PRÁCTICAS DE PROCESADORES DEL LENGUAJE

CURSO 2008/2009

PRÁCTICA 2: ANALIZADOR SINTÁCTICO Y TABLA DE SÍMBOLOS.

Objetivo de la Práctica

- Esta práctica tiene como primer objetivo la codificación de un analizador sintáctico mediante el uso del lenguaje de programación C y la herramienta de ayuda a la generación de analizadores sintácticos ***bison***.
- Este analizador será utilizado por el compilador del lenguaje ***ALFA*** que va a desarrollarse durante el curso.
- El segundo objetivo es la codificación de una tabla de símbolos. El alumno conoce el tipo de datos llamado tabla hash, los algoritmos para su gestión de la asignatura *Metodología y tecnología de la programación II (MTPII)* y su uso como tabla de símbolos de la parte de teoría de la asignatura *Procesadores de lenguaje*.

Desarrollo de la Práctica

Codificación de una Especificación para *bison*

- El alumno deberá escribir un fichero con nombre ***alfa.y*** que sea una especificación correcta para la entrada de la herramienta *bison*. Se recomienda al alumno que se familiarice primero con *bison* mediante las explicaciones que su profesor de prácticas realizará en su laboratorio.

Resolución de Conflictos

- Cuando se compile el fichero ***alfa.y*** pueden aparecer mensajes de conflictos. Estos conflictos se refieren a los que encuentra el autómata a pila generado por la herramienta ***bison***. Para obtener información acerca de ellos, se debe compilar el fichero ***alfa.y*** de la siguiente manera:

  bison -d -y -v alfa.y

  con el flag ***-v*** se genera el fichero ***y.output*** que contiene la descripción completa del autómata y de los conflictos (si existieran).

- El alumno debe asegurarse de eliminar los conflictos siguiendo las indicaciones de su profesor de prácticas.

Modificación de la Especificación para *flex*

- El alumno deberá modificar el fichero ***alfa.l*** de la primera práctica para poder ser enlazado con el fichero ***alfa.y***. Para ello, utilizará como guía las explicaciones que su profesor de prácticas realizará en su laboratorio.

Codificación de un Programa de Prueba

- El alumno deberá escribir (en C) un programa de prueba del analizador sintáctico generado con ***bison***, con los siguientes ***requisitos***:
  - Nombre del programa fuente: ***prueba_sintactico.c***
  - Al ejecutable correspondiente, se le invocará de la siguiente manera:

    prueba_sintactico [<nombre fichero entrada> [<nombre fichero salida>]]

- Es decir, el programa se puede ejecutar:
  - Con 0 argumentos, entonces se utilizan la entrada y la salida estándares.
  - Con 1 argumentos, entonces se utiliza la salida estándar y el argumento como nombre del fichero de entrada.
  - Con 2 argumentos, entonces el primero se interpreta como el nombre del fichero de entrada y el segundo como el nombre del fichero de salida.
- La estructura de los ficheros de entrada/salida se explican a continuación.

Descripción del Fichero de Entrada

- El fichero de entrada contiene un programa escrito en lenguaje ***ALFA*** (no necesariamente correcto).

Descripción del Fichero de Salida

- El fichero de salida se compone de un conjunto de líneas de dos tipos. En concreto:
  - Una línea por cada **TOKEN** deplazado (reconocido) por el analizador léxico.
  - Una línea por cada **PRODUCCIÓN** reducida en el proceso de análisis sintáctico.
- Cada línea correspondiente a un **TOKEN** desplazado debe contener la siguiente información y formato:

  D: <token>

  donde **<token>** es el fragmento de entrada reconocido por el analizador léxico y **D:** y **<token>** van separados por un tabulador.
- Cada línea correspondiente a una regla reducida tendrá el formato siguiente:

  R<nº regla>: <regla>

  donde **<nº regla>** es el número que identifica la regla que se reduce en la gramática de **ALFA** especificada en el enunciado de la primera práctica, y **<regla>** es el texto de la regla reducida según aparece en la gramática. **R<nº regla>** y **<regla>** van separados por un tabulador. Los elementos que forman **<regla>** irán separados por espacios.

Salidas especiales

Las líneas **R<nº regla>: <regla>** que debiesen contener elementos del tipo **<identificador>**, **<constante_entera>** o **<constante _real>** se describen a continuación:

- Cada aparición del símbolo no terminal **<identificador>** irá seguida de su correspondiente lexema atendiendo al siguiente formato:

  <identificador (lexema)>

  donde **lexema** es el lexema correspondiente al identificador.
- Cada aparición del símbolo no terminal **<constante_entera>** irá seguida de su correspondiente valor númerico atendiendo al siguiente formato:

  <constante_entera (valor)>

  donde **valor** es el valor número correspondiente a la constante entera.
- Cada aparición del símbolo no terminal **<constante_real>** irá seguida de su correspondiente valor número atendiendo al siguiente formato:

  <constante_entera (valor)>

  donde **valor** es el valor número correspondiente a la constante real, en el formato dado por el símbolo **%f** de la rutina fprintf del lenguaje de programación C.

- Estas salidas especiales se implementarán utilizando los atributos semánticos definidos por el alumno.
- en el fichero de entrada para bison"proporcionados por **bison**.

Gestión de errores

- Cuando se produzca un error sintáctico el analizador creado mediante **bison** deberá de imprimir una línea por la salida estándar de error con el siguiente formato:

  ERROR SINTACTICO:<nº linea>:<nº carácter>

  donde **<nº linea>** es la línea en el fichero de entrada donde aparece el último **TOKEN** reconocido por el

analizador léxico y **<nº carácter>** es el carácter en la correspondiente línea donde empieza el último **TOKEN** reconocido por el analizador léxico.

