| **enРБНФ №1**<br>**(опис синтаксису всіма допустимими засобами РБНФ)** | **РБНФ №2**<br>**(опис формальної граматики засобами РБНФ)** | **Формальна граматика** | **Формальна граматика з специфікацією lookahead у правилах для LL(2)-аналізатора** | /*<br>**Перевірка РБНФ №1 за допомогою коду**<br>*(помістити у файл "EBNF_N1.h")*<br>*/ | /*<br>**Перевірка РБНФ №2 за допомогою коду**<br>*(помістити у файл "EBNF_N2.h")*<br>*/ | /*<br>**Перевірка прототипу LL(2)-синтаксичного аналізатора (спеціальна структура) та прототипу лексичного аналізатора (регулярні вирази) за допомогою коду.** *Лексеми для синтаксичного аналізатора обробляються лексичним аналізатором, тому синтаксичний аналізатор не аналізує їх повторно (як показано в РБНФ).*<br>*(помістити у файл "LexicaByRegExAndSyntaxByLL2prototype.h")*<br>УВАГА: при копіюванні зважайте, щоб у кожному рядку після символу «\» не містилось жодних інших символів.<br>*/ |
|---|---|---|---|---|---|---|
| | | G = (N, T, P, S) | G = (N, T, P, S) | | | |
| | | S → program_rule | S → program_rule | | | |
| | | N = { labeled_point,<br>goto_label,<br>program_name,<br>value_type,<br>array_specify,<br>declaration_element,<br>array_specify__optional,<br>other_declaration_ident,<br>declaration,<br>other_declaration_ident__iteration,<br>index_action,<br>unary_operator,<br>unary_operation,<br>binary_operator,<br>binary_action,<br>left_expression,<br>group_expression,<br>index_action__optional,<br>expression,<br>binary_action__iteration,<br>expression_or_cond_block__with_optional_assign,<br>assign_to_right,<br>assign_to_right__optional,<br>if_expression,<br>body_for_true,<br>false_cond_block_without_else,<br>body_for_false,<br>cond_block,<br>false_cond_block_without_else__iteration,<br>body_for_false__optional,<br>cycle_begin_expression,<br>cycle_end_expression,<br>cycle_counter,<br>cycle_counter_lr_init,<br>cycle_counter_init,<br>cycle_counter_last_value,<br>cycle_body,<br>forto_direction,<br>forto_cycle,<br>continue_while,<br>break_while,<br>statement_in_while_and_if_body,<br>statement,<br>block_statements_in_while_and_if_body,<br>statement_in_while_and_if_body__iteration,<br>while_cycle_head_expression,<br>while_cycle,<br>repeat_until_cycle_cond,<br>repeat_until_cycle,<br>statements__or__block_statements,<br>block_statements,<br>input_rule,<br>argument_for_input,<br>output_rule,<br>statement__iteration,<br>expression__optional,<br>program_rule,<br>declaration__optional,<br>non_zero_digit,<br>digit__iteration,<br>digit,<br>unsigned_value,<br>value,<br>sign__optional,<br>sign,<br>ident,<br>letter_in_upper_case,<br>letter_in_lower_case,<br>sign_plus,<br>sign_minus } | N = { labeled_point,<br>goto_label,<br>program_name,<br>value_type,<br>array_specify,<br>declaration_element,<br>array_specify__optional,<br>other_declaration_ident,<br>declaration,<br>other_declaration_ident__iteration,<br>index_action,<br>unary_operator,<br>unary_operation,<br>binary_operator,<br>binary_action,<br>left_expression,<br>group_expression,<br>index_action__optional,<br>expression,<br>binary_action__iteration,<br>expression_or_cond_block__with_optional_assign,<br>assign_to_right,<br>assign_to_right__optional,<br>if_expression,<br>body_for_true,<br>false_cond_block_without_else,<br>body_for_false,<br>cond_block,<br>false_cond_block_without_else__iteration,<br>body_for_false__optional,<br>cycle_begin_expression,<br>cycle_end_expression,<br>cycle_counter,<br>cycle_counter_lr_init,<br>cycle_counter_init,<br>cycle_counter_last_value,<br>cycle_body,<br>forto_direction,<br>forto_cycle,<br>continue_while,<br>break_while,<br>statement_in_while_and_if_body,<br>statement,<br>block_statements_in_while_and_if_body,<br>statement_in_while_and_if_body__iteration,<br>while_cycle_head_expression,<br>while_cycle,<br>repeat_until_cycle_cond,<br>repeat_until_cycle,<br>statements__or__block_statements,<br>block_statements,<br>input_rule,<br>argument_for_input,<br>output_rule,<br>statement__iteration,<br>expression__optional,<br>program_rule,<br>declaration__optional,<br>non_zero_digit,<br>digit__iteration,<br>digit,<br>unsigned_value,<br>value,<br>sign__optional,<br>sign,<br>ident,<br>letter_in_upper_case,<br>letter_in_lower_case,<br>sign_plus,<br>sign_minus } | #define NONTERMINALS labeled_point, \<br>goto_label, \<br>program_name, \<br>value_type, \<br>array_specify, \<br>declaration_element, \<br>\<br>other_declaration_ident, \<br>declaration, \<br>\<br>index_action, \<br>unary_operator, \<br>unary_operation, \<br>binary_operator, \<br>binary_action, \<br>left_expression, \<br>group_expression, \<br>\<br>expression, \<br>\<br>expression_or_cond_block__with_optional_assign, \<br>assign_to_right, \<br>\<br>if_expression, \<br>body_for_true, \<br>false_cond_block_without_else, \<br>body_for_false, \<br>cond_block, \<br>\<br>\<br>cycle_begin_expression, \<br>cycle_end_expression, \<br>cycle_counter, \<br>cycle_counter_lr_init, \<br>cycle_counter_init, \<br>cycle_counter_last_value, \<br>cycle_body, \<br>forto_direction, \<br>forto_cycle, \<br>continue_while, \<br>break_while, \<br>statement_in_while_and_if_body, \<br>statement, \<br>block_statements_in_while_and_if_body, \<br>\<br>while_cycle_head_expression, \<br>while_cycle, \<br>repeat_until_cycle_cond, \<br>repeat_until_cycle, \<br>statements__or__block_statements, \<br>block_statements, \<br>input_rule, \<br>argument_for_input, \<br>output_rule, \<br>\<br>\<br>program_rule, \<br>\<br>non_zero_digit, \<br>digit__iteration, \<br>digit, \<br>unsigned_value, \<br>value, \<br>\<br>sign, \<br>ident, \<br>letter_in_upper_case, \<br>letter_in_lower_case, \<br>sign_plus, \<br>sign_minus | #define NONTERMINALS labeled_point, \<br>goto_label, \<br>program_name, \<br>value_type, \<br>array_specify, \<br>declaration_element, \<br>array_specify__optional, \<br>other_declaration_ident, \<br>declaration, \<br>other_declaration_ident__iteration, \<br>index_action, \<br>unary_operator, \<br>unary_operation, \<br>binary_operator, \<br>binary_action, \<br>left_expression, \<br>group_expression, \<br>index_action__optional, \<br>expression, \<br>binary_action__iteration, \<br>expression_or_cond_block__with_optional_assign, \<br>assign_to_right, \<br>assign_to_right__optional, \<br>if_expression, \<br>body_for_true, \<br>false_cond_block_without_else, \<br>body_for_false, \<br>cond_block, \<br>false_cond_block_without_else__iteration, \<br>body_for_false__optional, \<br>cycle_begin_expression, \<br>cycle_end_expression, \<br>cycle_counter, \<br>cycle_counter_lr_init, \<br>cycle_counter_init, \<br>cycle_counter_last_value, \<br>cycle_body, \<br>forto_direction, \<br>forto_cycle, \<br>continue_while, \<br>break_while, \<br>statement_in_while_and_if_body, \<br>statement, \<br>block_statements_in_while_and_if_body, \<br>statement_in_while_and_if_body__iteration, \<br>while_cycle_head_expression, \<br>while_cycle, \<br>repeat_until_cycle_cond, \<br>repeat_until_cycle, \<br>statements__or__block_statements, \<br>block_statements, \<br>input_rule, \<br>argument_for_input, \<br>output_rule, \<br>statement__iteration, \<br>expression__optional, \<br>program_rule, \<br>declaration__optional, \<br>non_zero_digit, \<br>digit__iteration, \<br>digit, \<br>unsigned_value, \<br>value, \<br>sign__optional, \<br>sign, \<br>ident, \<br>letter_in_upper_case, \<br>letter_in_lower_case, \<br>sign_plus, \<br>sign_minus | |
| | | T = {<br>":" ,<br>"GOTO" , | T = {<br>":" ,<br>"GOTO" , | #define TOKENS \<br>tokenCOLON, \<br>tokenGOTO, \ | #define TOKENS \<br>tokenCOLON, \<br>tokenGOTO, \ | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | "INTEGER16" ,<br>" , "<br>"NOT" ,<br>"AND" ,<br>"OR" ,<br>"==" ,<br>"!=" ,<br>"<=" ,<br>">=" ,<br>"+" ,<br>"-" ,<br>"*" ,<br>"DIV" ,<br>"MOD" ,<br>"(" ,<br>")" ,<br>"=:" ,<br>"ELSE" ,<br>"IF" ,<br>"DO" ,<br>"FOR" ,<br>"TO" ,<br>"DOWNTO" ,<br>"WHILE" ,<br>"CONTINUE" ,<br>"BREAK" ,<br>"EXIT" ,<br>"REPEAT" ,<br>"UNTIL" ,<br>"GET" ,<br>"PUT" ,<br>"NAME" ,<br>"BODY" ,<br>"DATA" ,<br>"BEGIN" ,<br>"END" ,<br>"{" ,<br>"}" ,<br>"[" ,<br>"]" ,<br>";" ,<br>"0",<br>"1",<br>"2",<br>"3",<br>"4",<br>"5",<br>"6",<br>"7",<br>"8",<br>"9",<br>"_",<br>"A",<br>"B",<br>"C",<br>"D",<br>"E",<br>"F",<br>"G",<br>"H",<br>"I",<br>"J",<br>"K",<br>"L",<br>"M",<br>"N",<br>"O",<br>"P",<br>"Q",<br>"R",<br>"S",<br>"T",<br>"U",<br>"V",<br>"W",<br>"X",<br>"Y",<br>"Z",<br>"a",<br>"b",<br>"c",<br>"d",<br>"e",<br>"f",<br>"g",<br>"h",<br>"i",<br>"j",<br>"k",<br>"l",<br>"m",<br>"n",<br>"o",<br>"p",<br>"q",<br>"r",<br>"s",<br>"t",<br>"u",<br>"v",<br>"w",<br>"x",<br>"y",<br>"z"<br>} | "INTEGER16" ,<br>" , "<br>"NOT" ,<br>"AND" ,<br>"OR" ,<br>"==" ,<br>"!