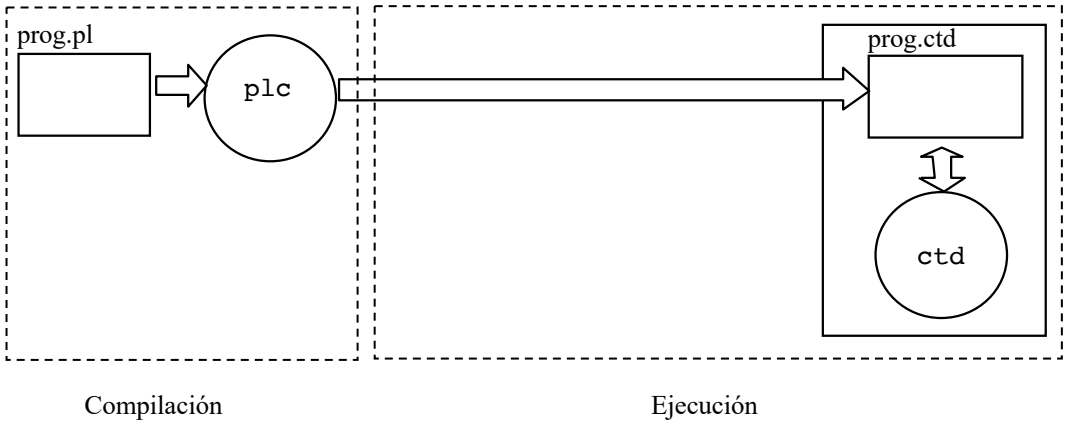
Practica Principal de Procesadores de Lenguajes

Esta practica consiste en la implementación mediante JFlex y Cup de un compilador de un pequeño lenguaje de programación, similar a C, denominado lenguaje PL. Para ser mas exactos, la práctica solo requiere la traducción del lenguaje PL a un lenguaje intermedio definido mediante instrucciones de tres direcciones que finalmente se interpretará mediante otro programa que simula su ejecución.

El esquema de compilación de Java es el siguiente:



Es decir, el código fuente de `Programa.pl` se traduce a código intermedio generando el fichero `Programa.ctd` . Este código de tres direcciones es la entrada a otro programa, denominado CTD, que ejecuta una a una las instrucciones.

SE PIDE: implementar con Java, JFlex y Cup el compilador del lenguaje fuente PL al código intermedio. Para ello, será necesario (al menos) implementar los ficheros `PLC.java`, `PLC.flex` y `PLC.cup`, que una vez compilados darán lugar a varias clases Java, entre ellas PLC.class. Es decir, construir el compilador que antes hemos denominado `plc`, (equivalente a "`java PLC`"), según las instrucciones indicadas a continuación.

En páginas siguientes se describen los lenguajes fuente (lenguaje PL) y objeto (lenguaje CTD).

EL CÓDIGO FUENTE (Lenguaje PL):

El lenguaje fuente tiene una sintaxis similar a C o JAVA y consiste en una lista de sentencias. El lenguaje no requiere la declaración de variables, ya que considera que todas las variables, y considera que todas las variables están definidas y tienen asignado un valor inicial 0. El lenguaje puede contener expresiones aritméticas simples (suma, resta, multiplicación y división); expresiones de asignación; y expresiones relacionales (menor, mayor, igual, etc.); así como sentencias de control ***if, if-else, while, do-while*** y ***for*** y una sentencia ***print*** de salida. Por ejemplo, el siguiente es un programa válido en PL:

prog.pl

```
i = 1;
x = 1+2*3;
factorial = 1;
while (i<x) {
     i = i+1;
     factorial = factorial * i;
}
if (factorial>=0)
     print(factorial);
else
     print(-1);
```

La siguiente gramática corresponde al lenguaje completo:

Lista_de_sent	→	*Sentencia*
	|	*Lista_de_sent Sentencia*
Sentencia	→	*Expresion* **;**
	|	**if (** *Condicion* **)** *Sentencia*
	|	**if (** *Condicion* **)** *Sentencia* **else** *Sentencia*
	|	**while (** *Condicion* **)** *Sentencia*
	|	**do** *Sentencia* **while (** *Condicion* **) ;**
	|	**for (** *Expresion* **;** *Condicion* **;** *Expresion* **)** *Sentencia*
	|	**print (** *Expresion* **) ;**
	|	**{** *Lista_de_sent* **}**
Expresion	→	*Expresion* + *Expresion*
	|	*Expresion* - *Expresion*
	|	*Expresion* * *Expresion*
	|	*Expresion* / *Expresion*
	|	- *Expresion*
	|	**(** *Expresion* **)**
	|	*IDENT* = *Expresion*
	|	*IDENT*
	|	*ENTERO*
Condicion	→	*Expresion* **==** *Expresion*
	|	*Expresion* **!=** *Expresion*
	|	*Expresion* **<** *Expresion*
	|	*Expresion* **<=** *Expresion*
	|	*Expresion* **>** *Expresion*
	|	*Expresion* **=>** *Expresion*
	|	**!** *Condcion*
	|	*Condicion* **&&** *Condicion*
	|	*Condicion* **||** *Condicion*
	|	**(** *Condicion* **)**

