Procesamiento de Lenguajes (PL)
Curso 2014/2015

Práctica 4: traductor ascendente usando `bison/flex`

Fecha y método de entrega

La práctica debe realizarse de forma individual, como las demás prácticas de la asignatura, y debe entregarse a través del servidor de prácticas del DLSI **antes de las 23:59 del 24 de abril de 2015**.

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.

Descripción de la práctica

La práctica consiste en implementar un traductor ascendente utilizando las herramientas `bison` y `flex`, versiones modernas de `yacc` y `lex`, para el mismo proceso de traducción de la práctica 3.

Gramática

La gramática que debe utilizarse como base para diseñar el traductor es la siguiente:

S	$\longrightarrow$	**program id pyc** *Vsp Bloque*
Vsp	$\longrightarrow$	*Vsp Unsp*
Vsp	$\longrightarrow$	*Unsp*
Unsp	$\longrightarrow$	**function id dosp** *Tipo* **pyc** *Vsp Bloque* **pyc**
Unsp	$\longrightarrow$	**var** *LV*
LV	$\longrightarrow$	*LV V*
LV	$\longrightarrow$	*V*
V	$\longrightarrow$	**Lid dosp** *Tipo* **pyc**
Lid	$\longrightarrow$	*Lid* **coma id**
Lid	$\longrightarrow$	**id**
Tipo	$\longrightarrow$	**integer**
Tipo	$\longrightarrow$	**real**
Bloque	$\longrightarrow$	**begin** *SInstr* **end**
SInstr	$\longrightarrow$	*SInstr* **pyc** *Instr*
SInstr	$\longrightarrow$	*Instr*
Instr	$\longrightarrow$	*Bloque*
Instr	$\longrightarrow$	**id asig** *E*
Instr	$\longrightarrow$	**if** *E* **then** *Instr ColaIf*
ColaIf	$\longrightarrow$	**endif**
ColaIf	$\longrightarrow$	**else** *Instr* **endif**
Instr	$\longrightarrow$	**while** *E* **do** *Instr*
Instr	$\longrightarrow$	**writeln pari** *E* **pard**
E	$\longrightarrow$	*Expr* **relop** *Expr*
E	$\longrightarrow$	*Expr*
Expr	$\longrightarrow$	*Expr* **addop** *Term*
Expr	$\longrightarrow$	*Term*
Term	$\longrightarrow$	*Term* **mulop** *Factor*
Term	$\longrightarrow$	*Factor*
Factor	$\longrightarrow$	**id**
Factor	$\longrightarrow$	**nentero**
Factor	$\longrightarrow$	**nreal**
Factor	$\longrightarrow$	**pari** *Expr* **pard**

Observa que, para facilitar el diseño del traductor (especialmente en lo relacionado con los atributos heredados y su implementación en traductores ascendentes), hay muchos no terminales que presentan recursividad por la izquierda (que **no** debes eliminar), pero sin embargo se ha eliminado el factor común por la izquierda en la regla del `if` para evitar conflictos en la construcción de las tablas con las acciones semánticas intermedias.

Mensajes de error

Los errores léxicos y semánticos deben generar exactamente el mismo mensaje que en la práctica 3, con el mismo formato. En el caso de los mensajes de error sintáctico, el mensaje de error debe ser simplemente:

```
Error sintactico (fila,columna): en 'lexema'
```

Esto se debe a que las tablas generadas por `bison` son difíciles de interpretar y por tanto, por simplificar, no es necesario mostrar los tokens que se esperaban en el lugar del token incorrecto.

Notas técnicas

1. Los programas `yacc` (`bison`) y `lex` (`flex`), a partir de una especificación léxica y un ETDS, generan un traductor ascendente basado en un analizador sintáctico LALR(1), escrito en C/C++. Junto con este enunciado se publicará una hoja técnica acerca de estas herramientas, con un ejemplo práctico.

2. Como en las prácticas anteriores, se publicará un autocorrector para facilitar la tarea de depurar la práctica.

3. La práctica debe estar contenida en estos ficheros (como en el ejemplo de la hoja técnica): `comun.h`, `plp4.l` y `plp4.y`

4. El proceso de traducción es el mismo que en la práctica 3, para lo que es necesario utilizar una tabla de símbolos (con básicamente dos funciones, `anyadir` y `buscar`)

5. En el proceso de traducción es necesario utilizar atributos heredados para colocar el prefijo adecuado a las variables al traducirlas (dicha traducción debe guardarse en la tabla de símbolos, como en la práctica 3), pero no hacen falta más atributos heredados. La gramática se ha diseñado para que sea fácil utilizar dichos atributos heredados, aunque hay que usar algunos marcadores.

6. Para comprobar que después de un programa correcto no aparecen más tokens en el fichero, es decir, que el siguiente token es el del final del fichero, es necesario hacer una comprobación en la acción semántica situada al final de la regla:

$$S \longrightarrow \textbf{program id pyc } \textit{Vsp Bloque}$$

   En dicha acción semántica se debe llamar directamente al analizador léxico:

```
int token = yylex();
```

   Si el valor de `token` es 0, el fichero termina correctamente. Si es cualquier otro valor, hay que generar un error sintáctico con la llamada "`yyerror("")`"