

РБНФ	Опис граматики за допомогою РБНФ	Опис граматика	Код для перевірки РБНФ	Код для перевірки граматики заданої за допомогою РБНФ
		$G = (N, T, P, S)$		
		$S \rightarrow \text{tokens_in_program};$		
		$N = \{$ tokens_in_program, token_iteration, token, keyword, ident, letter_in_lower_case, letter_in_upper_case, value, sign_optional, sign, sign_plus, sign_minus, unsigned_value, digit, digit_optional, non_zero_digit $\}$		
		$T = \{$ ".", "GOTO", "INTEGER16", ",", "NOT", "AND", "OR", "==", "!=", "<=", ">=", "<", ">", "+", "-", "*", "DIV", "MOD", "(", ")", "::", "ELSE", $\}$		

		"IF", "DO", "FOR", "TO", "DOWNTO", "WHILE", "CONTINUE", "BREAK", "EXIT", "REPEAT", "UNTIL", "GET", "PUT", "NAME", "BODY", "DATA", "BEGIN", "END", "{", "}", "[", "]", ",", "._", "+", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", " _", "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", "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" }		
keyword = ":" "GOTO" "INTEGER16" ","	keyword = ":" "GOTO" "INTEGER16" ","	keyword → ":" keyword → "GOTO" keyword → "INTEGER16" keyword → "," keyword → "NOT"	keyword = tokenCOLON tokenGOTO tokenINTEGER16 tokenCOMMA	keyword = tokenCOLON tokenGOTO tokenINTEGER16 tokenCOMMA

"NOT" "AND" "OR" "==" "!=" "<=" ">=" "<" ">" "+" "-" "*" "DIV" "MOD" "(" ")" "=" "ELSE" "IF" "DO" "FOR" "TO" "DOWNT0" "WHILE" "CONTINUE" "BREAK" "EXIT" "REPEAT" "UNTIL" "GET" "PUT" "NAME" "BODY" "DATA" "BEGIN" "END" "{" "}" "[" "]" ";", ";"	"NOT" "AND" "OR" "==" "!=" "<=" ">=" "<" ">" "+" "-" "*" "DIV" "MOD" "(" ")" "=" "ELSE" "IF" "DO" "FOR" "TO" "DOWNT0" "WHILE" "CONTINUE" "BREAK" "EXIT" "REPEAT" "UNTIL" "GET" "PUT" "NAME" "BODY" "DATA" "BEGIN" "END" "{" "}" "[" "]" ";", ";"	keyword → "AND" keyword → "OR" keyword → "==" keyword → "!=" keyword → "<=" keyword → ">=" keyword → "<" keyword → ">" keyword → "+" keyword → "-" keyword → "*" keyword → "DIV" keyword → "MOD" keyword → "(" keyword → ")" keyword → "=: " keyword → "ELSE" keyword → "IF" keyword → "DO" keyword → "FOR" keyword → "TO" keyword → "DOWNT0" keyword → "WHILE" keyword → "CONTINUE" keyword → "BREAK" keyword → "EXIT" keyword → "REPEAT" keyword → "UNTIL" keyword → "GET" keyword → "PUT" keyword → "NAME" keyword → "BODY" keyword → "DATA" keyword → "BEGIN" keyword → "END" keyword → "{" keyword → "}" keyword → "[" keyword → "]" keyword → ";," keyword → ";"	tokenNOT tokenAND tokenOR tokenEQUAL tokenNOTEQUAL tokenLESSOREQUAL tokenGREATEROREQUAL tokenLESS tokenGREATER tokenPLUS tokenMINUS tokenMUL tokenDIV tokenMOD tokenGROUPEXPRESSIONBEGIN tokenGROUPEXPRESSIONEND tokenLRBIND tokenELSE tokenIF tokenDO tokenFOR tokenTO tokenDOWNT0 tokenWHILE tokenCONTINUE tokenBREAK tokenEXIT tokenREPEAT tokenUNTIL tokenGET tokenPUT tokenNAME tokenBODY tokenDATA tokenBEGIN tokenEND tokenBEGINBLOCK tokenENDBLOCK tokenLEFTSQUAREBRACKETS tokenRIGHTSQUAREBRACKETS tokenSEMICOLON;	tokenNOT tokenAND tokenOR tokenEQUAL tokenNOTEQUAL tokenLESSOREQUAL tokenGREATEROREQUAL tokenLESS tokenGREATER tokenPLUS tokenMINUS tokenMUL tokenDIV tokenMOD tokenGROUPEXPRESSION BEGIN tokenGROUPEXPRESSION END tokenLRBIND tokenELSE tokenIF tokenDO tokenFOR tokenTO tokenDOWNT0 tokenWHILE tokenCONTINUE tokenBREAK tokenEXIT tokenREPEAT tokenUNTIL tokenGET tokenPUT tokenNAME tokenBODY tokenDATA tokenBEGIN tokenEND tokenBEGINBLOCK tokenENDBLOCK tokenLEFTSQUAREBRACK ETS tokenRIGHTSQUAREBRAC KETS tokenSEMICOLON;
tokens_in_program = {	tokens_in_progra	tokens_in_program → token_iteration	tokens_in_program = BOUNDARIES >> *(keyword ident	tokens_in_program =

