COMPILADORES E INTÉRPRETES

SEGUNDO CUATRIMESTRE DE 2016

MANUAL DE USUARIO

COMPILADOR MINIJAVA

MATIAS MARZULLO, 80902

Índice

Introducción

El siguiente manual presenta toda la información necesaria para que el programador utilice el compilador de MINIJAVA junto con la máquina virtual CEIVM.

El compilador de MINIJAVA es el encargado de traducir a un código intermedio programas fuente escritos en el Lenguaje MINIJAVA. Junto con la traducción del código, el compilador se encarga de realizar los controles léxicos, sintácticos y semánticos del programa fuente.

El código traducido será generado por el compilador y escrito en un archivo de salida.

Luego, el intérprete CEIVM podrá realizar la ejecución del programa tomando como entrada la salida del compilador.

Especificación del Lenguaje MINIJAVA

El Lenguaje cuenta con los siguientes elementos:

1. Declaración de entidades de tipos primitivos (*int, boolean, String y char*) y referencia. Se cuenta además con el tipo especial *void*.
2. Declaración de clases concretas y herencia entre las mismas.
3. Declaración de constructores, métodos y atributos de instancia.
4. Declaración de métodos de clase.
5. Declaración de variables locales en métodos y constructores.
6. Visibilidad *public* y *private* para variables de instancia.
7. Las siguientes sentencias y expresiones: Asignaciones. Invocación de métodos. Creación de instancias de clase. Sentencias compuestas. Sentencias condicionales (*if* (expr) sent o *if* (expr) sent *else* sent) Sentencias *while*. Sentencias de retorno de métodos. Expresiones aritméticas con los operadores: +, -, *, / y %. Expresiones booleanas con los operadores: && (and), || (or) y ! (not), y aquellas formadas utilizando los operadores relacionales: >, <, =, >=(≥), <= (≤) y != (≠). Expresiones *Cast* y el uso del operador ***instanceof***.
8. El siguiente conjunto de clases predefinidas:
   a) *Object*: La superclase de todas las clases de MINIJAVA (al estilo de la clase *java.lang.Object* de JAVA). En MINIJAVA, la clase *Object* no posee métodos ni atributos.
   b) *System*: Contiene métodos útiles para realizar entrada/salida (al estilo de la clase *java.lang.System* de JAVA). A diferencia de *java.lang.System*, esta clase solo brinda acceso a los *streams* de entrada (System.in) y salida (*System.out*) pero de manera oculta, proveyendo directamente los siguientes métodos:
      - *static int read():* lee el próximo byte del *stream* de entrada estándar (originalmente en la clase *java.io.InputStream*).
      - *static void print(boolean b)*: imprime un *boolean* por salida estándar (originalmente en la clase *java.io.PrintStream*).
      - *static void print(char c):* imprime un *char* por salida estándar (originalmente en la clase *java.io.PrintStream*).
      - *static void print(int i):* imprime un *int* por salida estándar (originalmente en la clase *java.io.PrintStream*).
      - *static void print(String s):* imprime un *String* por salida estándar (originalmente en la clase *java.io.PrintStream*).
      - *static void println():* imprime un separador de línea por salida estándar finalizando la línea actual (originalmente en la clase *java.io.PrintStream*).
      - *static void println(boolean b):* imprime un *boolean* por salida estándar y finaliza la línea actual (originalmente en la clase *java.io.PrintStream*).
      - *static void println(char c):* imprime un *char* por salida estándar y finaliza la línea actual (originalmente en la clase *java.io.PrintStream*).
      - *static void println(int i):* imprime un *int* por salida estándar y finaliza la línea actual (originalmente en la clase *java.io.PrintStream*).
      - *static void println(String s*): imprime un *String* por salida estándar y finaliza la línea actual (originalmente en la clase *java.io.PrintStream*).

Consideraciones sobre el Lenguaje

Es importante tener presente las siguientes restricciones del Lenguaje MINIJAVA con respecto al Lenguaje JAVA en el que se basa:

1. No se cuenta con declaraciones *package* ni con cláusulas *import*. Todas las declaraciones de clases se realizan en un único archivo fuente y todas las clases coexisten en un mismo espacio de nombres.
2. No se cuenta con la posibilidad de declarar clases abstractas ni interfaces.
3. No se cuenta con la posibilidad de utilizar modificadores de visibilidad para clases ni métodos, tienen directamente visibilidad pública.
4. Las variables de instancia no tienen visibilidad por defecto (esta debe ser explicita)
5. La declaración de métodos de instancia estará precedida por la palabra reservada *dynamic*, mientras que la declaración de métodos de clase (al igual que en JAVA) estará precedida por la palabra reservada *static*.
6. La asignación en MINIJAVA es una sentencia, a diferencia de en JAVA donde es un operador que se puede utilizar como parte de una expresión.
7. El casting se realiza mediante corchetes en vez de paréntesis. Por ejemplo una expresión ***cast*** sintácticamente valida en MINIJAVA puede ser [*JButton*] b (en JAVA seria (*JButton*) b)
8. No se permite declarar excepciones definidas por el usuario ni, en general, manejar excepciones a nivel del Lenguaje.

