‘program’ -> program *<LTID.new\_t;>* { ‘strct’; } ‘body’ $

‘strct’ -> struct ‘id’ *<LTID.set\_structname(‘id’); If (TSct.look\_s(‘id’) == true) { throw “second structure initialization”;} TSct.new\_s(‘id’); >* { [ {‘type’ ‘id’ *<if (TSct.look\_els(‘id’) == true) { throw “second initialization in a structure”;} if (LTID.get\_lextype == LEX\_STRUCT) { throw “struct in struct”; } TSct.new\_els(‘id’, LTID.get\_lextype());>* { , ‘id’ *<if (TSct.look\_els(‘id’) == true) { throw “second initialization in a structure”;} TSct.new\_els(‘id’, LTID.get\_lextype());>* } *<LTID.set\_lextype(LEX\_NULL)>* ; } ] } *<LTID.set\_lextype(LEX\_STRUCT);>* [‘id’ *<if (LTID.look\_id(‘id’) == true) { throw “second structure ‘id’ initialization”;} LTID.new\_id(‘id’); >* { , ‘id’ *<if (LTID.look\_id(‘id’) == true) { throw “second structure ’id’ initialization”;} LTID.new\_id(‘id’); >* } ] *<LTID.set\_lextype(LEX\_NULL);>* ;

‘body’ -> start { [ ‘definition’ <*\_prog.putlex(\_c\_lex);*> ; ] [ ‘instruction’; ] } { *<LTID.new\_t;>* ‘body’ *<LTID.del\_t;>* { [ ‘definition’<*\_prog.putlex(\_c\_lex);*>; ] [ ‘instruction’; ] } } finish

‘definition’ -> ‘type’ ‘variable’ {<*\_prog.putlex(Lex(LEX\_SEMICOLON));*> ,‘variable’}

‘type’ -> int *<LTID.set\_lextype(LEX\_INT);>* | string *<LTID.set\_lextype(LEX\_STRING);>* | bool *<LTID.set\_lextype(LEX\_BOOL);>* | real *<LTID.set\_lextype(LEX\_REAL);>* | struct *<LTID.set\_lextype(LEX\_STRUCT);>* ‘id’ *<LTID.set\_structname(‘id’); If (TSct.look\_s(‘id’) == false) { throw “undeclared structure”;}>*

‘variable’ -> ‘id’ *<if (LTID.look\_id(‘id’) == true) { throw “second initialization in a definition”;} LTID.new\_id(‘id’); \_st\_lex.push(‘id’); if (‘id’ ==LEX\_STRUCT) {\_st\_name.push(‘id’);} \_prog.putlex(Lex(POLIZ\_ADDRESS,\_c\_val));>* [= < *\_st\_lex.push(=); push(\_c\_lex);>* { ‘id’ *<if (LTID.look\_alltables(‘id’) == false) { throw “undeclared ‘id’ in a definition”;} \_st\_lex.push(‘id’); if (‘id’ ==LEX\_STRUCT) {\_st\_name.push(‘id’);} LTID.cmp\_three(); \_prog.putlex(Lex(POLIZ\_ADDRESS,\_c\_val));>* = <*\_st\_lex.push(=); push('\_c\_lex);>* } action < LTID.cmp\_three();>]

‘instruction’ -> ‘expression’ <*\_prog.putlex(\_c\_lex);*> | ‘stipulation’ | ‘cycle’ | ‘input\_output’ | ‘jump’

‘stipulation’ -> if (‘expression’ <*LTID.cmp\_bool()*;>) <*L2 = \_prog.getfree(); \_prog.blank(); \_prog.putlex(Lex(POLIZ\_FGO));*> [‘quant\_expr’] ; <*L3 = \_prog.getfree(); \_prog.blank(); \_prog.putlex(Lex(POLIZ\_GO)); \_prog\_putlex(Lex(POLIZ\_LABEL,\_prog.getfree()),L2);*> [ else [‘quant\_expr’] ; ] ] <*\_prog.putlex(Lex(POLIZ\_LABEL,\_prog.getfree()),L3);*>| case (‘expression’ <*LTID.cmp\_int();*>) of ‘variants’ end

