1. PROGRAM -> @1@ **program @1.1@** VAR\_DEFINITIONS; STATEMENTS **end@1.2@** FUNC\_DEFINITIONS @2@

curr\_tab = null

curr\_tab = make\_table(curr\_tab)

make\_table(curr\_tab)

pop table(curr\_tab)

curr\_tab = pop\_table(curr\_tab)

2.VAR\_DEFINITIONS -> VAR\_DEFINITION VAR\_DEFINITIONS\_t @1@

VAR\_DEFINITIONS . num\_of\_var = VAR\_DEFINITIONS\_t.num\_of\_var + VAR\_DEFINITION.num\_of\_var

2.1. VAR\_DEFINITIONS\_t -> ;VAR\_DEFINITIONS @1@ | ɛ @2@

VAR\_DEFINITIONS\_t . num\_of\_var = VAR\_DEFINITIONS.num\_of\_var

VAR\_DEFINITIONS\_t . num\_of\_var = 0

3. VAR\_DEFINITION - > TYPE@1@ VARIABLES\_LIST @2@

VARIABLES\_LIST . type = TYPE . type

VARIABLES\_LIST . action = DEFINE

VAR\_DEFINITION.num\_of\_var = VARIABLES\_LIST.num\_of\_var

4. TYPE -> **real** @1@| **integer** @2@

TYPE . type = real

TYPE . type = int

5. VARIABLES\_LIST - > @2@ VARIABLE VARIABLES\_LIST\_t @1@

VARIABLES\_LIST . num\_of\_var = VARIABLES\_LIST\_t.num\_of\_var+1

VARIABLE . type = VARIABLES\_LIST . type

VARIABLE . action = VARIABLES\_LIST . action

VARIABLES\_LIST\_t . type = VARIABLES\_LIST . type

VARIABLES\_LIST\_t . action = VARIABLES\_LIST . action

5.1. VARIABLES\_LIST\_t -> @3@ ,VARIABLE VARIABLES\_LIST\_t(1) @1@| ɛ@2@

VARIABLES\_LIST\_t . num\_of\_var = VARIABLES\_LIST\_t(1) . num\_of\_var + 1

VARIABLES\_LIST\_t . num\_of\_var = 0

VARIABLE .type = VARIABLES\_LIST\_t . type

VARIABLE .action = VARIABLES\_LIST\_t . action

VARIABLES\_LIST\_t(1).type = VARIABLES\_LIST\_t .type

VARIABLES\_LIST\_t(1).action = VARIABLES\_LIST\_t .action

6. VARIABLE -> id VARIABLE\_t @1@

if (VARIABLE.action == DEFINE){

inserted = insert(curr\_tab , id.lexme, VARIABLE . type, VARIABLE\_t.size)

if(!inserted){

print “ERROR name alredy define”

}

If(VARIABLE.action = USE) {

Entry\_of\_id = find(curr\_tab,id.lexeme)

If(Entry\_of\_id == NULL) {

Print “ERROR – VARIABLE NOT DEFINED”

}

}

6.1 VARIABLE\_t -> ɛ @1@ | [int\_number] @2@

VARIABLE\_t . size = -1

VARIABLE\_t . size = atoi(int\_number)

7. FUNC\_DEFINITIONS -> FUNC\_DEFINITION FUNC\_DEFINITIONS\_t

7.1 FUNC\_DEFINITIONS\_t -> FUNC\_DEFINITION FUNC\_DEFINITIONS\_t | ɛ

8. FUNC\_DEFINITION -> RETURNED\_TYPE id@0.1@ (PARAM\_DEFINITIONS)@1@ BLOCK @2@

curr\_tab =make\_table(curr\_tab) // that the parameter names will be inserted into the table current function

list\_param = get\_list(curr\_tab) // the parameters are located in this section

inserted = insert (curr\_tab->parent , id.lexme, RETURNED\_TYPE. type, PARAM\_DEFINITIONS.size,list\_param)

if (!inserted) {

print “ERROR alredy defined”

}

BLOCK.type\_needed = RETURNED\_TYPE.type

if RETURNED\_TYPE . type != void && BLOCK.type == empty {

print “ERROR – Function was supposed to return a value but didn’t returned anything”

}

curr\_tab = pop table(curr\_tab )

