1. PROGRAM -> {cur\_table = make\_table(cur\_table);} **program** {cur\_table = make\_table(cur\_table);} VAR\_DEFINITIONS; STATEMENTS **end** {cur\_table = pop\_table(cur\_table);} FUNC\_DEFINITIONS  
   {cur\_table = make\_table(cur\_table); cur\_table = pop\_table(cur\_table);   
   cur\_table = pop\_table(cur\_table);}
2. VAR\_DEFINITIONS -> VAR\_DEFINITION VAR\_DEFINITIONS'
3. VAR\_DEFINITIONS' -> ; VAR\_DEFINITIONS | {VAR\_DEFINITIONS'.type = NONE }
4. VAR\_DEFINITION -> TYPE {VAR\_DEFINITION.type = TYPE.type} VARIABLES\_LIST
5. TYPE -> **real** {TYPE.type = real}
6. TYPE -> **integer** {TYPE.type = integer}
7. VARIABLES\_LIST -> { VARIABLE.type = VARIABLES\_LIST.type } VARIABLE   
   {VARIABLES\_LIST'.type = VARIABLES\_LIST.type } VARIABLE\_LIST'
8. VARIABLES\_LIST' -> , { “,” || VARIABLES\_LIST.type = VARIABLES\_LIST'.type } VARIABLE VARIABLE\_LIST' |   
   {VARIABLES\_LIST'.type = NONE}
9. VARIABLE -> id VARIABLE' {VARIABALE.type = VARIABLE'.type}
10. VARIABLE' -> [int\_number] {“[“ || VARIABLE\_TAG.type = integer || “]” ; }
11. VARIABLE' ->
12. FUNC\_DEFINITIONS -> FUNC\_DEFINITION FUNC\_DEFINITIONS' {if cur\_token -> kind == id OR  
    cur\_token -> kind == integer OR cur\_token -> kind == void then cur\_table = make\_table(cur\_table)}
13. FUNC\_DEFINITIONS' -> FUNC\_DEFINITION FUNC\_DEFINITION' |   
    {FUNC\_DEFINITIONS'.type = NONE }
14. FUNC\_DEFINITION -> RETURNED\_TYPE id (PARAM\_DEFINITIONS)   
     BLOCK
15. RETURNED\_TYPE -> **void** | TYPE
16. PARAM\_DEFINITIONS -> VAR\_DEFINITIONS *|* { PARAM\_DEFINITIONS.type = NONE }
17. STATEMENTS -> STATEMENT; STATEMENTS'
18. STATAMENTS' -> STATEMENTS | { STATEMENTS'.type = NONE }
19. STATEMENT -> BLOCK {cur\_table = make\_table(cur\_table)} | **return** RETURN\_STATEMENT' | id ID\_STATAMENT'
20. RETURN\_STATEMENT' -> EXPRESSION |
21. ID\_STATAMENT' -> VARAIBLE' = EXPRESSION | (PARAMETERS\_LIST)
22. BLOCK -> { VAR\_DEFINITIONS; STATEMENTS }
23. PARAMETERS\_LIST -> VARIABLES\_LIST *|* {PARAMETES\_LIST.type = NONE}
24. EXPRESSION -> int\_number {EXPRESSION.type = integer || EXPRESSION'}
25. EXPRESSION -> real\_number
26. EXPRESSION -> id EXPRESSION' if(EXPRESSION.type = typeChecking(getidtype(table\_entry), id.type || EXPRESSION'.type) == ErrorType) then Printf(“ERROR”);}
27. EXPRESSION' -> VARIABLE' | ar\_op EXPRESSION {if(EXPRESSION\_TAG.type = typeChecking(EXPRESSION\_TAG.type, EXPRESSION.type) != integer) Then Print(“ERROR”);}

List of all semantic attributes and classification:

|  |  |
| --- | --- |
| **synthesized** | **Inherited** |
| TYPE.type | VAR\_DEFINITIONS.type |
| EXPRESSION.type | VAR\_DEFINITIONS'.type |
| STATEMENTS.type | STATEMENTS\_TAG.type |
|  | EXPRESSION\_TAG.type |
|  | STATEMENT.type |
|  | VARIABLE.type |
|  | VARIABLE'.type |
|  | VARIABLES\_LIST.type |
|  | VARIABLES\_LIST'.type |