=" ,<br>"<=" ,<br>">=" ,<br>"+" ,<br>"-" ,<br>"*" ,<br>"DIV" ,<br>"MOD" ,<br>"(" ,<br>")" ,<br>"=:" ,<br>"ELSE" ,<br>"IF" ,<br>"DO" ,<br>"FOR" ,<br>"TO" ,<br>"DOWNTO" ,<br>"WHILE" ,<br>"CONTINUE" ,<br>"BREAK" ,<br>"EXIT" ,<br>"REPEAT" ,<br>"UNTIL" ,<br>"GET" ,<br>"PUT" ,<br>"NAME" ,<br>"BODY" ,<br>"DATA" ,<br>"BEGIN" ,<br>"END" ,<br>"{" ,<br>"}" ,<br>"[" ,<br>"]" ,<br>";" ,<br>"0",<br>"1",<br>"2",<br>"3",<br>"4",<br>"5",<br>"6",<br>"7",<br>"8",<br>"9",<br>"_",<br>"A",<br>"B",<br>"C",<br>"D",<br>"E",<br>"F",<br>"G",<br>"H",<br>"I",<br>"J",<br>"K",<br>"L",<br>"M",<br>"N",<br>"O",<br>"P",<br>"Q",<br>"R",<br>"S",<br>"T",<br>"U",<br>"V",<br>"W",<br>"X",<br>"Y",<br>"Z",<br>"a",<br>"b",<br>"c",<br>"d",<br>"e",<br>"f",<br>"g",<br>"h",<br>"i",<br>"j",<br>"k",<br>"l",<br>"m",<br>"n",<br>"o",<br>"p",<br>"q",<br>"r",<br>"s",<br>"t",<br>"u",<br>"v",<br>"w",<br>"x",<br>"y",<br>"z"<br>} | tokenINTEGER16, \<br>tokenCOMMA, \<br>tokenNOT, \<br>tokenAND, \<br>tokenOR, \<br>tokenEQUAL, \<br>tokenNOTEQUAL, \<br>tokenLESSOREQUAL, \<br>tokenGREATEROREQUAL, \<br>tokenPLUS, \<br>tokenMINUS, \<br>tokenMUL, \<br>tokenDIV, \<br>tokenMOD, \<br>tokenGROUPEXPRESSIONBEGIN, \<br>tokenGROUPEXPRESSIONEND, \<br>tokenLRASSIGN, \<br>tokenELSE, \<br>tokenIF, \<br>tokenDO, \<br>tokenFOR, \<br>tokenTO, \<br>tokenDOWNTO, \<br>tokenWHILE, \<br>tokenCONTINUE, \<br>tokenBREAK, \<br>tokenEXIT, \<br>tokenREPEAT, \<br>tokenUNTIL, \<br>tokenGET, \<br>tokenPUT, \<br>tokenNAME, \<br>tokenBODY, \<br>tokenDATA, \<br>tokenBEGIN, \<br>tokenEND, \<br>tokenBEGINBLOCK, \<br>tokenENDBLOCK, \<br>tokenLEFTSQUAREBRACKETS, \<br>tokenRIGHTSQUAREBRACKETS, \<br>tokenSEMICOLON, \<br>digit_0, \<br>digit_1, \<br>digit_2, \<br>digit_3, \<br>digit_4, \<br>digit_5, \<br>digit_6, \<br>digit_7, \<br>digit_8, \<br>digit_9, \<br>tokenUNDERSCORE, \<br>A, \<br>B, \<br>C, \<br>D, \<br>E, \<br>F, \<br>G, \<br>H, \<br>I, \<br>J, \<br>K, \<br>L, \<br>M, \<br>N, \<br>O, \<br>P, \<br>Q, \<br>R, \<br>S, \<br>T, \<br>U, \<br>V, \<br>W, \<br>X, \<br>Y, \<br>Z, \<br>a, \<br>b, \<br>c, \<br>d, \<br>e, \<br>f, \<br>g, \<br>h, \<br>i, \<br>j, \<br>k, \<br>l, \<br>m, \<br>n, \<br>o, \<br>p, \<br>q, \<br>r, \<br>s, \<br>t, \<br>u, \<br>v, \<br>w, \<br>x, \<br>y, \<br>z | tokenINTEGER16, \<br>tokenCOMMA, \<br>tokenNOT, \<br>tokenAND, \<br>tokenOR, \<br>tokenEQUAL, \<br>tokenNOTEQUAL, \<br>tokenLESSOREQUAL, \<br>tokenGREATEROREQUAL, \<br>tokenPLUS, \<br>tokenMINUS, \<br>tokenMUL, \<br>tokenDIV, \<br>tokenMOD, \<br>tokenGROUPEXPRESSIONBEGIN, \<br>tokenGROUPEXPRESSIONEND, \<br>tokenLRASSIGN, \<br>tokenELSE, \<br>tokenIF, \<br>tokenDO, \<br>tokenFOR, \<br>tokenTO, \<br>tokenDOWNTO, \<br>tokenWHILE, \<br>tokenCONTINUE, \<br>tokenBREAK, \<br>tokenEXIT, \<br>tokenREPEAT, \<br>tokenUNTIL, \<br>tokenGET, \<br>tokenPUT, \<br>tokenNAME, \<br>tokenBODY, \<br>tokenDATA, \<br>tokenBEGIN, \<br>tokenEND, \<br>tokenBEGINBLOCK, \<br>tokenENDBLOCK, \<br>tokenLEFTSQUAREBRACKETS, \<br>tokenRIGHTSQUAREBRACKETS, \<br>tokenSEMICOLON, \<br>digit_0, \<br>digit_1, \<br>digit_2, \<br>digit_3, \<br>digit_4, \<br>digit_5, \<br>digit_6, \<br>digit_7, \<br>digit_8, \<br>digit_9, \<br>tokenUNDERSCORE, \<br>A, \<br>B, \<br>C, \<br>D, \<br>E, \<br>F, \<br>G, \<br>H, \<br>I, \<br>J, \<br>K, \<br>L, \<br>M, \<br>N, \<br>O, \<br>P, \<br>Q, \<br>R, \<br>S, \<br>T, \<br>U, \<br>V, \<br>W, \<br>X, \<br>Y, \<br>Z, \<br>a, \<br>b, \<br>c, \<br>d, \<br>e, \<br>f, \<br>g, \<br>h, \<br>i, \<br>j, \<br>k, \<br>l, \<br>m, \<br>n, \<br>o, \<br>p, \<br>q, \<br>r, \<br>s, \<br>t, \<br>u, \<br>v, \<br>w, \<br>x, \<br>y, \<br>z | |
| | | | | #define COMMENT_BEGIN_STR "#*"<br>#define COMMENT_END_STR "*#" | #define COMMENT_BEGIN_STR "#*"<br>#define COMMENT_END_STR "*#" | #define COMMENT_BEGIN_STR "#*"<br>#define COMMENT_END_STR "*#" |
| | | | | | | #define TOKENS_RE ";|:=|<=|*|-|*|,|==|!=|:|\\[|\\]|\\{|\\}|\\(|\\)|<=|>=|[_0-9A-Za-z]+|[^ \t\r\f\v\n]" |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | #define IDENTIFIERS_RE "_[A-Z][A-Z][A-Z][A-Z][A-Z][A-Z][A-Z]" |
| | | | | | | #define UNSIGNEDVALUES_RE "0[1-9][0-9]*" |
| | | | | tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES; | tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES; | #define T_BEGIN_GROUPEXPRESSION_0 "("<br>#define T_BEGIN_GROUPEXPRESSION_1 ""<br>#define T_BEGIN_GROUPEXPRESSION_2 ""<br>#define T_BEGIN_GROUPEXPRESSION_3 "" |
| | | | | tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES; | tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES; | #define T_END_GROUPEXPRESSION_0 ")"<br>#define T_END_GROUPEXPRESSION_1 ""<br>#define T_END_GROUPEXPRESSION_2 ""<br>#define T_END_GROUPEXPRESSION_3 "" |
| | | | | tokenLEFTSQUAREBRACKETS = "[" >> BOUNDARIES; | tokenLEFTSQUAREBRACKETS = "[" >> BOUNDARIES; | #define T_LEFT_SQUAREBRACKETS_0 "["<br>#define T_LEFT_SQUAREBRACKETS_1 ""<br>#define T_LEFT_SQUAREBRACKETS_2 ""<br>#define T_LEFT_SQUAREBRACKETS_3 "" |
| | | | | tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES; | tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES; | #define T_RIGHT_SQUAREBRACKETS_0 "]"<br>#define T_RIGHT_SQUAREBRACKETS_1 ""<br>#define T_RIGHT_SQUAREBRACKETS_2 ""<br>#define T_RIGHT_SQUAREBRACKETS_3 "" |
| | | | | tokenBEGINBLOCK = "{" >> BOUNDARIES; | tokenBEGINBLOCK = "{" >> BOUNDARIES; | #define T_BEGIN_BLOCK_0 "{"<br>#define T_BEGIN_BLOCK_1 ""<br>#define T_BEGIN_BLOCK_2 ""<br>#define T_BEGIN_BLOCK_3 "" |
| | | | | tokenENDBLOCK = "}" >> BOUNDARIES; | tokenENDBLOCK = "}" >> BOUNDARIES; | #define T_END_BLOCK_0 "}"<br>#define T_END_BLOCK_1 ""<br>#define T_END_BLOCK_2 ""<br>#define T_END_BLOCK_3 "" |
| | | | | tokenSEMICOLON = ";" >> BOUNDARIES; | tokenSEMICOLON = ";" >> BOUNDARIES; | #define T_SEMICOLON_0 ";"<br>#define T_SEMICOLON_1 ""<br>#define T_SEMICOLON_2 ""<br>#define T_SEMICOLON_3 "" |
| | | | | tokenCOLON = ":" >> BOUNDARIES; | tokenCOLON = ":" >> BOUNDARIES; | #define T_COLON_0 ":"<br>#define T_COLON_1 ""<br>#define T_COLON_2 ""<br>#define T_COLON_3 "" |
| | | | | tokenGOTO = "GOTO" >> STRICT_BOUNDARIES; | tokenGOTO = "GOTO" >> STRICT_BOUNDARIES; | #define T_GOTO_0 "GOTO"<br>#define T_GOTO_1 ""<br>#define T_GOTO_2 ""<br>#define T_GOTO_3 "" |
| | | | | tokenINTEGER16 = "INTEGER16" >> STRICT_BOUNDARIES; | tokenINTEGER16 = "INTEGER16" >><br>STRICT_BOUNDARIES; | #define T_DATA_TYPE_0 "INTEGER16"<br>#define T_DATA_TYPE_1 ""<br>#define T_DATA_TYPE_2 ""<br>#define T_DATA_TYPE_3 "" |
| | | | | tokenCOMMA = "," >> BOUNDARIES; | tokenCOMMA = "," >> BOUNDARIES; | #define T_COMA_0 ","<br>#define T_COMA_1 ""<br>#define T_COMA_2 ""<br>#define T_COMA_3 "" |
| | | | | | | #define T_BITWISE_NOT_0 "~"<br>#define T_BITWISE_NOT_1 ""<br>#define T_BITWISE_NOT_2 ""<br>#define T_BITWISE_NOT_3 "" |
| | | | | tokenNOT = "NOT" >> STRICT_BOUNDARIES; | tokenNOT = "NOT" >> STRICT_BOUNDARIES; | #define T_NOT_0 "NOT"<br>#define T_NOT_1 ""<br>#define T_NOT_2 ""<br>#define T_NOT_3 "" |
| | | | | | | #define T_BITWISE_AND_0 "&"<br>#define T_BITWISE_AND_1 ""<br>#define T_BITWISE_AND_2 ""<br>#define T_BITWISE_AND_3 "" |
| | | | | tokenAND = "AND" >> STRICT_BOUNDARIES; | tokenAND = "AND" >> STRICT_BOUNDARIES; | #define T_AND_0 "AND"<br>#define T_AND_1 ""<br>#define T_AND_2 ""<br>#define T_AND_3 "" |
| | | | | | | #define T_BITWISE_OR_0 "\|"<br>#define T_BITWISE_OR_1 ""<br>#define T_BITWISE_OR_2 ""<br>#define T_BITWISE_OR_3 "" |
| | | | | tokenOR = "OR" >> STRICT_BOUNDARIES; | tokenOR = "OR" >> STRICT_BOUNDARIES; | #define T_OR_0 "OR"<br>#define T_OR_1 ""<br>#define T_OR_2 ""<br>#define T_OR_3 "" |
| | | | | tokenEQUAL = "==" >> BOUNDARIES; | tokenEQUAL = "==" >> BOUNDARIES; | #define T_EQUAL_0 "=="<br>#define T_EQUAL_1 ""<br>#define T_EQUAL_2 ""<br>#define T_EQUAL_3 "" |
| | | | | tokenNOTEQUAL = "!=" >> BOUNDARIES; | tokenNOTEQUAL = "!=" >> BOUNDARIES; | #define T_NOT_EQUAL_0 "!="<br>#define T_NOT_EQUAL_1 ""<br>#define T_NOT_EQUAL_2 ""<br>#define T_NOT_EQUAL_3 "" |
| | | | | tokenLESSOREQUAL = "<=" >> BOUNDARIES; | tokenLESSOREQUAL = "<=" >> BOUNDARIES; | #define T_LESS_OR_EQUAL_0 "<="<br>#define T_LESS_OR_EQUAL_1 ""<br>#define T_LESS_OR_EQUAL_2 ""<br>#define T_LESS_OR_EQUAL_3 "" |
| | | | | tokenGREATEROREQUAL = ">=" >> BOUNDARIES; | tokenGREATEROREQUAL = ">=" >> BOUNDARIES; | #define T_GREATER_OR_EQUAL_0 ">="<br>#define T_GREATER_OR_EQUAL_1 ""<br>#define T_GREATER_OR_EQUAL_2 ""<br>#define T_GREATER_OR_EQUAL_3 "" |
| | | | | tokenPLUS = "+" >> BOUNDARIES; | tokenPLUS = "+" >> BOUNDARIES; | #define T_ADD_0 "+"<br>#define T_ADD_1 ""<br>#define T_ADD_2 ""<br>#define T_ADD_3 "" |
| | | | | tokenMINUS = "-" >> BOUNDARIES; | tokenMINUS = "-" >> BOUNDARIES; | #define T_SUB_0 "-"<br>#define T_SUB_1 ""<br>#define T_SUB_2 ""<br>#define T_SUB_3 "" |
| | | | | tokenMUL = "*" >> BOUNDARIES; | tokenMUL = "*" >> BOUNDARIES; | #define T_MUL_0 "*"<br>#define T_MUL_1 ""<br>#define T_MUL_2 ""<br>#define T_MUL_3 "" |
| | | | | tokenDIV = "DIV" >> STRICT_BOUNDARIES; | tokenDIV = "DIV" >> STRICT_BOUNDARIES; | #define T_DIV_0 "DIV"<br>#define T_DIV_1 ""<br>#define T_DIV_2 ""<br>#define T_DIV_3 "" |
| | | | | tokenMOD = "MOD" >> STRICT_BOUNDARIES; | tokenMOD = "MOD" >> STRICT_BOUNDARIES; | #define T_MOD_0 "MOD"<br>#define T_MOD_1 "" |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | #define T_MOD_2 ""<br>#define T_MOD_3 "" |
| | | | | tokenLRASSIGN = "=:" >> BOUNDARIES; | tokenLRASSIGN = "=:" >> BOUNDARIES; | #define T_LRASSIGN_0 "=:"<br>#define T_LRASSIGN_1 ""<br>#define T_LRASSIGN_2 ""<br>#define T_LRASSIGN_3 "" |
| | | | | | | #define T_THEN_BLOCK_0 "{"<br>#define T_THEN_BLOCK_1 ""<br>#define T_THEN_BLOCK_2 ""<br>#define T_THEN_BLOCK_3 "" |
| | | | | tokenELSE = "ELSE" >> STRICT_BOUNDARIES; | tokenELSE = "ELSE" >> STRICT_BOUNDARIES; | #define T_ELSE_BLOCK_0 "ELSE"<br>#define T_ELSE_BLOCK_1 T_BEGIN_BLOCK_0<br>#define T_ELSE_BLOCK_2 ""<br>#define T_ELSE_BLOCK_3 "" |
| | | | | tokenIF = "IF" >> STRICT_BOUNDARIES; | tokenIF = "IF" >> STRICT_BOUNDARIES; | #define T_IF_0 "IF"<br>#define T_IF_1 ""<br>#define T_IF_2 ""<br>#define T_IF_3 "" |
| | | | | | | #define T_ELSE_IF_0 T_ELSE_BLOCK_0<br>#define T_ELSE_IF_1 T_IF_0<br>#define T_ELSE_IF_2 ""<br>#define T_ELSE_IF_3 "" |
| | | | | tokenDO = "DO" >> STRICT_BOUNDARIES; | tokenDO = "DO" >> STRICT_BOUNDARIES; | #define T_DO_0 "DO"<br>#define T_DO_1 ""<br>#define T_DO_2 ""<br>#define T_DO_3 "" |
| | | | | tokenFOR = "FOR" >> STRICT_BOUNDARIES; | tokenFOR = "FOR" >> STRICT_BOUNDARIES; | #define T_FOR_0 "FOR"<br>#define T_FOR_1 ""<br>#define T_FOR_2 ""<br>#define T_FOR_3 "" |
| | | | | tokenTO = "TO" >> STRICT_BOUNDARIES; | tokenTO = "TO" >> STRICT_BOUNDARIES; | #define T_TO_0 "TO"<br>#define T_TO_1 ""<br>#define T_TO_2 ""<br>#define T_TO_3 "" |
| | | | | tokenDOWNTO = "DOWNTO" >> STRICT_BOUNDARIES; | tokenDOWNTO = "DOWNTO" >> STRICT_BOUNDARIES; | #define T_DOWNTO_0 "DOWNTO"<br>#define T_DOWNTO_1 ""<br>#define T_DOWNTO_2 ""<br>#define T_DOWNTO_3 "" |
| | | | | tokenWHILE = "WHILE" >> STRICT_BOUNDARIES; | tokenWHILE = "WHILE" >> STRICT_BOUNDARIES; | #define T_WHILE_0 "WHILE"<br>#define T_WHILE_1 ""<br>#define T_WHILE_2 ""<br>#define T_WHILE_3 "" |
| | | | | tokenCONTINUE = "CONTINUE" >> STRICT_BOUNDARIES; | tokenCONTINUE = "CONTINUE" >> STRICT_BOUNDARIES; | #define T_CONTINUE_WHILE_0 "CONTINUE"<br>#define T_CONTINUE_WHILE_1 ""<br>#define T_CONTINUE_WHILE_2 ""<br>#define T_CONTINUE_WHILE_3 "" |
