|  |  |
| --- | --- |
|  |  |
| 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 >> 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 >> 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"; |