University of Toronto

CSC 488S / CSC2107S Compilers and Interpreters **Winter 2008/2009**

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 ,             % conditional statement
                'if' expression 'then' statement 'else' statement ,
                'while' expression 'do' scope,                 % loop
                'repeat' scope 'until' expression ,
                'exit' ,                                       % exit from containing loop
                'exit' integer ,                               % exit from nested containing loop
                'exit' 'when' expression ,                     % exit from containing loop
                                                               % when expression is true
                'exit' integer 'when' expression ,             % exit from nested containing loop
                'return' expression ,                          % return from function or procedure
                                                               % expression ignored for procedure
                'write' output ,                               % print to standard output
                'read' input ,                                 % input from standard input
                procedurename ,                                % call procedure
                procedurename '(' arguments ')' ,
                scope ,                                        % embedded scope
                statement statement                            % sequence of statements

declaration:    'var' variablenames ':' type ,                 % declare variables
                'func' functionname ':' type scope ,           % declare function
                'func' functionname '(' parameters ')' ':' type scope ,
                'proc' procedurename scope ,                   % declare procedure
                'proc' procedurename '(' parameters ')' scope ,
                declaration declaration                        % sequence of declarations

variablenames:  variablename ,                                 % declare scalar variable
                variablename '[' integer '..' integer ']' ,    % declare one-dimensional array
                                                               % bounds integer .. integer inclusive
                variablenames ',' variablenames                % declare multiple variables

scope           'begin' declaration statement 'end' ,          % define new scope
                'begin' statement 'end' ,
                'begin' declaration 'end' ,
                'begin' 'end'
```

```
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

type:           'Integer' ,                                         % integer type
                'Boolean'                                           % Boolean type

arguments:      expression ,                                        % actual parameter expression
                arguments ',' arguments                             % actual parameter sequence

parameters:     parametername ':' type ,                            % declare formal parameter
                parameters ',' parameters                           % formal parameter sequence

variable:       variablename ,                                      % reference to scalar variable
                arrayname '[' expression ']'                        % reference to 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
                'not' expression ,                                  % Boolean not
                expression 'and' expression ,                       % Boolean and
                expression 'or' expression ,                        % Boolean or
                expression 'xor' expression ,                       % Boolean xor
                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 ')' ,
                '(' expression '?' expression ':' expression ')' ,  % conditional expression
                variable ,                                          % reference to variable
                functionname ,                                      % call of a function
                functionname '(' arguments ')' ,
                parametername                                       % 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
```