Procesamiento de Lenguajes (PL)
Curso 2016/2017

Práctica 5: traductor a código `m2r`

Fecha y método de entrega

La práctica debe realizarse de forma individual o por parejas[1], y debe entregarse a través del servidor de prácticas del DLSI **antes de las 23:59 del miércoles 31 de mayo de 2017**.

Al servidor de prácticas del DLSI se puede acceder de dos maneras:

- Desde la web del DLSI (http://www.dlsi.ua.es), en el apartado "Entrega de prácticas"
- Desde la URL http://pracdlsi.dlsi.ua.es

Una vez en el servidor, se debe elegir la asignatura PL y seguir las instrucciones. En el caso de que la práctica la haya realizado una pareja, debe entregarla solamente una de las dos personas de la pareja, y se debe poner en un comentario los nombres y DNIs de los dos componentes de la pareja. Si no aparece dicho comentario se entenderá que la práctica se entrega de forma individual.

Descripción de la práctica

- La práctica 5 consiste en realizar un compilador para el lenguaje fuente que se describe más adelante (un subconjunto de Java), que genere código para el lenguaje objeto `m2r`, utilizando como en la práctica anterior `bison` y `flex`. Los fuentes deben llamarse `plp5.l` y `plp5.y`. Para compilar la práctica se utilizarán las siguientes órdenes:

```
$ flex plp5.l
$ bison -d plp5.y
$ g++ -o plp5 plp5.tab.c lex.yy.c
```

- El compilador debe diseñarse teniendo en cuenta las siguientes restricciones:

  1. En ningún caso se permite que el código objeto se vaya imprimiendo por la salida estándar conforme se realiza el análisis, debe imprimirse al final.
  2. Tampoco se permite utilizar ningún *buffer* o *array* global para almacenar el código objeto generado, el código objeto que se vaya generando debe circular por los atributos de las variables de la gramática (en memoria dinámica), y al final, en la regla correspondiente al símbolo inicial, se imprimirá por la salida estándar el código objeto; éste debe ser el único momento en que se imprima algo por la salida estándar. Se recomienda utilizar el tipo `string` para almacenar el código generado.
  3. Se aconseja utilizar muy pocas variables globales, especialmente en las acciones semánticas de las reglas; en ese caso siempre es aconsejable utilizar atributos. El uso de variables globales está, en general, reñido con la existencia de estructuras recursivas tales como las expresiones o los bloques de instrucciones. Se pueden utilizar variables globales temporales mientras se usen sea únicamente dentro de una misma acción semántica, sin efectos fuera de ella.

- Si se produce algún tipo de error, léxico, sintáctico o semántico, el programa enviará un mensaje de error de una sola línea a la salida de error (`stderr`). Se proporcionará una función "`msgError`" que se encargará de generar los mensajes de error correctos, a la que se debe pasar la fila, la columna y el lexema del token más relacionado con el error; en el caso de los errores léxicos y sintácticos, el lexema será la cadena "`yytext`" que proporciona el analizador léxico.

[1] Una pareja está compuesta por exactamente dos personas, no tres, ni cuatro, ...

Especificación sintáctica

La sintaxis del lenguaje fuente puede ser representada por la siguiente gramática:

S	$\longrightarrow$	*Import Class*
Import	$\longrightarrow$	*Import* **import** *SecImp* **pyc**
Import	$\longrightarrow$	ϵ
SecImp	$\longrightarrow$	*SecImp* **punto id**
SecImp	$\longrightarrow$	*SecImp* **punto scanner**
SecImp	$\longrightarrow$	**id**
Class	$\longrightarrow$	**public class id llavei** *Main* **llaved**
Main	$\longrightarrow$	**public static void main pari string cori cord id pard** *Bloque*
Tipo	$\longrightarrow$	**int**
Tipo	$\longrightarrow$	**double**
Tipo	$\longrightarrow$	**boolean**
Bloque	$\longrightarrow$	**llavei** *BDecl SeqInstr* **llaved**
BDecl	$\longrightarrow$	*BDecl DVar*
BDecl	$\longrightarrow$	ϵ
DVar	$\longrightarrow$	*Tipo LIdent* **pyc**
DVar	$\longrightarrow$	*Tipo DimSN* **id asig new** *Tipo Dimensiones* **pyc**
DVar	$\longrightarrow$	**scanner id asig new scanner pari system punto in pard pyc**
DimSN	$\longrightarrow$	*DimSN* **cori cord**
DimSN	$\longrightarrow$	**cori cord**
Dimensiones	$\longrightarrow$	**cori entero cord** *Dimensiones*
Dimensiones	$\longrightarrow$	**cori entero cord**
LIdent	$\longrightarrow$	*LIdent* **coma** *Variable*
LIdent	$\longrightarrow$	*Variable*
Variable	$\longrightarrow$	**id**
SeqInstr	$\longrightarrow$	*SeqInstr Instr*
SeqInstr	$\longrightarrow$	ϵ
Instr	$\longrightarrow$	**pyc**
Instr	$\longrightarrow$	*Bloque*
Instr	$\longrightarrow$	*Ref* **asig** *Expr* **pyc**
Instr	$\longrightarrow$	**system punto out punto println pari** *Expr* **pard pyc**
Instr	$\longrightarrow$	**system punto out punto print pari** *Expr* **pard pyc**
Instr	$\longrightarrow$	**if pari** *Expr* **pard** *Instr*
Instr	$\longrightarrow$	**if pari** *Expr* **pard** *Instr* **else** *Instr*
Instr	$\longrightarrow$	**while pari** *Expr* **pard** *Instr*
Expr	$\longrightarrow$	*Expr* **or** *EConj*
Expr	$\longrightarrow$	*EConj*
EConj	$\longrightarrow$	*EConj* **and** *ERel*
EConj	$\longrightarrow$	*ERel*
ERel	$\longrightarrow$	*Esimple* **relop** *Esimple*
ERel	$\longrightarrow$	*Esimple*
Esimple	$\longrightarrow$	*Esimple* **addop** *Term*
Esimple	$\longrightarrow$	*Term*
Term	$\longrightarrow$	*Term* **mulop** *Factor*
Term	$\longrightarrow$	*Factor*
Factor	$\longrightarrow$	*Ref*
Factor	$\longrightarrow$	**id punto nextint pari pard**
Factor	$\longrightarrow$	**id punto nextdouble pari pard**
Factor	$\longrightarrow$	**nentero**
Factor	$\longrightarrow$	**nreal**
Factor	$\longrightarrow$	**true**
Factor	$\longrightarrow$	**false**
Factor	$\longrightarrow$	**pari** *Expr* **pard**
Factor	$\longrightarrow$	**not** *Factor*
Factor	$\longrightarrow$	**pari** *Tipo* **pard** *Expr*
Ref	$\longrightarrow$	**id**
Ref	$\longrightarrow$	*Ref* **cori** *Esimple* **cord**

Especificación léxica

Expresión regular	Componente léxico	Valor léxico entregado
`[ \n\t]+`	(ninguno)	
`boolean`	**boolean**	(palabra reservada)
`int`	**int**	(palabra reservada)
`double`	**double**	(palabra reservada)
`main`	**main**	(palabra reservada)
`System`	**system**	(palabra reservada)
`out`	**out**	(palabra reservada)
`in`	**in**	(palabra reservada)
`println`	**print**	(palabra reservada)
`print`	**println**	(palabra reservada)
`String`	**string**	(palabra reservada)
`class`	**class**	(palabra reservada)
`import`	**import**	(palabra reservada)
`new`	**new**	(palabra reservada)
`public`	**public**	(palabra reservada)
`static`	**static**	(palabra reservada)
`void`	**void**	(palabra reservada)
`Scanner`	**scanner**	(palabra reservada)
`nextInt`	**nextint**	(palabra reservada)
`nextDouble`	**nextdouble**	(palabra reservada)
`if`	**if**	(palabra reservada)
`else`	**else**	(palabra reservada)
`while`	**while**	(palabra reservada)
`true`	**ctebool**	(palabra reservada)
`false`	**ctebool**	(palabra reservada)
`[A-Za-z][0-9A-Za-z]*`	**id**	(nombre del ident.)
`[0-9]+`	**nentero**	(valor numérico)
`([0-9]+)"."([0-9]+)`	**nreal**	(valor numérico)
`,`	**coma**	
`;`	**pyc**	
`.`	**punto**	
`(`	**pari**	
`)`	**pard**	
`==`	**relop**	`==`
`!=`	**relop**	`!=`
`<`	**relop**	`<`
`<=`	**relop**	`<=`
`>`	**relop**	`>`
`>=`	**relop**	`>=`
`+`	**addop**	`+`
`-`	**addop**	`-`
`*`	**mulop**	`*`
`/`	**mulop**	`/`
`=`	**assig**	
`[`	**cori**	
`]`	**cord**	
`{`	**llavei**	
`}`	**llaved**	
`&&`	**and**	
`||`	**or**	
`!`	**not**	

Notas:

1. El analizador léxico también debe ignorar los blancos[2] y los comentarios, que comenzarán por '`//`' y terminarán al final de la línea.