| | | | | tokenBREAK = "BREAK" >> STRICT_BOUNDARIES; | tokenBREAK = "BREAK" >> STRICT_BOUNDARIES; | #define T_EXIT_WHILE_0 "BREAK"<br>#define T_EXIT_WHILE_1 ""<br>#define T_EXIT_WHILE_2 ""<br>#define T_EXIT_WHILE_3 "" |
| | | | | tokenEXIT = "EXIT" >> STRICT_BOUNDARIES; | tokenEXIT = "EXIT" >> STRICT_BOUNDARIES; | #define T_EXIT_0 "EXIT"<br>#define T_EXIT_1 ""<br>#define T_EXIT_2 ""<br>#define T_EXIT_3 "" |
| | | | | tokenREPEAT = "REPEAT" >> STRICT_BOUNDARIES; | tokenREPEAT = "REPEAT" >> STRICT_BOUNDARIES; | #define T_REPEAT_0 "REPEAT"<br>#define T_REPEAT_1 ""<br>#define T_REPEAT_2 ""<br>#define T_REPEAT_3 "" |
| | | | | tokenUNTIL = "UNTIL" >> STRICT_BOUNDARIES; | tokenUNTIL = "UNTIL" >> STRICT_BOUNDARIES; | #define T_UNTIL_0 "UNTIL"<br>#define T_UNTIL_1 ""<br>#define T_UNTIL_2 ""<br>#define T_UNTIL_3 "" |
| | | | | tokenGET = "GET" >> STRICT_BOUNDARIES; | tokenGET = "GET" >> STRICT_BOUNDARIES; | #define T_INPUT_0 "GET"<br>#define T_INPUT_1 ""<br>#define T_INPUT_2 ""<br>#define T_INPUT_3 "" |
| | | | | tokenPUT = "PUT" >> STRICT_BOUNDARIES; | tokenPUT = "PUT" >> STRICT_BOUNDARIES; | #define T_OUTPUT_0 "PUT"<br>#define T_OUTPUT_1 ""<br>#define T_OUTPUT_2 ""<br>#define T_OUTPUT_3 "" |
| | | | | tokenNAME = "NAME" >> STRICT_BOUNDARIES; | tokenNAME = "NAME" >> STRICT_BOUNDARIES; | #define T_NAME_0 "NAME"<br>#define T_NAME_1 ""<br>#define T_NAME_2 ""<br>#define T_NAME_3 "" |
| | | | | tokenBODY = "BODY" >> STRICT_BOUNDARIES; | tokenBODY = "BODY" >> STRICT_BOUNDARIES; | #define T_BODY_0 "BODY"<br>#define T_BODY_1 ""<br>#define T_BODY_2 ""<br>#define T_BODY_3 "" |
| | | | | tokenDATA = "DATA" >> STRICT_BOUNDARIES; | tokenDATA = "DATA" >> STRICT_BOUNDARIES; | #define T_DATA_0 "DATA"<br>#define T_DATA_1 ""<br>#define T_DATA_2 ""<br>#define T_DATA_3 "" |
| | | | | tokenBEGIN = "BEGIN" >> STRICT_BOUNDARIES; | tokenBEGIN = "BEGIN" >> STRICT_BOUNDARIES; | #define T_BEGIN_0 "BEGIN"<br>#define T_BEGIN_1 ""<br>#define T_BEGIN_2 ""<br>#define T_BEGIN_3 "" |
| | | | | tokenEND = "END" >> STRICT_BOUNDARIES; | tokenEND = "END" >> STRICT_BOUNDARIES; | #define T_END_0 "END"<br>#define T_END_1 ""<br>#define T_END_2 ""<br>#define T_END_3 "" |
| | | | | | | #define T_NULL_STATEMENT_0 "NULL"<br>#define T_NULL_STATEMENT_1 "STATEMENT"<br>#define T_NULL_STATEMENT_2 ""<br>#define T_NULL_STATEMENT_3 "" |
| | | | | | | #define GRAMMAR_LL2__2025 {\ |
| labeled_point = ident , ":"; | labeled_point = ident , ":"; | labeled_point → ident ":" | labeled_point(1: "ident_terminal") → ident ":" | labeled_point = ident >> tokenCOLON; | labeled_point = ident >> tokenCOLON; | { LA_IS, {"ident_terminal"}, { "labeled_point",{\<br>{ LA_IS, {""}, 2, {"ident", T_COLON_0}}\<br>}}},\ |
| goto_label = "GOTO" , ident; | goto_label = "GOTO" , ident; | goto_label → "GOTO" ident | goto_label(1: "GOTO") → "GOTO" ident | goto_label = tokenGOTO >> ident; | goto_label = tokenGOTO >> ident; | { LA_IS, {T_GOTO_0}, { "goto_label",{\<br>{ LA_IS, {""}, 2, {T_GOTO_0, "ident"}}\<br>}}},\ |
| program_name = ident; | program_name = ident; | program_name → ident | program_name(1: "ident_terminal") → ident | program_name = SAME_RULE(ident); | program_name = SAME_RULE(ident); | { LA_IS, {"ident_terminal"}, { "program_name",{\<br>{ LA_IS, {""}, 1, {"ident"}}\<br>}}},\ |
| value_type = "INTEGER16"; | value_type = "INTEGER16"; | value_type → "INTEGER16" | value_type(1: "INTEGER16") → "INTEGER16" | value_type = SAME_RULE(tokenINTEGER16); | value_type = SAME_RULE(tokenINTEGER16); | { LA_IS, {T_DATA_TYPE_0}, { "value_type",{\<br>{ LA_IS, {""}, 1, {T_DATA_TYPE_0}}\ |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | }}},\ |
| | array_specify = "[", unsigned_value, "]"; | array_specify → "[" unsigned_value "]" | array_specify(1: "[") → "[" unsigned_value "]" | | array_specify = "[" >> unsigned_value >> "]"; | { LA_IS, {"["}, { "array_specify",{\<br>{ LA_IS, {""}, 3, {"[", "unsigned_value", "]"}}\<br>}}},\ |
| declaration_element = ident, [ "[", unsigned_value , "]" ]; | declaration_element = ident,<br>array_specify__optional; | declaration_element → ident<br>array_specify__optional | declaration_element(1: "ident_terminal") → ident<br>array_specify__optional | declaration_element = ident >> -(tokenLEFTSQUAREBRACKETS >> unsigned_value >> tokenRIGHTSQUAREBRACKETS); | declaration_element = ident >><br>array_specify__optional; | { LA_IS, {"ident_terminal"}, { "declaration_element",{\<br>{ LA_IS, {""}, 2, {"ident", "array_specify__optional"}}\<br>}}},\ |
| | array_specify__optional = array_specify \| ε; | array_specify__optional → array_specify<br>array_specify__optional → ε | array_specify__optional(1: "[") → array_specify<br>array_specify__optional(1: !"[") → ε | | array_specify__optional = array_specify \| ""; | { LA_IS, {"["}, { "array_specify__optional",{\<br>{ LA_IS, {""}, 1, {"array_specify"}}\<br>}}},\<br>{ LA_NOT, {"["}, { "array_specify__optional",{\<br>{ LA_IS, {""}, 0, {""}}\<br>}}},\ |
| other_declaration_ident = "," , declaration_element; | other_declaration_ident = "," ,<br>declaration_element; | other_declaration_ident → ","<br>declaration_element | other_declaration_ident(1: ",") → ","<br>declaration_element | other_declaration_ident = tokenCOMMA >><br>declaration_element; | other_declaration_ident = tokenCOMMA >><br>declaration_element; | { LA_IS, {T_COMA_0}, { "other_declaration_ident",{\<br>{ LA_IS, {""}, 2, {T_COMA_0, "declaration_element"}}\<br>}}},\ |
| declaration = value_type , declaration_element , {other_declaration_ident}; | declaration = value_type , declaration_element ,<br>other_declaration_ident__iteration; | declaration → value_type declaration_element<br>other_declaration_ident__iteration | declaration(1: "INTEGER16") → value_type<br>declaration_element<br>other_declaration_ident__iteration | declaration = value_type >> declaration_element >><br>*other_declaration_ident; | declaration = value_type >> declaration_element >><br>other_declaration_ident__iteration; | { LA_IS, {T_DATA_TYPE_0}, { "declaration",{\<br>{ LA_IS, {""}, 3, {"value_type", "declaration_element",<br>"other_declaration_ident__iteration"}}\<br>}}},\ |
| | other_declaration_ident__iteration =<br>other_declaration_ident,<br>other_declaration_ident__iteration \| ε; | other_declaration_ident__iteration →<br>other_declaration_ident<br>other_declaration_ident__iteration<br>false_cond_block_without_else__iteration → ε | other_declaration_ident__iteration(1: ",") →<br>other_declaration_ident<br>other_declaration_ident__iteration<br>false_cond_block_without_else__iteration(1: !",") →<br>ε | | other_declaration_ident__iteration =<br>other_declaration_ident >><br>other_declaration_ident__iteration \| ""; | { LA_IS, { T_COMA_0 }, {<br>"other_declaration_ident__iteration",{\<br>{ LA_IS, {""}, 2, { "other_declaration_ident",<br>"other_declaration_ident__iteration" }}\<br>}}},\<br>{ LA_NOT, { T_COMA_0 }, {<br>"other_declaration_ident__iteration",{\<br>{ LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| index_action = "[" , expression , "]"; | index_action = "[" , expression , "]"; | index_action → "[" expression "]" | index_action(1: "[") → "[" expression "]" | index_action = tokenLEFTSQUAREBRACKETS >> expression >><br>tokenRIGHTSQUAREBRACKETS; | index_action = tokenLEFTSQUAREBRACKETS >><br>expression >> tokenRIGHTSQUAREBRACKETS; | { LA_IS, { "[" }, { "index_action",{\<br>{ LA_IS, {""}, 3, { "[", "expression", "]" }}\<br>}}},\ |
| unary_operator = "NOT"; | unary_operator = "NOT"; | unary_operator → "NOT" | unary_operator(1: "NOT") → "NOT" | unary_operator = SAME_RULE(tokenNOT); | unary_operator = SAME_RULE(tokenNOT); | { LA_IS, { T_NOT_0 }, { "unary_operator",{\<br>{ LA_IS, {""}, 1, { T_NOT_0 }}\<br>}}},\ |
| unary_operation = unary_operator , expression; | unary_operation = unary_operator , expression; | unary_operation → unary_operator expression | unary_operation(1: "NOT") → unary_operator<br>expression | unary_operation = unary_operator >> expression; | unary_operation = unary_operator >> expression; | { LA_IS, { T_NOT_0 }, { "unary_operation",{\<br>{ LA_IS, {""}, 2, { "unary_operator", "expression" }}\<br>}}},\ |
| binary_operator = "AND" \| "OR" \| "==" \| "!=" \| "<=" \| ">=" \| "+" \| "-" \| "*" \| "DIV" \| "MOD"; | binary_operator = "AND" \| "OR" \| "==" \| "!=" \|<br>"<=" \| ">=" \| "+" \| "-" \| "*" \| "DIV" \| "MOD"; | binary_operator → "AND"<br>binary_operator → "OR"<br>binary_operator → "=="<br>binary_operator → "!="<br>binary_operator → "<="<br>binary_operator → ">="<br>binary_operator → "+"<br>binary_operator → "-"<br>binary_operator → "*"<br>binary_operator → "DIV"<br>binary_operator → "MOD" | binary_operator(1: "AND") → "AND"<br>binary_operator(1: "OR") → "OR"<br>binary_operator(1: "==") → "=="<br>binary_operator(1: "!=") → "!="<br>binary_operator(1: "<=") → "<="<br>binary_operator(1: ">=") → ">="<br>binary_operator(1: "+") → "+"<br>binary_operator(1: "-") → "-"<br>binary_operator(1: "*") → "*"<br>binary_operator(1: "DIV") → "DIV"<br>binary_operator(1: "MOD") → "MOD" | binary_operator = tokenAND \| tokenOR \| tokenEQUAL \| tokenNOTEQUAL \| tokenLESSOREQUAL \| tokenGREATEROREQUAL \| tokenPLUS \| tokenMINUS \| tokenMUL \| tokenDIV \| tokenMOD; | binary_operator = tokenAND \| tokenOR \| tokenEQUAL<br>\| tokenNOTEQUAL \| tokenLESSOREQUAL \|<br>tokenGREATEROREQUAL \| tokenPLUS \| tokenMINUS \|<br>tokenMUL \| tokenDIV \| tokenMOD; | { LA_IS, { T_AND_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_AND_0 }}\<br>}}},\<br>{ LA_IS, { T_OR_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_OR_0 }}\<br>}}},\<br>{ LA_IS, { T_EQUAL_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_EQUAL_0 }}\<br>}}},\<br>{ LA_IS, { T_NOT_EQUAL_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_NOT_EQUAL_0 }}\<br>}}},\<br>{ LA_IS, { T_LESS_OR_EQUAL_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_LESS_OR_EQUAL_0 }}\<br>}}},\<br>{ LA_IS, { T_GREATER_OR_EQUAL_0 }, {<br>"binary_operator",{\<br>{ LA_IS, {""}, 1, { T_GREATER_OR_EQUAL_0 }}\<br>}}},\<br>{ LA_IS, { T_ADD_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_ADD_0 }}\<br>}}},\<br>{ LA_IS, { T_SUB_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_SUB_0 }}\<br>}}},\<br>{ LA_IS, { T_MUL_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_MUL_0 }}\<br>}}},\<br>{ LA_IS, { T_DIV_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_DIV_0 }}\<br>}}},\<br>{ LA_IS, { T_MOD_0 }, { "binary_operator",{\<br>{ LA_IS, {""}, 1, { T_MOD_0 }}\<br>}}},\ |
