|  |  |
| --- | --- |
| **РБНФ** | **Код для перевірки РБНФ** |
| program\_name = ident; | program\_name = SAME\_RULE(ident); |
| value\_type = "INTEGER16"; | value\_type = SAME\_RULE(tokenINT2); |
| declaration\_element = ident , [ "[", unsigned\_value , "]" ]; | declaration\_element = ident >> -(tokenLEFTSQUAREBRACKETS >> unsigned\_value >> tokenRIGHTSQUAREBRACKETS); |
| other\_declaration\_ident = "," , declaration\_element; | other\_declaration\_ident = tokenCOMMA >> declaration\_element; |
| declaration = value\_type , declaration\_element , {other\_declaration\_ident}; | declaration = value\_type >> declaration\_element >> \*other\_declaration\_ident; |
| index\_action = "[" , expression , "]"; | index\_action = tokenLEFTSQUAREBRACKETS >> expression >> tokenRIGHTSQUAREBRACKETS; |
| unary\_operator = "NOT"; | unary\_operator = SAME\_RULE(tokenNOT); |
| unary\_operation = unary\_operator , expression; | unary\_operation = unary\_operator >> expression; |
| binary\_operator = "AND" | "OR" | "==" | "!=" | "<=" | ">=" | "<" | ">" | "+" | "-" | "\*" | "DIV" | "MOD"; | binary\_operator = tokenAND | tokenOR | tokenEQUAL | tokenNOTEQUAL | tokenLESS | tokenGREATER | tokenPLUS | tokenMINUS | tokenMUL | tokenDIV | tokenMOD; |
| binary\_action = binary\_operator , expression; | binary\_action = binary\_operator >> expression; |
| left\_expression = group\_expression | unary\_operation | ident , [index\_action] | value | cond\_block\_\_with\_optionally\_return\_value; | left\_expression = group\_expression | unary\_operation | ident >> -index\_action | value; |
| expression = left\_expression , {binary\_action}; | expression = left\_expression >> \*binary\_action; |
| group\_expression = "(" , expression , ")"; | group\_expression = tokenGROUPEXPRESSIONBEGIN >> expression >> tokenGROUPEXPRESSIONEND; |
| bind\_left\_to\_right = expression , "=:" , ident , [index\_action]; | bind\_left\_to\_right = expression >> tokenLRBIND >> ident >> -index\_action; |
| if\_expression = expression; | if\_expression = SAME\_RULE(expression); |
| body\_for\_true\_\_with\_optionally\_return\_value = block\_statements\_\_with\_optionally\_return\_value; | body\_for\_true = SAME\_RULE(block\_statements\_in\_while\_and\_if\_body); |
| false\_cond\_block\_without\_else\_\_with\_optionally\_return\_value = "ELSE" , "IF" , if\_expression , body\_for\_true\_\_with\_optionally\_return\_value; | false\_cond\_block\_without\_else = tokenELSE >> cond\_block; |
| body\_for\_false\_\_with\_optionally\_return\_value = "ELSE" , block\_statements\_\_with\_optionally\_return\_value; | body\_for\_false = tokenELSE >> block\_statements\_in\_while\_and\_if\_body; |
| cond\_block\_\_with\_optionally\_return\_value = "IF" , if\_expression , body\_for\_true\_\_with\_optionally\_return\_value , {false\_cond\_block\_without\_else\_\_with\_optionally\_return\_value} , [body\_for\_false\_\_with\_optionally\_return\_value]; | cond\_block = tokenIF >> if\_expression >> body\_for\_true >> \*false\_cond\_block\_without\_else >> (-body\_for\_false); |
