<u>РБНФ</u>	Код для перевірки РБНФ
program_name = ident;	program_name = SAME_RULE(ident);
value_type = "INTEGER16";	value_type = SAME_RULE(tokenINTEGER_2);
	declaration_element = ident >> -(tokenLEFTSQUAREBRACKETS >>
declaration_element = ident , ["[", unsigned_value , "]"];	unsigned_value >> tokenRIGHTSQUAREBRACKETS);
other_declaration_ident = "," , declaration_element;	other_declaration_ident = tokenCOMMA >> declaration_element;
	declaration = value_type >> declaration_element >>
declaration = value_type , declaration_element , {other_declaration_ident};	*other_declaration_ident;
	index_action = tokenLEFTSQUAREBRACKETS >> expression >>
index_action = "[" , expression , "]";	tokenRIGHTSQUAREBRACKETS;
unary_operator = "NOT";	unary_operator = SAME_RULE(tokenNOT);
unary_operation = unary_operator , expression;	unary_operation = unary_operator >> expression;
	binary_operator = tokenAND tokenOR tokenEQ tokenNOTEQUAL
binary_operator = "AND" "OR" "==" "!=" "<=" ">=" "<" ">" "+"	tokenLESS tokenGR tokenPLUS tokenMINUS tokenMUL tokenDIV
"-" "*" "DIV" "MOD";	tokenMOD;
binary_action = binary_operator , expression;	binary_action = binary_operator >> expression;
left_expression = group_expression unary_operation ident , [index_action]	left_expression = group_expression unary_operation ident >> -
value cond_blockwith_optionally_return_value;	index_action value cond_blockwith_optionally_return_value;
expression = left_expression , {binary_action};	expression = left_expression >> *binary_action;
	group_expression = tokenGROUPEXPRESSIONBEGIN >> expression >>
group_expression = "(", 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_truewith_optionally_return_value =	body_for_truewith_optionally_return_value =
block_statementswith_optionally_return_value;	SAME_RULE(block_statementswith_optionally_return_value);
false_cond_block_without_elsewith_optionally_return_value = "ELSE" , "IF"	false_cond_block_without_elsewith_optionally_return_value = tokenELSE
, if_expression , body_for_truewith_optionally_return_value;	>> tokenIF >> if_expression >> body_for_truewith_optionally_return_value;
body_for_falsewith_optionally_return_value = "ELSE",	
block_statementswith_optionally_return_value;	body_for_falsewith_optionally_return_value = tokenELSE >>
	block_statementswith_optionally_return_value;
cond_blockwith_optionally_return_value = "IF", if_expression,	cond_blockwith_optionally_return_value = tokenIF >> if_expression >>
body_for_truewith_optionally_return_value ,	body_for_truewith_optionally_return_value >>
{false_cond_block_without_elsewith_optionally_return_value},	*false_cond_block_without_elsewith_optionally_return_value >> -

[body_for_falsewith_optionally_return_value];	body_for_falsewith_optionally_return_value;
cond_blockwith_optionally_return_value_and_optionally_bind =	cond_blockwith_optionally_return_value_and_optionally_bind =
cond_blockwith_optionally_return_value , [tokenLRBIND , ident ,	cond_blockwith_optionally_return_value >> -(tokenLRBIND >> ident >> -
[index_action]];	index_action);
	continue_while = SAME_RULE(tokenCONTINUE);
	break_while = SAME_RULE(tokenBREAK);
	statement_in_while_and_if_body = statement continue_while
statement_in_while_and_if_body = statement "CONTINUE" "BREAK";	break_while;