keyword ident value);	m = token_iteration ;		value);	SAME_RULE(token_iteration);
	token = keyword ident value;	token → keyword ident value;		token = keyword ident value;
	token_iteration = token , token_iteration ε;	token_iteration → token token_iteration token_iteration → ε		token_iteration = token >> token_iteration "";
digit = "0" non_zero_digit;	digit = digit_0 non_zero_digit;	digit → "0" digit → non_zero_digit	digit = digit_0 non_zero_digit;	digit = digit_0 non_zero_digit;
	digit_optional = digit ε;	digit_optional → digit; digit_optional → ε;		digit_optional = digit "";
non_zero_digit = "1" "2" "3" "4" "5" "6" "7" "8" "9";		non_zero_digit → "1" non_zero_digit → "2" non_zero_digit → "3" non_zero_digit → "4" non_zero_digit → "5" non_zero_digit → "6" non_zero_digit → "7" non_zero_digit → "8" non_zero_digit → "9"	non_zero_digit = digit_1 digit_2 digit_3 digit_4 digit_5 digit_6 digit_7 digit_8 digit_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_optional "0";	unsigned_value → non_zero_digit , digit_optional unsigned_value → "0"	unsigned_value = ((non_zero_digit >> *digit) digit_0) >> BOUNDARIES;	unsigned_value = non_zero_digit >> (digit_optional digit_0) >> BOUNDARIES;
value = [sign] , unsigned_value;		value → sign_optional unsigned_value;	value = -sign >> unsigned_value >> BOUNDARIES;	value = sign_optional >> unsigned_value >> BOUNDARIES;
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" letter_in_lower_case → "b" letter_in_lower_case → "c" letter_in_lower_case → "d" letter_in_lower_case → "e" letter_in_lower_case → "f" letter_in_lower_case → "g" letter_in_lower_case → "h" letter_in_lower_case → "i" letter_in_lower_case → "j" letter_in_lower_case → "k" letter_in_lower_case → "l" letter_in_lower_case → "m" letter_in_lower_case → "n" letter_in_lower_case → "o" letter_in_lower_case → "p" letter_in_lower_case → "q" letter_in_lower_case → "r" letter_in_lower_case → "r"	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 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 → "s" letter_in_lower_case → "t" letter_in_lower_case → "u" letter_in_lower_case → "v" letter_in_lower_case → "w" letter_in_lower_case → "x" letter_in_lower_case → "y" letter_in_lower_case → "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" letter_in_upper_case → "B" letter_in_upper_case → "C" letter_in_upper_case → "D" letter_in_upper_case → "E" letter_in_upper_case → "F" letter_in_upper_case → "G" letter_in_upper_case → "H" letter_in_upper_case → "I" letter_in_upper_case → "J" letter_in_upper_case → "K" letter_in_upper_case → "L" letter_in_upper_case → "M" letter_in_upper_case → "N" letter_in_upper_case → "O" letter_in_upper_case → "P" letter_in_upper_case → "Q" letter_in_upper_case → "R" letter_in_upper_case → "S" letter_in_upper_case → "T" letter_in_upper_case → "U" letter_in_upper_case → "V" letter_in_upper_case → "W" letter_in_upper_case → "X" letter_in_upper_case → "Y" letter_in_upper_case → "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, 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, 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 → "_", digit, letter_in_upper_case, letter_in_upper_case;	ident = tokenUNDERSCORE >> digit >> letter_in_upper_case >> letter_in_upper_case >> STRICT_BOUNDARIES;	ident = tokenUNDERSCORE >> digit >> letter_in_upper_case >> letter_in_upper_case >> STRICT_BOUNDARIES;

