

РБНФ	Код для перевірки РБНФ
labeled point = ident , ":";	labeled point = ident >> tokenCOLON;
program_name = ident;	program_name = SAME_RULE(ident);
value_type = "INTEGER16";	value_type = SAME_RULE(tokenINTEGER);
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 = "&&" " " "eq" "ne" "lt" "gt" "+" "-" "*" "/" "%";	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 cond block with optionally return 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_valu e =	body_for_true__with_optionally_return_value = SAME_RULE(block statements with optionally return value);

block_statements__with_optionally_return_value;	
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__with_optionally_return_value = tokenELSE >> tokenIF >> if_expression >> body_for_true__with_optionally_return_value;
body_for_false__with_optionally_return_value = "else" , block_statements__with_optionally_return_value;	body_for_false__with_optionally_return_value = tokenELSE >> block_statements__with_optionally_return_value;
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__with_optionally_return_value = tokenIF >> 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__with_optionally_return_value_and_optionally_bind = cond_block__with_optionally_return_value , [tokenLRBIND , ident , [index_action]];	cond_block__with_optionally_return_value_and_optionally_bind = cond_block__with_optionally_return_value >> -(tokenLRBIND >> ident >> -index_action);
statement_in_while_and_if_body = statement "CONTINUE" "BREAK";	statement_in_while_and_if_body = statement continue_while break_while;
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 = "input" , (ident , [index_action] "(" , ident , [index_action] , ")");	input = tokenGET >> (ident >> -index_action tokenGROUPEXPRESSIONBEGIN >> ident >> -index_action >> tokenGROUPEXPRESSIONEND);
output = "output" , expression;	output = tokenPUT >> expression;

statement = bind_left_to_right cond_block_with_optionally_return_value_and_optionally_bind repeat_until_cycle input output ";" ;	statement = bind_left_to_right cond_block_with_optionally_return_value_and_optionally_bind repeat_until_cycle input output tokenSEMICOLON ;
block_statements = "{" , {statement} , "}" ;	block_statements = tokenBEGINBLOCK >> *statement >> tokenENDBLOCK ;
block_statements__with_optionally_return_value = "{" , {statement_in_while_and_if_body} , [expression] , "}" ;	block_statements__with_optionally_return_value = tokenBEGINBLOCK >> *statement_in_while_and_if_body >> -expression >> tokenENDBLOCK ;
program = "NAME" , "DATA" , [declaration] , ";" , "begin" , {statement} , "end" ;	program = BOUNDARIES >> tokenNAME >> tokenDATA >> (-declaration) >> tokenSEMICOLON >> tokenBEGIN >> *statement >> tokenEND ;
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_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 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_lower_case , letter_in_lower_case , letter_in_lower_case , digit , digit ;	ident = letter_in_lower_case >> letter_in_lower_case >> letter_in_lower_case >> digit >> digit >> STRICT BOUNDARIES ;
sign = "+" "-" ;	sign = sign_plus sign_minus ;
	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;
	tokenINTEGER = "int" >> STRICT BOUNDARIES;
	tokenCOMMA = "," >> BOUNDARIES;
	tokenNOT = "!!" >> STRICT BOUNDARIES;
	tokenAND = "&&" >> STRICT BOUNDARIES;
	tokenOR = " " >> STRICT BOUNDARIES;
	tokenEQUAL = "eq" >> BOUNDARIES;
	tokenNOTEQUAL = "ne" >> BOUNDARIES;
	tokenLESS = "lt" >> BOUNDARIES;
	tokenGREATER = "gt" >> BOUNDARIES;
	tokenPLUS = "+" >> BOUNDARIES;
	tokenMINUS = "-" >> BOUNDARIES;
	tokenMUL = "*" >> BOUNDARIES;
	tokenDIV = "/" >> STRICT BOUNDARIES;
	tokenMOD = "%" >> STRICT BOUNDARIES;
	tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES;
	tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES;
	tokenLRBIND = "=: " >> BOUNDARIES;
	tokenELSE = "else" >> STRICT BOUNDARIES;
	tokenIF = "if" >> STRICT BOUNDARIES;
	tokenREPEAT = "REPEAT" >> STRICT BOUNDARIES;
	tokenUNTIL = "until" >> STRICT BOUNDARIES;

	tokenGET = "input" >> STRICT_BOUNDARIES;
	tokenPUT = "output" >> 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 = "";
	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";