University of Toronto

CSC 488S / CSC2107S Compilers and Interpreters **Winter 2014/2015**

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 ,	% assignment
	'**if**' expression '**then**' statement '**end**' ,	% conditional statement
	'**if**' expression '**then**' statement '**else**' statement '**end**' ,	
	'**while**' expression '**do**' statement '**end**' ,	% conditional loop
		% loop while expression is true
	'**loop**' statement '**end**' ,	% infinite loop
	'**exit**' ,	% exit from containing loop
	'**exit**' '**when**' expression ,	% exit from containing loop
		% when expression is true
	'**return**' '(' expression ')' ,	% return from function
	'**return**' ,	% return from a procedure
	'**put**' output ,	% print to standard output
	'**get**' input ,	% input from standard input
	procedurename ,	% call procedure
	procedurename '(' arguments ')' ,	
	scope ,	% embedded scope
	type variablenames ,	% declare variables
	type '**function**' functionname scope ,	% declare function without parameters
	type '**function**' functionname '(' parameters ')' scope ,	% declare function with parameters
	'**procedure**' procedurename scope ,	% declare procedure without parameters
	'**procedure**' procedurename '(' parameters ')' scope ,	% declare procedure with parameters
	statement statement	% sequence of statements
variablenames:	variablename ,	% declare scalar variable
	variablename '[' bound ']' ,	% declare one dimensional array
	variablename '[' bound ',' bound ']' ,	% declare two-dimensional array
	variablenames ',' variablenames	% declare multiple variables
bound	integer ,	% bounds 1 .. *integer* inclusive
	generalBound '.' '.' generalBound	% bounds left bound .. right bound
generalBound	integer ' ,	% positive integer bound
	'–' integer	% negative integer bound
scope	'**begin**' statement '**end**' ,	% define new scope
	'**begin**' '**end**'	

output:	expression ,	% integer expression to be printed
	text ,	% string constant to be printed
	'**skip**' ,	% skip to new line
	output ',' output	% output sequence
input:	variable ,	% input to this integer variable
	input ',' input	% input sequence
type:	'**integer**' ,	% integer type
	'**boolean**'	% Boolean type
arguments:	expression ,	% actual parameter expression
	arguments ',' arguments	% actual parameter sequence
parameters:	type parametername ,	% declare formal parameter
	parameters ',' parameters	% formal parameter sequence
variable:	variablename ,	% reference to scalar variable
	arrayname '[' expression ']' ,	% reference to 1-dimensional array element
	arrayname '[' expression ',' expression ']'	% reference to 2-dimensional array element
expression:	integer ,	% literal constant
	'-' expression ,	% unary minus
	expression '+' expression ,	% addition
	expression '-' expression ,	% subtraction
	expression '*' expression ,	% multiplication
	expression '/' expression ,	% division
	'**true**' ,	% Boolean constant true
	'**false**' ,	% Boolean constant false
	'!' expression ,	% Boolean not
	expression '&' expression ,	% Boolean and
	expression '|' expression ,	% Boolean or
	expression '=' expression ,	% equality comparison
	expression '!' '=' expression ,	% inequality comparison
	expression '<' expression ,	% less than comparison
	expression '<' '=' expression ,	% less than or equal comparison
	expression '>' expression ,	% greater than comparison
	expression '>' '=' expression ,	% greater than or equal comparison
	'(' expression ')' ,	% parenthesized expression
	'{' statement '**yields**' expression '}' ,	% anonymous parameterless function
	variable ,	% reference to variable
	functionname ,	% call a function without arguments
	functionname '(' arguments ')' ,	% call a function with arguments
	parametername	% 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 , digits and underscores. Examples: sum, sum_0, I, XYZANY, CsC488s .
Function and procedure parameters are passed by value, entire arrays may not be passed as parameters.

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 length of a text is 255 characters. Texts may only be used in the **put** 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. Because variable, function and procedure declarations are a form of statement, these declarations can occur anywhere in a scope. If an identifier is declared (as a variable, function or procedure) in a scope, the declaration must precede all uses of the identifier in the scope.

The number of elements in a one or two dimensional 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 indices A[2], A[3], A[4] and A[5] with total size of 4.
B[-2 .. 1] has 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. = ! ’ = < < = > > =
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.
The & and | operators are *conditional* as in C and Java.