2. Hay componentes léxicos y reglas de la gramática cuyo único objetivo es conseguir que un programa en este lenguaje fuente pueda ser compilado también con un compilador de Java, como por ejemplo la importación de paquetes.En muchos casos, la práctica producirá un error sintáctico cuando un compilador de Java produciría un error semántico.

[2]Los tabuladores se contarán como un espacio en blanco.

Especificación semántica

Las reglas semánticas de este lenguaje son similares a las de Java, en muchos casos es necesario hacer conversiones de tipos, y en algunos casos se debe producir un mensaje de error semántico cuando aparezca alguna construcción no permitida. Las reglas se pueden resumir como:

Declaración de variables

- No es posible declarar dos veces un símbolo en el mismo ámbito con el mismo nombre, aunque sea con el mismo tipo (recuérdese que no se debe distinguir entre mayúsculas y minúsculas). Además, en Java no está permitido declarar en un ámbito una variable con el mismo nombre que otra variable de un ámbito exterior (no se permite *ocultar* variables).
- No es posible utilizar una variable sin haberla declarado previamente. Se pueden declarar variables de tipo simple (entero, real o booleano), variables de tipo `Scanner`, y arrays.
- Las variables de tipo `Scanner` sólo pueden utilizarse para leer con los métodos `nextInt` y `nextDouble`, nada más. Además, dichos métodos sólo pueden utilizarse con variables de tipo `Scanner`, como es lógico.
- Si al declarar una variable el espacio ocupado por ésta sobrepasa el tamaño máximo de memoria para variables (siempre inferior a 16384, que es el tamaño de la memoria de la máquina virtual), el compilador debe producir un mensaje de error indicando el lexema exacto de la variable que ya no cabe en memoria. El tamaño de todos los tipos básicos es 1, es decir, una variable simple ocupa una única posición, ya sea de tipo entero, real o booleano.
- De las 16384 posiciones de memoria de la máquina virtual, se deben destinar 16000 a variables declaradas por el programa fuente, y 384 a variables temporales.

Instrucciones

Asignación: en esta instrucción tanto la referencia que aparece a la izquierda del operador "`=`" como la expresión de la derecha deben ser del mismo tipo. Solamente hay una excepción a esta regla, y consiste en que está permitida también la asignación cuando la referencia es real y la expresión es entera (que debe convertirse a real antes de ser asignada). No es posible asignar valor a las variables de tipo `Scanner` (excepto en la declaración, por supuesto).

Lectura y escritura: al escribir valores booleanos se escribirá un 0 si es `false`, y un 1 si es `true`. Solamente es posible leer valores enteros o reales, y para ello se llamará a los métodos `nextInt` y `nextDouble` utilizando una variable de tipo `Scanner`.

Control de flujo: las instrucciones "`if`" y "`while`" tienen una semántica similar a la de Java, se exige que el tipo de la expresión sea booleano; en caso contrario, se debe producir un error semántico.

Bloques: las declaraciones de variables sólo son válidas en el cuerpo del bloque, es decir, una vez que se cierra el bloque, el compilador debe *olvidar* todas las variables declaradas en él. Además, no es posible declarar en un bloque variables con el mismo nombre que las variables de otros bloques de niveles superiores (en C/C++ sí está permitido, en Java no).

Expresiones

Operadores booleanos: en los operadores "`||`", "`&&`" y "`!`" los operandos deben ser booleanos. Por simplificar, la evaluación de los operadores "`||`" y "`&&`" debe ser como el resto de operadores binarios, es decir, no debe implementarse la evaluación en cortocircuito presente en Java y otros lenguajes derivados de C.

Operadores relacionales: Estos operadores permiten comparar valores numéricos (enteros o reales) entre sí, y también valores booleanos entre sí (en cuyo caso se considera que "`false`" es menor que "`true`"). No se permite comparar valores de distinto tipo (excepto valores enteros y reales, por supuesto). El resultado de una operación relacional siempre es de tipo booleano.

Operadores aritméticos: solamente pueden utilizarse con valores enteros o reales. La división entre valores enteros siempre es una división entera; si alguno de los operandos es real, la división es real (y puede ser necesario convertir uno de los operandos a real).

