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 → 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 = { "LONG" , "INT" , "," , "NOT" , "AND" , "OR" , "EQ" , "NE" , "<" , ">" , "ADD" , "SUB" , "MUL" , "DIV" , "MOD" , "(" , ")" , "->" , "ELSE" , "IF" , "FOR" , "DOWNTO" , "DO" , "SCAN" , "PRINT" , "PROGRAM" , "VAR" , "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" }		
keyword = "LONG" | "INT" | "NOT" | "AND" | "OR" | "EQ" | "NE" | "<" | ">" | "ADD" | "SUB" | "MUL" | "DIV" | "MOD" | "(" | ")" | "->" | "ELSE" | "IF" | "FOR" |	keyword = "LONG" | "INT" | "NOT" | "AND" | "OR" | "EQ" | "NE" | "<" | ">" | "ADD" | "SUB"	Keyword = "LONG" , Keyword = "INT" , Keyword = "," , Keyword = "NOT" , Keyword = "AND" , Keyword = "OR" , Keyword = "EQ" , Keyword = "NE" , Keyword = "<" , Keyword = ">" , Keyword = "ADD" , Keyword = "SUB" , Keyword = "MUL" , Keyword = "DIV" , Keyword = "MOD" , Keyword = "(" , Keyword = ")" , Keyword = "->" , Keyword =	keyword = tokenINTEGER16 | tokenCOMMA | tokenNOT | tokenAND | tokenOR | tokenEQUAL |	keyword = tokenINTEGER16 | tokenCOMMA | tokenNOT | tokenAND | tokenOR | tokenEQUAL | tokenNOTEQUAL |

"DOWNTO" | "DO" | "SCAN" | "PRINT" | "PROGRAM" | "VAR" | "BEGIN" | "END" | "{" | "}" | "[" | "]" | ";" ;	| "MUL" | "DIV" | "MOD" | "(" | ")" | "->" | "ELSE" | "IF" | "FOR" | "DOWNTO" | "DO" | "SCAN" | "PRINT" | "PROGRAM" | "VAR" | "BEGIN" | "END" | "{" | "}" | "[" | "]" | ";" ;	"ELSE" , Keyword = "IF" , Keyword = "FOR" , Keyword = "DOWNTO" , Keyword = "DO" , Keyword = "SCAN" , Keyword = "PRINT" , Keyword = "PROGRAM" , Keyword = "VAR" , Keyword = "BEGIN" , Keyword = "END" , Keyword = "{" , Keyword = "}" , Keyword = "[" , Keyword = "]" , Keyword = ";"	tokenNOTEQUAL | tokenLESSOREQUAL | tokenGREATEROREQUAL | tokenPLUS | tokenMUL | tokenDIV | tokenMOD | tokenGROUPEXPRESSIONBEGIN | tokenGROUPEXPRESSIONEND | tokenLRBIND | tokenMINUS | tokenELSE | tokenIF | tokenWHILE | tokenCONTINUE | tokenBREAK | tokenGET | tokenPUT | tokenNAME | tokenDATA | tokenBEGIN | tokenEND | tokenBEGINBLOCK | tokenENDBLOCK | tokenLEFTSQUAREBRACKETS | tokenRIGHTSQUAREBRACKETS | tokenSEMICOLON;	tokenLESS | tokenGREATER | tokenPLUS | tokenMUL | tokenDIV | tokenMOD | tokenGROUPEXPRESSIONBEGIN | tokenGROUPEXPRESSIONEND | tokenLRBIND | tokenMINUS | tokenELSE | tokenIF | tokenFOR | tokenDOWNTO | tokenDO | tokenSCAN | tokenPRINT | tokenNAME | tokenDATA | tokenBEGIN | tokenEND | tokenBEGINBLOCK | tokenENDBLOCK | tokenLEFTSQUAREBRACKETS | tokenRIGHTSQUAREBRACKETS | tokenSEMICOLON;
tokens_in_program = { keyword | ident | value} ;	tokens_in_program = token_iteration ;	tokens_in_program → token_iteration	tokens_in_program = BOUNDARIES >> *(keyword | ident | value);	tokens_in_program = 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_optional → digit;		digit_optional = digit |

	digit | ε;	digit_optional → ε;		"";
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" | 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"; "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_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	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;