9. Además de lo indicado anteriormente, no se permite el uso de los modificadores *transcient* ni *volatile* para campos, ni de los modificadores *synchronized* ni *native* para métodos.

10. El método *main* en programas MINIJAVA tiene un encabezado sin parámetros, es decir, respeta la forma:

    void static main()

MINIJAVA sólo provee las características indicadas en la sección precedente. Características no indicadas allí, tales como por ej., genericidad, anotaciones, *break*, *continue*, tipos primitivos de punto flotante, etc., no están soportadas por el Lenguaje.

Gramática

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
<Class> es un símbolo no terminal (además la primer letra es una mayúscula)
∈ representa la cadena vacía
X^* *representa cero o más ocurrencias de X*
X^+ *representa una o más ocurrencias de X*
$X^?$ *representa cero o una ocurrencia de X*
$X \rightarrow Y$ *representa una producción*
$X \rightarrow Y \mid Z$ *es una abreviación de* $X \rightarrow Y$ *o* $X \rightarrow Z$

Producciones BNF-Extendida

A continuación se detallan las producciones de la Gramática de MINIJAVA junto con una descripción y ejemplos:

$<Inicial> \rightarrow <Clase>+$ Puede declararse más de una clase, pero como mínimo una.
$<Clase> \rightarrow$ **class idClase** $<Herencia>^?$ { $<Miembro>^*$ }

$<Herencia> \rightarrow extends$ **idClase**
La declaración de una clase puede tener o no herencia (pudiendo heredar solo de una clase). Y puede tener un cuerpo, definido como Miembro, o el mismo puede no estar, teniendo en este caso una clase vacía.

Ejemplo con Herencia:
```
class A extends B { }
```

Ejemplo sin Herencia:
```
class A {}
```

$<Miembro> \rightarrow <Atributo> \mid <Ctor> \mid <Método>$
Un Miembro puede ser la declaración de un atributo, de un constructor o de un método.

$<Atributo> \rightarrow <Visibilidad> <Tipo> <ListaDecVars> ;$
Un atributo es declarado comenzando con la visibilidad, seguida del tipo, una lista de variables y finaliza con el carácter ";"

$<Método> \rightarrow <FormaMetodo> <TipoMetodo> idMetVar <ArgsFormales> <Bloque>$
La declaración de un método consta de la forma, seguido del tipo del método, su identificador, los argumentos formales, es decir, aquellos correspondientes a la declaración, y el cuerpo del mismo, denominado Bloque.
Ejemplo:

```
static int sumar(int n1, int n2){
        int aux;
        aux= n1 + n2;
        return aux;
}
```

$<Ctor> \rightarrow$ **idClase** $<ArgsFormales> <Bloque>$
Un constructor se declara de la misma manera que un método, sólo que en este caso, no se cuenta con la forma ni con el tipo. En un constructor, no pueden haber sentencias return, como si hay en el ejemplo anterior.
$<Visibilidad> \rightarrow public \mid private$
$<ArgsFormales> \rightarrow (<ListaArgsFormales>^?)$

Los argumentos formales de un método o un constructor son declarados como una lista de argumentos formales, la cual puede, o no, estar presente. La misma es encerrada entre los caracteres "(" y ")". Ejemplo sin lista de argumentos formales:

```
static int invertir(){
  return 0;
}
```

En el ejemplo para la declaración de método, se encuentra la otra declaración posible para los argumentos formales.

$<ListaArgsFormales> \rightarrow <ArgFormal>$
$<ListaArgsFormales> \rightarrow <ArgFormal>\ ,\ <ListaArgsFormales>$
Una lista de argumentos formales está constituida por muchos argumentos formales separados por el carácter ",", o sólo por un argumento formal.

$<ArgFormal> \rightarrow <Tipo>\ \mathbf{idMetVar}$
Un argumento formal se declara con su tipo seguido de su identificador asociado.
Ejemplo:

```
int a
```

$<FormaMetodo> \rightarrow static \mid dynamic$
La forma de un método está constituido o bien por la palabra reservada "static" o por la palabra "dynamic", no puede ser otra palabra.

