PROGRAM 🡪 BLOCK

BLOCK 🡪 **block** { currSymTable = createSymbolTable(currSymTable); } DEFINITIONS;   
 **begin** COMMANDS;   
 **end** { currSymTable = popSymbolTable(currSymTable); }

DEFINITIONS 🡪 DEFINITION DEFINITIONS\_EXT

DEFINITIONS\_EXT 🡪 𝞮

DEFINITIONS\_EXT 🡪 ; DEFINITION DEFINITIONS\_EXT

DEFINITION 🡪 VAR\_DEFINITION

DEFINITION 🡪 TYPE\_DEFINITION

VAR\_DEFINITION 🡪 id { if (currTableLookup(id.lexValue) != NULL)   
 errorMessage(“Same ID exists in this block!”); }   
 : VAR\_DEFINITION\_EXT { idEntry = insertToTable(currSymTable, id.lexValue);  
 setIdType(idEntry, VAR\_DEFINITION\_EXT.type); }

VAR\_DEFINITION\_EXT 🡪 BASIC\_TYPE { VAR\_DEFINITION\_EXT.type = BASIC\_TYPE.type }

VAR\_DEFINITION\_EXT 🡪 type\_name   
 { if (entryPlace = currTableLookup(currSymTable, type\_name.lexValue) == NULL)  
 errorMessage(“Type is not defined!”);  
 else  
 VAR\_DEFINITION\_EXT.type = type\_name.lexValue }

TYPE\_DEFINITION 🡪 **type** type\_name **is** TYPE\_INDICATOR  
 { idEntry = insertToTable(currSymTable, type\_name.lexValue);  
 setIdType(idEntry, TYPE\_INDICATOR.type);   
 if (TYPE\_INDICATOR.type == ARRAY)  
 idEntry🡪size = TYPE\_INDICATOR.arraySize; }

TYPE\_INDICATOR **🡪** BASIC\_TYPE { TYPE\_INDICATOR.type = BASIC\_TYPE.type }

TYPE\_INDICATOR **🡪** ARRAY\_TYPE { TYPE\_INDICATOR.type = ARRAY\_TYPE.type  
 TYPE\_INDICATOR.arraySize = ARRAY\_TYPE.arraySize }

TYPE\_INDICATOR **🡪** POINTER\_TYPE { TYPE\_INDICATOR.type = POINTER\_TYPE.type }

BASIC\_TYPE 🡪 **integer** { BASIC\_TYPE.type = integer.type }

BASIC\_TYPE 🡪 **real** { BASIC\_TYPE.type = real.type }

ARRAY\_TYPE 🡪 **array** [SIZE] **of** BASIC\_TYPE { ARRAY\_TYPE.arraySize = SIZE.size   
 ARRAY\_TYPE.type = BASIC\_TYPE.type }

POINTER\_TYPE 🡪 ^ VAR\_DEFINITION\_EXT { POINTER\_TYPE.type = VAR\_DEFINITION\_EXT.type }

SIZE 🡪 int\_num { SIZE.size = int\_num.lexValue }

COMMANDS 🡪 COMMAND COMMANDS\_EXT

COMMANDS\_EXT 🡪 𝞮

COMMANDS\_EXT 🡪 ; COMMAND COMMANDS\_EXT

COMMAND 🡪 RECEIVER = EXPRESSION { switch(RECEIVER.category)  
 { case BASIC:   
 if (RECEIVER.type != EXPRESSION.type)  
 errorMessage(“Different types!”);  
 case POINTER:  
 if (EXPRESSION.type != ADDRESS)  
 errorMessage(“Different types!”);  
 case ARRAY:  
 errorMessage(“Unable assign to array type!”);  
 }   
 }

COMMAND 🡪 **when** (EXPRESSION rel\_op EXPRESSION) { if (EXPRESSION.type != EXPRESSION.type)  
 errorMessage(“Different types!”); }