9. RETURNED\_TYPE -> **void** @1@ | TYPE @2@

RETURNED\_TYPE . type = void

RETURNED\_TYPE . type = TYPE . type

10. PARAM\_DEFINITIONS -> ɛ @1@| VAR\_DEFINITIONS @2@

PARAM\_DEFINITIONS . num\_of\_var= 0

PARAM\_DEFINITIONS .num\_of\_var = VAR\_DEFINITIONS .num\_of\_var

11. STATEMENTS -> @0.1@STATEMENT; STATEMENTS\_t @1@

STATEMENT.type\_needed = STATEMENTS.type\_needed

STATEMENS\_t.type\_needed = STATEMENTS.type\_needed

if( STATEMENTS . type== STATEMENT . type ){

STATEMENTS.type = STATEMENT.type

}

Else If(STATEMENT.type == empty ){

STATEMENTS.type = STATEMENTS\_t.type

}

Else If(STATEMENTS\_t.type == empty ){

STATEMENTS.type = STATEMENT.type

}

else

if(STATEMENT.type == ERROR){

STATEMENTS.type = STATEMENTS\_t.type

}

Else if (STATEMENTS\_t.type = ERROR){

STATEMENTS.type = STATEMENT.type

}

11.1 STATEMENTS\_t -> ɛ @1@| STATEMENTS@2@

STATEMENTS\_t . type = empty

STATEMENTS\_t . type= STATEMENTS . type

STATEMENTS . type\_needed= STATEMENTS\_t . type\_needed

12. STATEMENT -> **return** STATEMENT\_t @1@ | id @2@ STATEMENT\_t2 | @3@ BLOCK @4@

STATEMENT.type = STATEMENT\_t.type

If(STATEMENT\_t.type != STATEMENT.type\_needed){

Print “ERROR – Missmatch returned type ”

STATEMENT.type = ERROR

}

Else{

STATEMENT.type = STATEMENT\_t.type

}

id\_table\_entry = find(curr\_tab, id.lexme)

if id\_table\_entry == null

print “ERROR id dont definded”

STATEMENT\_t2 . entry = id\_table\_entry

STATEMENT.type = EMPTY

curr\_tab = make\_table(curr\_table)

curr\_tab = pop\_table(curr\_tab)

STATEMENT.type = BLOCK.type

12.1 STATEMENT\_t -> ɛ @1@|EXPRESSION @2@

STATEMENT\_t . type = void

STATEMENT\_t . type= EXPRESSION .type

12.2 STATEMENT\_t2 -> VARIABLE\_t @1@ = EXPRESSION@2@ | (PARAMETERS\_LIST) @3@

check\_use(STATEMENT\_t2.entry ,VARIABLE\_t.size) // function is showed in the end of this file

If ( EXPRESSION.type != ERROR && STATEMENT\_t2.entry != null){

if (( STATEMENT\_t2.entry->type == INTEGER) && (Expression.type != INTEGER) )

print “ERROR” (MISSMATCH of type)

}

If(STATEMENT\_t2.entry != null) {

If(STATEMENT\_t2.entry->kind != FUNCTION){

Print “ERROR - missmatch kind – kind is not a FUNCTION”

}

Elseif(entry->num\_param != PARAMETERS\_LIST.num\_of\_var){

Print “ERROR – missmatch number of parameters”

}

}

13. BLOCK -> { VAR\_DEFINITIONS;@0.1@ STATEMENTS }@1@

STATEMENTS.type\_needed = BLOCK.type\_needed

BLOCK . type = STATEMENTS . type

14. PARAMETERS\_LIST -> ɛ @1@ | @1.1@VARIABLES\_LIST @2@

VARIABLES\_LIST.action = USE

PARAMETERS\_LIST . num\_of\_var = 0

PARAMETERS\_LIST . num\_of\_var = VARIABLES\_LIST . num\_of\_var

15. EXPRESSION -> id @1@ EXPRESSION\_t @2@| int\_number @3@ | real\_number@4@

Id\_table\_entry = find(curr\_table, id.lexeme)

If(id\_table\_entry == null)

Print “ERROR - name not defined” return ERROR

EXPRESSION\_t.entry = id\_table\_entry

If(id\_table\_entry != null && EXPRESSION\_t.type != ERROR)

If((id\_table\_entry ->type== integer) && (EXPRESSION\_t.type == integer)

EXPRESSION.type = integer

Else( EXPRESSION.type = real)

Else{

EXPRESSION.type = ERROR

}

EXPRESSION.type = INTEGER ;

EXPRESSION.type = REAL

15.1. EXPRESSION\_t -> VARIABLE\_t @1@|ar\_op EXPRESSION @2@

EXPRESSION\_t.type = Check\_use(EXPRESSION\_t.entry,VARIABLE\_t.size) // implemented below

If(Check\_use(EXPRESSION\_t.entry,-1)==ERROR || EXPRESSION.type == ERROR){

EXPRESSION\_t.type = ERROR

}

Else{

EXPRESSION\_t.type = EXPRESSION.type

}

EXTRA :

Check\_use(Entry\_of\_id,Size){

If(entry\_of\_id != NULL ){

If(entry\_of\_id->kind == FUNCTION){Print “ERROR - ID IS FUNCTION AND NOT VARIABLE}

Elseif(entry\_of\_id->kind == ARRAY && size == -1 ){Print “ERROR – ID IS ARRAY NOT VARIABLE”}

Elseif(entry\_of\_id->kind == VARIABLE && size != -1){Print “ERROR -ID IS VARIABLE, NOT ARRAY”}

Elseif(entry\_of\_id->kind == ARRAY && size >= entry\_of\_id ->size ){Print “ERROR -OUT OF BOUNDS”}

RETURN “ERROR”

}

Else{

RETURN entry\_of\_id->type

}