$<TipoMetodo> \rightarrow <Tipo> \mid void$
El tipo de un método puede ser void o de tipo "Tipo" (definido más adelante).

$<Tipo> \rightarrow <TipoPrimitivo> \mid \mathbf{idClase}$
Un tipo puede ser un Tipo Primitivo o un Tipo Clase.

$<TipoPrimitivo> \rightarrow boolean \mid char \mid int \mid String$
Un tipo primitivo está dado por las palabras reservadas "boolean", "char", "int" o "String", cualquier otra palabra no será considerada un tipo primitivo.

$<ListaDecVars> \rightarrow \mathbf{idMetVar} \mid \mathbf{idMetVar},\ <ListaDecVars>$
Una declaración de una lista de variables está dada por un solo idMetVar, o por muchos idMetVar separados por el carácter ",".

$<Bloque> \rightarrow \{\ <Sentencia>^*\ \}$
Un bloque está dado por cero o más sentencias encerradas entre los caracteres "{" al principio y "}" al final. Un ejemplo de bloque con más de una sentencia se muestra en el ejemplo dado en la declaración de Método.

$<Sentencia> \rightarrow ;$
$<Sentencia> \rightarrow <Asignación> ;$
$<Sentencia> \rightarrow <SentenciaSimple> ;$
$<Sentencia> \rightarrow <Tipo>\ \ <ListaDecVars> ;$
$<Sentencia> \rightarrow if\ (\ <Expresión>\ )\ <Sentencia>$
$<Sentencia> \rightarrow if\ (\ <Expresión>\ )\ <Sentencia>\ else\ \ <Sentencia>$
$<Sentencia> \rightarrow while\ (\ <Expresión>\ )\ <Sentencia>$
$<Sentencia> \rightarrow <Bloque>$
$<Sentencia> \rightarrow return\ \ <Expresión>^?\ ;$
La declaración de Sentencias está dada por sólo un carácter ";", una asignación, una sentencia simple finalizada en ";", declaraciones de variables, los dos tipos de sentencias *if* posibles, es decir, con y sin el

"*else*", la sentencia "*while*", un bloque o la sentencia "*return*" siguiendo la especificación recién mostrada. Ejemplos de sentencias:

- `return;`
- ```
  if(a == 2)
     b = true;
  else b = false;
     a = 1;
  ```
- `int a;`
- ```
  while(a < 10){
     b = b * 10 + a;
     a = a +1;
   }
  ```

$<Asignación> \rightarrow <Primario> = <Expresión>$

La declaración de una asignación está dada por un <Primario> (definido más adelante) seguido del carácter "=", y este, a su vez es seguido de una <Expresión>.

Ejemplos:

a=2; m1.a = 5;

$<SentenciaLlamada> \rightarrow <Primario>$

$<Expresión> \rightarrow <ExpOr>$

$<ExpOr> \rightarrow <ExpOr> \; || \; <ExpAnd> \mid <ExpAnd>$

$<ExpAnd> \rightarrow <ExpAnd> \; \&\& \; <ExpIg> \mid <ExpIg>$

$<ExpIg> \rightarrow <ExpIg> <OpIg> <ExpComp> \mid <ExpComp>$

$<ExpComp> \rightarrow <ExpAd> <OpComp> <ExpAd> \mid <ExpAd> \; instanceof \; \mathbf{idClase} \mid <ExpAd>$

$<ExpAd> \rightarrow <ExpAd> <OpAd> <ExpMul> \mid <ExpMul>$

$<ExpMul> \rightarrow <ExpMul> <OpMul> <ExpUn> \mid <ExpUn>$

$<ExpUn> \rightarrow <OpUn> <ExpUn> \mid <ExpCast>$

$<ExpCast> \rightarrow [\; \mathbf{idClase} \;] <Operando> \mid < Operando>$

Estas son las distintas expresiones que se pueden generar, ya sea con un operador Unario o Binario.

