

CSC 488S/2107S 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 'fi', 'if' expression 'then' statement 'else' statement 'fi', 'loop' statement 'pool' 'exit' , 'exit' integer , 'exit' 'when' expression , 'exit' integer 'when' expression , 'result' expression , 'return' , 'put' output , 'get' input , procedurename '(' argumentList ')', scope , statement statement	% assignment % conditional statement & infinite loop % 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
scope	'begin' declaration statement 'end' , 'begin' statement 'end'	% define new scope
declaration:	type variablenames , functionHead scope , procedureHead scope , 'forward' functionHead , 'forward' procedureHead , declaration declaration	% declare variables % declare function % declare procedure % forward function declaration % forward procedure declaration % sequence of declarations
functionHead procedureHead	type functionname '(' parameterList ')' 'proc' procedurename '(' parameterList ')'	% declare function head % declare procedure head
variablenames:	variablename , variablename '[' integer ']' , variablename '[' bound ':' ':' bound ']' , variablenames ':' variablenames	% 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 % declare multiple variables
bound	integer , '-' integer	% positive integer bound % negative integer bound

type:	'integer' , 'boolean'	% integer type % Boolean type
output:	expression , text , 'skip' , 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
argumentList	arguments , % EMPTY	% arguments to function/procedure
arguments:	expression , arguments ' ' arguments	% actual parameter expression % actual parameter sequence
parameterList	parameters , % EMPTY	% formal parameters of function/procedure
parameters:	type parametername , 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' , '!' expression , expression '&' expression , expression ' ' expression , expression '=' expression , expression '!' '=' expression , expression '<' expression , expression '<' '=' expression , expression '>' expression , expression '>' '=' expression , '(' expression ') ' , variable , functionname '(' argumentList ') ' , 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 % 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`, `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 **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 ; 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 **forward** declaration allows the name and parameter list of functions and procedures to be predeclared before the actual declaration of the function or procedure. This feature is intended to facilitate writing of mutually recursive functions/procedures. The forward and actual declarations of a function or procedure must occur in the same scope.

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 coercions.. The precedence of operators is:

0. unary -
1. * /
2. + binary -
3. = # < <= > >=
4. !
5. &
6. |

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.