El generador de analizadores sintácticos *Yacc*
IV

Teoría de autómatas y lenguajes formales
Alma María Pisabarro Marrón (alma@infor.uva.es)
Universidad de Valladolid

1. Tipos de datos de los valores semánticos

Los tokens y los auxiliares pueden tener valores semánticos asociados. Por defecto estos valores son enteros aunque Yacc soporta otros tipos de valores incluyendo estructuras y uniones de varios tipos. El usuario declara que tipo de valor está asociado a cada símbolo terminal y a cada no terminal y Yacc sustituye $$, $1, ... por su valor semántico correspondiente (véase el apartado de acciones de Yacc)

Para especificar el tipo de los símbolos se utilizan las siguientes directivas en la sección de definiciones de Yacc:

- **`%token`** para los símbolos terminales
- **`%type`** para los símbolos auxiliares

`YYSTYPE` es el tipo de datos de la pila semántica. Si se desea cambiar el tipo de datos por defecto (enteros) a otro cualquiera podemos incluir la siguiente línea en la sección de definiciones de Yacc:

```
%{ #define YYSTYPE tipo %}
```

Si en la pila se desean varios tipos de datos distintos se puede utilizar la directiva **`%union`** en la sección de definiciones de Yacc:

```
%union      {
            cuerpo de la unión...
            }
```

Esto genera en el código fuente c (en y.tab.c) la siguiente estructura:

```
typedef union          {
                       cuerpo de la unión...
                       } YYSTYPE
```

Ejemplo1: En la siguiente sección de definiciones de un programa Yacc se utiliza la directiva `%token` para indicar que el valor semántico asociado al símbolo terminal `NUMERO` es de tipo `double` y la directiva `%type` para indicar que el valor semántico asociado al símbolo auxiliar `exp` es también de tipo `double`

```
%union {
       double     valor;
       int        indice;
       }
%token <valor>    NUMERO
%type  <valor>    exp
```

2. Valores semánticos de los tokens

Cada nodo del árbol de derivación que el analizador sintáctico construye, o más bien recorre, para una cadena de entrada concreta puede tener asociado un valor denominado valor semántico. El valor de un nodo puede depender de los valores de otros, y las reglas que permiten calcular unos en función de otros, llamadas como ya se ha visto reglas semánticas, se expresan mediante acciones en C asociadas a cada regla sintáctica. Estas acciones pueden utilizar, a parte de valores constantes, los valores de los símbolos auxiliares o terminales que intervengan en dicha regla sintáctica. A estos

efectos, como ya se ha visto, el valor asociado al antecedente de la regla es la variable `$$`, y el valor asociado al símbolo que ocupa la posición `i` del consecuente es `$i`. Si no especifica otra cosa, el tipo de los valores semánticos es entero.
Para los terminales, que normalmente tienen un valor semántico que solo depende de si mismos, es el analizador léxico el que debe obtener dicho valor y depositarlo en una variable externa al analizador léxico y al analizador sintáctico. En un entorno Lex-Yacc esa variable se llama **`yyval`** o **`yylval`**.

Como ya sabemos cada vez que el analizador sintáctico generado con Yacc necesita un token de la entrada realiza una llamada a la función `yylex()` que le devuelve un número entero que indica el tipo de terminal que aparece en la entrada. Si además el terminal tiene asociado una valor semántico debe asignarse a la variable externa `yylval`. Por ejemplo, para la gramática de expresiones aritméticas, si en la entrada aparece la secuencia de caracteres `'9.87'` `yylex`() devolverá que ha aparecido un token de tipo `NUMERO` con un valor semántico asociado que es el valor del número decimal con el que se corresponde la cadena, en este caso 9.87.

Para que coincidan los números asociados a cada token, (no el valor semántico sino el que define el tipo de token), en el analizador léxico generado con Lex y en el sintáctico generado con Yacc es imprescindible que la compilación se realice siguiendo uno de los dos métodos que ya hemos visto.

El tipo de datos que se pueden devolver en `yylval` es entero por defecto pero puede modificarse como se indicó en el apartado anterior. Si hemos modificado el tipo de datos a una unión debe asignarse el valor al tipo de datos correcto. Por ejemplo si `yylval` es una unión de `double` y de `int`, como se muestra a continuación, podrían realizarse las asignaciones que le siguen en el analizador léxico:

```
%union {
       double     valor;
       int        indice;
       }
...
yylval.valor = 5.8;
yylval.indice = 7;
```

Ejemplo2: El siguiente es un ejemplo de asignación de valores semánticos en un analizador léxico, construido en C sin utilizar Lex, que reconoce dígitos.

```
yylex (){
      extern int yylval;
      int c;
      …
      c = getchar();
      …
      switch (c){
        case '0':
        case '1':
        …
        case '9':  yylval = c-'0';    /*valor semántico asociado al token*/
                   return (DIGITO);  /*tipode token*/
      }
}
```

3. Paso de valores entre Lex y Yacc

Los siguientes ficheros fuente Lex y Yacc generan un programa capaz de concatenar cadenas de texto o de sumar números enteros positivos. Este ejemplo ilustra el paso de valores de diferentes tipos entre Lex y Yacc, así como la asignación de valores a los símbolos no terminales.
Para simplificar no se ha tenido en cuenta la posibilidad de que la cadena pueda superar los 100

caractéres (lo que generaría un error de ejecución con este ejemplo) ni de que los datos numéricos se salgan del rango de los enteros.

Ejemplo3: El siguiente ejemplo muestra los ficheros fuente Lex y Yacc necesarios para obtener un programa que sume números enteros, si la entrada es numérica, o bien concatene cadenas de caracteres si la entrada es una suma de cadenas. En cualquier caso la expresión de entrada debe finalizar con un retorno de carro. En la entrada las cadenas deben ir sin comillas.

Fichero fuente Lex

```
%{
#include "y.tab.h"
%}
%%
[a-zA-Z]+{strcpy(yylval.cadena, yytext);
          return(CAD);};
[0-9]+    {yylval.numero=atoi(yytext);
          return(NUM);};
"+"       {return('+');};
"\n"      {return('\n');};
.         ;
```

Fichero fuente Yacc

```
%union    {
          char cadena[100];
          int numero;
          }
%token <numero> NUM
%token <cadena> CAD

%type <numero> NUM
%type <cadena> CAD
%type <numero> expnum
%type <cadena> expcad

%left '+'

%%
entrada   : expnum '\n'         {printf("Resultado:%d", $1);};
          : expcad '\n'         {printf("Resultado:%s", $1);};
          ;
expnum    : expnum '+' expnum   {$$=$1+$3;}
          | NUM                 {$$=S1;}
          ;
expcad    : expcad '+' expcad   {strcat($1,$3);
                                 strcpy($$,$1);}
          | CAD                 {strcpy($$,S1);}
          ;
%%
main()
          {
          yyparse();
          }
```