Context-free grammar for Minijava variant (version komp12.3)

Reserved words are **bold face**. Terminal and non-terminal symbols are *italics*. Literal strings which are not reserved words are in `typewriter face`.

Please see the course project pages for the latest version of this grammar.

Program	→	*MainClass ClassDecl**
MainClass	→	**class** *id* { **public static void** `main` (**String** `[ ]` *id*) { *VarDecl* Stmt** } }
ClassDecl	→	**class** *id* { *VarDecl* MethodDecl** }
VarDecl	→	*Type id* ;
MethodDecl	→	**public** *Type id* (*FormalList*) { *VarDecl* Stmt** **return** *Exp* ; }
FormalList	→	*Type id FormalRest**
	→	
FormalRest	→	, *Type id*
Type	→	**int** `[ ]`
	→	**boolean**
	→	**int**
	→	*id*
Stmt	→	{ *Stmt** }
	→	**if** `(` *Exp* `)` *Stmt* **else** *Stmt*
	→	**while** `(` *Exp* `)` *Stmt*
	→	**System.out.println** `(` *Exp* `)` ;
	→	*id* `=` *Exp* ;
	→	*id* `[` *Exp* `]` `=` *Exp* ;
Exp	→	*Exp Op Exp*
	→	*Exp* [*Exp*]
	→	*Exp* . **length**
	→	*Exp* . *id* (*ExpList*)
	→	*int_lit*
	→	**true**
	→	**false**
	→	*id*
	→	**this**
	→	**new int** [*Exp*]
	→	**new** *id* ( )
	→	! *Exp*
	→	(*Exp*)
Op	→	&&
	→	<
	→	+
	→	−
	→	*
ExpList	→	*Exp ExpRest**
	→	
ExpRest	→	, *Exp*

Grammar extensions

These are *grammar* extensions. For a list of all types of extensions, please see project web pages.

Extension 15p:

Stmt → **if** (*Exp*) *Stmt*

Extension 15p/5p (5p if combined with X86_64 and INT32, else 15p):

Type → **long [ ]**
→ **long**
Exp → *long_lit*

Extension 20p (syntax checks) + 10p/30p (see course project web pages for point rules):

ClassDecl → **class** *id* **extends** *id* { *VarDecl* MethodDecl** }

Extension 20p. Replace first *Stmt* production:

Stmt → { *VarDecl* Stmt** }

(Please note that Java does not permit reuse of an identifier in a nested block; we should keep to that restriction for Minijava.)

Extension 1p per operator:

Op → <=
→ >
→ >=
→ ==
→ !=

Extension 2p:

Op → ||

Extension Xp (suggest your own extension!)

Lexicals

id := [a-zA-Z_][a-zA-Z0-9_]*
int_lit := 0 | [1-9][0-9]*
long_lit := 0 | [1-9][0-9]*[lL]

Comments should be handled like in Java (i.e., no comment nesting like in Appel!):
/* this is a comment */
// and so is this

Context rules and Semantics

Minijava does not have method overloading.

The semantics of a Minijava program are defined by Java's semantics.

A program that is invalid Java is also invalid Minijava. Student Minijava compilers do not need to reject Minijava programs with potential variable reads prior to their their first initialisation.