- Cuando se produzca un error léxico, el analizador léxico implementado en la práctica 1 deberá imprimir la siguiente línea por la salida estándar de error:

  ERROR LEXICO:<nº linea>:<nº carácter>

  donde **<nº linea>** es la línea en el fichero de entrada donde aparece el **TOKEN** erróneo y **<nº carácter>** es el carácter en la correspondiente línea donde empieza el **TOKEN** erróneo.

- En el caso de que se imprima un mensaje de error léxico, no deberá de imprimirse otro mensaje de error sintáctico.

Ejemplo

```
user@foobar:~$ cat 1.alf
main {
	int x, resultado;
	scanf x;
	resultado = x * x + 1.5 * 2;
	printf resultado;
}
user@foobar:~$ ./prueba_sintactico 1.alf 1.sin
user@foobar:~$ cat 1.sin
D:	main
D:	{
D:	int
R10:	<tipo> ::= int
D:	x
R9:	<clase_escalar> ::= <tipo>
R5:	<clase> ::= <clase_escalar>
R106:	<identificador (x)> ::= TOK_IDENTIFICADOR
D:	,
D:	resultado
R106:	<identificador (resultado)> ::= TOK_IDENTIFICADOR
D:	;
R18:	<identificadores> ::= <identificador (resultado)>
R19:	<identificadores> ::= <identificador (x)> , <identificadores>
R4:	<declaracion> ::= <clase> <identificadores> ;
D:	scanf
R2:	<declaraciones> ::= <declaracion>
R21:	<funciones> ::=
D:	x
R106:	<identificador (x)> ::= TOK_IDENTIFICADOR
D:	;
R54:	<lectura> ::= scanf <identificador (x)>
R35:	<sentencia_simple> ::= <lectura>
```

R32: <sentencia> ::= <sentencia_simple> ;

D: resultado

R106: <identificador (resultado)> ::= TOK_IDENTIFICADOR

D: =

D: x

R106: <identificador (x)> ::= TOK_IDENTIFICADOR

D: *

R80: <exp> ::= <identificador (x)>

D: x

R106: <identificador (x)> ::= TOK_IDENTIFICADOR

D: +

R80: <exp> ::= <identificador (x)>

R75: <exp> ::= <exp> * <exp>

D: 1.5

R105: <constante_real (1.500000)> ::= TOK_CONSTANTE_REAL

R101: <constante> ::= <constante_real (1.500000)>

R81: <exp> ::= <constante>

D: *

D: 2

R104: <constante_entera (2)> ::= TOK_CONSTANTE_ENTERA

R100: <constante> ::= <constante_entera (2)>

R81: <exp> ::= <constante>

D: ;

R75: <exp> ::= <exp> * <exp>

R72: <exp> ::= <exp> + <exp>

R43: <asignacion> ::= <identificador (resultado)> = <exp>

R34: <sentencia_simple> ::= <asignacion>

R32: <sentencia> ::= <sentencia_simple> ;

D: printf

D: resultado

R106: <identificador (resultado)> ::= TOK_IDENTIFICADOR

D: ;

R80: <exp> ::= <identificador (resultado)>

R56: <escritura> ::= printf <exp>

R36: <sentencia_simple> ::= <escritura>

R32: <sentencia> ::= <sentencia_simple> ;

D: }

R30: <sentencias> ::= <sentencia>

R31: <sentencias> ::= <sentencia> <sentencias>

R31: <sentencias> ::= <sentencia> <sentencias>

R1: <programa> ::= main { <declaraciones> <funciones> <sentencias> }

user@foobar:~$

Implementación de una tabla de símbolos

- El alumno debe escribir los ficheros y módulos C que considere necesarios para la definición e implementación de la tabla de símbolos. Para todas sus decisiones de diseño utilizará como referencia las indicaciones que conoce de la parte de teoría y las explicaciones que su profesor de prácticas realizará en su laboratorio.

Autocorrección de la práctica

- En la página web del laboratorio de Procesadores de Lenguaje hay disponible un enlace para que el alumno se descargue el fichero comprimido **autocorreccion_sintactico.zip**. Este fichero contiene 10 programas escritos en **ALFA** y sus correspondientes 10 ficheros de salida que debe generar el programa **prueba_sintactico** desarrollado en esta segunda práctica. El alumno debe comparar las salidas que obtiene su programa prueba_sintactico con las salidas proporcionadas en **autocorreccion_sintactico.zip**. Para realizar la comparación en Linux se puede utilizar el comando **diff**, que compara el contenido de dos ficheros.
- Observe que la salida contiene dos tipos de información:
    - La secuencia de acciones de reducción y desplazamiento
    - En las líneas correspondientes a reducciones, el texto de las producciones reducidas

  Sólo es relevante para la corrección de su práctica la primera información, ya que las diferencias en el texto que haya escrito para representar las producciones no hacen que su práctica sea incorrecta. La detallada descripción del enunciado es para facilitar la comparación automática mediante diff. Por las mismas razones tenga en cuenta que es posible que el texto de las producciones de alguno de los casos de prueba contenga alguna errata.

Sugerencia

- Se recomienda al alumno que escriba un fichero compatible con la herramienta **make** y con el nombre **Makefile** que para el objetivo **all** genere el ejecutable de nombre **prueba_sintactico**.