Extensión de la Práctica Principal de Procesadores de Lenguajes (Examen febrero 2016)

Esta práctica consiste en la implementación mediante JFlex y Cup de un compilador de un pequeño lenguaje de programación, similar a C, denominado lenguaje PLX. El lenguaje PLX es una extensión del lenguaje PL que se describe como práctica de la asignatura, pero en esta versión extendida se requieren algunas funciones adicionales. Se presupone que todos los elementos del lenguaje PL están presentes en el lenguaje PLX y que no se modifica su funcionamiento al incluir los nuevos elementos de esta extensión.

EL CÓDIGO FUENTE (Lenguaje PLX):

El lenguaje PLX incluye todas las sentencias definidas en el lenguaje PL y algunas más. Asimismo, se modifica ligeramente el lenguaje intermedio CTD, de manera que soporte algunas instrucciones adicionales.

* Eliminación de comentarios. El fichero fuente puede incluir comentarios, tanto en estilo C, como en estilo C++, que no generan ningún código. (5 puntos)

Código fuente (PLX)	Código intermedio (CTD)	Resultado de la ejecución

`// Esto es un comentario`
`print(1+2*3); // print(7);`
	`t0 = 2 * 3;`
	`t1 = 1 + t0;`
	`print t1;`
		7

`/* print(1+2*3) */`
`print(1+2*3); /* => 7 */`
	`t0 = 2 * 3;`
	`t1 = 1 + t0;`
	`print t1;`
		7

`print(0);`
`/* Los comentarios`
`  pueden tener`
`  varias lineas */`
`print(1);`
	`print 0;`
	`print 1;`
		0
		1

`print(0); // /* comentarios */`
`  /* print(2); // print(3); */`
`  /**/ print(4);`
`print(1);`
	`print 0;`
	`print 4;`
	`print 1;`
		0
		4
		1

`print(0);`
`/* Los comentarios`
`  /* sean del tipo que sean */`
`  no se pueden anidar`
`*/`
`print(1);`
	`print 0;`
	`error;`
	…	

* Expresiones de asignación y operación sobre una variable. Los operadores +=, -=, *= y /= al igual que las asignaciones, requieren como parte izquierda una variable, son asociativos por la derecha y tienen la misma prioridad que el operador de asignación. (5 puntos)

Código fuente (PLX)	Código intermedio (CTD)	Resultado de la ejecución

`int x;`
`x=1;`
`x+=2+3*4;`
`print(x);`
	`x = 1;`
	`t0 = 3 * 4;`
	`t1 = 2 + t0;`
	`x = x + t1;`
	`print x;`
		15

`int x;`
`x=5;`
`int y;`
`y=6;`
`y += 1 + x += 1+2*3;`
`print (x*y);`
	`x = 5;`
	`y = 6;`
	`t0 = 2 * 3;`
	`t1 = 1 + t0;`
	`x = x + t1;`
	`t2 = 1 + x;`
	`y = y + t2;`
	`t3 = x * y;`
	`print t3;`
		228

`int x;`
`int y;`
`x = y+=3 += 2;`
`print (x*y);`
	…
	`error;`
	…
		--

`int x;`
`int y;`
`x=2;`
`y=3;`
`x += y+=y+2 + y+=y+1;`
`print (x*y);`
	`x = 2;`
	`y = 3;`
	`$t0 = y + 2;`
	`$t1 = y + 1;`
	`y = y + $t1;`
	`$t2 = $t0 + y;`
	`y = y + $t2;`
	`x = x + y;`
	`$t3 = x * y;`
	`print $t3;`
		399

`int x;`
`int y;`
`int z;`
`x = 1;`
`y = 2;`
`z = 3;`
`x *= y -= z += x *= y += z;`
`print (x*y*z+x+y+z);`
	`x = 1;`
	`y = 2;`
	`z = 3;`
	`y = y + z;`
	`x = x * y;`
	`z = z + x;`
	`y = y - z;`
	`x = x * y;`
	`$t0 = x * y;`
	`$t1 = $t0 * z;`
	`$t2 = $t1 + x;`
	`$t3 = $t2 + y;`
	`$t4 = $t3 + z;`
	`print $t4;`
		350

* Constantes de tipo ***array*** *unidimensional,* inicialización y uso de variables de tipo ***array*** *unidimensional*. Para ello se usarán expresiones separadas por comas y contenidas entre llaves. Las expresiones pueden ser calculadas en tiempo de ejecución. Para obtener la máxima calificación debe comprobarse que el rango del ***array*** coincide con el asignado. Se acepta que el rango asignado sea menor o igual, pero no que sea mayor. Deben poder combinarse las asignaciones entre variables de tipo *array* y *arrays constantes* definidos mediante llaves, tal y como puede hacerse en la declaración e inicialización de variables tipo *array* en el lenguaje Java. En PLX se acepta este tipo de expresiones también en las sentencias de asignación. (5 puntos).
NOTA: Las pruebas de este apartado se corresponde exactamente con los que aparecen en el examen de la asignatura en febrero de 2015.

