University of Toronto

CSC 488S / CSC2107S Compilers and Interpreters **Winter 2013/2014**

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 ,	% assignment
	’**if**’ expression ’**then**’ statement ’**fi**’ ,	% conditional statement
	’**if**’ expression ’**then**’ statement ’**else**’ statement ’**fi**’ ,	
	’**while**’ expression ’**do**’ statement ’**end**’ ,	% loop while expression is true
	’**repeat**’ statement ’**until**’ expression ,	% loop until expression is true
	’**exit**’ ,	% exit from containing loop
	’**exit**’ ’**when**’ expression ,	% exit from containing loop % when expression is true
	’**result**’ expression ,	% return from function
	’**return**’ ,	% return from a procedure
	’**put**’ output ,	% print to standard output
	’**get**’ input ,	% input from standard input
	procedurename ’(’ argumentList ’)’ ,	% call procedure
	scope ,	% embedded scope
	statement statement	% sequence of statements
scope	’{’ declaration statement ’}’ ,	% define new scope
	’{’ statement ’}’ ,	
	’{’ ’}’	% empty scope
declaration:	’**var**’ variablenames ’:’ type ,	% declare variables
	functionHead scope ,	% declare function
	procedureHead scope ,	% declare procedure
	’**forward**’ functionHead ,	% forward function declaration
	’**forward**’ procedureHead ,	% forward procedure declaration
	declaration declaration	% sequence of declarations
functionHead	’**func**’ functionname ’(’ parameterList ’)’ ’:’ type	% declare function head
procedureHead	’**proc**’ procedurename ’(’ parameterList ’)’	% declare procedure head
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 ,	% bound 1 .. *integer* inclusive
	generalBound ’.’ ’.’ generalBound	% bounds left bound .. right bound inclusive
generalBound	integer ,	% positive integer bound
	’-’ integer	% negative integer bound

type:	’**integer**’ ,	% integer type
	’**boolean**’	% Boolean type
output:	expression ,	% integer expression to be printed
	text ,	% string constant to be printed
	’**newline**’ ,	% skip to new line
	output ’,’ output	% output sequence
input:	variable ,	% input to this integer variable
	input ’,’ input	% input sequence
argumentList	arguments ,	% arguments to function/procedure
	% EMPTY	
arguments:	expression ,	% actual parameter expression
	arguments ’,’ arguments	% actual parameter sequence
parameterList	parameters ,	% formal parameters of function/procedure
	% EMPTY	
parameters:	parametername ’:’ type,	% declare formal parameter
	parameters ’,’ parameters	% formal parameter sequence
variable:	variablename ,	% reference to scalar variable
	parametername ,	% reference to parameter
	arrayname ’[’ expression ’]’ ,	% reference to one dimensional array element
	arrayname ’[’ expression ’,’ expression ’]’	% reference to two dimensional array element
expression:	integer ,	% 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
	’**not**’ expression ,	% Boolean not
	expression ’**and**’ expression ,	% Boolean and
	expression ’**or**’ expression ,	% Boolean or
	expression ’=’ expression ,	% equality comparison
	expression ’**not**’ ’=’ 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 ’)’ ,	
	’(’ expression ’?’ expression ’:’ expression ’)’ ,	% conditional expression
	variable ,	% reference to variable
	functionname ’(’ argumentList ’)’ ,	% call of a function
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 .
Every identifier must be declared before it is used.

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 **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.

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.
Function and procedure arguments are passed by value.

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 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.	**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.
The **and** , **or** , **not** operators are conditional as in C or Java.

Multiple statements are permitted in the body of **if** , **while** and **repeat** statements.