**do** COMMANDS;

**default** COMMANDS;

**end\_when**

COMMAND 🡪 **for** (id = EXPRESSION   
 { if (IdEntry = currTableLookup(currSymTable, id.lexeme)) != VARIABLE   
 errorMessage(“ID is not a variable!”); }  
 ; id rel\_op EXPRESSION  
 { if (IdEntry = currTableLookup(currSymTable, id.lexeme)) != VARIABLE   
 errorMessage(“ID is not a variable!”); }   
 ; id++)  
 { if (IdEntry = currTableLookup(currSymTable, id.lexeme)) != VARIABLE   
 errorMessage(“ID is not a variable!”); }

COMMANDS;

**end\_for**

COMMAND 🡪 id  
 { if (IdEntry = currTableLookup(currSymTable, id.lexeme)) != VARIABLE   
 errorMessage(“ID is not a variable!”); }  
 = **malloc**(**size\_of**(type\_name))   
 { if (IdEntry = currTableLookup(currSymTable, type\_name.lexeme)) != VARIABLE   
 errorMessage(“type\_name is not a variable!”); }

COMMAND 🡪 **free**(id) { if (IdEntry = currTableLookup(currSymTable, id.lexeme) != VARIABLE)  
 errorMessage(“Cannot free a non veriable!”); }

COMMAND 🡪 BLOCK

RECEIVER 🡪 id RECEIVER\_EXT { idEntry = currTableLookup(currSymTable, id.lexeme);  
 switch(RECEIVER\_EXT.type)  
 { case ARRAY:   
 if (idEntry🡪category != ARRAY)  
 errorMessage(“Must be array type!”);  
 case POINTER:   
 if (idEntry🡪category != POINTER)  
 errorMessage(“Must be pointer type!”); }  
 RECEIVER.category = idEntry🡪category }

RECEIVER\_EXT 🡪 𝞮 { RECEIVER\_EXT.type = NULLABLE }

RECEIVER\_EXT 🡪 [EXPRESSION { if (EXPRESSION.type != INTEGER)   
 errorMessage(“Array index must be integer!”);}  
 ] { RECEIVER\_EXT.type = ARRAY }

RECEIVER\_EXT 🡪 ^ { RECEIVER\_EXT.type = POINTER }

EXPRESSION 🡪 int\_num { EXPRESSION.type = INTEGER }

EXPRESSION 🡪 real\_num { EXPRESSION.type = REAL }

EXPRESSION 🡪 &id { EXPRESSION.type = ADDRESS }

EXPRESSION 🡪 **size\_of**(type\_name) { EXPRESSION.type = type\_name.lexValue }

EXPRESSION 🡪 id EXPRESSION\_EXT { switch(EXPRESSION\_EXT.type)  
 case ARRAY:   
 if (idEntry🡪category != ARRAY)  
 errorMessage(“Must be array type!”);  
 case POINTER:   
 if (idEntry🡪category != POINTER)  
 errorMessage(“Must be pointer type!”);  
 }

EXPRESSION\_EXT 🡪 𝞮 { EXPRESSION\_EXT.type = NULLABLE }

EXPRESSION\_EXT 🡪 [EXPRESSION { if (EXPRESSION.type != INTEGER)  
 errorMessage(“Array index must be an integer!”); }   
 ] { EXPRESSION\_EXT.type = ARRAY }

EXPRESSION\_EXT 🡪 ^ { EXPRESSION\_EXT.type = POINTER }

EXPRESSION\_EXT 🡪 ar\_op EXPRESSION { EXPRESSION\_EXT.type = EXPRESSION.type }

**List of synthesized items**: VAR\_DEFINITION\_EXT.type, BASIC\_TYPE.type, TYPE\_INDICATOR.type, TYPE\_INDICATOR.arraySize, ARRAY\_TYPE.type, POINTER\_TYPE.type,

SIZE.size, ARRAY\_TYPE.arraySize, EXPRESSION.type, RECEIVER\_EXT.type, EXPRESSION\_EXT.type, RECEIVER.category.