

CSC 488 Source Language Reference Grammar

Meta Notation: Alternatives within each rule are separated by commas.

Terminal symbols (except identifier, integer and text) are enclosed in single quote marks (').

% Comments extend to end of line and are not part of the grammar.

The Source Language

program:	scope	% main program
statement:	variable ':' '=' expression , 'if' expression 'then' statement , 'if' expression 'then' statement 'else' statement , 'while' expression 'do' statement , 'repeat' statement 'until' expression , 'exit' , 'exit' integer , 'exit' 'when' expression , 'exit' integer 'when' expression , 'return' 'with' expression , 'return' , 'write' output , 'read' input , procedurename , procedurename '(' arguments ')', scope , statement statement	% assignment % conditional statement % loop while expression is true % loop until expression is true % exit from containing loop % exit from <i>integer</i> loops % exit from containing loop % when expression is true % exit from <i>integer</i> loops % when expression is true % return from function % return from a procedure % print to standard output % input from standard input % call procedure % embedded scope % sequence of statements
declaration:	'var' variablenames ':' type , 'function' functionname ':' type scope , 'function' functionname '(' parameters ')' ':' type scope , 'procedure' procedurename scope , 'procedure' procedurename '(' parameters ')' scope , declaration declaration	% declare variables % declare function % declare procedure % sequence of declarations
variablenames:	variablename , variablename '[' integer ']' , variablename '[' bound ':' ':' bound ']' ,	% declare scalar variable % declare one dimensional array % bounds 1 .. <i>integer</i> inclusive % declare one-dimensional array % bounds <i>integer</i> .. <i>integer</i> inclusive
bound	variablenames ':' variablenames integer , '-' integer	% declare multiple variables % positive integer bound % negative integer bound
scope	'{' declaration statement '}' , '{' statement '}' , '{' '}'	% define new scope % empty scope

output:	expression , text , 'newline' , output ' , ' output	% integer expression to be printed % string constant to be printed % skip to new line % output sequence
input:	variable , input ' , ' input	% input to this integer variable % input sequence
type:	'Integer' , 'Boolean'	% integer type % Boolean type
arguments:	expression , arguments ' , ' arguments	% actual parameter expression % actual parameter sequence
parameters:	parametername ':' type , parameters ' , ' parameters	% declare formal parameter % formal parameter sequence
variable:	variablename , arrayname '[' expression ']'	% reference to scalar variable % reference to array element
expression:	integer , '-' expression , expression '+' expression , expression '-' expression , expression '*' expression , expression '/' expression , 'true' , 'false' , 'not' expression , expression 'and' expression , expression 'or' expression , expression '=' expression , expression 'not' '=' expression , expression '<' expression , expression '<' '=' expression , expression '>' expression , expression '>' '=' expression , '(' expression ')', '(' expression '?' expression ':' expression ')', variable , functionname , functionname '(' arguments ')', parametername	% integer literal constant % unary minus % addition % subtraction % multiplication % division % Boolean constant true % Boolean constant false % Boolean not % Boolean and % Boolean or % equality comparison % inequality comparison % less than comparison % less than or equal comparison % greater than comparison % greater than or equal comparison % conditional expression % reference to variable % call of a function % reference to a parameter
variablename:	identifier	
arrayname:	identifier	
functionname:	identifier	
parametername:	identifier	
procedurename:	identifier	

Notes

Identifiers are similar to identifiers in Java. Identifiers start with an upper or lower case letter and may contain letters or digits, as well as underscore .. Examples: sum, sum_0, I, XYZANY, CsC488 .

Function and procedure parameters are passed by value.

integer in the grammar stands for positive literal constants in the usual decimal notation. Examples: 0, 1, 100, 32767. Negative integer constants are expressions involving the unary minus operator.

The range of values for the **Integer** type is -32767 .. 32767.

A **text** is a string of characters enclosed in double quotes ("). Examples: "Compilers & Interpreters", "Hello World". The maximum allowable length of a text is 255 characters. Texts may only be used in the **write** statement.

Comments start with a '%' and continue to the end of the current line.

Lexical tokens may be separated by blanks, tabs, comments, or line boundaries. An identifier or reserved word must be separated from a following identifier, reserved word or integer; two integers must be separated, as well. In all other cases, tokens need not be separated. No token, text or comment can be continued across a line boundary.

Every identifier must be declared before it is used.

The number of elements in an array is specified in two ways:

a) by a single integer, which implies a lower bound of one.

For example A[3] has legal indices A[1], A[2], A[3] with a total size of 3.

b) by a pair of integers given in the array declaration.

The first integer is the lower bound and the second integer is the upper bound.

The lower bound must be less than or equal to the upper bound.

For example A [2 .. 5] has legal indices A[2], A[3], A[4] and A[5] with total size of 4.

B[-2 .. 1] has legal indices B[-2], B[-1], B [0] and B [1] with a total size of 4.

There are no type conversions. The precedence of operators is:

0. unary -
1. * /
2. + binary -
3. = **not** = < < = > > =
4. **not**
5. **and**
6. **or**

The operators of levels 1, 2, 5 and 6 associate from left to right.

The operators of level 3 do not associate, so a=b=c is illegal.

if-then-else statements have the usual structure; hence, an **if** statement can be followed either by a single statement, or by multiple statements wrapped in a scope. In particular, the following is not legal (the parser should report an error when reading line 4):

0. **if** expression
1. **then**
2. statement
3. statement
4. **else**
5. statement
6. statement