

COMPILADORES E INTÉRPRETES

Proyecto N° 1

Sintaxis de MiniJava

Segundo Cuatrimestre de 2014

1. Introducción

Este documento describe la sintaxis de **MiniJava**. Como se menciona en el documento de especificación del proyecto, **MiniJava** es un lenguaje que puede verse como una simplificación de Java, donde por una parte no se consideran elementos avanzados como Genericidad, Excepciones o Hilos, y por otra parte la estructura de su sintaxis es más estricta.

En este documento, en primer lugar, se presentan los elementos léxicos correspondientes a la sintaxis del lenguaje, donde en particular se identificarán las palabras reservadas, forma de los identificadores y literales del lenguaje. Luego se presentará la estructura sintáctica del lenguaje mediante su gramática; en esta última se podrán identificar especialmente aquellos elementos que alejan a **MiniJava** de Java.

2. Componentes Léxicos

El primer paso para compilar un programa de **MiniJava** radica en convertir una entrada de caracteres en una entrada de tokens, donde cada token se corresponderá con un lexema de **MiniJava**. Los lexemas de **MiniJava** son palabras clave, nombres de tipos primitivos, literales, símbolos de puntuación, operadores e identificadores. Cada uno de estos elementos se describe en las siguientes secciones.

2.1. Espacios en Blanco

Los espacios en blanco en **MiniJava**, al igual que en Java, simplemente hacen más fácil la lectura del programa por un humano y **no** son tokens. Un espacio en blanco es un espacio, tab o un salto de línea.

2.2. Palabras Clave

MiniJava tiene las siguientes palabras clave:

class	extends	varinst	varlocal	static	dynamic
void	boolean	char	int	String	
if	else	while	for	return	
this	new	null	true	false	

Note que varias de las palabras clave originales de Java no están presentes en **MiniJava**. Un compilador de **MiniJava** podría considerar esas palabras clave no presentes como prohibidas en **MiniJava**. Por otra parte, observe que **MiniJava** tiene palabras clave que no están presentes en Java, como por ejemplo **varinst**, **varlocal** o **dynamic**. Estas palabras, como se podrá ver en la Sección 3, ayudarán a dar una estructura más estricta a los programas **MiniJava**.

2.3. Identificadores

Un identificador en **MiniJava** es una letra o un *underscore* (`_`) seguido de cero o más letras, dígitos y underscores. Un identificador no puede ser una palabra reservada. Es importante notar que los identificadores son case-sensitive.

2.4. Comentarios

Los comentarios son ignorados por el analizador léxico. **MiniJava**, al igual que Java, tiene dos estilos de comentarios:

- `/* comentario */` Comentarios multi-línea: Todos los caracteres desde `/*` hasta `*/` son ignorados.
- `// comentario` Comentario simple: Todos los caracteres desde `//` hasta el final de la línea son ignorados.

2.5. Literales

En **MiniJava** hay cinco tipos de literales, los cuales se corresponden a los tipos primitivos del lenguaje.

2.5.1. Literales Enteros

Un literal entero es una secuencia de uno o más dígitos. El valor de un literal entero corresponde a su interpretación estándar en base 10.

2.5.2. Literales Caracteres

Un literal caracter puede ser:

- `'x'` donde x es cualquier caracter excepto la barra invertida (`\`), el salto de línea, o la comilla simple (`'`). El valor del literal es el valor del caracter x .
- `'\x'` donde x es cualquier caracter excepto `n` o `t`. El valor del literal es el valor del caracter x .
- `'\t'`. El valor del literal es el valor del caracter Tab.
- `'\n'`. El valor del literal es el valor del caracter salto de línea.

2.5.3. Literales String

Un literal string se representa mediante una comilla doble (`"`) seguida de una secuencia de caracteres y finaliza con otra comilla doble (`"`). El valor del literal corresponde a la cadena de caracteres entre las comillas. Se restringe el uso del caracter de salto de línea y las comillas dobles como parte de la cadena de un literal string.

2.5.4. Literales Booleanos

Los literales booleanos se representan mediante las palabras reservadas **true** y **false**.

2.5.5. Literal Nulo

El literal nulo se representa mediante la palabra reservada **null**.

