University of Toronto

CSC 488 / CSC2107 Compilers and Interpreters **Winter 2022**

CSC 488 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 (').
'//' comment extend to end of line and are not part of the grammar.

The Source Language – MiniC

program:	preamble declaration	// main program
preamble:	'#include' '"minicio.h"'	// optional system i/o header
	ε	
declaration:	vardecl,	// variable declarations
	returntype functionname '(' parameters ')' scope ,	// declare and define function
	returntype functionname '(' parameters ')' ';' ,	// declare function only
	declaration declaration	// sequence of declarations
vardecl:	vartype variablelist ';' ,	// declare variables
	vardecl vardecl	// list of var declarations
scope	'{' vardecl statement '}' ,	// define new scope
	'{' statement '}' ,	// sequence of statements
	'{' '}'	// empty scope
statement:	expr ';',	// expression statements
	'**if**' '(' expr ')' statement ,	// conditional statement
	'**if**' '(' expr ')' statement '**else**' statement ,	// conditional with else
	'**for**' '(' expropt ';' expropt ';' expropt ')' statement,	// for loop
	'**while**' '(' expropt ')' statement,	// while loop
	'**break**' ';',	// exit from containing loop
	'**return**' ';',	// return from the function
	'**return**' expr ';',	
	scope ,	// embedded scope
	statement statement	// sequence of statements
variablelist:	variablename ,	// declare scalar variable
	variablename '[' integer ']' ,	// declare an array
	variablelist ',' variablelist	// list of variables
vartype:	'**int**' ,	// integer type
	'**bool**'	// Boolean type
returntype:	'**void**' ,	// void type
	vartype	// other variable types

```
parameters:       ε,                                  // empty
                  parameterlist                       // parameter sequence

parameterlist:    vartype parametername,              // declare formal parameter
                  parameterlist ',' parameterlist     // parameter sequence

expropt:          expr,                               // expression or empty
                  ε

expr:             integer ,                           // integer literal constant
                  '-' expr ,                          // unary minus
                  expr '+' expr ,                     // addition
                  expr '-' expr ,                     // subtraction
                  expr '*' expr ,                     // multiplication
                  expr '/' expr ,                     // division
                  'true' ,                            // Boolean constant true
                  'false' ,                           // Boolean constant false
                  '!' expr ,                          // Boolean not
                  expr '&&' expr ,                    // Conditional Boolean and
                  expr '||' expr ,                    // Conditional Boolean or
                  expr '==' expr ,                    // equality comparison
                  expr '!=' expr ,                    // inequality comparison
                  expr '<' expr ,                     // less than comparison
                  expr '<' '=' expr ,                 // less than or equal comparison
                  expr '>' expr ,                     // greater than comparison
                  expr '>' '=' expr ,                 // greater than or equal comparison
                  '(' expr ')' ,
                  variable ,                          // reference to variable
                  functionname '(' arguments ')' ,    // call of a function
                  variable '=' expr ,                 // assignment expression
                  parametername                       // reference to a parameter

variable:         variablename ,                      // reference to scalar variable
                  arrayname '[' expr ']'              // reference to 1-dimensional array element

arguments:        ε,                                  // empty
                  argumentlist                        // actual parameter sequence

argumentlist:     expr ,                              // actual parameter expression
                  argumentlist ',' argumentlist       // actual parameter sequence

variablename:     identifier
arrayname:        identifier
functionname:     identifier
parametername:    identifier
```

Notes

MiniC is a subset of C language in C99 standard. A valid MiniC program is always a valid C program in the C99 standard with the same semantic meanings (but a valid C program may not be a valid MiniC program). Identifiers are same 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 parameters are passed by value.

in miniC we do not have the idea of pointers and passing by pointers

integer in the grammar stands for positive literal constants in the usual decimal notation. Examples: 0, 1, 100, 32767, and 100000. Negative integer constants are expressions involving the unary minus operator.
The range of values for the **int** type is -2^{31} .. $2^{31}-1$.
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 number of elements in an array is specified by a single integer. The index of an array in MiniC always starts from 0. For example A[3] has legal indices A[0], A[1], A[2] with a total size of 3.
There are no type conversions. The precedence of operators is:

0. unary -
1. * /
2. + binary -
3. == != < <= > >=
4. !
5. &&
6. ||
7. =

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 assignment operator of level 7 associates from right to left.
The '&&' and '`||`' operators are *conditional* as in C and Java.

if-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, this example is not legal (the parser should report an error when reading line 3):

1. **if** (expression)
2. statement
3. statement
4. **else**
5. statement
6. statement

Runtime for I/O

MiniC has a small system runtime library to handle I/O operations. The library contains three functions: `getint()`, which reads an integer from the standard input (i.e., stdin), `putint(x)`, which writes the ineger value `x` to the standard output followed by a space, and `putnewline()`, which writes a newline to the standard output. The type signature of these two functions are as follows:

```
int getint();

void putint(int x);

void putnewline();
```