block_statements_in_while_and_if_body = "{",	block_statements_in_while_and_if_body = tokenBEGINBLOCK >>
{statement_in_while_and_if_body}, "}";	*statement_in_while_and_if_body >> tokenENDBLOCK;
while_cycle_head_expression = expression;	while_cycle_head_expression = SAME_RULE(expression);
while_cycle = "WHILE" , while_cycle_head_expression ,	while_cycle = tokenWHILE >> while_cycle_head_expression >>
block_statements_in_while_and_if_body;	block_statements_in_while_and_if_body;
	input = tokenGET >> (ident >> -index_action tokenGROUPEXPRESSIONBEGIN
<pre>input = "GET" , (ident , [index_action] "(" , ident , [index_action] , ")");</pre>	>> ident >> -index_action >> tokenGROUPEXPRESSIONEND);
output = "PUT", expression;	output = tokenPUT >> expression;
statement = bind_left_to_right	
cond_blockwith_optionally_return_value_and_optionally_bind	statement = bind_left_to_right
forto_cycle while_cycle repeat_until_cycle labeled_point goto_label	cond_blockwith_optionally_return_value_and_optionally_bind
input output ";";	while_cycle input output tokenSEMICOLON;
block_statements = "{" , {statement} , "}";	block_statements = tokenBEGINBLOCK >> *statement >> tokenENDBLOCK;
block_statementswith_optionally_return_value = "{",	block_statementswith_optionally_return_value = tokenBEGINBLOCK >>
{statement_in_while_and_if_body}, [expression], "}";	*statement_in_while_and_if_body >> -expression >> tokenENDBLOCK;
<pre>program = "NAME", program_name, ";", "BODY", "DATA", [declaration], ";"</pre>	program = BOUNDARIES >> tokenNAME >> tokenDATA >> declaration >>
, {statement} , "END";	tokenSEMICOLON >> tokenBODY >> *statement >> tokenEND;
	digit = digit_0 digit_1 digit_2 digit_3 digit_4 digit_5 digit_6 digit_7
digit = "0" "1" "2" "3" "4" "5" "6" "7" "8" "9";	digit_8 digit_9;
	non_zero_digit = digit_1 digit_2 digit_3 digit_4 digit_5 digit_6
non_zero_digit = "1" "2" "3" "4" "5" "6" "7" "8" "9";	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"	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;

"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"	letter_in_upper_case = A B C D E F G H I J K L M N O
"Y" "Z";	P Q R S T U V W X Y Z;
ident = "_", letter_in_upper_case, letter_in_upper_case,	ident = tokenUNDERSCORE >> letter_in_lower_case >> non_zero_digit >>
letter_in_upper_case , letter_in_upper_case , letter_in_upper_case ,	BOUNDARIES;
letter_in_upper_case , letter_in_upper_case;	
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';
	tokenINTEGER_2 = "int_2" >> STRICT_BOUNDARIES;
	tokenCOMMA = "," >> BOUNDARIES;
	tokenNOT = "!" >> STRICT_BOUNDARIES;
	tokenAND = "and" >> STRICT_BOUNDARIES;
	tokenOR = "or" >> STRICT_BOUNDARIES;
	tokenEQ "eq" >> BOUNDARIES;
	tokenNOTEQUAL = "noteq" >> BOUNDARIES;
	tokenLESS = "less" >> BOUNDARIES;
	tokenGREATER = "gr" >> BOUNDARIES;
	tokenPLUS = "+" >> BOUNDARIES;
	tokenMINUS = "-" >> BOUNDARIES;