‘cycle’ -> while <*L0 = \_prog.getfree(); \_st\_labels.push(L0); ends = NULL; \_st\_ends.push(ends(по ссылке!!!));*> (‘expression’ <(LTID.cmp\_bool();>) <*L1 = \_prog.getfree(); \_prog.blank(); \_prog.putlex(Lex(POLIZ\_FGO));*> [‘quant\_expr’] <*\_prog.putlex(Lex(POLIZ\_LABEL,L0)); \_prog.putlex(Lex(POLIZ\_GO)); \_prog.putlex(Lex(POLIZ\_LABEL,\_prog.getfree()),L1); while (ends != NULL) { \_prog.putlex(Lex(POLIZ\_LABEL\_prog.getfree()),ends->label); tmp = ends->next; delete ends; ends = tmp;} \_st\_labels.pop(); \_st\_ends.pop();*> ; | do <*L0 = \_prog.getfree(); \_st\_labels.push(L0); ends = NULL; \_st\_ends.push(ends(по ссылке!!!));*> ‘quant\_expr’ ; while (‘expression’ <LTID.cmp\_bool();>) <*L1 = \_prog.getfree(); \_prog.blank(); \_prog.putlex(Lex(POLIZ\_FGO)); \_prog.putlex(Lex(POLIZ\_LABEL,L0)); \_prog.putlex(Lex(POLIZ\_GO)); \_prog.putlex(Lex(POLIZ\_LABEL,\_prog.getfree()),L1); while (ends != NULL) { \_prog.putlex(Lex(POLIZ\_LABEL\_prog.getfree()),ends->label); tmp = ends->next; delete ends; ends = tmp;} \_st\_labels.pop(); \_st\_ends.pop();*> | for ([‘expression’<*\_prog.putlex(\_c\_lex);*>] ; <*L0 = \_prog.getfree(); \_st\_labels.push(L0); ends = NULL; \_st\_ends.push(ends(по ссылке!!!));*> [‘expression’ <LTID.cmp\_bool();>]<*если выражения нет, то \_prog.putlex(Lex(LEX\_TRUE,1));*> ; <*L1 = \_prog.getfree(); \_prog.blank(); \_prog.putlex(Lex(POLIZ\_FGO)); L2 = \_prog.getfree(); \_prog.blank(); \_prog.putlex(Lex(POLIZ\_GO)); L3 = \_prog.getfree();*> [‘expression’<*\_prog.putlex(Lex(LEX\_SEMICOLON));*>]) <*\_prog.putlex(Lex(POLIZ\_LABEL,L0)); \_prog.putlex(Lex(POLIZ\_GO)); \_prog.putlex(Lex(POLIZ\_LABEL,\_prog.getfree()),L2);*> ‘quant\_expr’ <*\_prog.putlex(Lex(POLIZ\_LABEL,L3)); \_prog.putlex(Lex(POLIZ\_GO)); \_prog.putlex(Lex(POLIZ\_LABEL,\_prog.getfree()),L1); while (ends != NULL) { \_prog.putlex(Lex(POLIZ\_LABEL\_prog.getfree()),ends->label); tmp = ends->next; delete ends; ends = tmp;} \_st\_labels.pop(); \_st\_ends.pop();*> ;

‘input\_output’ -> read ( ‘id’ *<if (LTID.look\_alltables(‘id’) == false) { throw “undeclared ‘id’ ”;} \_prog.putlex(Lex(POLIZ\_ADRESS,\_c\_val));*>*'*) <*\_prog.putlex(Lex(LEX\_READ));*> | write (‘expression’ { , ‘expression’}) <*\_prog.putlex(Lex(LEX\_WRITE));*>