2.6. Puntuación

MiniJava cuenta con los siguientes símbolos de puntuación, utilizados para estructurar los programas: `() { } ; , .`

2.7. Operadores

El lenguaje MiniJava cuenta con los siguientes operadores:

<code>></code>	<code><</code>	<code>!</code>	
<code>==</code>	<code>>=</code>	<code><=</code>	<code>!=</code>
<code>+</code>	<code>-</code>	<code>*</code>	<code>/</code>
<code>&&</code>	<code> </code>	<code>%</code>	

2.8. Asignación

Al igual que en Java, en MiniJava el símbolo de asignación es el `=`.

2.9. Otros caracteres

Cualquier otra entrada que no conforme con las reglas anteriormente presentadas debe generar un error.

3. Gramática de MiniJava

Cualquier programa MiniJava sintácticamente válido debe ser producto de la gramática que se presenta en esta sección. La gramática sigue la notación BNF-extendida, donde:

terminal	es un símbolo terminal
<code><Class></code>	es un símbolo no terminal (además la primer letra es una mayúscula)
<code>ε</code>	representa la cadena vacía
<code>X*</code>	representa cero o más ocurrencias de X
<code>X+</code>	representa una o más ocurrencias de X
<code>X?</code>	representa cero o una ocurrencia de X
<code>X → Y</code>	representa una producción
<code>X → Y Z</code>	es una abreviación de <code>X → Y</code> o <code>X → Z</code>

4. Producciones

Las producciones de la Gramática de MiniJava son:

```
<Inicial> → <Clase>+

<Clase> → class identificador <Herencia>? { <Miembro>* }

<Herencia> → extends identificador

<Miembro> → <Atributo> | <Ctor> | <Metodo>

<Atributo> → varinst <Tipo> <ListaDecVars> ;

<Metodo> → <ModMetodo> <TipoMetodo> identificador <ArgsFormales> <Bloque>

<Ctor> → identificador <ArgsFormales> <Bloque>
```