tokenMDL = """ >> BOUNDARIES; tokenMOD = "%" >> STRICT_BOUNDARIES; tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES; tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES; tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES; tokenLRBIND = ">> >> SUDINDARIES; tokenLRBIND = "S >> STRICT_BOUNDARIES; tokenELSE = "else" >> STRICT_BOUNDARIES; tokenWHILE = "while" >> STRICT_BOUNDARIES; tokenWHILE = "while" >> STRICT_BOUNDARIES; tokenGNTINUE = "continue" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenDEDOPT = "startprogram" >> STRICT_BOUNDARIES; tokenDEDOPT = "startprogram" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenENDEOPT = "startprogram" >> STRICT_BOUNDARIES; tokenENDEOCK = "" >> BOUNDARIES; tokenENDEOCK = "" >> BOUNDARIES; stokenENDEOCK = "" >> BOUNDARIES; tokenENDEOCK = "" >> BOUNDARIES; stokenENDEOCK = ""	
tokenMOD = "%" >> STRICT_BOUNDARIES; tokenGROUPEXPRESSIONEEGIn = "(" >> BOUNDARIES; tokenGROUPEXPRESSIONEEGIn = "(" >> BOUNDARIES; tokentEISIND = "->" >> BOUNDARIES; tokentEISIND = "->" >> BOUNDARIES; tokentEISE = "else" >> STRICT_BOUNDARIES; tokentEISE = "else" >> STRICT_BOUNDARIES; tokentEISE = "i" >> STRICT_BOUNDARIES; tokenWHILE = "while" >> STRICT_BOUNDARIES; tokenDREAK = "break" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenBET = "get" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenDAME = "startprogram" >> STRICT_BOUNDARIES; tokenBOD = "startblock" >> STRICT_BOUNDARIES; tokenBOD = "startblock" >> STRICT_BOUNDARIES; tokenBOD = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenENDBLOCK = "\" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "\" >> BOUNDARIES; tokenSEMICOLON = ";" >> BOUNDARY >> (BOUNDARY) (I(qi:aipha qi:char_("_"))); BOUNDARY => BOUNDARY -> PACE BOUNDARY_INB BOUNDARY SPACE BOUNDARY_INB BOUNDARY SPACE = "\"; BOUNDARY SPACE = "\"; BOUNDARY == "\t";	tokenMUL = "*" >> BOUNDARIES;
tokenGROUPEXPRESSIONBEGIN = "(">> BOUNDARIES; tokenGROUPEXPRESSIONEND = "]" >> BOUNDARIES; tokenLRBIND = "->">> BOUNDARIES; tokenELSE = 'Estes" >> STRICT_BOUNDARIES; tokenELSE = 'Estes" >> STRICT_BOUNDARIES; tokenWHILE = "while" >> STRICT_BOUNDARIES; tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenBREAK = "pet" >> STRICT_BOUNDARIES; tokenBREAK = "pet" >> STRICT_BOUNDARIES; tokenANAME = "startpogram" >> STRICT_BOUNDARIES; tokenBODY = "startpogram" >> STRICT_BOUNDARIES; tokenBODY = "startplock" >> STRICT_BOUNDARIES; tokenBODY = "nonDolock" >> STRICT_BOUNDARIES; tokenENDATA = "variable" >> STRICT_BOUNDARIES; tokenENDBLOCK = "" >> BOUNDARIES; tokenENDBLOCK = "" >> BOUNDARY >> "(BOUNDARY) NO_BOUNDARY NO_BOUNDARY NO_BOUNDARY NO_BOUNDARY NO_BOUNDARY NO_BOUNDARY NO_BOUNDARY NO_BOUNDA	_
tokenRBOUPEXPRESSIONEND = ")" >> BOUNDARIES; tokenLBRIND = "->" >> BOUNDARIES; tokenESE = "else" >> STRICT_BOUNDARIES; tokenESE = "else" >> STRICT_BOUNDARIES; tokenCONTINUE = "while" >> STRICT_BOUNDARIES; tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenBOD = "startblock" >> STRICT_BOUNDARIES; tokenBOD = "endblock" >> STRICT_BOUNDARIES; tokenBOD = "endblock" >> STRICT_BOUNDARIES; tokenBODEOCK = "j" >> BOUNDARIES; tokenENDBLOCK = "j" >> BOUNDARIES; tokenENDBLOCK = "j" >> BOUNDARIES; tokenENGHTSQUAREBRACKETS = "[" >> BOUNDARIES; tokenSEMICOLON = "," >> BOUNDARIES; strict_BOUNDARY >> "(BOUNDARY) (!(qi::alpha qi:char_("_"))); BOUNDARY = BOUNDARY >> *(BOUNDARY TAB BOUNDARY TAB BOUNDARY TAB BOUNDARY TAB BOUNDARY TAB BOUNDARY SPACE BOUNDARY TAB BOUNDARY SPACE BOUNDARY TAB BOUNDARY SPACE BOUNDARY TAB BOUNDARY SPACE = "";	tokenMOD = "%" >> STRICT_BOUNDARIES;
