1. El programa generado por YACC

Análisis estructural

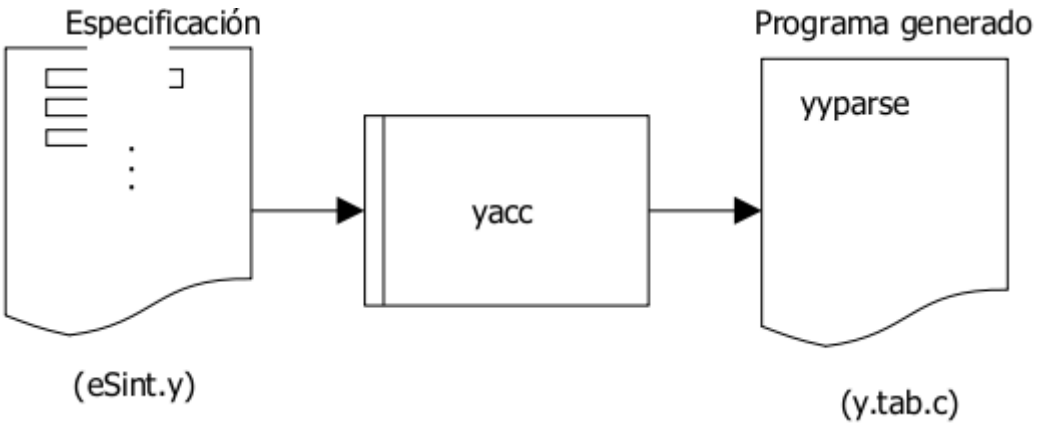
El problema que se pretende resolver mediante el programa generado por Yacc consiste en la realización de un análisis estructural de un texto de entrada.

El programa generado recibe como entrada una secuencia representativa de los componentes elementales de un texto, y comprueba si esa secuencia se ajusta a la estructura definida por una gramática de contexto independiente; los componentes elementales recibidos son los símbolos terminales de la gramática.

Con otras palabras: el programa generado por Yacc analiza sintácticamente un texto; para ello, recibe como entrada la secuencia de piezas sintácticas proporcionada por un analizador lexicográfico encargado de la lectura del texto de entrada. Se trata de un análisis jerárquico que complementa el análisis lineal realizado por un programa generado por Lex.

Entrada y salida para el traductor Yacc

En el siguiente esquema se muestra, de una manera simplificada, lo que constituye la entrada y la salida para el traductor/generador Yacc.



La entrada a Yacc es una especificación sintáctica (estructural) escrita en forma de gramática de contexto independiente (codificada de manera semejante a la notación BNF-No ampliada). Las gramáticas son mecanismos adecuados para la especificación de las características sintácticas de los lenguajes de programación y se representan con una notación textual fácil de procesar; por estos motivos resultan apropiadas para emplearlas como entrada a Yacc para generar analizadores sintácticos.

La salida es un programa escrito en C cuya parte central es una función llamada yyparse que realiza el análisis sintáctico de la secuencia de piezas sintácticas recibidas.

La gramática de entrada a Yacc ha de ser exclusivamente sintáctica: se trata de generar un analizador sintáctico; la descripción de las características lexicográficas no debe de considerarse aquí. Aunque se volverá sobre ello, conviene apuntar ahora que en la notación Yacc no se tiene una facilidad análoga a la notación BNF-Ampliada para escribir gramáticas de manera más compacta.

Aunque en este momento no pueda entenderse el motivo, para conseguir un funcionamiento más eficaz del analizador generado, conviene que la gramática de entrada proporcionada a Yacc sea recursiva por la izquierda; es decir, en caso de que la gramática tenga que ser recursiva, es preferible que lo sea por la izquierda a que lo sea por la derecha.