$\langle \text{ArgsFormales} \rangle \rightarrow (\langle \text{ListaArgsFormales} \rangle^?)$
 $\langle \text{ListaArgsFormales} \rangle \rightarrow \langle \text{ArgFormal} \rangle$
 $\langle \text{ListaArgsFormales} \rangle \rightarrow \langle \text{ArgFormal} \rangle , \langle \text{ListaArgsFormales} \rangle$
 $\langle \text{ArgFormal} \rangle \rightarrow \langle \text{Tipo} \rangle \text{ identificador}$
 $\langle \text{ModMetodo} \rangle \rightarrow \text{static} \mid \text{dynamic}$
 $\langle \text{TipoMetodo} \rangle \rightarrow \langle \text{Tipo} \rangle \mid \text{void}$
 $\langle \text{Tipo} \rangle \rightarrow \langle \text{TipoPrimitivo} \rangle \mid \text{identificador}$
 $\langle \text{TipoPrimitivo} \rangle \rightarrow \text{boolean} \mid \text{char} \mid \text{int} \mid \text{String}$
 $\langle \text{ListaDecVars} \rangle \rightarrow \text{identificador}$
 $\langle \text{ListaDecVars} \rangle \rightarrow \text{identificador} , \langle \text{ListaDecVars} \rangle$
 $\langle \text{Bloque} \rangle \rightarrow \{ \langle \text{Sentencia} \rangle^* \}$
 $\langle \text{Sentencia} \rangle \rightarrow ;$
 $\langle \text{Sentencia} \rangle \rightarrow \langle \text{Asignacion} \rangle ;$
 $\langle \text{Sentencia} \rangle \rightarrow \langle \text{SentenciaSimple} \rangle ;$
 $\langle \text{Sentencia} \rangle \rightarrow \text{varlocal} \langle \text{Tipo} \rangle \langle \text{ListaDecVars} \rangle ;$
 $\langle \text{Sentencia} \rangle \rightarrow \text{if} (\langle \text{Expresion} \rangle) \langle \text{Sentencia} \rangle$
 $\langle \text{Sentencia} \rangle \rightarrow \text{if} (\langle \text{Expresion} \rangle) \langle \text{Sentencia} \rangle \text{ else } \langle \text{Sentencia} \rangle$
 $\langle \text{Sentencia} \rangle \rightarrow \text{while} (\langle \text{Expresion} \rangle) \langle \text{Sentencia} \rangle$
 $\langle \text{Sentencia} \rangle \rightarrow \text{for} (\langle \text{Asignacion} \rangle ; \langle \text{Expresion} \rangle ; \langle \text{Asignacion} \rangle) \langle \text{Sentencia} \rangle$
 $\langle \text{Sentencia} \rangle \rightarrow \langle \text{Bloque} \rangle$
 $\langle \text{Sentencia} \rangle \rightarrow \text{return} \langle \text{Expresion} \rangle^? ;$
 $\langle \text{Asignacion} \rangle \rightarrow \langle \text{LadoIzquierdo} \rangle = \langle \text{Expresion} \rangle$
 $\langle \text{LadoIzquierdo} \rangle \rightarrow \text{identificador} \langle \text{IdEncadenado} \rangle^*$
 $\langle \text{IdEncadenado} \rangle \rightarrow . \text{ identificador}$
 $\langle \text{SentenciaSimple} \rangle \rightarrow (\langle \text{Expresion} \rangle)$
 $\langle \text{Expresion} \rangle \rightarrow \langle \text{Expresion} \rangle \langle \text{OperadorBinario} \rangle \langle \text{Expresion} \rangle$
 $\langle \text{Expresion} \rangle \rightarrow \langle \text{OperadorUnario} \rangle \langle \text{Expresion} \rangle$
 $\langle \text{Expresion} \rangle \rightarrow \langle \text{Operando} \rangle$
 $\langle \text{OperadorBinario} \rangle \rightarrow \mid \mid \mid \&\& \mid == \mid != \mid < \mid > \mid <= \mid >= \mid + \mid - \mid * \mid / \mid \%$
 $\langle \text{OperadorUnario} \rangle \rightarrow ! \mid + \mid -$
 $\langle \text{Operando} \rangle \rightarrow \text{this}$
 $\langle \text{Operando} \rangle \rightarrow \langle \text{Literal} \rangle$
 $\langle \text{Operando} \rangle \rightarrow \langle \text{Primario} \rangle$
 $\langle \text{Literal} \rangle \rightarrow \text{null} \mid \text{true} \mid \text{false} \mid \text{intLiteral} \mid \text{charLiteral} \mid \text{stringLiteral}$
 $\langle \text{Primario} \rangle \rightarrow (\langle \text{Expresion} \rangle) \langle \text{LlamadaoIdEncadenado} \rangle^*$
 $\langle \text{Primario} \rangle \rightarrow \text{identificador} \langle \text{LlamadaoIdEncadenado} \rangle^*$
 $\langle \text{Primario} \rangle \rightarrow \text{new identificador} \langle \text{ArgsActuales} \rangle \langle \text{LlamadaoIdEncadenado} \rangle^*$
 $\langle \text{Primario} \rangle \rightarrow \text{identificador} \langle \text{ArgsActuales} \rangle \langle \text{LlamadaoIdEncadenado} \rangle^*$

$\langle LlamadaoIdEncadenado \rangle \rightarrow . \text{identificador} \langle ArgsActuales \rangle$
 $\langle LlamadaoIdEncadenado \rangle \rightarrow . \text{identificador}$

$\langle ArgsActuales \rangle \rightarrow (\langle ListaExps \rangle^?)$

$\langle ListaExps \rangle \rightarrow \langle Expresion \rangle$

$\langle ListaExps \rangle \rightarrow \langle Expresion \rangle , \langle ListaExps \rangle$

4.1. Precedencia y Asociatividad de los Operadores

Los operadores binarios y unarios utilizados en MiniJava siguen las reglas de precedencia y asociatividad presentadas en la siguiente tabla. Note que 0 denota la mayor precedencia, mientras que 6 la menor.

Operador	Precedencia	Asociatividad
!	0	Ninguna
- Unario	0	Ninguna
+ Unario	0	Ninguna
*	1	Izquierda
/	1	Izquierda
%	1	Izquierda
+	2	Izquierda
-	2	Izquierda
<	3	No asociativo
>	3	No asociativo
<=	3	No asociativo
>=	3	No asociativo
==	4	Izquierda
!=	4	Izquierda
&&	5	Izquierda
	6	Izquierda

Por ejemplo, la expresión

$$a * b + c == a - 2 > f || g$$

debe ser reconocida como:

$$(((a * b) + c) == ((a - 2) > f)) || g)$$