| cond\_block\_\_with\_optionally\_return\_value\_and\_optionally\_bind = cond\_block\_\_with\_optionally\_return\_value , [tokenLRBIND , ident , [index\_action]]; | block\_statements\_in\_while\_and\_if\_body = tokenBEGINBLOCK >> \*statement\_in\_while\_and\_if\_body >> tokenENDBLOCK; |
| statement\_in\_while\_and\_if\_body = statement | "CONTINUE" | "BREAK"; | statement\_in\_while\_and\_if\_body = statement | break\_while; |
| block\_statements\_in\_while\_and\_if\_body = "{" , {statement\_in\_while\_and\_if\_body} , "}"; | block\_statements\_in\_while\_and\_if\_body = tokenBEGINBLOCK >> \*statement\_in\_while\_and\_if\_body >> tokenENDBLOCK; |
| repeat\_until\_cycle\_cond = expression; | repeat\_until\_cycle\_cond = SAME\_RULE(expression); |
| repeat\_until\_cycle = "REPEAT" , ({statement} | block\_statements) , "UNTIL" , repeat\_until\_cycle\_cond; | repeat\_until\_cycle = tokenREPEAT >> (\*statement | block\_statements) >> tokenUNTIL >> repeat\_until\_cycle\_cond; |
| input = "GET" , ( ident , [index\_action] | "(" , ident , [index\_action] , ")" ); | input = tokenINPUT >> tokenGROUPEXPRESSIONBEGIN >> ident >> tokenGROUPEXPRESSIONEND; |
| output = "PUT" , expression; | output = tokenOUTPUT >> tokenGROUPEXPRESSIONBEGIN >> expression >> tokenGROUPEXPRESSIONEND; |
| statement = bind\_left\_to\_right | cond\_block\_\_with\_optionally\_return\_value\_and\_optionally\_bind | forto\_cycle | while\_cycle | repeat\_until\_cycle | labeled\_point | goto\_label | input | output | ";"; | statement = (bind\_left\_to\_right | cond\_block |repeat\_until\_cycle | input | output)>> -tokenSEMICOLON; |
| block\_statements = "{" , {statement} , "}"; | block\_statements = tokenBEGINBLOCK >> \*statement >> tokenENDBLOCK; |
| program = "NAME" , program\_name , ";" , "BODY" , "DATA", [declaration] , ";" , {statement} , "END"; | program = BOUNDARIES >> tokenPROGRAM >> ident >> tokenSEMICOLON>> tokenBEGIN >> tokenVAR >> (-declaration) >> tokenSEMICOLON>> \*statement >> tokenEND >> tokenSEMICOLON; |
| digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"; | digit = digit\_0 | digit\_1 | digit\_2 | digit\_3 | digit\_4 | digit\_5 | digit\_6 | digit\_7 | digit\_8 | digit\_9; |
| non\_zero\_digit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"; | non\_zero\_digit = digit\_1 | digit\_2 | digit\_3 | digit\_4 | digit\_5 | digit\_6 | digit\_7 | digit\_8 | digit\_9; |
| unsigned\_value = (non\_zero\_digit , {digit}) | "0"; | unsigned\_value = ((non\_zero\_digit >> \*digit) | digit\_0) >> BOUNDARIES; |
| value = [sign] , unsigned\_value; | value = (-sign) >> unsigned\_value >> BOUNDARIES; |
| 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; |