		→ "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"		
ident = "_" , letter_in_upper_case , letter_in_upper_case;	ident = "_" , letter_in_upper _case , letter_in_upper _case; pper_case;	ident → "_" letter_in_upper_case letter_in_upper_case	ident = tokenUNDERSCORE >> letter_in_upper_case >> letter_in_upper_case >> STRICT_BOUNDARIES;	ident = tokenUNDERSCORE >> letter_in_upper_case >> letter_in_upper_case >> STRICT_BOUNDARIES;
sign = "ADD" | "SUB";	sign = "+" | "-";	sign → "ADD" sign → "SUB"	sign = sign_plus | sign_minus;	sign = sign_plus | sign_minus;
	sign_optional = sign | ε;	sign_optional → sign sign_optional → ε		sign_optional = sign | "";
			sign_plus = SAME_RULE(tokenPLUS);	sign_plus = SAME_RULE(tokenPLUS);
			sign_minus = SAME_RULE(tokenMINUS);	sign_minus = SAME_RULE(tokenMINUS);
			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';
			tokenINTEGER16 = "LONG" >> BOUNDARIES >> "INT" >> STRICT_BOUNDARIES;	tokenINTEGER16 = "LONG" >> “INT” STRICT_BOUNDARIES;

			tokenCOMMA = "," >> BOUNDARIES;	tokenCOMMA = "," >> BOUNDARIES;
			tokenNOT = "NOT" >> STRICT_BOUNDARIES;	tokenNOT = "NOT" >> STRICT_BOUNDARIES;
			tokenAND = "AND" >> STRICT_BOUNDARIES;	tokenAND = "AND" >> STRICT_BOUNDARIES;
			tokenOR = "OR" >> STRICT_BOUNDARIES;	tokenOR = "OR" >> STRICT_BOUNDARIES;
			tokenEQUAL = "EQ" >> BOUNDARIES;	tokenEQUAL = "EQ" >> BOUNDARIES;
			tokenNOTEQUAL = "NE" >> BOUNDARIES;	tokenNOTEQUAL = "NE" >> BOUNDARIES;
			TokenLESSOREQUAL = "<" >> BOUNDARIES;	TokenLESSOREQUAL = "<" >> BOUNDARIES;
			TokenGREATEROREQUAL = ">" >> BOUNDARIES;	TokenGREATEROREQUAL = ">" >> BOUNDARIES;
			tokenPLUS = "ADD" >> BOUNDARIES;	tokenPLUS = "ADD" >> BOUNDARIES;
			tokenMINUS = "SUB" >> BOUNDARIES;	tokenMINUS = "SUB" >> BOUNDARIES;
			tokenMUL = "MUL" >> BOUNDARIES;	tokenMUL = "MUL" >> 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;

			tokenLRBIND = "->" >> BOUNDARIES;	tokenLRBIND = "->" >> BOUNDARIES;
			tokenELSE = "ELSE" >> STRICT_BOUNDARIES;	tokenELSE = "ELSE" >> STRICT_BOUNDARIES;
			tokenIF = "IF" >> STRICT_BOUNDARIES;	tokenIF = "IF" >> STRICT_BOUNDARIES;
			tokenFOR = "FOR" >> STRICT_BOUNDARIES;	tokenFOR = "FOR" >> STRICT_BOUNDARIES;
			tokenDOWNTO = "DOWNTO" >> STRICT_BOUNDARIES;	tokenDOWNTO = "DOWNTO" >> STRICT_BOUNDARIES;
			tokenDO = "DO" >> STRICT_BOUNDARIES;	tokenSCAN = "SCAN" >> STRICT_BOUNDARIES;
			tokenSCAN = "SCAN" >> STRICT_BOUNDARIES;	tokenGET = "Input" >> STRICT_BOUNDARIES;
			tokenPRINT = "PRINT" >> STRICT_BOUNDARIES;	tokenPRINT = "PRINT" >> STRICT_BOUNDARIES;
			tokenNAME = "PROGRAM" >> STRICT_BOUNDARIES;	tokenNAME = "PROGRAM" >> STRICT_BOUNDARIES;
			tokenDATA = "VAR" >> STRICT_BOUNDARIES;	tokenDATA = "VAR" >> STRICT_BOUNDARIES;
			tokenBEGIN = "BEGIN" >> STRICT_BOUNDARIES;	tokenBEGIN = "BEGIN" >> STRICT_BOUNDARIES;
			tokenEND = "END" >> 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_RETURN | BOUNDARY_LINE_FEED | BOUNDARY_NULL;
			BOUNDARY_SPACE = " ";	BOUNDARY_SPACE = " ";
			BOUNDARY_TAB = "\t";	BOUNDARY_TAB = "\t";
			BOUNDARY_CARRIAGE_RETURN = "\r";	BOUNDARY_CARRIAGE_RETURN = "\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";
			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::ostringstream& error_stream) : cwgrammar::base_type(tokens_in_program), error_stream_(error_stream) {
        keyword =
            tokenINTEGER16 |
            tokenCOMMA |
            tokenNOT |
            tokenAND |
            tokenOR |
```