	letter_in_upper_case;			
sign = "+" "-";	sign = "+" "-";	sign_optional → "+" sign_optional → "-"	unary_minus = qi::char_('-') >> (qi::char_('-'));	unary_minus = "--";
	sign_optional = sign ε;	sign_optional → sign sign_optional → ε	digit_0 = '0';	digit_0 = '0';
			digit_1 = '1';	digit_1 = '1';
			digit_2 = '2';	digit_2 = '2';
			digit_3 = '3';	digit_3 = '3';
			digit_4 = '4';	digit_4 = '4';
			digit_5 = '5';	digit_5 = '5';
			digit_6 = '6';	digit_6 = '6';
			digit_7 = '7';	digit_7 = '7';
			digit_8 = '8';	digit_8 = '8';
			digit_9 = '9';	digit_9 = '9';
			tokenCOLON = ":" >> BOUNDARIES;	tokenCOLON = ":" >> BOUNDARIES;
			tokenINTEGER2 = "Integer_2" >> STRICT_BOUNDARIES;	tokenINTEGER16 = "INTEGER16" >> STRICT_BOUNDARIES;
			tokenCOMMA = "," >> BOUNDARIES;	tokenCOMMA = "," >> BOUNDARIES;
			tokenNOT = "!" >> BOUNDARIES;	tokenNOT = "NOT" >> STRICT_BOUNDARIES;
			tokenAND = "&" >> BOUNDARIES;	tokenAND = "AND" >> STRICT_BOUNDARIES;
			tokenOR = " " >> BOUNDARIES;	tokenOR = "OR" >> STRICT_BOUNDARIES;
			tokenEQUAL = "==" >> BOUNDARIES;	tokenEQUAL = "==" >> BOUNDARIES;
			tokenNOTEQUAL = "!=" >> BOUNDARIES;	tokenNOTEQUAL = "!=" >> BOUNDARIES;
			tokenLESS = "<" >> BOUNDARIES;	tokenLESS = "<" >> BOUNDARIES;
			tokenGREATER = ">" >> BOUNDARIES;	tokenGREATER = ">" >> BOUNDARIES;
			tokenPLUS = "++" >> BOUNDARIES;	tokenPLUS = "+" >> BOUNDARIES;

			tokenMINUS = "--" >> BOUNDARIES;	tokenMINUS = "-" >> BOUNDARIES;
			tokenMUL = "***" >> BOUNDARIES;	tokenMUL = "*" >> BOUNDARIES;
			tokenDIV = "Div" >> STRICT_BOUNDARIES;	tokenDIV = "DIV" >> STRICT_BOUNDARIES;
			tokenMOD = "Mod" >> STRICT_BOUNDARIES;	tokenMOD = "MOD" >> STRICT_BOUNDARIES;
			tokenGROUPEXPRESSIONBEGIN = "{" >> BOUNDARIES;	tokenGROUPEXPRESSIONBEGIN = "{" >> BOUNDARIES;
			tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES;	tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES;
			tokenBIND = "->" >> BOUNDARIES;	tokenLRBIND = "=:." >> BOUNDARIES;
			tokenELSE = "Else" >> STRICT_BOUNDARIES;	tokenELSE = "ELSE" >> STRICT_BOUNDARIES;
			tokenIF = "If" >> STRICT_BOUNDARIES;	tokenIF = "IF" >> STRICT_BOUNDARIES;
			tokenDO = "Do" >> STRICT_BOUNDARIES;	tokenDO = "DO" >> STRICT_BOUNDARIES;
			tokenFOR = "For" >> STRICT_BOUNDARIES;	tokenFOR = "FOR" >> STRICT_BOUNDARIES;
			tokenDOWNT0 = "Downto" >> STRICT_BOUNDARIES;	tokenDOWNT0 = "DOWNT0" >> STRICT_BOUNDARIES;
			tokenREAD = "Read" >> STRICT_BOUNDARIES;	tokenGET = "GET" >> STRICT_BOUNDARIES;
			tokenWRITE = "Write" >> STRICT_BOUNDARIES;	tokenPUT = "PUT" >> STRICT_BOUNDARIES;
			tokenPROGRAM = "#Program" >> STRICT_BOUNDARIES;	tokenNAME = "NAME" >> STRICT_BOUNDARIES;
			tokenVARIABLE = "Variable" >> STRICT_BOUNDARIES;	tokenDATA = "DATA" >> STRICT_BOUNDARIES;
			tokenSTART = "Start" >> STRICT_BOUNDARIES;	tokenBEGIN = "BEGIN" >> STRICT_BOUNDARIES;
			tokenSTOP = "Stop" >> STRICT_BOUNDARIES;	tokenEND = "END" >> STRICT_BOUNDARIES;
			tokenBEGINBLOCK = "{" >> BOUNDARIES;	tokenBEGINBLOCK = "{" >> BOUNDARIES;
			tokenENDBLOCK = "}" >> BOUNDARIES;	tokenENDBLOCK = "}" >> BOUNDARIES;
			tokenLEFTSQUAREBRACKETS = "[" >> BOUNDARIES;	tokenLEFTSQUAREBRACKETS = "[" >> BOUNDARIES;

			tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES;	tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES;
			tokenSEMICOLON = ";" >> BOUNDARIES;	tokenSEMICOLON = ";" >> BOUNDARIES;
			STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_("_ ")));	STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_("_ ")));
			BOUNDARIES = (BOUNDARY >> *(BOUNDARY) NO_BOUNDARY);	BOUNDARIES = (BOUNDARY >> *(BOUNDARY) NO_BOUNDARY);
			BOUNDARY = BOUNDARY_SPACE BOUNDARY_TAB BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_NULL;	BOUNDARY = BOUNDARY_SPACE BOUNDARY_TAB BOUNDARY_CARRIAGE_R ETURN BOUNDARY_LINE_FEED BOUNDARY_NULL;
			BOUNDARY_SPACE = " ";	BOUNDARY_SPACE = " ";
			BOUNDARY_TAB = "\t";	BOUNDARY_TAB = "\t";
			BOUNDARY_CARRIAGE_RETURN = "\r";	BOUNDARY_CARRIAGE_R ETURN = "\r";
			BOUNDARY_LINE_FEED = "\n";	BOUNDARY_LINE_FEED = "\n";
			BOUNDARY_NULL = "\0";	BOUNDARY_NULL = "\0";
			NO_BOUNDARY = "";	NO_BOUNDARY = "";
			tokenUNDERSCORE = "_";	tokenUNDERSCORE = "_";
			A = "A";	A = "A";
			B = "B";	B = "B";
			C = "C";	C = "C";
			D = "D";	D = "D";
			E = "E";	E = "E";