| binary_action = binary_operator , expression; | binary_action = binary_operator , expression; | binary_action → binary_operator expression | binary_action(1: "AND", "OR", "==", "!=", "<=", ">=",<br>"+", "-", "*", "DIV", "MOD") → binary_operator<br>expression | binary_action = binary_operator >> expression; | binary_action = binary_operator >> expression; | { LA_IS, { T_AND_0, T_OR_0, T_EQUAL_0,<br>T_NOT_EQUAL_0, T_LESS_OR_EQUAL_0,<br>T_GREATER_OR_EQUAL_0, T_ADD_0, T_SUB_0,<br>T_MUL_0, T_DIV_0, T_MOD_0 }, { "binary_action",{\<br>{ LA_IS, {""}, 2, { "binary_operator", "expression" }}\<br>}}},\ |
| left_expression = group_expression \| unary_operation \| cond_block \| value \| ident , {index_action}; | left_expression = group_expression \|<br>unary_operation \| cond_block \| value \| ident ,<br>index_action__optional; | left_expression → group_expression<br>left_expression → unary_operation<br>left_expression → cond_block<br>left_expression → value<br>left_expression → ident , index_action__optional | left_expression(1: "(") → group_expression<br>left_expression(1: "NOT") → unary_operation<br>left_expression(1: "IF") → cond_block<br>left_expression(1: "**0**", "**1**", "**2**", "**3**", "**4**", "**5**", "**6**", "**7**",<br>"**8**", "**9**") → value<br>left_expression(1: "+", "-"; 2: "**0**", "**1**", "**2**", "**3**", "**4**",<br>"**5**", "**6**", "**7**", "**8**", "**9**") → value<br>left_expression(1: "_") → ident ,<br>index_action__optional | left_expression = group_expression \| unary_operation \|<br>cond_block \| value \| ident >> -index_action; | left_expression = group_expression \| unary_operation<br>\| cond_block \| value \| ident >><br>index_action__optional; | {LA_IS, { "(" }, { "left_expression",{\<br>{LA_IS, { "" }, 1, { "group_expression" }}\<br>}}},\<br>{LA_IS, { T_NOT_0 }, { "left_expression",{\<br>{LA_IS, { "" }, 1, { "unary_operation" }}\<br>}}},\<br>{LA_IS, { T_IF_0 }, { "left_expression",{\<br>{LA_IS, { "" }, 1, { "cond_block" }}\<br>}}},\<br>{LA_IS, { "unsigned_value_terminal" }, {<br>"left_expression",{\<br>{LA_IS, {""}, 1, { "value" }}\<br>}}},\<br>{LA_IS, { T_ADD_0, T_SUB_0 }, { "left_expression",{\<br>{LA_IS, { "unsigned_value_terminal"}, 1, { "value" }},\<br>/*{LA_NOT, { "unsigned_value_terminal" }, 1, {<br>"unary_operation" }}*/\<br>}}},\<br>{LA_IS, { "ident_terminal" }, { "left_expression",{\<br>{LA_IS, {""}, 2, { "ident", "index_action__optional" }}\<br>}}},\ |
| | index_action__optional = index_action \| ε; | index_action__optional → index_action<br>index_action__optional → ε | index_action__optional(1: "[") → index_action<br>index_action__optional(1: !"[") → ε | | index_action__optional = index_action \| ""; | {LA_IS, { "[" }, { "index_action__optional",{\<br>{LA_IS, {""}, 1, { "index_action" }}\<br>}}},\<br>{LA_NOT, { "[" }, { "index_action__optional",{\<br>{LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| expression = left_expression , {binary_action}; | expression = left_expression ,<br>binary_action__iteration; | expression → left_expression<br>binary_action__iteration | expression(1: "(", "NOT", "+", "-", "_", "**0**", "**1**", "**2**",<br>"**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "IF") → | expression = left_expression >> *binary_action; | expression = left_expression >><br>binary_action__iteration; | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0,<br>"ident_terminal", "unsigned_value_terminal", T_IF_0 }, |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | left_expression binary_action__iteration | | | { "expression",{\<br>{LA_IS, {""}, 2, { "left_expression",<br>"binary_action__iteration" }}\<br>}}},\ |
| | binary_action__iteration = binary_action,<br>binary_action__iteration \| ε; | binary_action__iteration → binary_action<br>binary_action__iteration<br>binary_action__iteration → ε | binary_action__iteration(1: "AND", "OR", "==", "!=", "<=", ">=", "+", "-", "*", "DIV", "MOD") →<br>binary_action binary_action__iteration<br>binary_action__iteration(1: !"AND", !"OR", !"==", !"!=", !"<=", !">=", !"+", !"-", !"*", !"DIV", !"MOD") →<br>ε | | binary_action__iteration = binary_action >><br>binary_action__iteration \| ""; | {LA_IS, { T_AND_0, T_OR_0, T_EQUAL_0,<br>T_NOT_EQUAL_0, T_LESS_OR_EQUAL_0,<br>T_GREATER_OR_EQUAL_0, T_ADD_0, T_SUB_0,<br>T_MUL_0, T_DIV_0, T_MOD_0 }, {<br>"binary_action__iteration",{\<br>{LA_IS, {""}, 2, { "binary_action",<br>"binary_action__iteration" }}\<br>}}},\<br>{LA_NOT, { T_AND_0, T_OR_0, T_EQUAL_0,<br>T_NOT_EQUAL_0, T_LESS_OR_EQUAL_0,<br>T_GREATER_OR_EQUAL_0, T_ADD_0, T_SUB_0,<br>T_MUL_0, T_DIV_0, T_MOD_0 }, {<br>"binary_action__iteration",{\<br>{LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| group_expression = "(" , expression , ")"; | group_expression = "(" , expression , ")"; | group_expression → "(" expression ")" | group_expression(1: "(") → "(" expression ")" | group_expression = tokenGROUPEXPRESSIONBEGIN >><br>expression >> tokenGROUPEXPRESSIONEND; | group_expression = tokenGROUPEXPRESSIONBEGIN >><br>expression >> tokenGROUPEXPRESSIONEND; | {LA_IS, { "(" }, { "group_expression",{\<br>{LA_IS, {""}, 3, { "(", "expression", ")" }}\<br>}}},\ |
| expression_or_cond_block__with_optional_assign = expression , ["=:" , ident , [index_action]]; | expression_or_cond_block__with_optional_assign<br>= expression , assign_to_right__optional; | expression_or_cond_block__with_optional_assign<br>→ expression assign_to_right__optional | expression_or_cond_block__with_optional_assign(1:<br>"(", "NOT", "+", "-", "_", "**0**", "**1**", "**2**", "**3**", "**4**", "**5**",<br>"**6**", "**7**", "**8**", "**9**", "IF") → expression<br>assign_to_right__optional | expression_or_cond_block__with_optional_assign = expression<br>>> -(tokenLRASSIGN >> ident >> -index_action); | expression_or_cond_block__with_optional_assign =<br>expression >> assign_to_right__optional; | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0,<br>"ident_terminal", "unsigned_value_terminal", T_IF_0 },<br>{ "expression_or_cond_block__with_optional_assign",{\<br>{LA_IS, {""}, 2, { "expression",<br>"assign_to_right__optional" }}\<br>}}},\ |
| | assign_to_right = "=:" , ident ,<br>index_action__optional; | assign_to_right → "=:" ident<br>index_action__optional | assign_to_right(1: "=:") → "=:" ident<br>index_action__optional | | assign_to_right = tokenLRASSIGN >> ident >><br>index_action__optional; | {LA_IS, { T_LRASSIGN_0 }, { "assign_to_right",{\<br>{LA_IS, {""}, 3, { T_LRASSIGN_0, "ident",<br>"index_action__optional" }}\<br>}}},\ |
| | assign_to_right__optional = assign_to_right \| ε; | assign_to_right__optional → assign_to_right<br>assign_to_right__optional → ε; | assign_to_right__optional(1: "=:") → assign_to_right<br>assign_to_right__optional(1: !"=:") → ε; | | assign_to_right__optional = assign_to_right \| ""; | { LA_IS, { T_LRASSIGN_0 }, {<br>"assign_to_right__optional",{\<br>{ LA_IS, {""}, 1, { "assign_to_right" }}\<br>}}},\<br>{ LA_NOT, { T_LRASSIGN_0 }, {<br>"assign_to_right__optional",{\<br>{ LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| if_expression = expression; | if_expression = expression; | if_expression → expression | if_expression(1: "(", "NOT", "+", "-", "_", "**0**", "**1**", "**2**",<br>"**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "IF") → expression | if_expression = SAME_RULE(expression); | if_expression = SAME_RULE(expression); | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0,<br>"ident_terminal", "unsigned_value_terminal", T_IF_0 },<br>{ "if_expression",{\<br>{LA_IS, {""}, 1, { "expression" }}\<br>}}},\ |
| body_for_true = block_statements_in_while_and_if_body; | body_for_true =<br>block_statements_in_while_and_if_body; | body_for_true →<br>block_statements_in_while_and_if_body | body_for_true(1: "{") →<br>block_statements_in_while_and_if_body | body_for_true =<br>SAME_RULE(block_statements_in_while_and_if_body); | body_for_true =<br>SAME_RULE(block_statements_in_while_and_if_body); | {LA_IS, { T_BEGIN_BLOCK_0 }, { "body_for_true",{\<br>{LA_IS, {""}, 1, {<br>"block_statements_in_while_and_if_body" }}\<br>}}},\ |
| false_cond_block_without_else = "ELSE" , "IF" , if_expression , body_for_true; | false_cond_block_without_else = "ELSE" , "IF" ,<br>if_expression , body_for_true; | false_cond_block_without_else → "ELSE" "IF"<br>if_expression body_for_true | false_cond_block_without_else(1: "ELSE") → "ELSE"<br>"IF" if_expression body_for_true | false_cond_block_without_else = tokenELSE >> tokenIF<br>>> if_expression >> body_for_true; | false_cond_block_without_else = tokenELSE >> tokenIF >><br>if_expression >> body_for_true; | {LA_IS, { T_ELSE_IF_0 }, {<br>"false_cond_block_without_else",{\<br>{LA_IS, {""}, 4, { T_ELSE_IF_0, T_ELSE_IF_1,<br>"if_expression", "body_for_true" }}\<br>}}},\ |
| body_for_false = "ELSE" , block_statements_in_while_and_if_body; | body_for_false = "ELSE" ,<br>block_statements_in_while_and_if_body; | body_for_false → "ELSE"<br>block_statements_in_while_and_if_body | body_for_false(1: "ELSE") → "ELSE"<br>block_statements_in_while_and_if_body | body_for_false = tokenELSE >><br>block_statements_in_while_and_if_body; | body_for_false = tokenELSE >><br>block_statements_in_while_and_if_body; | {LA_IS, { T_ELSE_BLOCK_0 }, { "body_for_false",{\<br>{LA_IS, {""}, 2, { T_ELSE_BLOCK_0, "block_statements"<br>}}\<br>}}},\ |
