labeled_point = ident >> tokenCOLON;	labeled_point = ident >> tokenCOLON;
goto_label = tokenGOTO >> ident;	goto_label = tokenGOTO >> ident;
program_name = SAME_RULE(ident);	program_name = SAME_RULE(ident);
value_type = SAME_RULE(tokenINTEGER16);	value_type = SAME_RULE(tokenINTEGER16);
declaration_element = ident >> -(tokenLEFTSQUAREBRACKETS >> unsigned_value >> tokenRIGHTSQUAREBRACKETS);	declaration_element = ident >> -(tokenLEFTSQUAREBRACKETS >> unsigned_value >> tokenRIGHTSQUAREBRACKETS);
other_declaration_ident = tokenCOMMA >> 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 = tokenLEFTSQUAREBRACKETS >> expression >> tokenRIGHTSQUAREBRACKETS;	index_action = tokenLEFTSQUAREBRACKETS >> expression >> tokenRIGHTSQUAREBRACKETS;
unary_operator = tokenNOT | tokenMINUS | tokenPLUS;	unary_operator = tokenNOT | tokenMINUS | tokenPLUS;
unary_operation = unary_operator >> expression;	unary_operation = unary_operator >> expression;
binary_operator = tokenAND | tokenOR | tokenEQUAL | tokenNOTEQUAL | tokenLESSOREQUAL | tokenGREATEROREQUAL | tokenPLUS | tokenMINUS | tokenMUL | tokenDIV | tokenMOD;	binary_operator = tokenAND | tokenOR | tokenEQUAL | tokenNOTEQUAL | tokenLESSOREQUAL | tokenGREATEROREQUAL | 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;	left_expression = group_expression | unary_operation | ident >> -index_action | value;
expression = left_expression >> *binary_action;	expression = left_expression >> *binary_action;
group_expression = tokenGROUPEXPRESSIONBEGIN >> expression >> tokenGROUPEXPRESSIONEND;	group_expression = tokenGROUPEXPRESSIONBEGIN >> expression >> tokenGROUPEXPRESSIONEND;
bind_right_to_left = ident >> -index_action >> tokenRLBIND >> expression;	bind_right_to_left = ident >> -index_action >> tokenRLBIND >> expression;
bind_left_to_right = expression >> tokenLRBIND >> ident >> -index_action;	bind_left_to_right = expression >> tokenLRBIND >> ident >> -index_action;
if_expression = SAME_RULE(expression);	if_expression = SAME_RULE(expression);
body_for_true = SAME_RULE(block_statements_in_while_and_if_body);	body_for_true = SAME_RULE(block_statements_in_while_and_if_body);
false_cond_block = tokenELSE >> cond_block;	false_cond_block = tokenELSE >> cond_block;
body_for_false = tokenELSE >> block_statements_in_while_and_if_body;	body_for_false = tokenELSE >> block_statements_in_while_and_if_body;

cond_block = tokenIF >> if_expression >> body_for_true >> *false_cond_block >> (-body_for_false);	cond_block = tokenIF >> if_expression >> body_for_true >> *false_cond_block >> (-body_for_false);
cycle_begin_expression = SAME_RULE(expression);	cycle_begin_expression = SAME_RULE(expression);
cycle_end_expression = SAME_RULE(expression);	cycle_end_expression = SAME_RULE(expression);
cycle_counter = SAME_RULE(ident);	cycle_counter = SAME_RULE(ident);
cycle_counter_rl_init = cycle_counter >> tokenRLBIND >> cycle_begin_expression;	cycle_counter_rl_init = cycle_counter >> tokenRLBIND >> cycle_begin_expression;
cycle_counter_lr_init = cycle_begin_expression >> tokenLRBIND >> cycle_counter;	cycle_counter_lr_init = cycle_begin_expression >> tokenLRBIND >> cycle_counter;
cycle_counter_init = cycle_counter_rl_init | cycle_counter_lr_init;	cycle_counter_init = cycle_counter_rl_init | cycle_counter_lr_init;
cycle_counter_last_value = SAME_RULE(cycle_end_expression);	cycle_counter_last_value = SAME_RULE(cycle_end_expression);
cycle_body = tokenDO >> (statement | block_statements);	cycle_body = tokenDO >> (statement | block_statements);
forto_cycle = tokenFOR >> cycle_counter_init >> tokenTO >> cycle_counter_last_value >> cycle_body;	forto_cycle = tokenFOR >> cycle_counter_init >> tokenTO >> cycle_counter_last_value >> cycle_body;
continue_while = SAME_RULE(tokenCONTINUE);	continue_while = SAME_RULE(tokenCONTINUE);
break_while = SAME_RULE(tokenBREAK);	break_while = SAME_RULE(tokenBREAK);
statement_in_while_and_if_body = statement | continue_while | break_while;	statement_in_while_and_if_body = statement | continue_while | break_while;
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 >> tokenENDBLOCK;
while_cycle_head_expression = SAME_RULE(expression);	while_cycle_head_expression = SAME_RULE(expression);
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;
repeat_until_cycle_cond = SAME_RULE(expression);	repeat_until_cycle_cond = SAME_RULE(expression);
repeat_until_cycle = tokenREPEAT >> (statement | block_statements) >> tokenUNTIL >> repeat_until_cycle_cond;	repeat_until_cycle = tokenREPEAT >> (statement | block_statements) >> tokenUNTIL >> repeat_until_cycle_cond;
input = tokenGET >> (ident >> -index_action | tokenGROUPEXPRESSIONBEGIN >> ident >> -index_action >> tokenGROUPEXPRESSIONEND);	input = tokenGET >> (ident >> -index_action | tokenGROUPEXPRESSIONBEGIN >> ident >> -index_action >> tokenGROUPEXPRESSIONEND);
output = tokenPUT >> expression;	output = tokenPUT >> expression;
statement = bind_right_to_left | bind_left_to_right | cond_block | forto_cycle | while_cycle | repeat_until_cycle | labeled_point | goto_label | input | output | tokenSEMICOLON;	statement = bind_right_to_left | bind_left_to_right | cond_block | forto_cycle | while_cycle | repeat_until_cycle | labeled_point | goto_label | input | output | tokenSEMICOLON;