| ident = "\_" , letter\_in\_upper\_case , letter\_in\_upper\_case ; | ident = letter\_in\_upper\_case >> letter\_in\_upper\_case >> BOUNDARIES; |
| sign = "+" | "-"; | sign = sign\_plus | sign\_minus; |
|  | sign\_plus = '+' >> BOUNDARIES; |
|  | sign\_minus = '-' >> BOUNDARIES; |
|  | digit\_0 = '0'; |
|  | digit\_1 = '1'; |
|  | digit\_2 = '2'; |
|  | digit\_3 = '3'; |
|  | digit\_4 = '4'; |
|  | digit\_5 = '5'; |
|  | digit\_6 = '6'; |
|  | digit\_7 = '7'; |
|  | digit\_8 = '8'; |
|  | digit\_9 = '9'; |
|  | tokenCOLON = ":" >> BOUNDARIES; |
|  | tokenINT2 = "INT\_2" >> STRICT\_BOUNDARIES; |
|  | tokenCOMMA = "," >> BOUNDARIES; |
|  | tokenNOT = "!!" >> STRICT\_BOUNDARIES; |
|  | tokenAND = "&&" >> STRICT\_BOUNDARIES; |
|  | tokenOR = "||" >> STRICT\_BOUNDARIES; |
|  | tokenEQUAL = "=" >> BOUNDARIES; |
|  | tokenNOTEQUAL = "<>" >> BOUNDARIES; |
|  | tokenLESS = "LT" >> BOUNDARIES; |
|  | tokenGREATER = "GT" >> BOUNDARIES; |
|  | tokenPLUS = "+" >> BOUNDARIES; |
|  | tokenMINUS = "-" >> BOUNDARIES; |
|  | tokenMUL = "\*" >> BOUNDARIES; |
|  | tokenDIV = "DIV" >> STRICT\_BOUNDARIES; |
|  | tokenMOD = "MOD" >> STRICT\_BOUNDARIES; |
|  | tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES; |
|  | tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES; |
|  | tokenLRBIND = "->" >> BOUNDARIES; |
|  | tokenELSE = "ELSE" >> STRICT\_BOUNDARIES; |
|  | tokenIF = "IF" >> STRICT\_BOUNDARIES; |
|  | tokenREPEAT = "REPEAT" >> STRICT\_BOUNDARIES; |
|  | tokenUNTIL = "UNTIL" >> STRICT\_BOUNDARIES; |
|  | tokenINPUT = "INPUT" >> STRICT\_BOUNDARIES; |
|  | tokenOUTPUT = "OUTPUT" >> STRICT\_BOUNDARIES; |
|  | tokenPROGRAM = "PROGRAM" >> STRICT\_BOUNDARIES; |
|  | tokenBEGIN = "BEGIN" >> STRICT\_BOUNDARIES; |
|  | tokenVAR = "VAR" >> STRICT\_BOUNDARIES; |
|  | tokenEND = "END" >> STRICT\_BOUNDARIES; |
|  | tokenBEGINBLOCK = "{" >> BOUNDARIES; |
|  | tokenENDBLOCK = "}" >> BOUNDARIES; |
|  | tokenLEFTSQUAREBRACKETS = "[" >> BOUNDARIES; |
|  | tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES; |
|  | tokenSEMICOLON = ";" >> BOUNDARIES; |
|  | STRICT\_BOUNDARIES = (BOUNDARY >> \*(BOUNDARY)) | (!(qi::alpha | qi::char\_("\_"))); |
|  | BOUNDARIES = (BOUNDARY >> \*(BOUNDARY) | NO\_BOUNDARY); |
|  | BOUNDARY = BOUNDARY\_SPACE | BOUNDARY\_TAB | BOUNDARY\_CARRIAGE\_RETURN | BOUNDARY\_LINE\_FEED | BOUNDARY\_NULL; |
|  | BOUNDARY\_SPACE = " "; |
|  | BOUNDARY\_TAB = "\t"; |
|  | BOUNDARY\_CARRIAGE\_RETURN = "\r"; |
|  | BOUNDARY\_LINE\_FEED = "\n"; |
|  | BOUNDARY\_NULL = "\0"; |
|  | NO\_BOUNDARY = ""; |
|  | A = "A"; |
|  | B = "B"; |
|  | C = "C"; |
|  | D = "D"; |
|  | E = "E"; |
|  | F = "F"; |
|  | G = "G"; |
|  | H = "H"; |
|  | I = "I"; |
|  | J = "J"; |
|  | K = "K"; |
|  | L = "L"; |
|  | M = "M"; |
|  | N = "N"; |
|  | O = "O"; |
|  | P = "P"; |
|  | Q = "Q"; |
|  | R = "R"; |
|  | S = "S"; |
|  | T = "T"; |
|  | U = "U"; |
|  | V = "V"; |
|  | W = "W"; |
|  | X = "X"; |
|  | Y = "Y"; |
|  | Z = "Z"; |