```
            tokenEQUAL |
            tokenNOTEQUAL |
            tokenLESS |
            tokenGREATER |
            tokenPLUS |
            tokenMUL |
            tokenDIV |
            tokenMOD |
            tokenGROUPEXPRESSIONBEGIN |
            tokenGROUPEXPRESSIONEND |
            tokenLRBIND |
            tokenMINUS |
            tokenELSE |
            tokenIF |
            tokenFOR |
            tokenDOWNTO |
            tokenDO |
            tokenSCAN |
            tokenPRINT |
            tokenNAME |
            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;


        // VARIANT 2 IDENTIFIER: _[A-Z][A-Z]
        ident = tokenUNDERSCORE >> 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';


        // UPDATED KEYWORDS FOR VARIANT 2
        tokenINTEGER16 = "LONG" >> BOUNDARIES >> "INT" >> STRICT_BOUNDARIES;
        tokenCOMMA = "," >> BOUNDARIES;
        tokenNOT = "NOT" >> STRICT_BOUNDARIES;
        tokenAND = "AND" >> STRICT_BOUNDARIES;
        tokenOR = "OR" >> STRICT_BOUNDARIES;
        tokenEQUAL = "EQ" >> BOUNDARIES;
        tokenNOTEQUAL = "NE" >> BOUNDARIES;
        tokenLESS = "<" >> BOUNDARIES;
        tokenGREATER = ">" >> BOUNDARIES;
```

```
        tokenPLUS = "ADD" >> BOUNDARIES;
        tokenMINUS = "SUB" >> BOUNDARIES;
        tokenMUL = "MUL" >> BOUNDARIES;
        tokenDIV = "DIV" >> STRICT_BOUNDARIES;
        tokenMOD = "MOD" >> STRICT_BOUNDARIES;
        tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES;
        tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES;
        tokenLRBIND = "->" >> BOUNDARIES;
        tokenELSE = "ELSE" >> STRICT_BOUNDARIES;
        tokenIF = "IF" >> STRICT_BOUNDARIES;
        tokenFOR = "FOR" >> STRICT_BOUNDARIES;
        tokenDOWNTO = "DOWNTO" >> STRICT_BOUNDARIES;
        tokenDO = "DO" >> STRICT_BOUNDARIES;
        tokenSCAN = "SCAN" >> STRICT_BOUNDARIES;
        tokenPRINT = "PRINT" >> STRICT_BOUNDARIES;
        tokenNAME = "PROGRAM" >> STRICT_BOUNDARIES;
        tokenDATA = "VAR" >> 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,
        unsigned_value,
        value,
        sign_optional,
        sign,
        sign_plus,
        sign_minus,
        digit,
        digit_optional,
        non_zero_digit,
        //
        tokenCOLON, tokenGOTO, tokenINTEGER16, tokenCOMMA, tokenNOT, tokenAND, tokenOR, tokenEQUAL, tokenNOTEQUAL,
        tokenLESS,
        tokenGREATER,
        tokenPLUS, tokenMINUS, tokenMUL, tokenDIV, tokenMOD, tokenGROUPEXPRESSIONBEGIN, tokenGROUPEXPRESSIONEND, tokenLRBIND,
        tokenELSE, tokenIF, tokenDO, tokenFOR, tokenTO, tokenDOWNTO, tokenWHILE, tokenCONTINUE, tokenBREAK, tokenEXIT,
tokenREPEAT, tokenUNTIL, tokenSCAN, tokenPRINT, 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;

};
```