<statement> ::= <while-loop>

statement.returnTyep = while-loop.returnTyep

<statement> ::= <if-else>

statement.returnTyep = if-else.returnType

<statement> ::= <return>**;**

statement.returnTyep ::= return.returnType

<statement> ::= <jump>**;**

statement.returnTyep = jump.returnTyep

<statement> ::= <assignment>**;** | <declaration>**;** | <in>**;** | <out>**;**|<call-func>**;**

statement.returnTyep = **NOJUMP**

<while-loop> ::= **while (** <condition> **)** <block>

block.EnterPoint = loop;

loop:

if(condition.Bool!=**True**) goto out;

<block>

while-loop.returnType = block.returnType

if(while-loop.returnType==**break**||while-loop.returnType==**return**) goto out

goto loop;

out:;

<if-else> ::= **if (** <condition> **)**  <block1> {**else**  <block2>}

if(condition.Bool!=True) goto else;

<block1>

goto out;

else:

<block2>

out:;

<block> ::= **{** {<statement>} **}**

**for** i in Statements

block.returnType = statement.return.Type

if(block.returnType==**continue**||block.returnType==**break**) return block.returnType

<out> ::= **print(**<expression>|<string>**)**

out.Val = expression.Val | string.Val

<in> ::= **scan(**<variable >**)**

variable.Val = input.Val

<assignment> ::= <variable> **=** <expression>

variable.Val = expression.Val

<declaration> ::= <type> <variable> <rest-variable>

for i in variable.length

variables.NameArray.add(variable.Name)

variables = variables.NameArray+rest-variable.NameArray

declaration.ValList = ValList(variables,type.type)

LocalScope.add(declaration.ValList)

<rest-variable1> ::= **,**<variable>< rest-variable2>|空

rest-variable1.NameArray.add(variable.Name)

rest-variable1.NameArray = rest-variable2.NameArray+ rest-variable1.NameArray

<variable> ::= <identifier> { **[** <inumber> **]** }

variable.Name=identifier.String

variable.Length = inumber.Val

<expression> ::= <calculation> | <character>

expression.Val = calculation.Val | character.Val

<calculation> ::= <item>((**+**|**-**)<calculation >) | <item>

calculation.Val = item.Val +|- calculation.Val | item.Val

<item> ::= (<factor> (**\***|**-**)<item>) | <factor>

item.Val = factor.Val \*|/ item.Val | factor.Val

<factor> ::= (<variable > | <number> | <call-func>) | **(** <calculation> **)**

factor.Val = variable.Val | number.Val | call-func.Val | calculation.Val

<condition> ::= (<sub-con> (**&&** | **||**) <condition>) | <sub-con>

condition.Bool = sub-con.Bool (**&&**|**||**) condition.Bool | sub-con.Bool

<sub-con> ::= <expression1><relation-key><expression2>

if(!(relation-key.operator(expression1.Val, expression2.Val))) goto False;

sub-con.Bool = **True**

goto out

False: sub-con.Bool = **False**

out:;

<number> ::= (+|-)<number> | (<inumber> | <fnumber>)

number.Val = (+|-)number.Val | inumber.Val | fnumber.Val

<relation-key> ::= **<** | **>** | **<=** | **>=** | **==** | **<>**

relation-key.operator = **<** | **>** | **<=** | **>=** | **==** | **<>**

<type> ::= **int** | **real | char**

type.type = **int** | **real | char**

<jump> ::= **break** | **continue**

jump.returnType = **break** | **continue**

<return>::= **return** (<expression>|<blank>)

return.returnType = **RETURN**

rax.Val = expression.Val | NULL

<def-func>::=**def** <identifier> **(**<def-val-list>**)** <block>

def-func.ValList = def-val-list.ValList

def-func.Name = identifier

def-func.EnterPoint = block.Node

funcList.add(def-func)

<def-val-list>::=(<declaration>**;**)\*|<blank>

for i in declarations

def-val-list.ValList = declaration.ValList + def-val-list.ValList

<call-func>::=<identifier>**(**<val-list>**)**

def-func = funcList.get(identifier)

def-func.ValList.Val = val-list.ValList.Val

goto def-func.EnterPoint

<val-list>::=<variable> <rest-variable>|<blank>

val-list.ValList.add(variable.Val)

val-list.ValList = val-list.ValList+rest-variable.ValList