Código fuente (PLX)	Código intermedio (CTD)	Resultado de la ejecución

`int a[3];`
`a = {1,2,3};`
`print(a[0]+a[1]*a[2]);`
		7

`int a[3] = {1,2,3};` `print(a[0]+a[1]*a[2]);`		**7**

`int a[2];`
`a = {1,2,3};`
`print(a[0]+a[1]*a[2]);`
	…
	`error;`
	…
		--

`int i;`
`int a[3];`
`for(i=0; i<5; i=i+1) {`
`  a[(i+i)/2] = i*i;`
`  print (a[i]);`
`}`
	…
	`# Comprobacion de rango`
	`  if (t2 < 0) goto L4;`
	`  if (3 < t2) goto L4;`
	`  if (3 == t2) goto L4;`
	`  goto L5;`
	`L4:`
	`  error;`
	`  halt;`
	`L5:`
	….
		0
		1
		4
		runtime error

`int a[3];`
`int b[2];`
`int suma;`
`a = {1,2,3};`
`b = a;`
`int i;`
`for(i=0; i<3; i=i+1) {`
`  suma = suma + b[i];`
`}`
`print(suma);`
	…
	`# las matrices no son compatibles`
	`error;`
	…
		--

`int a[5];`
`a[3] = 4;`
`a = {1,2,3};`
`a[4] = 5;`
`print(a[0]+a[1]*a[2]+a[3]*a[4]);`
		27

`int a[3];`
`int b[3];`
`int c[3];`
`int p;`
`c = b = a = {1,2,3};`
`int i;`
`for(i=0; i<3; i=i+1) {`
`  p = p+ a[i]*b[i]+c[i];`
`}`
`print(p);`
		20

* Introducción del tipo ***array*** multidimensional. Las dimensiones se especifican en la declaración y deben ser constantes. Para obtener la máxima puntuación en este apartado debe implementarse también la comprobación de rangos en tiempo de ejecución. (5 puntos)

Código fuente (PLX)	Código intermedio (CTD)	Resultado de la ejecución
`int a[2][3];` `print( a[0][0] );`		**0**

`int a[2][3];`
`print( a[1][2] );`
`print( a[2][2] );`
		0
		runtime error

`int a[3][2];`
`a[0][0] = 0;`
`a[0][1] = 1;`
`a[1][0] = 10;`
`a[1][1] = 11;`
`a[2][0] = 20;`
`a[2][1] = 21;`
`print(a[2][1]*a[1][1]);`
		231

`int a[3][2];`
`int i;`
`int j;`
`for (i=1; i<=3; i=i+1) {`
`  for(j=1; j<=2; j=j+1) {`
`  a[i-1][j-1] = 2*(i-1)+j;`
`  }`
`}`
`int factorial = 1;`
`for (i=1; i<=3; i=i+1) {`
`  for(j=1; j<=2; j=j+1) {`
`  factorial = factorial * a[i-1][j-1];`
`  }`
`}`
`print(factorial);`
		720

`int a[3][2][5];`
`int i;`
`int j;`
`int k;`
`int conta;`
`conta=0;`
`for(i=0;i<3;i=i+1) {`
`  for(j=0;j<2; j=j+1) {`
`  for(k=0; k<5; k=k+1) {`
`  a[i][j][k] = conta;`
`  conta = conta+1;`
`  }`
`  }`
`}`
`print(a[2][1][3]);`
		28

Asignación de valores a una variable de tipo ***array*** multimensional. Para ello se usarán expresiones separadas por comas y contenidas entre llaves con cualquier grado de anidación. Los valores que se usen pueden ser calculados en tiempo de ejecución. Para obtener la máxima calificación debe comprobarse que el rango del ***array*** coincide con el asignado y que los componentes de cada *subarray* son homogéneos. No se muestra el código generado por motivos de espacio. (6 puntos)

Código fuente (PLX)	Código intermedio (CTD)	Resultado de la ejecución

`int a[2][3];`
`a = { {1,2,3},{4,5,6} };`
`int b[3][2] = { {1,2},{3,4},{5,6} };`
`print( a[1][2] + b[2][1] );`
		12