tokenLRBIND = "->" >> BOUNDARIES; tokenELSE = "else" >> STRICT_BOUNDARIES; tokenIF = "if" >> STRICT_BOUNDARIES; tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenBEAK = "break" >> STRICT_BOUNDARIES; tokenBEEAK = "break" >> STRICT_BOUNDARIES; tokenPUT = "get" >> STRICT_BOUNDARIES; tokenPUT = "yet" >> STRICT_BOUNDARIES; tokenPUT = "yet" >> STRICT_BOUNDARIES; tokenPUT = "yet" >> STRICT_BOUNDARIES; tokenDMA = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenBEGINBLOCK = "(" >> BOUNDARIES; tokenBEGINBLOCK = "(" >> BOUNDARIES; tokenEBEGINBLOCK = "(" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(q:::alpha q::char_("_"))); BOUNDARY = BOUNDARY SPACE BOUNDARY _IND_BOUNDARY); BOUNDARY = BOUNDARY _SPACE BOUNDARY _IND_BOUNDARY); BOUNDARY _SPACE = " ";	tokenGROUPEXPRESSIONBEGIN = "(" >> BOUNDARIES;
tokenELSE = "else" >> STRICT_BOUNDARIES; tokenWHILE = "while" >> STRICT_BOUNDARIES; tokenWHILE = "continue" >> STRICT_BOUNDARIES; tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenBEEAK = "break" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenENDBLOCK = "\" >> SOUNDARIES; tokenENDBLOCK = "\" >> BOUNDARIES; strict_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_("\"))) BOUNDARY = BOUNDARY >> *(BOUNDARY NO_BOUNDARY); BOUNDARY = BOUNDARY SPACE BOUNDARY INO_BOUNDARY); BOUNDARY = BOUNDARY_SPACE BOUNDARY_LINE_FEED BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_SPACE = "\"; BOUNDARY_TAB = "\t";	tokenGROUPEXPRESSIONEND = ")" >> BOUNDARIES;
tokenIF = "if" >> STRICT_BOUNDARIES; tokenWHILE = "while" >> STRICT_BOUNDARIES; tokenGONTINUE = "continue" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenAMME = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startplock" >> STRICT_BOUNDARIES; tokenBOT = "startplock" >> STRICT_BOUNDARIES; tokenBOT = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenENDBLOCK = "(" >> BOUNDARIES; tokenENDBLOCK = "(" >> BOUNDARIES; tokenERIGHBLOCK = "(" >> BOUNDARIES; tokenERIGHTSQUAREBRACKETS = "(" >> BOUNDARIES; tokenEIGHTSQUAREBRACKETS = "(" >> BOUNDARIES; stokenSEMICOLON = ";" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_("_"))); BOUNDARIES = (BOUNDARY >> *(BOUNDARY_TAB BOUNDARY = BOUNDARY_LINE_FEED BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_SPACE = ""; BOUNDARY_PACE = ""; BOUNDARY_TAB = "\t";	tokenLRBIND = "->" >> BOUNDARIES;
tokenWHILE = "while" >> STRICT_BOUNDARIES; tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenENDBLOCK = "(" >> BOUNDARIES; tokenENDBLOCK = "(" >> BOUNDARIES; tokenLEFTSQUAREBRACKETS = "[" >> BOUNDARIES; tokenLEFTSQUAREBRACKETS = "]" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES; stokenSEMICOLON = ";" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_("_"))); BOUNDARIES = (BOUNDARY >> PACE BOUNDARY); BOUNDARY = BOUNDARY SPACE BOUNDARY_LINE_FEED BOUNDARY_NULL; BOUNDARY_NULL; BOUNDARY_NULL; BOUNDARY_PACE = " "; BOUNDARY_PACE = " "; BOUNDARY_PACE = " ";	tokenELSE = "else" >> STRICT_BOUNDARIES;
tokenCONTINUE = "continue" >> STRICT_BOUNDARIES; tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startplock" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenENDBLOCK = "{" >> BOUNDARIES; tokenENDBLOCK = "{" >> BOUNDARIES; tokenENDBLOCK = "{" >> BOUNDARIES; tokenERIGHISQUAREBRACKETS = "[" >> BOUNDARIES; tokenRIGHISQUAREBRACKETS = "[" >> BOUNDARIES; tokenRIGHISQUAREBRACKETS = "[" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_[" = "))); BOUNDARY = BOUNDARY -> *(BOUNDARY -TAB BOUNDARY -> BOUNDARY -TAB BOUNDARY -> BOUNDARY -TAB BOUNDARY -> BOUNDARY	tokenIF = "if" >> STRICT_BOUNDARIES;