block_statements = tokenBEGINBLOCK >> *statement >> tokenENDBLOCK;	block_statements = tokenBEGINBLOCK >> *statement >> tokenENDBLOCK;
program = tokenNAME >> program_name >> tokenSEMICOLON >> tokenBODY >> tokenDATA >> (-declaration) >> tokenSEMICOLON >> *statement >> tokenEND;	program = tokenNAME >> program_name >> tokenSEMICOLON >> tokenBODY >> tokenDATA >> (-declaration) >> tokenSEMICOLON >> *statement >> tokenEND;
digit = digit_0 | digit_1 | digit_2 | digit_3 | digit_4 | digit_5 | digit_6 | digit_7 | digit_8 | digit_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 = 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) | digit_0) >> BOUNDARIES;	unsigned_value = ((non_zero_digit >> *digit) | digit_0) >> BOUNDARIES;
value = (-sign) >> unsigned_value >> BOUNDARIES;	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 = 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 >> letter_in_upper_case >> STRICT_BOUNDARIES;	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 >> letter_in_upper_case >> STRICT_BOUNDARIES;
sign = sign_plus | sign_minus;	sign = sign_plus | sign_minus;
sign_plus = '-' >> BOUNDARIES;	sign_plus = '-' >> BOUNDARIES;
sign_minus = '+' >> BOUNDARIES;	sign_minus = '+' >> BOUNDARIES;
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;
tokenGOTO = "GOTO" >> STRICT_BOUNDARIES;	tokenGOTO = "GOTO" >> STRICT_BOUNDARIES;
tokenINTEGER16 = "INTEGER16" >> STRICT_BOUNDARIES;	tokenINTEGER16 = "INTEGER16" >> 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 = "==" >> BOUNDARIES;	tokenEQUAL = "==" >> BOUNDARIES;
tokenNOTEQUAL = "!=" >> BOUNDARIES;	tokenNOTEQUAL = "!=" >> BOUNDARIES;
tokenLESSOREQUAL = "<=" >> BOUNDARIES;	tokenLESSOREQUAL = "<=" >> BOUNDARIES;
tokenGREATEROREQUAL = ">=" >> BOUNDARIES;	tokenGREATEROREQUAL = ">=" >> 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;
tokenRLBIND = ":=" >> BOUNDARIES;	tokenRLBIND = ":=" >> BOUNDARIES;
tokenLRBIND = "=:" >> 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;
tokenTO = "TO" >> STRICT_BOUNDARIES;	tokenTO = "TO" >> STRICT_BOUNDARIES;
tokenWHILE = "WHILE" >> STRICT_BOUNDARIES;	tokenWHILE = "WHILE" >> STRICT_BOUNDARIES;
tokenCONTINUE = "CONTINUE" >> STRICT_BOUNDARIES;	tokenCONTINUE = "CONTINUE" >> STRICT_BOUNDARIES;
tokenBREAK = "BREAK" >> STRICT_BOUNDARIES;	tokenBREAK = "BREAK" >> STRICT_BOUNDARIES;
tokenEXIT = "EXIT" >> STRICT_BOUNDARIES;	tokenEXIT = "EXIT" >> STRICT_BOUNDARIES;
tokenREPEAT = "REPEAT" >> STRICT_BOUNDARIES;	tokenREPEAT = "REPEAT" >> STRICT_BOUNDARIES;
tokenUNTIL = "UNTIL" >> STRICT_BOUNDARIES;	tokenUNTIL = "UNTIL" >> STRICT_BOUNDARIES;

tokenGET = "GET" >> STRICT_BOUNDARIES;	tokenGET = "GET" >> STRICT_BOUNDARIES;
tokenPUT = "PUT" >> STRICT_BOUNDARIES;	tokenPUT = "PUT" >> STRICT_BOUNDARIES;
tokenNAME = "NAME" >> STRICT_BOUNDARIES;	tokenNAME = "NAME" >> STRICT_BOUNDARIES;
tokenBODY = "BODY" >> STRICT_BOUNDARIES;	tokenBODY = "BODY" >> STRICT_BOUNDARIES;
tokenDATA = "DATA" >> STRICT_BOUNDARIES;	tokenDATA = "DATA" >> 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";