| cond_block = "IF" , if_expression , body_for_true, { false_cond_block_without_else } , [body_for_false]; | cond_block = "IF" , if_expression , body_for_true,<br>false_cond_block_without_else__iteration ,<br>body_for_false__optional; | cond_block → "IF" if_expression body_for_true<br>false_cond_block_without_else__iteration<br>body_for_false__optional | cond_block(1: "IF") → "IF" if_expression<br>body_for_true<br>false_cond_block_without_else__iteration<br>body_for_false__optional | cond_block = tokenIF >> if_expression >> body_for_true >><br>*false_cond_block_without_else >> -body_for_false; | cond_block = tokenIF >> if_expression >><br>body_for_true >><br>false_cond_block_without_else__iteration >><br>body_for_false__optional; | {LA_IS, { T_IF_0 }, { "cond_block",{\<br>{LA_IS, {""}, 5, { T_IF_0, "if_expression",<br>"body_for_true",<br>"false_cond_block_without_else__iteration",<br>"body_for_false__optional" }}\<br>}}},\ |
| | false_cond_block_without_else__iteration =<br>false_cond_block_without_else,<br>false_cond_block_without_else__iteration \| ε; | false_cond_block_without_else__iteration →<br>false_cond_block_without_else<br>false_cond_block_without_else__iteration<br>false_cond_block_without_else__iteration → ε | false_cond_block_without_else__iteration(1: "ELSE";<br>2: "IF") → false_cond_block_without_else<br>false_cond_block_without_else__iteration<br>false_cond_block_without_else__iteration(1: "ELSE";<br>2: !"IF") → ε<br>false_cond_block_without_else__iteration(1:<br>!"ELSE") → ε | | false_cond_block_without_else__iteration =<br>false_cond_block_without_else >><br>false_cond_block_without_else__iteration \| ""; | {LA_IS, { T_ELSE_IF_0 }, {<br>"false_cond_block_without_else__iteration",{\<br>{LA_IS, {T_ELSE_IF_1}, 2, {<br>"false_cond_block_without_else",<br>"false_cond_block_without_else__iteration" }},\<br>{LA_NOT, { T_ELSE_IF_1 }, 0, { "" }}\<br>}}},\<br>{LA_NOT, { T_ELSE_IF_0 }, {<br>"false_cond_block_without_else__iteration",{\<br>{LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| | body_for_false__optional = body_for_false \| ε; | body_for_false__optional → body_for_false<br>body_for_false__optional → ε | body_for_false__optional(1: "FALSE") →<br>body_for_false<br>body_for_false__optional(1: !"FALSE") → ε | | body_for_false__optional = body_for_false \| ""; | {LA_IS, { T_ELSE_BLOCK_0 }, {<br>"body_for_false__optional",{\<br>{LA_IS, {""}, 1, { "body_for_false" }}\<br>}}},\<br>{LA_NOT, { T_ELSE_BLOCK_0 }, {<br>"body_for_false__optional",{\<br>{LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| cycle_begin_expression = expression; | cycle_begin_expression = expression; | cycle_begin_expression → expression | cycle_begin_expression(1: "(", "NOT", "+", "-", "_",<br>"**0**", "**1**", "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "IF") →<br>expression | cycle_begin_expression = SAME_RULE(expression); | cycle_begin_expression = SAME_RULE(expression); | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0,<br>"ident_terminal", "unsigned_value_terminal", T_IF_0 },<br>{ "cycle_begin_expression",{\<br>{LA_IS, {""}, 1, { "expression" }}\<br>}}},\ |
| cycle_end_expression = expression; | cycle_end_expression = expression; | cycle_end_expression → expression | cycle_end_expression(1: "(", "NOT", "+", "-", "_", "**0**",<br>"**1**", "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "IF") →<br>expression | cycle_end_expression = SAME_RULE(expression); | cycle_end_expression = SAME_RULE(expression); | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0,<br>"ident_terminal", "unsigned_value_terminal", T_IF_0 },<br>{ "cycle_end_expression",{\<br>{LA_IS, {""}, 1, { "expression" }}\<br>}}},\ |
| cycle_counter = ident; | cycle_counter = ident; | cycle_counter → ident | cycle_counter(1: "_") → ident | cycle_counter = SAME_RULE(ident); | cycle_counter = SAME_RULE(ident); | {LA_IS, { "ident_terminal" }, { "cycle_counter",{\<br>{LA_IS, {""}, 1, { "ident" }}\<br>}}},\ |
| cycle_counter_lr_init = cycle_begin_expression , "=:" , cycle_counter; | cycle_counter_lr_init = cycle_begin_expression ,<br>"=:" , cycle_counter; | cycle_counter_lr_init → cycle_begin_expression<br>"=:" cycle_counter | cycle_counter_lr_init(1: "(", "NOT", "+", "-", "_", "**0**",<br>"**1**", "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "IF") →<br>cycle_begin_expression "=:" cycle_counter | cycle_counter_lr_init = cycle_begin_expression >><br>tokenLRASSIGN >> cycle_counter; | cycle_counter_lr_init = cycle_begin_expression >><br>tokenLRASSIGN >> cycle_counter; | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0,<br>"ident_terminal", "unsigned_value_terminal", T_IF_0 },<br>{ "cycle_counter_lr_init",{\<br>{LA_IS, {""}, 3, { "cycle_begin_expression",<br>T_LRASSIGN_0, "cycle_counter" }}\<br>}}},\ |
| cycle_counter_init = cycle_counter_lr_init; | cycle_counter_init = cycle_counter_lr_init; | cycle_counter_init → cycle_counter_lr_init | cycle_counter_init(1: "(", "NOT", "+", "-", "_", "**0**",<br>"**1**", "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "IF") →<br>cycle_counter_lr_init | cycle_counter_init = SAME_RULE(cycle_counter_lr_init); | cycle_counter_init =<br>SAME_RULE(cycle_counter_lr_init); | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0,<br>"ident_terminal", "unsigned_value_terminal", T_IF_0 },<br>{ "cycle_counter_init",{\<br>{LA_IS, { "" }, 1, { "cycle_counter_lr_init" }}\<br>}}},\ |

| | | | | | | |
|---|---|---|---|---|---|---|
| cycle_counter_last_value = cycle_end_expression; | cycle_counter_last_value = cycle_end_expression; | cycle_counter_last_value → cycle_end_expression | cycle_counter_last_value(1: "(", "NOT", "+", "-", " ", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "IF") → cycle_end_expression | cycle_counter_last_value = SAME_RULE(cycle_end_expression); | cycle_counter_last_value = SAME_RULE(cycle_end_expression); | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0, "ident_terminal", "unsigned_value_terminal", T_IF_0 }, { "cycle_counter_last_value",{\ {LA_IS, {""}, 1, { "cycle_end_expression" }}\ }}},\ |
| cycle_body = "DO" , ({statement} \| block_statements); | cycle_body = "DO" , statements__or__block_statements; | cycle_body → "DO" statements__or__block_statements | cycle_body(1: "DO") → "DO" statements__or__block_statements | cycle_body = tokenDO >> (statement \| block_statements); | cycle_body = tokenDO >> statements__or__block_statements; | {LA_IS, { T_DO_0 }, { "cycle_body",{\ {LA_IS, {""}, 2, { T_DO_0, "statements__or__block_statements" }}\ }}},\ |
| | forto_direction = "TO" \| "DOWNTO"; | forto_direction → "TO" forto_direction → "DOWNTO" | forto_direction(1: "TO") → "TO" forto_direction(1: "DOWNTO") → "DOWNTO" | | forto_direction = tokenTO \| tokenDOWNTO; | {LA_IS, { T_TO_0 }, { "forto_direction",{\ {LA_IS, {""}, 1, { T_TO_0 }}\ }}},\ {LA_IS, { T_DOWNTO_0 }, { "forto_direction",{\ {LA_IS, {""}, 1, { T_DOWNTO_0 }}\ }}},\ |
| forto_cycle = "FOR" , cycle_counter_init , ("TO" \| "DOWNTO") , cycle_counter_last_value , cycle_body; | forto_cycle = "FOR" , cycle_counter_init , forto_direction, cycle_counter_last_value , cycle_body; | forto_cycle → "FOR" cycle_counter_init forto_direction, cycle_counter_last_value cycle_body | forto_cycle(1: "FOR") → "FOR" cycle_counter_init forto_direction, cycle_counter_last_value cycle_body | forto_cycle = tokenFOR >> cycle_counter_init >> (tokenTO \| tokenDOWNTO) >> cycle_counter_last_value >> cycle_body; | forto_cycle = tokenFOR >> cycle_counter_init >> forto_direction >> cycle_counter_last_value >> cycle_body; | {LA_IS, { T_FOR_0 }, { "forto_cycle",{\ {LA_IS, {""}, 5, { T_FOR_0, "cycle_counter_init", "forto_direction", "cycle_counter_last_value", "cycle_body" }}\ }}},\ |
| | continue_while = "CONTINUE"; | continue_while → "CONTINUE" | continue_while(1: "CONTINUE") → "CONTINUE" | continue_while = SAME_RULE(tokenCONTINUE); | continue_while = SAME_RULE(tokenCONTINUE); | {LA_IS, { T_CONTINUE_WHILE_0 }, { "continue_while",{\ {LA_IS, {""}, 1, { T_CONTINUE_WHILE_0 }}\ }}},\ |
| | break_while = "BREAK"; | break_while → "BREAK" | break_while(1: "BREAK") → "BREAK" | break_while = SAME_RULE(tokenBREAK); | break_while = SAME_RULE(tokenBREAK); | {LA_IS, { T_EXIT_WHILE_0 }, { "break_while",{\ {LA_IS, {""}, 1, { T_EXIT_WHILE_0 }}\ }}},\ |
| statement_in_while_and_if_body = statement \| "CONTINUE" \| "BREAK"; | statement_in_while_and_if_body = statement \| continue_while \| break_while; | statement_in_while_and_if_body → statement statement_in_while_and_if_body → continue_while statement_in_while_and_if_body → break_while | statement_in_while_and_if_body(1: "_", "(", "NOT", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "+", "-", "IF", "FOR", "WHILE", "REPEAT", "GOTO", "GET", "PUT", ";") → statement statement_in_while_and_if_body(1: "CONTINUE") → continue_while statement_in_while_and_if_body(1: "BREAK") → break_while | statement_in_while_and_if_body = statement \| continue_while \| break_while; | statement_in_while_and_if_body = statement \| continue_while \| break_while; | {LA_IS, { "ident_terminal", "(", T_NOT_0, "unsigned_value_terminal", T_ADD_0, T_SUB_0, T_IF_0, T_FOR_0, T_WHILE_0, T_REPEAT_0, T_GOTO_0, T_INPUT_0, T_OUTPUT_0, T_SEMICOLON_0 }, { "statement_in_while_and_if_body",{\ {LA_IS, {""}, 1, { "statement" }}\ }}},\ {LA_IS, { T_CONTINUE_WHILE_0 }, { "statement_in_while_and_if_body",{\ {LA_IS, {""}, 1, { "continue_while" }}\ }}},\ {LA_IS, { T_EXIT_WHILE_0 }, { "statement_in_while_and_if_body",{\ {LA_IS, {""}, 1, { "break_while" }}\ }}},\ |