tokenBREAK = "break" >> STRICT_BOUNDARIES; tokenGET = "get" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenBATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenBEGINBLOCK = "{" >> BOUNDARIES; tokenBEGINBLOCK = "{" >> BOUNDARIES; tokenENDBLOCK = "}" >> BOUNDARIES; tokenENDBLOCK = "]" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "[" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "[" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!{qi::alpha qi::char_("_"))); BOUNDARY = BOUNDARY >> *(BOUNDARY TAB BOUNDARY CARRIAGAE RETURN BOUNDARY_LINE_FEED BOUNDARY_SPACE = ""; BOUNDARY_SPACE = ""; BOUNDARY_SPACE = ""; BOUNDARY_SPACE = "";	tokenWHILE = "while" >> STRICT_BOUNDARIES;
tokenGET = "get" >> STRICT_BOUNDARIES; tokenPUT = "put" >> STRICT_BOUNDARIES; tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenENDBLOCK = "(" >> BOUNDARIES; tokenENDBLOCK = "(" >> BOUNDARIES; tokenENDBLOCK = ")" >> BOUNDARIES; tokenENDBLOCK = ")" >> BOUNDARIES; tokenIGHTSQUAREBRACKETS = "[" >> BOUNDARIES; tokenSEMICOLON = ";" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_(" _"))); BOUNDARIES = (BOUNDARY >> *(BOUNDARY) NO_BOUNDARY); BOUNDARY = BOUNDARY => BOUNDARY_SPACE BOUNDARY_INE_FEED BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_NULL; BOUNDARY_SPACE = ""; BOUNDARY_TAB = "\t";	tokenCONTINUE = "continue" >> STRICT_BOUNDARIES;
tokenPUT = "put" >> STRICT_BOUNDARIES; tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenENDBLOCK = "\{" >> BOUNDARIES; tokenENDBLOCK = "\{" >> BOUNDARIES; tokenENDBLOCK = "\{" >> BOUNDARIES; tokenENFSQUAREBRACKETS = "\{" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "\{" >> BOUNDARIES; tokenSEMICOLON = "\{" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (\{\{\frac{1}{2}\}\}\}\}\] BOUNDARIES = (BOUNDARY >> *\{\frac{1}{2}\}\}\} BOUNDARY = BOUNDARY = BOUNDARY \{\frac{1}{2}\}\}\] BOUNDARY CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_SPACE = ""; BOUNDARY_SPACE = "\{\frac{1}{2}\}\} BOUNDARY_TAB = "\{\frac{1}{2}\}\}	tokenBREAK = "break" >> STRICT_BOUNDARIES;
tokenNAME = "startprogram" >> STRICT_BOUNDARIES; tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenBEGINBLOCK = "{" >> BOUNDARIES; tokenENDBLOCK = "}" >> BOUNDARIES; tokenENDBLOCK = "}" >> BOUNDARIES; tokenEFTSQUAREBRACKETS = "[" >> BOUNDARIES; tokenIGHTSQUAREBRACKETS = "]" >> BOUNDARIES; tokenSEMICOLON = ";" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!{qi::alpha qi::char_{(" _")})}; BOUNDARY = BOUNDARY >> *(BOUNDARY _TAB BOUNDARY _TAB BOUNDARY _TAB BOUNDARY _TAB = "\text{TEED} BOUNDARY _NULL; BOUNDARY_SPACE = ""; BOUNDARY_TAB = "\text{TED} = "\text{TED} BOUNDARY_TAB = "\text{TE} = "\text{TE} BOUNDARY_TAB = "\text{TE} = "\text{TE}	tokenGET = "get" >> STRICT_BOUNDARIES;