			F = "F";	F = "F";
			G = "G";	G = "G";
			H = "H";	H = "H";
			I = "I";	I = "I";
			J = "J";	J = "J";
			K = "K";	K = "K";
			L = "L";	L = "L";
			M = "M";	M = "M";
			N = "N";	N = "N";
			O = "O";	O = "O";
			P = "P";	P = "P";
			Q = "Q";	Q = "Q";
			R = "R";	R = "R";
			S = "S";	S = "S";
			T = "T";	T = "T";
			U = "U";	U = "U";
			V = "V";	V = "V";
			W = "W";	W = "W";
			X = "X";	X = "X";
			Y = "Y";	Y = "Y";
			Z = "Z";	Z = "Z";

```
namespace qi = boost::spirit::qi;
namespace phx = boost::phoenix;

#define SAME_RULE(RULE) ((RULE) | (RULE))
template <typename Iterator>
struct cwgrammar : qi::grammar<Iterator> {
    cwgrammar(std::ostream& error_stream) : cwgrammar::base_type(tokens_in_program), error_stream_(error_stream) {
        keyword =
            tokenCOLON |
            tokenGOTO |
            tokenINTEGER16 |
            tokenCOMMA |
            tokenNOT |
            tokenAND |
            tokenOR |
            tokenEQUAL |
            tokenNOTEQUAL |
            tokenLESSOREQUAL |
            tokenGREATEROREQUAL |
            tokenLESS |
            tokenGREATER |
            tokenPLUS |
            tokenMINUS |
            tokenMUL |
            tokenDIV |
            tokenMOD |
            tokenGROUPEXPRESSIONBEGIN |
            tokenGROUPEXPRESSIONEND |
            tokenLRBIND |
            tokenELSE |
            tokenIF |
            tokenDO |
            tokenFOR |
            tokenTO |
            tokenDOWNT0 |
```

```

tokenWHILE |
tokenCONTINUE |
tokenBREAK |
tokenEXIT |
tokenREPEAT |
tokenUNTIL |
tokenGET |
tokenPUT |
tokenNAME |
tokenBODY |
tokenDATA |
tokenBEGIN |
tokenEND |
tokenBEGINBLOCK |
tokenENDBLOCK |
tokenLEFTSQUAREBRACKETS |
tokenRIGHTSQUAREBRACKETS |
tokenSEMICOLON;
tokens_in_program = SAME_RULE(token_iteration);
token = keyword | ident | value;
token_iteration = token >> token_iteration | "";
digit = digit_0 | non_zero_digit;
digit_optional = digit | "";
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_optional | digit_0) >> BOUNDARIES;
value = sign_optional >> unsigned_value >> BOUNDARIES;
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;
ident = tokenUNDERSCORE >> 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 >> STRICT_BOUNDARIES;
sign = sign_plus | sign_minus;
sign_optional = sign | "";
sign_plus = SAME_RULE(tokenPLUS);
sign_minus = SAME_RULE(tokenMINUS);
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;
tokenGOTO = "GOTO" >> STRICT_BOUNDARIES;
tokenINTEGER16 = "INTEGER16" >> STRICT_BOUNDARIES;
tokenCOMMA = "," >> BOUNDARIES;
tokenNOT = "NOT" >> STRICT_BOUNDARIES;
tokenAND = "AND" >> STRICT_BOUNDARIES;
tokenOR = "OR" >> STRICT_BOUNDARIES;
tokenEQUAL = "==" >> BOUNDARIES;
tokenNOTEQUAL = "!=" >> BOUNDARIES;
tokenLESSOREQUAL = "<=" >> BOUNDARIES;
tokenGREATEROREQUAL = ">=" >> BOUNDARIES;
tokenLESS = "<" >> BOUNDARIES;
tokenGREATER = ">" >> 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;
tokenDO = "DO" >> STRICT_BOUNDARIES;
tokenFOR = "FOR" >> STRICT_BOUNDARIES;
tokenTO = "TO" >> STRICT_BOUNDARIES;
tokenDOWNT0 = "DOWNT0" >> STRICT_BOUNDARIES;
tokenWHILE = "WHILE" >> STRICT_BOUNDARIES;
tokenCONTINUE = "CONTINUE" >> STRICT_BOUNDARIES;
tokenBREAK = "BREAK" >> STRICT_BOUNDARIES;
tokenEXIT = "EXIT" >> STRICT_BOUNDARIES;
tokenREPEAT = "REPEAT" >> STRICT_BOUNDARIES;
tokenUNTIL = "UNTIL" >> STRICT_BOUNDARIES;
tokenGET = "GET" >> STRICT_BOUNDARIES;
tokenPUT = "PUT" >> STRICT_BOUNDARIES;
tokenNAME = "NAME" >> STRICT_BOUNDARIES;
tokenBODY = "BODY" >> STRICT_BOUNDARIES;
tokenDATA = "DATA" >> STRICT_BOUNDARIES;
```

```
tokenBEGIN = "BEGIN" >> 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 = "";
tokenUNDERSCORE = "_";
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";  
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";
```

```
}  
std::ostringstream& error_stream_;
```

```
qi::rule<Iterator>  
tokens_in_program,  
token_iteration,  
token,  
keyword,  
ident,  
letter_in_lower_case,  
letter_in_upper_case,  
value,
```

```

    sign_optional,
    sign,
    sign_plus,
    sign_minus,
    unsigned_value,
    digit,
    digit_optional,
    non_zero_digit,
    //
    tokenCOLON, tokenGOTO, tokenINTEGER16, tokenCOMMA, tokenNOT, tokenAND, tokenOR, tokenEQUAL, tokenNOTEQUAL,
    tokenLESSOREQUAL,
    tokenGREATEROREQUAL,
    tokenLESS,
    tokenGREATER,
    tokenPLUS, tokenMINUS, tokenMUL, tokenDIV, tokenMOD, tokenGROUPEXPRESSIONBEGIN, tokenGROUPEXPRESSIONEND, tokenLRBIND,
    tokenELSE, tokenIF, tokenDO, tokenFOR, tokenTO, tokenDOWNT0, tokenWHILE, tokenCONTINUE, tokenBREAK, tokenEXIT, tokenREPEAT,
tokenUNTIL, tokenGET, tokenPUT, tokenNAME, tokenBODY, tokenDATA, tokenBEGIN, tokenEND, tokenBEGINBLOCK, tokenENDBLOCK,
tokenLEFTSQUAREBRACKETS, tokenRIGHTSQUAREBRACKETS, tokenSEMICOLON,
    //
    STRICT_BOUNDARIES, BOUNDARIES, BOUNDARY, BOUNDARY_SPACE, BOUNDARY_TAB, BOUNDARY_CARRIAGE_RETURN, BOUNDARY_LINE_FEED, BOUNDARY_NULL,
    NO_BOUNDARY,
    //
    digit_0, digit_1, digit_2, digit_3, digit_4, digit_5, digit_6, digit_7, digit_8, digit_9,
    //
    tokenUNDERSCORE,
    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, 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;
};

```