| block_statements_in_while_and_if_body = "{" , {statement_in_while_and_if_body} , "}"; | block_statements_in_while_and_if_body = "{" , statement_in_while_and_if_body__iteration , "}"; | block_statements_in_while_and_if_body → "{" statement_in_while_and_if_body__iteration "}" | block_statements_in_while_and_if_body(1: "{") → "{" statement_in_while_and_if_body__iteration "}" | block_statements_in_while_and_if_body = tokenBEGINBLOCK >> *statement_in_while_and_if_body >> tokenENDBLOCK; | block_statements_in_while_and_if_body = tokenBEGINBLOCK >> statement_in_while_and_if_body__iteration >> tokenENDBLOCK; | {LA_IS, { T_BEGIN_BLOCK_0 }, { "block_statements_in_while_and_if_body",{\ {LA_IS, {""}, 3, { T_BEGIN_BLOCK_0, "statement_in_while_and_if_body__iteration", T_END_BLOCK_0 }}\ }}},\ |
| | statement_in_while_and_if_body__iteration = statement_in_while_and_if_body , statement_in_while_and_if_body__iteration \| ε; | statement_in_while_and_if_body__iteration → statement_in_while_and_if_body statement_in_while_and_if_body__iteration statement_in_while_and_if_body__iteration → ε | statement_in_while_and_if_body__iteration(1: "_", "(", "NOT", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "+", "-", "IF", "FOR", "WHILE", "REPEAT", "GOTO", "GET", "PUT", ";", "CONTINUE", ";") → statement_in_while_and_if_body statement_in_while_and_if_body__iteration statement_in_while_and_if_body__iteration(1: !"_", !"(", !"NOT", !"0", !"1", !"2", !"3", !"4", !"5", !"6", !"7", !"8", !"9",!"+", !"-", !"IF", !"FOR", !"WHILE", !"REPEAT", !"GOTO", !"GET", !"PUT", !";", !"CONTINUE", !"BREAK") → ε | | statement_in_while_and_if_body__iteration = statement_in_while_and_if_body >> statement_in_while_and_if_body__iteration \| ""; | {LA_IS, { "ident_terminal", "(", T_NOT_0, "unsigned_value_terminal", T_ADD_0, T_SUB_0, T_IF_0, T_FOR_0, T_WHILE_0, T_REPEAT_0, T_GOTO_0, T_INPUT_0, T_OUTPUT_0, T_SEMICOLON_0, T_CONTINUE_WHILE_0, T_EXIT_WHILE_0 }, { "statement_in_while_and_if_body__iteration",{\ {LA_IS, {""}, 2, { "statement_in_while_and_if_body", "statement_in_while_and_if_body__iteration" }}\ }}},\ {LA_NOT, { "ident_terminal", "(", T_NOT_0, "unsigned_value_terminal", T_ADD_0, T_SUB_0, T_IF_0, T_FOR_0, T_WHILE_0, T_REPEAT_0, T_GOTO_0, T_INPUT_0, T_OUTPUT_0, T_SEMICOLON_0, T_CONTINUE_WHILE_0, T_EXIT_WHILE_0 }, { "statement_in_while_and_if_body__iteration",{\ {LA_IS, {""}, 0, { "" }}\ }}},\ |
| while_cycle_head_expression = expression; | while_cycle_head_expression = expression; | while_cycle_head_expression → expression | while_cycle_head_expression(1: "(", "NOT", "+", "-", "_", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "IF") → expression | while_cycle_head_expression = SAME_RULE(expression); | while_cycle_head_expression = SAME_RULE(expression); | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0, "ident_terminal", "unsigned_value_terminal", T_IF_0 }, { "while_cycle_head_expression",{\ {LA_IS, {""}, 1, { "expression" }}\ }}},\ |
| while_cycle = "WHILE" , while_cycle_head_expression , block_statements_in_while_and_if_body; | while_cycle = "WHILE" , while_cycle_head_expression , block_statements_in_while_and_if_body; | while_cycle → "WHILE" while_cycle_head_expression block_statements_in_while_and_if_body | while_cycle(1: "WHILE") → "WHILE" while_cycle_head_expression block_statements_in_while_and_if_body | while_cycle = tokenWHILE >> while_cycle_head_expression >> block_statements_in_while_and_if_body; | while_cycle = tokenWHILE >> while_cycle_head_expression >> block_statements_in_while_and_if_body; | {LA_IS, { T_WHILE_0 }, { "while_cycle",{\ {LA_IS, {""}, 3, { T_WHILE_0, "while_cycle_head_expression", "block_statements_in_while_and_if_body" }}\ }}},\ |
| repeat_until_cycle_cond = expression; | repeat_until_cycle_cond = expression; | repeat_until_cycle_cond → expression | repeat_until_cycle_cond(1: "(", "NOT", "+", "-", "_", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "IF") → expression | repeat_until_cycle_cond = SAME_RULE(expression); | repeat_until_cycle_cond = SAME_RULE(expression); | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0, "ident_terminal", "unsigned_value_terminal", T_IF_0 }, { "repeat_until_cycle_cond",{\ {LA_IS, {""}, 1, { "expression" }}\ }}},\ |
| repeat_until_cycle = "REPEAT" , ({statement} \| block_statements) , "UNTIL" , repeat_until_cycle_cond; | repeat_until_cycle = "REPEAT" , statements__or__block_statements, "UNTIL" , repeat_until_cycle_cond; | repeat_until_cycle → "REPEAT" statements__or__block_statements "UNTIL" repeat_until_cycle_cond | repeat_until_cycle(1: "REPEAT") → "REPEAT" statements__or__block_statements "UNTIL" repeat_until_cycle_cond | repeat_until_cycle = tokenREPEAT >> (*statement \| block_statements) >> tokenUNTIL >> repeat_until_cycle_cond; | repeat_until_cycle = tokenREPEAT >> statements__or__block_statements >> tokenUNTIL >> repeat_until_cycle_cond; | {LA_IS, { T_REPEAT_0 }, { "repeat_until_cycle",{\ {LA_IS, {""}, 4, { T_REPEAT_0, "statements__or__block_statements", T_UNTIL_0, "repeat_until_cycle_cond" }}\ }}},\ |
| | statements__or__block_statements = statement__iteration \| block_statements; | statements__or__block_statements → statement__iteration statements__or__block_statements → block_statements | statements__or__block_statements(1: "_", "(", "NOT", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "+", "-", "IF", "FOR", "WHILE", "REPEAT", "GOTO", "GET", "PUT", ";") → statement__iteration statements__or__block_statements(1: "{") → block_statements | | statements__or__block_statements = statement__iteration \| block_statements; | {LA_IS, { "ident_terminal", "(", T_NOT_0, "unsigned_value_terminal", T_ADD_0, T_SUB_0, T_IF_0, T_FOR_0, T_WHILE_0, T_REPEAT_0, T_GOTO_0, T_INPUT_0, T_OUTPUT_0, T_SEMICOLON_0 }, { "statements__or__block_statements",{\ {LA_IS, {""}, 1, { "statement__iteration" }}\ }}},\ {LA_IS, { T_BEGIN_BLOCK_0 }, { "statements__or__block_statements",{\ {LA_IS, {""}, 1, { "block_statements" }}\ }}},\ |
| input_rule = "GET" , ( ident , [index_action] \| "(" , ident , [index_action] , ")" ); | input_rule = "GET", argument_for_input; | input_rule → "GET" argument_for_input | input_rule(1: "GET") → "GET" argument_for_input | input_rule = tokenGET >> (ident >> -index_action \| tokenGROUPEXPRESSIONBEGIN >> ident >> -index_action >> tokenGROUPEXPRESSIONEND); | input_rule = tokenGET >> argument_for_input; | {LA_IS, { T_INPUT_0 }, { "input_rule",{\ {LA_IS, {""}, 2, { T_INPUT_0, "argument_for_input" }}\ }}},\ |
| | argument_for_input = ident , index_action__optional; argument_for_input = "(", "ident", "index_action__optional", ")"; | argument_for_input → ident index_action__optional argument_for_input → "(" "ident" "index_action__optional" ")" | argument_for_input(1: "_") → ident index_action__optional argument_for_input(1: "(") → "(" "ident" "index_action__optional" ")" | | argument_for_input = ident >> index_action__optional \| tokenGROUPEXPRESSIONBEGIN >> ident >> index_action__optional >> tokenGROUPEXPRESSIONEND; | {LA_IS, { "**ident_terminal**" }, { "argument_for_input",{\ {LA_IS, {""}, 2, { "ident", "index_action__optional" }}\ }}},\ {LA_IS, { "(" }, { "argument_for_input",{\ {LA_IS, {""}, 4, { "(", "ident", "index_action__optional", ")" }}\ }}},\ |
| output_rule = "PUT" , expression; | output_rule = "PUT", expression; | output_rule → "PUT" expression | output(1: "PUT") → "PUT" expression | output_rule = tokenPUT >> expression; | output_rule = tokenPUT >> expression; | {LA_IS, { T_OUTPUT_0 }, { "output_rule", {\ {LA_IS, { "" }, 2, {T_OUTPUT_0, "expression"} }\ |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | }}},\ |
| statement = labeled_point \| expression_or_cond_block__with_optional_assign \| goto_label \| forto_cycle \| while_cycle \| repeat_until_cycle \| input_rule \| output_rule \| ";"; | statement = labeled_point \| expression_or_cond_block__with_optional_assign \| goto_label \| forto_cycle \| while_cycle \| repeat_until_cycle \| input_rule \| output_rule \| ";"; | statement → labeled_point<br>statement → expression_or_cond_block__with_optional_assign<br>statement → goto_label<br>statement → forto_cycle<br>statement → while_cycle<br>statement → repeat_until_cycle<br>statement → input_rule<br>statement → output_rule<br>statement → ";" | statement(1: "_", ":") → labeled_point<br>statement(1: "_"; 2: !":") → expression_or_cond_block__with_optional_assign<br>statement(1: "(", "NOT", "+", "-", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "IF") → expression_or_cond_block__with_optional_assign<br>statement(1: "GOTO") → goto_label<br>statement(1: "FOR") → forto_cycle<br>statement(1: "WHILE") → while_cycle<br>statement(1: "REPEAT") → repeat_until_cycle<br>statement(1: "GET") → input_rule<br>statement(1: "PUT") → output_rule<br>statement(1: ";") → ";" | statement = labeled_point /* labeled_point first*/ \| expression_or_cond_block__with_optional_assign \| goto_label \| forto_cycle \| while_cycle \| repeat_until_cycle \| input_rule \| output_rule \| tokenSEMICOLON; | statement = labeled_point /* labeled_point first*/ \| expression_or_cond_block__with_optional_assign \| goto_label \| forto_cycle \| while_cycle \| repeat_until_cycle \| input_rule \| output_rule \| tokenSEMICOLON; | {LA_IS, { "**ident_terminal**" }, { "statement", {\<br>{ LA_IS, { T_COLON_0 }, 1, {"labeled_point"}},\<br>{ LA_NOT, { T_COLON_0 }, 1, {"expression_or_cond_block__with_optional_assign"}}\<br>}}},\<br>{LA_IS, { "(", T_NOT_0, "unsigned_value_terminal", T_ADD_0, T_SUB_0, T_IF_0 }, { "statement", {\<br>{ LA_IS, {""}, 1, {"expression_or_cond_block__with_optional_assign"}}\<br>}}},\<br>{LA_IS, { T_GOTO_0 }, { "statement",{\<br>{LA_IS, {""}, 1, {"goto_label"}}\<br>}}},\<br>{LA_IS, { T_FOR_0 }, { "statement",{\<br>{LA_IS, {""}, 1, {"forto_cycle"}}\<br>}}},\<br>{LA_IS, { T_WHILE_0 }, { "statement",{\<br>{LA_IS, {""}, 1, {"while_cycle"}}\<br>}}},\<br>{LA_IS, { T_REPEAT_0 }, { "statement",{\<br>{LA_IS, {""}, 1, {"repeat_until_cycle"}}\<br>}}},\<br>{LA_IS, { T_INPUT_0 }, { "statement",{\<br>{LA_IS, {""}, 1, {"input_rule"}}\<br>}}},\<br>{LA_IS, { T_OUTPUT_0 }, { "statement",{\<br>{LA_IS, {""}, 1, {"output_rule"}}\<br>}}},\<br>{LA_IS, { T_SEMICOLON_0 }, { "statement",{\<br>{LA_IS, {""}, 1, {";"}}\<br>}}},\ |