Arrays:

- En las declaraciones de variables de tipo *array*, el número de dimensiones del tipo debe coincidir con el número de dimensiones del array creado con new, y los tipos deben ser iguales; además, los números entre corchetes deben ser estríctamente mayor que cero (aunque Java permite arrays de tamaño 0). El rango de posiciones del *array* será, como en Java, de 0 a $n - 1$.
- No se permite utilizar una variable de tipo *array* con más ni con menos índices de los necesarios (según la declaración de la variable) para obtener un valor de tipo simple.
- No se permite poner índices (corchetes) a variables que no sean de tipo *array*.
- No está permitida la asignación entre valores de tipo *array*. Las asignaciones deben realizarse siempre con valores de tipo simple.
- La expresión entre corchetes (el índice) debe ser de tipo entero.
- En principio, el número de dimensiones que puede tener una variable de tipo *array* no está limitado más que por la memoria disponible. Por este motivo es aconsejable utilizar una tabla de tipos.

Conversión de tipo: es posible, utilizando un *cast*, convertir un factor de una expresión de un tipo simple (entero, real o booleano) a otro; en ese caso, los valores `true` y `false` se convertirían a 1 y 0, respectivamente, y los valores númericos se convertirán a `true` si son distintos de 0, y a `false` si son 0.

Mensajes de error

En la web de la asignatura se publicará un fichero con el código de una función para emitir mensajes de error (léxico, sintáctico y semántico). Los errores semánticos son:

`ERRYADECL`: se emite cuando ya se ha declarado una variable con el mismo identificador (en el ámbito actual o en un ámbito exterior). El lexema, la fila y la columna se deben referir al identificador que se va a declarar.

`ERRNODECL`: se emite cuando se utiliza un identificador y no ha sido declarado.

`ERR_NOSC`: este error se produce cuando se intenta utilizar los métodos `nextInt` o `nextDouble` con una variable que no es de tipo `Scanner`

`ERR_SCVAR`: este error se produce cuando se intenta utilizar una variable de tipo `Scanner` (sin la llamada a los métodos `nextInt` o `nextDouble`) en una expresión o en una asignación.

`ERR_TIPOSDECLARRAY`: se emite cuando, en una declaración de array, los tipos antes y después de `new` no coinciden. El lexema, la fila y la columna se deben referir al identificador que se va a declarar.

`ERR_DIMSDECLARRAY`: se emite cuando, en una declaración de array, el número de dimensiones antes y después de `new` no coinciden.

`ERRDIM`: se emite cuando, en una declaración de array, alguna de las dimensiones no es mayor que 0. El lexema, la fila y la columna se deben referir al número.

`ERR_TIPOSASIG`: si en una asignación los tipos de la referencia y la expresión no son compatibles, se emite este error. El lexema, la fila y la columna se deben referir al operador de asignación.

`ERR_TIPOS`: se emite cuando los dos operandos de un operador relacional no son compatibles. El lexema, la fila y la columna se deben referir al operador relacional.

`ERR_TIPOSIFW`: se emite cuando la expresión de un `if` o un `while` no es de tipo booleano. El lexema, la fila y la columna se deben referir al `if` o al `while`.

`ERR_OPNOBOOL`: se produce cuando alguno de los operandos de un operador booleano no es de tipo booleano. El lexema, la fila y la columna se deben referir al operador booleano.

`ERR_NUM`: se produce cuando alguno de los operandos de un operador numérico no es de tipo entero o real. El lexema, la fila y la columna se deben referir al operador.

`ERRFALTAN`: se produce cuando en una referencia a un array faltan índices por poner. El lexema, la fila y la columna se deben referir al último "`]`", o bien al identificador si no hay corchetes.

`ERRSOBRAN`: se produce cuando en una referencia a un array sobran índices. El lexema, la fila y la columna se deben referir al primer "`[`" que sobra.

`ERR_EXP_ENT`: se produce cuando en una referencia a un array la expresión entre corchetes no es de tipo entero.

`ERR_NOCABE`: este error se debe emitir cuando se declara una variable que no cabe en el espacio de memoria de la máquina virtual (16000 posiciones).

`ERR_MAXTMP`: este error se debe emitir cuando no es posible obtener ninguna variable temporal más, porque se han agotado las 384 posiciones disponibles.

`ERR_MAXVAR`, `ERR_MAXTIPOS`: si se utilizan vectores de tamaño fijo para la tabla de símbolos y la tabla de tipos, y alguno de ellos se llena, se debe emitir este error.