‘jump’ -> continue < *if (\_st\_labels.isempty()) { throw SemErr(“'continue' is out of the loop”);}* *\_prog.putlex(Lex(POLIZ\_LABEL,\_st\_labels.peek())); \_prog.putlex(Lex(POLIZ\_GO));*> | break < *if (\_st\_labels.isempty()) { throw SemErr(“'break' is out of the loop”);} ends.add(\_prog.getfree());\_prog.blank(); \_prog.putlex(Lex(POLIZ\_GO));*>

‘expression’ -> [ ‘id’ *<if (LTID.look\_alltables(‘id’) == false) { throw “undeclared ‘id’ ”;} \_st\_lex.push(‘id’); if (‘id’ == LEX\_STRUCT) {\_st\_name.push(‘id’);} \_prog.putlex(Lex(POLIZ\_ADDRESS,\_c\_val));>*  = <*\_st\_lex.push(=); push(\_c\_lex);*>{ ‘id’ *<if (LTID.look\_alltables(‘id’) == false) { throw “undeclared ‘id’ ”;} \_st\_lex.push(‘id’); if (‘id’ ==LEX\_STRUCT) {\_st\_name.push(‘id’);} LTID.*cmp\_three(); *\_prog.putlex(Lex(POLIZ\_ADDRESS,\_c\_val));>*  = *<\_st\_lex.push(=); push(\_c\_lex);>*} ] ‘action’ <LTID.cmp\_three();>

‘action’ -> ‘bool\_sum’ { or <*\_st\_lex.push(or); push(\_c\_lex);>*  ‘bool\_sum’ <LTID.cmp\_three();> }

‘bool\_sum’ -> ‘bool\_mul’ { and <*\_st\_lex.push(and); push(\_c\_lex);>* ‘bool\_mul’ <LTID.cmp\_three();>}

‘bool\_mul’ -> ‘cmp\_variable’ { <*\_st\_lex.push(\_c\_type); push(\_c\_lex);>* ‘cmp\_instr’ ‘cmp\_variable’ <LTID.cmp\_three();>}

‘cmp\_variable’ -> ‘sum’ { <*\_st\_lex.push(\_c\_type); push(\_c\_lex);>* ‘plus\_minus’ ‘sum’ <LTID.cmp\_three();>}

‘sum’ -> ‘mul\_with\_sign’ { <*\_st\_lex.push(\_c\_type); push(\_c\_lex);>* ‘mul\_div\_exc’ ‘mul\_with\_sign‘<LTID.cmp\_three();>}

‘mul\_with\_sign ‘ -> [ <*\_st\_lex.push(\_c\_type); push(\_c\_lex);>* ‘un\_not’ { <*\_st\_lex.push(\_c\_type); push(\_c\_lex);>* ‘un\_not’ < LTID.cmp\_two();> } ] ‘mul’ < LTID.cmp\_two();>

‘mul’ -> ‘id’ *<if (LTID.look\_alltables(‘id’) == false) { throw “undeclared ‘id’ ”;}>* <*\_st\_lex.push(‘id’); if (‘id’ ==LEX\_STRUCT) {\_st\_name.push(‘id’); \_prog.putlex(Lex(POLIZ\_ADDRESS,\_c\_val));} else { \_prog.putlex(Lex(LEX\_ID,\_c\_val));}>*  | ’const’ <*\_st\_lex.push(‘const’); \_prog.putlex(Lex(\_c\_type,\_c\_val));>*  | ( ‘action’ )

‘quant\_expr’ -> *<LTID.new\_t;>* ‘body’ *<LTID.del\_t;>* | *<LTID.new\_t;>* ‘instruction’ <*\_prog.putlex(\_c\_lex);*>*<LTID.del\_t;>*

‘variants’ -> ‘variant’ { ‘variant’ }

‘variant’ -> ‘const’ { , ‘const’ } : ‘instruction’