`int a[3][2] = { {1,2,3},{4,5,6} };` `print( a[0][0] + a[1][1] );`	… `error;`	

`int x = 3;`
`int a[3][3] = { {1+2,1*9,15-x},`
`{x*x,1+2,(x+1)*(x+1)},`
`{9,-x,1}`
`};`
`print( a[1-2+3][x-2] );`
		-3

`int a[2][3];`
`int suma;`
`int i;`
`int j;`
`int signo = 1;`
`a = { {1,2,3},{4,5,6} };`
`for(i=0; i<2; i=i+1) {`
`for(j=0; j<3; j=j+1) {`
`if (signo==1) signo=-1; else signo=1;`
`suma = suma + signo*a[i][j];`
`}`
`}`
`print(suma);`
		3

`int a[2][3][5];`
`int b[3][2][5];`
`int suma;`
`int prod=1;`
`int i;`
`int j;`
`int k;`
`b = { {{1,2,3,4,5},{2,4,3,2,1}},`
`{{3,6,5,4,3},{4,2,3,5,4}},`
`{{5,4,6,3,2},{6,4,3,2,1}} } ;`
`for(i=0; i<3; i=i+1) {`
`for(j=0; j<2; j=j+1) {`
`for(k=0; k<5; k=k+1) {`
`suma = suma + b[i][j][k];`
`prod = prod + b[i][j][k] * b[i][j][k];`
`}`
`}`
`}`
`print(suma);`
`print(prod);`
		102
		411

`int a[2][3][5];`
`int b[3][2][5] = { {{1,2,3,4,5},{2,4,3,2,1}},`
`{{3,6,5,4,3},{4,2,3,5,4}},`
`{{5,4,6,3,2},{6,4,3,2,1}}`
`} ;`
`int suma;`
`int prod=1;`
`int i;`
`int j;`
`int k;`
`for(i=0; i<3; i=i+1) {`
`for(j=0; j<2; j=j+1) {`
`for(k=0; k<5; k=k+1) {`
`suma = suma + b[i][j][k];`
`prod = prod + b[i][j][k] * b[i][j][k];`
`}`
`}`
`}`
`print(suma);`
`print(prod);`
		102
		411

* Asignación de *arrays* con variables de tipo *array*. En este caso deben combinarse las asignaciones entre variables de tipo *array* y *arrays constantes* definidos mediante llaves, tal y como puede hacerse en la declaración e inicialización de variables tipo *array* en el lenguaje Java. En PLX se acepta este tipo de expresiones también en las sentencias de asignación. Ver ejemplos. No se muestra el código *ctd* generado por motivos de espacio. (4 puntos)

Código fuente (PLX)	Código intermedio (CTD)	Resultado de la ejecución

`int a[3];`
`int b[3];`
`int suma;`
`int i;`
`a = {1,2,3};`
`b = a;`
`for(i=0; i<3; i=i+1) {`
`  suma = suma + b[i];`
`}`
`print(suma);`
		6

`int a[3];`
`int b[3];`
`int c[2][3];`
`int suma;`
`int i;`
`int j;`
`a = {1,2,3};`
`b = {4,5,6};`
`c = {a,b};`
`int signo;`
`for(i=0; i<2; i=i+1) {`
`  for(j=0; j<3; j=j+1) {`
`  if (signo==1) signo = -1; else signo = 1;`
`  suma = suma + signo * c[i][j];`
`  }`
`}`
`print(suma);`
		-3

`int a[3] = {1,2,3};`
`int b[3] = {4,5,6};`
`int c[2][3] = {a,b};`
`int d[2][3] = c;`
`int suma;`
`int i;`
`int j;`
`int signo;`
`for(i=0; i<2; i=i+1) {`
`  for(j=0; j<3; j=j+1) {`
`  if (signo==1) signo = -1; else signo = 1;`
`  suma = suma + signo*d[i][j];`
`  }`
`}`
`print(suma);`
		-3

`int x1[5] = {1,2,3,4,5};`
`int x2[5] = {6,7,8,9,10};`
`int x3[5] = {11,12,13,14,15};`
`int a[3][5] = {x1,x2,x3};`
`int b[3][5] = {x3,x2,{1,2,3,4,5}} ;`
`int c[2][3][5] = {a,b};`
`int prod;`
`int i;`
`int j;`
`int k;`
`for(i=0; i<2; i=i+1) {`
`  for(j=0; j<3; j=j+1) {`
`  for(k=0; k<5; k=k+1) {`
`  prod = prod + i*j*k * c[i][j][k] ;`
`  }`
`  }`
`}`
`print(prod);`
		170

EL CÓDIGO OBJETO (Código de tres direcciones):

El código objeto es a su vez una extensión del código intermedio utilizado en la practica principal de la asignatura. Se añaden algunas instrucciones necesarias para generar el código requerido por el lenguaje PLX. Todas las variables del código intermedio se considera que están previamente definidas y que su valor inicial es 0.