| | statement__iteration = statement, statement__iteration \| ε; | statement__iteration → statement<br>statement__iteration<br>statement__iteration → ε | statement__iteration(1: "_", "(", "NOT", "**0**", "**1**", "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "+", "-", "IF", "FOR", "WHILE", "REPEAT", "GOTO", "GET", "PUT", ";") → statement statement__iteration<br>statement__iteration(1: !"_", !"(", !"NOT", !"**0**", !"**1**", !"**2**", !"**3**", !"**4**", !"**5**", !"**6**", !"**7**", !"**8**", !"**9**", !"+", !"-", !"IF", !"FOR", !"WHILE", !"REPEAT", !"GOTO", !"GET", !"PUT", !";") → ε | | statement__iteration = statement >> statement__iteration \| ""; | { LA_IS, { "ident_terminal", "(", T_NOT_0, "unsigned_value_terminal", T_ADD_0, T_SUB_0, T_IF_0, T_FOR_0, T_WHILE_0, T_REPEAT_0, T_GOTO_0, T_INPUT_0, T_OUTPUT_0, T_SEMICOLON_0 }, { "statement__iteration",{\<br>{ LA_IS, {""}, 2, { "statement", "statement__iteration" }}\<br>}}},\<br>{ LA_NOT, { "ident_terminal", "(", T_NOT_0, "unsigned_value_terminal", T_ADD_0, T_SUB_0, T_IF_0, T_FOR_0, T_WHILE_0, T_REPEAT_0, T_GOTO_0, T_INPUT_0, T_OUTPUT_0, T_SEMICOLON_0 }, { "statement__iteration",{\<br>{ LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| block_statements = "{" , {statement} , "}"; | block_statements = "{" , statement__iteration , "}"; | block_statements → "{" statement__iteration "}" | block_statements(1: "{") → "{" statement__iteration "}" | block_statements = tokenBEGINBLOCK >> *statement >> tokenENDBLOCK; | block_statements = tokenBEGINBLOCK >> statement__iteration >> tokenENDBLOCK; | { LA_IS, { T_BEGIN_BLOCK_0 }, { "block_statements",{\<br>{ LA_IS, {""}, 3, { T_BEGIN_BLOCK_0, "statement__iteration", T_END_BLOCK_0 }}\<br>}}},\ |
| | expression__optional = expression \| ""; | expression__optional → expression<br>expression__optional → ε | expression__optional(1: "(", "NOT", "+", "-", "_", "**0**", "**1**", "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "IF") → expression<br>expression__optional(1: !"(", !"NOT", !"+", !"-", !"_", !"**0**", !"**1**", !"**2**", !"**3**", !"**4**", !"**5**", !"**6**", !"**7**", !"**8**", !"**9**", !"IF") → ε | | expression__optional = expression \| ""; | {LA_IS, { "(", T_NOT_0, T_ADD_0, T_SUB_0, "ident_terminal", "unsigned_value_terminal", T_IF_0 }, { "expression__optional",{\<br>{LA_IS, {""}, 1, { "expression" }}\<br>}}},\<br>{LA_NOT, { "(", T_NOT_0, T_ADD_0, T_SUB_0, "ident_terminal", "unsigned_value_terminal", T_IF_0 }, { "expression__optional",{\<br>{LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| program_rule = "NAME" , program_name , ";" , "BODY" , "DATA", [declaration] , ";" , {statement} , "END"; | program_rule = "NAME" , program_name , ";" , "BODY" , "DATA", declaration__optional , ";" , statement__iteration , "END"; | program_rule → "NAME" program_name ";" "BODY" "DATA" declaration__optional ";" statement__iteration "END" | program_rule(1: "NAME") → "NAME" program_name ";" "BODY" "DATA" declaration__optional ";" statement__iteration "END" | program_rule = BOUNDARIES >> tokenNAME >> program_name >> tokenSEMICOLON >> tokenBODY >> tokenDATA >> (-declaration) >> tokenSEMICOLON >> *statement >> tokenEND; | program_rule = BOUNDARIES >> tokenNAME >> program_name >> tokenSEMICOLON >> tokenBODY >> tokenDATA >> declaration__optional >> tokenSEMICOLON >> statement__iteration >> tokenEND; | { LA_IS, { T_NAME_0 }, { "program_rule",{\<br>{ LA_IS, {""}, 9, { T_NAME_0, "program_name", T_SEMICOLON_0, T_BODY_0, T_DATA_0, "declaration__optional", T_SEMICOLON_0, "statement__iteration", T_END_0 }}\<br>}}},\ |
| | declaration__optional = declaration \| ""; | declaration__optional → declaration<br>declaration__optional → ε | declaration__optional(1: "INTEGER16") → declaration<br>declaration__optional(1: !"INTEGER16") → ε | | declaration__optional = declaration \| ""; | { LA_IS, { T_DATA_TYPE_0 }, { "declaration__optional",{\<br>{ LA_IS, {""}, 1, { "declaration" }}\<br>}}},\<br>{ LA_NOT, { T_DATA_TYPE_0 }, { "declaration__optional",{\<br>{ LA_IS, {""}, 0, { "" }}\<br>}}},\ |
| value = [sign] , unsigned_value; | value = sign__optional, unsigned_value; | value → sign__optional unsigned_value | value(1: "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "+", "-") → sign__optional unsigned_value | value = -sign >> unsigned_value >> BOUNDARIES; | value = sign__optional >> unsigned_value >> BOUNDARIES; | {LA_IS, { "unsigned_value_terminal", T_ADD_0, T_SUB_0 }, { "value", {\<br>{LA_IS, { "" }, 2, { "sign__optional", "unsigned_value" }}\<br>}}},\ |
| | sign__optional = sign \| ε; | sign__optional → sign<br>sign__optional → ε | sign__optional(1: "+", "-") → sign<br>sign__optional(1: !"+", !"-") → ε | | sign__optional = sign \| ""; | {LA_IS, { T_ADD_0, T_SUB_0 }, { "sign__optional", {\<br>{LA_IS, { "" }, 1, { "sign" }}\<br>}}},\<br>{LA_NOT, { T_ADD_0, T_SUB_0 }, { "sign__optional",{\<br>{LA_IS, { "" }, 0, { "" }}\<br>}}},\ |
| sign = sign_plus \| sign_minus; | sign = sign_plus \| sign_minus; | sign → sign_plus<br>sign → sign_minus | sign(1: "+") → sign_plus<br>sign(1: "-") → sign_minus | sign = sign_plus \| sign_minus; | sign = sign_plus \| sign_minus; | {LA_IS, { T_ADD_0 }, { "sign", {\<br>{LA_IS, {""}, 1, { "sign_plus" }}\<br>}}},\<br>{LA_IS, { T_SUB_0 }, { "sign", {\<br>{LA_IS, {""}, 1, { "sign_minus" } }\<br>}}},\ |
| sign_plus = "+"; | sign_plus = "+"; | sign_plus → "+" | sign_plus(1: "+") → "+" | sign_plus = SAME_RULE(tokenPLUS); | sign_plus = SAME_RULE(tokenPLUS); | {LA_IS, { T_ADD_0 }, { "sign_plus", {\<br>{LA_IS, {""}, 1, {T_ADD_0}}\<br>}}},\ |
| sign_minus = "-"; | sign_minus = "-"; | sign_minus → "-" | sign_minus(1: "-") → "-" | sign_minus = SAME_RULE(tokenMINUS); | sign_minus = SAME_RULE(tokenMINUS); | {LA_IS, { T_SUB_0 }, { "sign_minus", {\<br>{LA_IS, {""}, 1, {T_SUB_0} }\<br>}}},\ |
| unsigned_value = non_zero_digit , {digit} \| "0"; | unsigned_value = non_zero_digit , digit__iteration \| "0"; | unsigned_value → non_zero_digit digit__iteration<br>unsigned_value → "0" | unsigned_value(1: "1", "2", "3", "4", "5", "6", "7", "8", "9") → non_zero_digit digit__iteration<br>unsigned_value(1: "0") → "0" | unsigned_value = (non_zero_digit >> *digit \| digit_0) >> BOUNDARIES; | unsigned_value = (non_zero_digit >> digit__iteration \| digit_0) >> BOUNDARIES; | **/* <u>unsigned_value_token</u> represents <u>unsigned_value</u> in lexical analyzer */**<br>{LA_IS, { "**unsigned_value_terminal**" }, { "unsigned_value", {\<br>{LA_IS, { "" }, 1, { "**unsigned_value_terminal**" }}\<br>}}},\ |
| | digit__iteration = digit, digit__iteration \| ε; | digit__iteration → digit digit__iteration<br>digit__iteration → ε | digit__iteration(1: "0", "1", "2", "3", "4", "5", "6", "7", "8", "9") → digit digit__iteration<br>digit__iteration(1: !"0", !"1", !"2", !"3", !"4", !"5", !"6", !"7", !"8", !"9") → ε | | digit__iteration = digit >> digit__iteration \| ""; | \ |

| | | | | | | |
|---|---|---|---|---|---|---|
| digit = "0" \| non_zero_digit; | digit = "0" \| non_zero_digit; | digit → "0"<br>digit → non_zero_digit | digit(1: "0") → "0"<br>digit(1: "1", "2", "3", "4", "5", "6", "7", "8", "9",) → non_zero_digit | digit_0 = '0';<br>digit = digit_0 \| non_zero_digit; | digit_0 = '0';<br>digit = digit_0 \| non_zero_digit; | \ |
| non_zero_digit = "1" \| "2" \| "3" \| "4" \| "5" \| "6" \| "7" \| "8" \| "9"; | non_zero_digit = "1" \| "2" \| "3" \| "4" \| "5" \| "6" \| "7" \| "8" \| "9"; | non_zero_digit → "1"<br>non_zero_digit → "2"<br>non_zero_digit → "3"<br>non_zero_digit → "4"<br>non_zero_digit → "5"<br>non_zero_digit → "6"<br>non_zero_digit → "7"<br>non_zero_digit → "8"<br>non_zero_digit → "9" | non_zero_digit(1: "1") → "1"<br>non_zero_digit(1: "2") → "2"<br>non_zero_digit(1: "3") → "3"<br>non_zero_digit(1: "4") → "4"<br>non_zero_digit(1: "5") → "5"<br>non_zero_digit(1: "6") → "6"<br>non_zero_digit(1: "7") → "7"<br>non_zero_digit(1: "8") → "8"<br>non_zero_digit(1: "9") → "9" | digit_1 = '1';<br>digit_2 = '2';<br>digit_3 = '3';<br>digit_4 = '4';<br>digit_5 = '5';<br>digit_6 = '6';<br>digit_7 = '7';<br>digit_8 = '8';<br>digit_9 = '9';<br>non_zero_digit = digit_1 \| digit_2 \| digit_3 \| digit_4 \| digit_5 \| digit_6 \| digit_7 \| digit_8 \| digit_9; | digit_1 = '1';<br>digit_2 = '2';<br>digit_3 = '3';<br>digit_4 = '4';<br>digit_5 = '5';<br>digit_6 = '6';<br>digit_7 = '7';<br>digit_8 = '8';<br>digit_9 = '9';<br>non_zero_digit = digit_1 \| digit_2 \| digit_3 \| digit_4 \| digit_5 \| digit_6 \| digit_7 \| digit_8 \| digit_9; | \ |
