )   
  
=> Function\_Call= ^ Identifier \( ( Identifier ( \, Identifier )\* ) | ε \)$

Term: maybe Number or Identifier or function call. (e.g. 441 | var1 | sum(a,b) | …)

=>Term= (Number| identifiers|function\_call)+

Arithmatic\_Operator: any arithmetic operation (+ | - | \* | / )

=> Arthimatic\_Operator =(+ | - | \* | / )

tokens

Equation: starts with Term or left bracket “(“ followed by one or more Arithmatic\_Operator and Term. with right bracket “)” for each left bracket (e.g. 3+5 | x +1 | (2+3)\*10 | …)

=> Equation= **(( '(' (T A)+ T ')' | T ) ( (A T) | (A'('(T A)+ T')') )\*)**

// A= Arithmatic\_Operator .

// T=Term.

Expression: may be a String, Term or Equation (e.g. “hi” | counter | 404 | 2+3 | …)

=> Expression = (String |Term | Equation)

Assignment\_Statement: starts with Identifier then assignment operator “:=” followed by Expression (e.g. x := 1 | y:= 2+3 | z := 2+3\*2+(2-3)/1 | …)

=> (identifiers := expression)

//token

Datatype: set of reserved keywords (int, float, string)

=>Datatype = int | float | String | char | Double

Declaration\_Statement: starts with Datatype then one or more identifiers (assignment statement might exist) separated by coma and ends with semi-colon. (e.g. int x; | float x1,x2:=1,xy:=3; | …)

=>Declaration\_Statement=Datatype(identifiers)(= . )\*(, (identifiers)(= . )\*)\*;$

Write\_Statement: starts with reserved keyword “write” followed by an Expression or endl and ends with semi-colon (e.g. write x; | write 5; | write 3+5; | write “Hello World”; | …)

=> Write\_Statement= ^(write)(Expression | endl);$

Read\_Statement: starts with reserved keyword “read” followed by an Identifier and ends with semi-colon (e.g. read x; | …)

=> Read\_Statement= ^(Read)(Expression | endl);$

Return\_Statement: starts with reserved keyword “return” followed by Expression then ends with semi-colon (e.g. return a+b; | return 5; | return “Hi”; | …)

=> Return\_Statement= ^(Return)(Expression | endl);$

Condition\_Operator: ( less than “<” | greater than “>” | is equal “=” | not equal “<>”)

=> Condition\_Operator= (< | > | =| <>)

//token

Condition: starts with Identifier then Condition\_Operator then Term (e.g. z1 <> 10)

=>Condition = Identifiers Condition\_Operator Term

Boolean\_Operator: AND operator “&&” and OR operator “||”

=> Boolean\_Operator= (&& | ||)

//token

Condition\_Statement: starts with Condition followed by zero or more Boolean\_Operator and Condition (e.g. x < 5 && x > 1)

=>condition\_statment=Condition(BooleanOperator Condition )\*

If\_Statement: starts with reserved keyword “if” followed by Condition\_Statement then reserved keyword “then” followed by set of Statements (i.e. any type of statement: write, read, assignment, declaration, …) then Else\_If\_Statment or Else\_Statment or reserved keyword “end”

=>If\_Statement= If Condition\_Statement then (Assignment\_Statement | Declaration\_Statement |Write\_Statement |Read\_Statement |Return\_Statement)+ (Else\_If\_Statment\* | Else\_Statment ?| end)

Else\_If\_Statement: same as if statement but starts with reserved keyword “elseif”

=> Else\_If\_Statement= elseif Condition\_Statement then (Assignment\_Statement | Declaration\_Statement |Write\_Statement |Read\_Statement |Return\_Statement)+ (Else\_If\_Statment\* | Else\_Statment ?| end)

Else\_Statement: starts with reserved keyword “else” followed by a set of Statements then ends with reserved keyword “end”

=>Else\_Statement= else (Assignment\_Statement | Declaration\_Statement |Write\_Statement |Read\_Statement |Return\_Statement)+ end

Repeat\_Statement: starts with reserved keyword “repeat” followed by a set of Statements then reserved keyword “until” followed by Condition\_Statement

=>Repeat\_Statement= repeat (Assignment\_Statement | Declaration\_Statement |Write\_Statement |Read\_Statement |Return\_Statement)+ until Condition\_Statement

FunctionName: same as Identifier

=> FunctionName= Identifier

Parameter: starts with Datatype followed by Identifier   
(e.g. int x)

=>Parameter = DataType Identifier+

Function\_Declaration: starts with Datatype followed by FunctionName followed by “(“ then zero or more Parameter separated by “,” then “)” (e.g. int sum(int a, int b) | …)

=> Function\_Declaration=DataType Identifier+ '(' (Paramete | Parameter,Parameter)\* ')'

Function\_Body: starts with curly bracket “{” then a set of Statements followed by Return\_Statement and ends with “}”

=>Function\_Body = '{' Statements\* ReturnStatement '}'

Function\_Statement: starts with Function\_Declaration followed by Function\_Body

=>Function\_Statement = Function\_Declaration Function\_Body

Main\_Function: starts with Datatype followed by reserved keyword “main” then “()” followed by Function\_Body

=>Main\_Function = DataType ReservedWord "()" Function\_Body

Program: has zero or more Function\_Statement followed by Main\_Function

=> Program = Function\_Statements\* Main\_Function