El conjunto de instrucciones del código ensamblador, y su semántica son las siguientes:

Instrucción	Acción
`x = a ;`	Asigna el valor de a en la variable x
`x = a + b ;`	Suma los valores de a y b, y el resultado lo asigna a la variable x
`x = a - b ;`	Resta los valores de a y b, y el resultado lo asigna a la variable x
`x = a * b ;`	Multiplica los valores de a y b, y el resultado lo asigna a la variable x
`x = a / b ;`	Divide (div. entera) los valores de a y b , y el resultado lo asigna a la variable x
`x = y[a] ;`	Obtiene el a-ésimo valor del array y, asignando el contenido en x
`x[a] = b ;`	Coloca el valor b en la a-ésima posición del array x
`goto l ;`	Salto incondicional a la posición marcada con la sentencia "`label l`"
`if (a == b) goto l ;`	Salta a la posición marcada con la sentencia "`label l`", si y solo si el valor de a es igual que el valor de b
`if (a < b) goto l ;`	Salta a la posición marcada con la sentencia "`label l`", si y solo si el valor de a es estrictamente menor que el valor de b.
`l:`	Indica una posición de salto.
`label l ;`	Indica una posición de salto. Es otra forma sintáctica equivalente a la anterior.
`print a ;`	Imprime el valor de a
`error ;`	Indica una situación de error, pero no detiene la ejecución.
`halt ;`	Detiene la ejecución. Si no aparece esta instrucción la ejecución se detiene cuando se alcanza la última instrucción de la lista.
`# ….`	Cualquier línea que comience con un # se considera un comentario.

En donde `a,b` representan tanto variables como constantes enteras o reales, `x,y` representan siempre una variable y `l` representa una etiqueta de salto.

IMPLEMENTACIÓN DE LA PRÁCTICA:

Se proporciona una solución compilada del ejercicio (versiones para Linux, Windows y Mac). Esto puede servir de ayuda para comprobar los casos de prueba y las instrucciones intermedio,. No es necesario que el código generado sea idéntico al que se propone como ejemplo (que de hecho no es óptimo), basta con que sea equivalente, es decir que dé los mismos resultados al ejecutar los casos de prueba. Para compilar y ejecutar un programa en lenguaje PLX, pueden utilizarse las instrucciones

	Linux
Compilación	`java PLXC prog.plx prog.ctd`
Ejecución	`./ctd prog.ctd`

En donde `prog.plx` contiene el código fuente en PLX, `prog.ctd` es un fichero de texto que contiene el código intermedio válido según las reglas gramaticales de este lenguaje. El programa `plxc` es un *script* del *shell* del sistema operativo que llama a (`java PLXC`), que es el programa que se pide construir en este ejercicio. El programa `ctd` es un interprete del código intermedio. Asimismo, para mayor comodidad se proporciona otro *script del shell* denominado `plx` que compila y ejecuta en un solo paso, y al que se pasa el nombre del fichero sin extensión.

	Linux
Compilación + Ejecución	`./plx prog`

NOTAS IMPORTANTES:

1. Toda práctica debe contener al menos tres ficheros denominados "`PLXC.java`", "`PLXC.flex`" y "`PLXC.cup`", correspondientes respectivamente al programa principal y a las especificaciones en JFlex y Cup. Para realizar la compilación se utilizarán las siguientes instrucciones:

```
cup PLXC.cup
jflex PLXC.flex
javac *.java
```

   y para compilar y ejecutar el programa en PLX

```
java PLXC prog.plx prog.ctd
./ctd prog.ctd
```

2. Puede ocurrir que al descargar los ficheros y descomprimirlos en Linux se haya perdido el carácter de fichero ejecutable. Para poder ejecutarlos debe modificar los permisos:

```
chmod +x plx plxc ctd
chmod +x plx-linux plxc-linux ctd-linux
```

3. El programa `ctd`, interprete del código intermedio, tiene una opción –v para generar trazas que pueden ayudar en la depuración de errores:

```
./ctd –v prog.ctd
```

4. Los ejemplos que se proponen como casos de prueba no definen exhaustivamente el lenguaje. Para implementar esta práctica es necesario generar otros casos de prueba de manera que se garantice un funcionamiento en todos los casos posibles, y no solo en este limitado banco de pruebas.

5. En todas las pruebas en donde el código **plx** produce un "*error*", para comprobar que el compilador realmente detecta el error, se probará también que el código corregido compila adecuadamente, y si no es así la prueba no se considerará correcta.