Ejemplos:

```
3*3 , a || b, b && c, (con a,b,c booleans).
a instanceof d
!(2 == 2)
```

$<OpIg> \rightarrow == \mid !=$

$<OpComp> \rightarrow < \mid > \mid <= \mid >=$

$<OpAd> \rightarrow + \mid -$

$<OpUn> \rightarrow + \mid - \mid !$

$<OpMul> \rightarrow * \mid / \mid \%$

$<Operando> \rightarrow <Literal>$

$<Operando> \rightarrow <Primario>$

$<Literal> \rightarrow null \mid true \mid false \mid intLiteral \mid charLiteral \mid stringLiteral$

$<Primario> \rightarrow (<Expresión>) <LlamadaoIdEncadenado>*$

$<Primario> \rightarrow this <LlamadaoIdEncadenado>*$

$<Primario> \rightarrow \mathbf{idMetVar} <LlamadaoIdEncadenado>*$

$<Primario> \rightarrow(\; \mathbf{idClase}. <Llamada>) <LlamadaoIdEncadenado>*$

$<Primario> \rightarrow new \; \mathbf{idClase} <ArgsActuales> <LlamadaoIdEncadenado>*$

$<Primario> \rightarrow <Llamada> <LlamadaoIdEncadenado>*$

Un primario, siempre finaliza con cero o más llamadas o identificadores encadenados, y puede comenzar con:

- una expresión encerrada entre los caracteres "(" y ")",
- por un objeto actual (this),
- por un idMetVar,

- dado por un idClase, seguido de ".", seguido de una llamada, todo encerrado entre los caracteres "(" y ")",
- por la palabra reservada "new" seguida de un identificador idClase, argumentos actuales,
- o, por último, por una llamada

Ejemplos:

```
(this).metodo()
a.metodo()
new Clase(true,2)
metodo(true,2).m2().a
```

$<LlamadaoIdEncadenado> \rightarrow . <Llamada>$ | $. \mathbf{idMetVar}$

Una llamada o identificador encadenado comienza con punto y puede ser, un idMetVar o bien una llamada.

Ejemplo:

```
.a
.a(arg1,arg2)
```

$<Llamada> \rightarrow \mathbf{idMetVar}\ <ArgsActuales>$

Una llamada es un idMetVar seguido de argumentos (métodos)

$<ArgsActuales> \rightarrow (\ <ListaExps>^{?}\ )$

La declaración de los Argumentos Actuales está dada por cero o una Lista de Expresiones encerrada entre los caracteres "(" y ")".

Ejemplos:

```
(arg1,arg2)   , ()
```

$<ListaExps> \rightarrow <Expresión> \mid <Expresión>\ ,\ <ListaExps>$

Una lista de expresiones está dada por solo una expresión o por muchas expresiones separadas por el carácter ",".

Ejecución del Compilador

El compilador se distribuye en un archivo ejecutable para entornos JAVA. Para ejecutarlo, utilizar en línea de comandos el siguiente comando:

```
> java –jar <PROGRAM_NAME> <IN_FILE> [<OUT_FILE>]
```

Parámetros de Entada

<IN_FILE>: es el archivo de entrada correspondiente al programa escrito en Lenguaje MINIJAVA y es el que se desea compilar.
[<OUT_FILE>]: es opcional, es el archivo de salida que corresponde al código generado como resultado de la compilación del parámetro <IN_FILE> escrito en Lenguaje CEIASM (Lenguaje definido por la cátedra). Si se omite este parámetro, la salida se realizará con la creación de un archivo automático.

Ejemplo (y excepciones/restricciones de la gramática)

Los códigos fuentes que son válidos para el compilador se pueden deducir fácilmente de la gramática. Sin embargo, a continuación se muestran algunos ejemplos de códigos válidos, junto a algunas condiciones especiales que deben cumplir (excepciones a la gramática).

El nuevo archivo se ubicará en la misma carpeta que el archivo fuente.

Luego, para ejecutar el código intermedio, debe invocar a la Máquina Virtual de Compiladores e Intérpretes (CEIVM) que le fue provista junto al compilador, ejecutando el comando:

```
> java –jar CeiVM-cei2011.jar <IN_FILE> [<OUT_FILE>]
```

El modo de uso del front-end es el siguiente:

```
>java -jar CeIVM-cei2011.jar <archivo CeIASM>
```

La máquina virtual provee una funcionalidad adicional que permite realizar un listado de la información de ensamblado, linkeo y carga que puede ser útil al momento de corroborar que la configuración del ambiente de tiempo de ejecución y la generación de código es correcta. Además, brinda información de carga esencial para ubicar una instrucción a partir del valor del pc (errores de tiempo de ejecución). Para generar este archivo el modo de uso es alguno de los siguientes:

```
>java -jar CeIVM-cei2011.jar <archivo CeIASM> -v

>java -jar CeIVM-cei2011.jar <archivo CeIASM> -v <archivoListado>
```

La primera alternativa genera el listado en un archivo cuyo nombre resulta de añadir una extensión al nombre del archivo CEIASM. La segunda alternativa permite especificar un nombre para el archivo de listado.