tokenBODY = "startblock" >> STRICT_BOUNDARIES; tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenBEGINBLOCK = "{" >> BOUNDARIES; tokenENDBLOCK = "}" >> BOUNDARIES; tokenENDBLOCK = "}" >> BOUNDARIES; tokenETSQUAREBRACKETS = "[" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES; tokenSEMICOLON = ";" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_("_"))); BOUNDARIES = (BOUNDARY >> *(BOUNDARY) NO_BOUNDARY); BOUNDARY = BOUNDARY = BOUNDARY SPACE BOUNDARY LINE_FEED BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_NULL; BOUNDARY_SPACE = " "; BOUNDARY_TAB = "\t";	tokenPUT = "put" >> STRICT_BOUNDARIES;
tokenDATA = "variable" >> STRICT_BOUNDARIES; tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenBEGINBLOCK = "\{" >> BOUNDARIES; tokenENDBLOCK = "\{" >> BOUNDARIES; tokenENDBLOCK = "\{" >> BOUNDARIES; tokenEFTSQUAREBRACKETS = "\{" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "\{" >> BOUNDARIES; tokenSEMICOLON = ";" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (\{\{\frac{1}{2}\}\}\}\}\] BOUNDARIES = (BOUNDARY >> *\{\frac{1}{2}\}\}\}\] BOUNDARY = BOUNDARY = BOUNDARY_SPACE BOUNDARY_TAB BOUNDARY_NULL; BOUNDARY_SPACE = ""; BOUNDARY_TAB = "\{\frac{1}{2}\}\}\}	tokenNAME = "startprogram" >> STRICT_BOUNDARIES;
tokenEND = "endblock" >> STRICT_BOUNDARIES; tokenBEGINBLOCK = "\" >> BOUNDARIES; tokenENDBLOCK = "\" >> BOUNDARIES; tokenENDBLOCK = "\" >> BOUNDARIES; tokenLEFTSQUAREBRACKETS = "\" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "\" >> BOUNDARIES; tokenSEMICOLON = ";" >> BOUNDARIES; STRICT_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";	tokenBODY = "startblock" >> 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";	tokenDATA = "variable" >> STRICT_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";	tokenEND = "endblock" >> STRICT_BOUNDARIES;
tokenLEFTSQUAREBRACKETS = "[" >> BOUNDARIES; tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES; tokenSEMICOLON = ";" >> BOUNDARIES; STRICT_BOUNDARIES = (BOUNDARY >> *(BOUNDARY)) (!(qi::alpha qi::char_("_"))); BOUNDARIES = (BOUNDARY >> *(BOUNDARY) NO_BOUNDARY); BOUNDARY = BOUNDARY = BOUNDARY_TAB BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_NULL; BOUNDARY_SPACE = " "; BOUNDARY_TAB = "\t";	tokenBEGINBLOCK = "{" >> 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";	tokenENDBLOCK = "}" >> 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";	tokenLEFTSQUAREBRACKETS = "[" >> 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";	tokenRIGHTSQUAREBRACKETS = "]" >> BOUNDARIES;
qi::char_("_"))); BOUNDARIES = (BOUNDARY) NO_BOUNDARY); BOUNDARY = BOUNDARY_SPACE BOUNDARY_TAB BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_NULL; BOUNDARY_SPACE = " "; BOUNDARY_TAB = "\t";	tokenSEMICOLON = ";" >> BOUNDARIES;
BOUNDARY = BOUNDARY_SPACE BOUNDARY_TAB BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_NULL; BOUNDARY_SPACE = " "; BOUNDARY_TAB = "\t";	
BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED BOUNDARY_NULL; BOUNDARY_SPACE = " "; BOUNDARY_TAB = "\t";	BOUNDARIES = (BOUNDARY >> *(BOUNDARY) NO_BOUNDARY);
BOUNDARY_NULL; BOUNDARY_SPACE = " "; BOUNDARY_TAB = "\t";	BOUNDARY = BOUNDARY_SPACE BOUNDARY_TAB
BOUNDARY_SPACE = " "; BOUNDARY_TAB = "\t";	BOUNDARY_CARRIAGE_RETURN BOUNDARY_LINE_FEED
BOUNDARY_TAB = "\t";	BOUNDARY_NULL;
	BOUNDARY_SPACE = " ";
BOUNDARY_CARRIAGE_RETURN = "\r";	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";
I = "I";
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";