Desde el punto de vista léxico, las reglas para componer los identificadores, las constantes numéricas, y demás elementos del lenguaje, son las mismas que en el lenguaje C o JAVA.

EL CÓDIGO OBJETO (Código de tres direcciones):

El código objeto esta formado por una secuencia de sentencias de código intermedio que se obtienen a partir de un conjunto reducido de instrucciones de tres direcciones, que operan sobre constantes de números enteros o variables enteras. 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
`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
`halt ;`	Detiene la ejecución. Si no aparece esta instrucción la ejecución se detiene cuando se alcanza la última instrucción de la lista.

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

Desde un punto de vista meramente sintáctico, el código intermedio se representa mediante un fichero de texto en el cual cada sentencia ocupa exactamente una línea. Todas las instrucciones de código intermedio terminan en punto y coma. Los nombres de las variables comienzan siempre por una letra (mayúscula o minúscula), seguida de una o mas letras o números. Estas mismas reglas se usan para los nombres de las etiquetas. Se excluyen como nombres validos de variables y etiquetas las palabras reservadas del lenguaje intermedio. En los ejemplos siguientes se muestra el código fuente en PL, el correspondiente código intermedio en formato CTD y el resultado de la ejecución del mismo:

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

`f=1;`
`for (i=1; i<4; i=i+1) {`
`  f = f*i;`
`}`
`print (f);`
	`  f = 1;`
	`  i = 1;`
	`L0:`
	`  if (i < 4) goto L1;`
	`  goto L2;`
	`L3:`
	`  t0 = i + 1;`
	`  i = t0;`
	`  goto L0;`
	`L1:`
	`  t1 = f * i;`
	`  f = t1;`
	`  goto L3;`
	`L2:`
	`  print f;`
		6

`equis=64;`
`while (equis>1) {`
`  print (equis=equis/2);`
`}`
	`equis = 64;`
	`label L0;`
	`if (1 < equis) goto L2;`
	`goto L3;`
	`label L2;`
	`t0 = equis / 2;`
	`equis = t0;`
	`print equis;`
	`goto L0;`
	`label L3;`
	`halt;`
		32
		16
		8
		4
		2
		1

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 PL, deben utilizarse las instrucciones

	Linux
Compilación	`./plc prog.pl prog.ctd`
Ejecución	`./ctd prog.ctd`

En donde `prog.pl` contiene el código fuente en PL, `prog.ctd` es un fichero de texto que contiene el código intermedio válido según las reglas gramaticales de este lenguaje. El programa `plc` es un *script* del *shell* del sistema operativo que llama a (`java PLC`), que es el programa que se pide construir en este ejercicio. El programa `ctd` es un intérprete del código intermedio.

EVALUACIÓN DE ESTE EJERCICIO:

Para la corrección de este ejercicio, se tendrán en cuenta resultados parciales según el número de casos de prueba que supere el compilador, de acuerdo a los siguientes criterios:

a) Una única sentencia ***print*** que contenga expresiones con números enteros. Se deben implementar los operadores aritméticos +, - , *, /, el operador menos unario, y los paréntesis. (6 puntos)
b) Programas con varias sentencias de asignación y sentencias ***print***, incluyendo bloques de sentencias compuestas. (2 puntos)
c) Condiciones basadas en expresiones relacionales y sentencias ***if***. Se deben implementar los seis operadores relacionales <=, <, ==, !=, >, >= (4 puntos)
d) Condiciones compuestas, basadas en expresiones lógicas de conjunción, disyunción y negación con evaluación en cortocircuito. (4 puntos)
e) Sentencias ***if-else***. La parte ***else*** es opcional, al igual que en C o Java. (4 puntos)
f) Sentencias ***while*** (2 puntos)
g) Sentencias ***do-while*** (2 puntos)
h) Sentencias ***for***. (2 puntos)
i) Combinación de diverso tipo de sentencias. (4 puntos)

Se proporcionan ejemplos de casos de prueba para cada apartado, (excepto para el ultimo).

NOTAS IMPORTANTES:

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

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

2. Los ejemplos que se proponen como casos de prueba no definen exhaustivamente el lenguaje. Para implementar esta practica 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.