| ident = "_" , letter_in_upper_case , letter_in_upper_case , letter_in_upper_case , letter_in_upper_case , letter_in_upper_case , letter_in_upper_case , letter_in_upper_case , letter_in_upper_case; | ident = "_" , letter_in_upper_case ,<br>letter_in_upper_case , letter_in_upper_case ,<br>letter_in_upper_case , letter_in_upper_case ,<br>letter_in_upper_case , letter_in_upper_case; | ident → "_" letter_in_upper_case<br>letter_in_upper_case letter_in_upper_case<br>letter_in_upper_case letter_in_upper_case<br>letter_in_upper_case letter_in_upper_case | Ident(1: "_") → "_" letter_in_upper_case<br>letter_in_upper_case letter_in_upper_case<br>letter_in_upper_case letter_in_upper_case<br>letter_in_upper_case letter_in_upper_case | tokenUNDERSCORE = "_";<br>ident =<br>!(<br>tokenCOLON \|<br>tokenGOTO \|<br>tokenINTEGER16 \|<br>tokenCOMMA \|<br>tokenNOT \|<br>tokenAND \|<br>tokenOR \|<br>tokenEQUAL \|<br>tokenNOTEQUAL \|<br>tokenLESSOREQUAL \|<br>tokenGREATEROREQUAL \|<br>tokenPLUS \|<br>tokenMINUS \|<br>tokenMUL \|<br>tokenDIV \|<br>tokenMOD \|<br>tokenGROUPEXPRESSIONBEGIN \|<br>tokenGROUPEXPRESSIONEND \|<br>tokenLRASSIGN \|<br>tokenELSE \|<br>tokenIF \|<br>tokenDO \|<br>tokenFOR \|<br>tokenTO \|<br>tokenDOWNTO \|<br>tokenWHILE \|<br>tokenCONTINUE \|<br>tokenBREAK \|<br>tokenEXIT \|<br>tokenREPEAT \|<br>tokenUNTIL \|<br>tokenGET \|<br>tokenPUT \|<br>tokenNAME \|<br>tokenBODY \|<br>tokenDATA \|<br>tokenBEGIN \|<br>tokenEND \|<br>tokenBEGINBLOCK \|<br>tokenENDBLOCK \|<br>tokenLEFTSQUAREBRACKETS \|<br>tokenRIGHTSQUAREBRACKETS \|<br>tokenSEMICOLON<br>) >><br>tokenUNDERSCORE >> letter_in_upper_case >><br>letter_in_upper_case >> letter_in_upper_case >><br>letter_in_upper_case >> letter_in_upper_case >><br>letter_in_upper_case >> letter_in_upper_case >><br>STRICT_BOUNDARIES; | tokenUNDERSCORE = "_";<br>ident =<br>!(<br>tokenCOLON \|<br>tokenGOTO \|<br>tokenINTEGER16 \|<br>tokenCOMMA \|<br>tokenNOT \|<br>tokenAND \|<br>tokenOR \|<br>tokenEQUAL \|<br>tokenNOTEQUAL \|<br>tokenLESSOREQUAL \|<br>tokenGREATEROREQUAL \|<br>tokenPLUS \|<br>tokenMINUS \|<br>tokenMUL \|<br>tokenDIV \|<br>tokenMOD \|<br>tokenGROUPEXPRESSIONBEGIN \|<br>tokenGROUPEXPRESSIONEND \|<br>tokenLRASSIGN \|<br>tokenELSE \|<br>tokenIF \|<br>tokenDO \|<br>tokenFOR \|<br>tokenTO \|<br>tokenDOWNTO \|<br>tokenWHILE \|<br>tokenCONTINUE \|<br>tokenBREAK \|<br>tokenEXIT \|<br>tokenREPEAT \|<br>tokenUNTIL \|<br>tokenGET \|<br>tokenPUT \|<br>tokenNAME \|<br>tokenBODY \|<br>tokenDATA \|<br>tokenBEGIN \|<br>tokenEND \|<br>tokenBEGINBLOCK \|<br>tokenENDBLOCK \|<br>tokenLEFTSQUAREBRACKETS \|<br>tokenRIGHTSQUAREBRACKETS \|<br>tokenSEMICOLON<br>) >><br>tokenUNDERSCORE >> letter_in_upper_case >><br>letter_in_upper_case >> letter_in_upper_case >><br>letter_in_upper_case >> letter_in_upper_case >><br>letter_in_upper_case >> letter_in_upper_case >><br>STRICT_BOUNDARIES; | /* **ident_token** **represents** **ident** **in lexical analyzer** */\<br>{LA_IS, { "ident_terminal" }, { "ident", {\<br>{LA_IS, {""}, 1, {"ident_terminal"} }\<br>}}},\ |
| letter_in_lower_case = "a" \| "b" \| "c" \| "d" \| "e" \| "f" \| "g" \| "h" \| "i" \| "j" \| "k" \| "l" \| "m" \| "n" \| "o" \| "p" \| "q" \| "r" \| "s" \| "t" \| "u" \| "v" \| "w" \| "x" \| "y" \| "z"; | letter_in_lower_case = "a" \| "b" \| "c" \| "d" \| "e" \| "f" \| "g" \| "h" \| "i" \| "j" \| "k" \| "l" \| "m" \| "n" \| "o" \| "p" \| "q" \| "r" \| "s" \| "t" \| "u" \| "v" \| "w" \| "x" \| "y" \| "z"; | letter_in_lower_case → "a"<br>letter_in_lower_case → "b"<br>letter_in_lower_case → "c"<br>letter_in_lower_case → "d"<br>letter_in_lower_case → "e"<br>letter_in_lower_case → "f"<br>letter_in_lower_case → "g"<br>letter_in_lower_case → "h"<br>letter_in_lower_case → "i"<br>letter_in_lower_case → "j"<br>letter_in_lower_case → "k"<br>letter_in_lower_case → "l"<br>letter_in_lower_case → "m"<br>letter_in_lower_case → "n"<br>letter_in_lower_case → "o"<br>letter_in_lower_case → "p"<br>letter_in_lower_case → "q"<br>letter_in_lower_case → "r"<br>letter_in_lower_case → "s"<br>letter_in_lower_case → "t"<br>letter_in_lower_case → "u"<br>letter_in_lower_case → "v"<br>letter_in_lower_case → "w"<br>letter_in_lower_case → "x"<br>letter_in_lower_case → "y"<br>letter_in_lower_case → "z" | letter_in_lower_case(1: "a") → "a"<br>letter_in_lower_case(1: "b") → "b"<br>letter_in_lower_case(1: "c") → "c"<br>letter_in_lower_case(1: "d") → "d"<br>letter_in_lower_case(1: "e") → "e"<br>letter_in_lower_case(1: "f") → "f"<br>letter_in_lower_case(1: "g") → "g"<br>letter_in_lower_case(1: "h") → "h"<br>letter_in_lower_case(1: "i") → "i"<br>letter_in_lower_case(1: "j") → "j"<br>letter_in_lower_case(1: "k") → "k"<br>letter_in_lower_case(1: "l") → "l"<br>letter_in_lower_case(1: "m") → "m"<br>letter_in_lower_case(1: "n") → "n"<br>letter_in_lower_case(1: "o") → "o"<br>letter_in_lower_case(1: "p") → "p"<br>letter_in_lower_case(1: "q") → "q"<br>letter_in_lower_case(1: "r") → "r"<br>letter_in_lower_case(1: "s") → "s"<br>letter_in_lower_case(1: "t") → "t"<br>letter_in_lower_case(1: "u") → "u"<br>letter_in_lower_case(1: "v") → "v"<br>letter_in_lower_case(1: "w") → "w"<br>letter_in_lower_case(1: "x") → "x"<br>letter_in_lower_case(1: "y") → "y"<br>letter_in_lower_case(1: "z") → "z" | A = "A";<br>B = "B";<br>C = "C";<br>D = "D";<br>E = "E";<br>F = "F";<br>G = "G";<br>H = "H";<br>I = "I";<br>J = "J";<br>K = "K";<br>L = "L";<br>M = "M";<br>N = "N";<br>O = "O";<br>P = "P";<br>Q = "Q";<br>R = "R";<br>S = "S";<br>T = "T";<br>U = "U";<br>V = "V";<br>W = "W";<br>X = "X";<br>Y = "Y";<br>Z = "Z";<br>letter_in_lower_case = a \| b \| c \| d \| e \| f \| g \| h \| i \| j \| k \| l \| m \| n \| o \| p \| q \| r \| s \| t \| u \| v \| w \| x \| y \| z; | A = "A";<br>B = "B";<br>C = "C";<br>D = "D";<br>E = "E";<br>F = "F";<br>G = "G";<br>H = "H";<br>I = "I";<br>J = "J";<br>K = "K";<br>L = "L";<br>M = "M";<br>N = "N";<br>O = "O";<br>P = "P";<br>Q = "Q";<br>R = "R";<br>S = "S";<br>T = "T";<br>U = "U";<br>V = "V";<br>W = "W";<br>X = "X";<br>Y = "Y";<br>Z = "Z";<br>letter_in_lower_case = a \| b \| c \| d \| e \| f \| g \| h \| i \| j \| k \| l \| m \| n \| o \| p \| q \| r \| s \| t \| u \| v \| w \| x \| y \| z; | \ |
| letter_in_upper_case = "A" \| "B" \| "C" \| "D" \| "E" \| "F" \| "G" \| "H" \| "I" \| "J" \| "K" \| "L" \| "M" \| "N" \| "O" \| "P" \| "Q" \| "R" \| "S" \| "T" \| "U" \| "V" \| "W" \| "X" \| "Y" \| "Z"; | letter_in_upper_case = "A" \| "B" \| "C" \| "D" \| "E" \| "F" \| "G" \| "H" \| "I" \| "J" \| "K" \| "L" \| "M" \| "N" \| "O" \| "P" \| "Q" \| "R" \| "S" \| "T" \| "U" \| "V" \| "W" \| "X" \| "Y" \| "Z"; | letter_in_upper_case → "A"<br>letter_in_upper_case → "B"<br>letter_in_upper_case → "C"<br>letter_in_upper_case → "D"<br>letter_in_upper_case → "E"<br>letter_in_upper_case → "F"<br>letter_in_upper_case → "G"<br>letter_in_upper_case → "H"<br>letter_in_upper_case → "I"<br>letter_in_upper_case → "J"<br>letter_in_upper_case → "K"<br>letter_in_upper_case → "L"<br>letter_in_upper_case → "M"<br>letter_in_upper_case → "N" | letter_in_upper_case(1: "A") → "A"<br>letter_in_upper_case(1: "B") → "B"<br>letter_in_upper_case(1: "C") → "C"<br>letter_in_upper_case(1: "D") → "D"<br>letter_in_upper_case(1: "E") → "E"<br>letter_in_upper_case(1: "F") → "F"<br>letter_in_upper_case(1: "G") → "G"<br>letter_in_upper_case(1: "H") → "H"<br>letter_in_upper_case(1: "I") → "I"<br>letter_in_upper_case(1: "J") → "J"<br>letter_in_upper_case(1: "K") → "K"<br>letter_in_upper_case(1: "L") → "L"<br>letter_in_upper_case(1: "M") → "M"<br>letter_in_upper_case(1: "N") → "N" | a = "a";<br>b = "b";<br>c = "c";<br>d = "d";<br>e = "e";<br>f = "f";<br>g = "g";<br>h = "h";<br>i = "i";<br>j = "j";<br>k = "k";<br>l = "l";<br>m = "m";<br>n = "n"; | a = "a";<br>b = "b";<br>c = "c";<br>d = "d";<br>e = "e";<br>f = "f";<br>g = "g";<br>h = "h";<br>i = "i";<br>j = "j";<br>k = "k";<br>l = "l";<br>m = "m";<br>n = "n"; | \ |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | letter_in_upper_case → "O"<br>letter_in_upper_case → "P"<br>letter_in_upper_case → "Q"<br>letter_in_upper_case → "R"<br>letter_in_upper_case → "S"<br>letter_in_upper_case → "T"<br>letter_in_upper_case → "U"<br>letter_in_upper_case → "V"<br>letter_in_upper_case → "W"<br>letter_in_upper_case → "X"<br>letter_in_upper_case → "Y"<br>letter_in_upper_case → "Z" | letter_in_upper_case(1: "O") → "O"<br>letter_in_upper_case(1: "P") → "P"<br>letter_in_upper_case(1: "Q") → "Q"<br>letter_in_upper_case(1: "R") → "R"<br>letter_in_upper_case(1: "S") → "S"<br>letter_in_upper_case(1: "T") → "T"<br>letter_in_upper_case(1: "U") → "U"<br>letter_in_upper_case(1: "V") → "V"<br>letter_in_upper_case(1: "W") → "W"<br>letter_in_upper_case(1: "X") → "X"<br>letter_in_upper_case(1: "Y") → "Y"<br>letter_in_upper_case(1: "Z") → "Z" | o = "o";<br>p = "p";<br>q = "q";<br>r = "r";<br>s = "s";<br>t = "t";<br>u = "u";<br>v = "v";<br>w = "w";<br>x = "x";<br>y = "y";<br>z = "z";<br>letter_in_upper_case = A \| B \| C \| D \| E \| F \| G \| H \| I \| J \| K \| L \| M \| N \| O \| P \| Q \| R \| S \| T \| U \| V \| W \| X \| Y \| Z; | o = "o";<br>p = "p";<br>q = "q";<br>r = "r";<br>s = "s";<br>t = "t";<br>u = "u";<br>v = "v";<br>w = "w";<br>x = "x";<br>y = "y";<br>z = "z";<br>letter_in_upper_case = A \| B \| C \| D \| E \| F \| G \| H \| I \| J \| K \| L \| M \| N \| O \| P \| Q \| R \| S \| T \| U \| V \| W \| X \| Y \| Z; | |
| | | | | STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) \|<br>(!(qi::alpha \| qi::char_("_")));<br>BOUNDARIES = (BOUNDARY >> *(BOUNDARY) \|<br>NO_BOUNDARY);<br>BOUNDARY = BOUNDARY__SPACE \| BOUNDARY__TAB \|<br>BOUNDARY__VERTICAL_TAB \| BOUNDARY__FORM_FEED \|<br>BOUNDARY__CARRIAGE_RETURN \| BOUNDARY__LINE_FEED \|<br>BOUNDARY__NULL;<br>BOUNDARY__SPACE = " ";<br>BOUNDARY__TAB = "\t";<br>BOUNDARY__VERTICAL_TAB = "\v";<br>BOUNDARY__FORM_FEED = "\f";<br>BOUNDARY__CARRIAGE_RETURN = "\r";<br>BOUNDARY__LINE_FEED = "\n";<br>BOUNDARY__NULL = "\0";<br>NO_BOUNDARY = "";<br>#define WHITESPACES \<br>STRICT_BOUNDARIES, \<br>BOUNDARIES, \<br>BOUNDARY, \<br>BOUNDARY__SPACE, \<br>BOUNDARY__TAB, \<br>BOUNDARY__VERTICAL_TAB, \<br>BOUNDARY__FORM_FEED, \<br>BOUNDARY__CARRIAGE_RETURN, \<br>BOUNDARY__LINE_FEED, \<br>BOUNDARY__NULL, \<br>NO_BOUNDARY | STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY))<br>\| (!(qi::alpha \| qi::char_("_")));<br>BOUNDARIES = (BOUNDARY >> *(BOUNDARY) \|<br>NO_BOUNDARY);<br>BOUNDARY = BOUNDARY__SPACE \| BOUNDARY__TAB<br>\| BOUNDARY__VERTICAL_TAB \|<br>BOUNDARY__FORM_FEED \|<br>BOUNDARY__CARRIAGE_RETURN \|<br>BOUNDARY__LINE_FEED \| BOUNDARY__NULL;<br>BOUNDARY__SPACE = " ";<br>BOUNDARY__TAB = "\t";<br>BOUNDARY__VERTICAL_TAB = "\v";<br>BOUNDARY__FORM_FEED = "\f";<br>BOUNDARY__CARRIAGE_RETURN = "\r";<br>BOUNDARY__LINE_FEED = "\n";<br>BOUNDARY__NULL = "\0";<br>NO_BOUNDARY = "";<br>#define WHITESPACES \<br>STRICT_BOUNDARIES, \<br>BOUNDARIES, \<br>BOUNDARY, \<br>BOUNDARY__SPACE, \<br>BOUNDARY__TAB, \<br>BOUNDARY__VERTICAL_TAB, \<br>BOUNDARY__FORM_FEED, \<br>BOUNDARY__CARRIAGE_RETURN, \<br>BOUNDARY__LINE_FEED, \<br>BOUNDARY__NULL, \<br>NO_BOUNDARY | \ |
| | | | | | | \<br>\<br>\<br>{ LA_IS, { T_NAME_0 }, { "program____part1",{\<br>{ LA_IS, {""}, 7, { T_NAME_0, "program_name",<br>T_SEMICOLON_0, T_BODY_0, T_DATA_0,<br>"declaration_optional", T_SEMICOLON_0 }}\<br>}}},\<br>\<br>},\<br>"program_rule" |