

CSC 488S 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' scope, 'repeat' scope 'until' expression , 'exit' , 'exit' integer , 'exit' 'when' expression , 'exit' integer 'when' expression , 'return' expression , 'write' output , 'read' input , procedurename , procedurename '(' arguments ') ' , scope , statement statement	% assignment % conditional statement % loop % exit from containing loop % exit from nested containing loop % exit from containing loop % when expression is true % exit from nested containing loop % return from function or procedure % expression ignored for procedure % print to standard output % input from standard input % call procedure % embedded scope % sequence of statements
declaration:	'var' variablenames ':' type , 'func' functionname ':' type scope , 'func' functionname '(' parameters ')' ':' type scope , 'proc' procedurename scope , 'proc' procedurename '(' parameters ')' scope , declaration declaration	% declare variables % declare function % declare procedure % sequence of declarations
variablenames:	variablename , variablename '[' integer '..' integer ']' , variablenames ',' variablenames	% declare scalar variable % declare one-dimensional array % bounds integer .. integer inclusive % declare multiple variables
scope	'begin' declaration statement 'end' , 'begin' statement 'end' , 'begin' declaration 'end' , 'begin' 'end'	% define new 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 'xor' expression , expression '=' expression , expression '<>' expression , expression '<' expression , expression '<=' expression , expression '>' expression , expression '>=' expression , '(' expression ')', '(' expression '?' expression ':' expression ')', variable , functionname , functionname '(' arguments ')', parametername	% literal constant % unary minus % addition % subtraction % multiplication % division % Boolean constant true % Boolean constant false % Boolean not % Boolean and % Boolean or % Boolean xor % 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 C. 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`, `CsC488s` .

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 **int** type is `-32767 .. 32767`.

A text is a string of characters enclosed in double quotes (`"`). Examples: `"sun rising"`, `"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 keyword must be separated from a following identifier, keyword, integer, or newline token; in all other cases, tokens need not be separated. No token, text or comment can be continued across a line boundary.

Each identifier must be declared before it is used.

The number of elements in an array is specified 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.

For example `A [2..5]` has indices `A[2]`, `A[3]`, `A[4]` and `A[5]` with total length 4.

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

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

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