

Context-free grammar for Minijava variant (version komp12.1)

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.

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

Grammar extensions

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

Extension 15p:

$$Stmt \rightarrow \text{if } (Exp) Stmt$$

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

$$Type \rightarrow \text{long } [\]$$
$$\rightarrow \text{long}$$
$$Exp \rightarrow long_lit$$

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

$$ClassDecl \rightarrow \text{class } id \text{ extends } id \{ VarDecl^* MethodDecl^* \}$$

Extension 20p. Replace first *Stmt* production:

$$Stmt \rightarrow \{ VarDecl^* Stmt^* \}$$

(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:

$$Op \rightarrow \leq$$
$$\rightarrow >$$
$$\rightarrow \geq$$
$$\rightarrow =$$
$$\rightarrow \neq$$

Extension 2p:

$$Op \rightarrow ||$$

Extension *X*p (suggest your own extension!)

Lexicals

$$id := [a-zA-Z][a-zA-Z0-9]^*$$
$$int_lit := 0 \mid [1-9][0-9]^*$$
$$long_lit := 0 \mid [1-9][0-9]^*[iL]$$

Context rules and Semantics

Minijava does not have method overloading.

The semantics of a Minijava program is 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 first initialisation.