El funcionamiento del analizador generado se apoya en:

1. Una estructura de datos, la pila de estados, cuyo contenido es un reflejo de las situaciones por las que se va pasando durante el proceso de análisis

2. Unas funciones de transición (las funciones goto y action, implementadas mediante tablas) que son indicativas de la evolución que debe tomar el proceso de análisis conforme se van recibiendo las sucesivas piezas sintácticas.

Llamadas a la función generada por Yacc

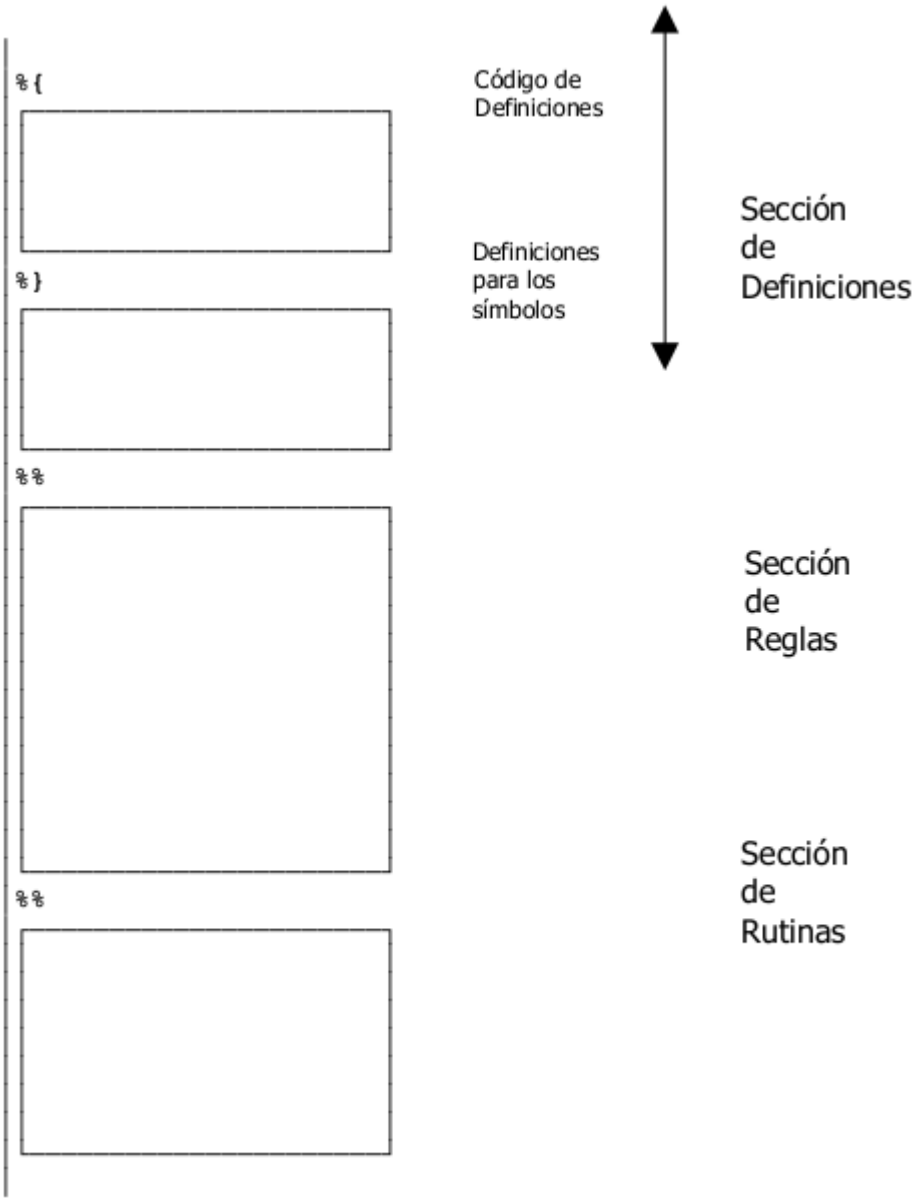
Mediante una única llamada a la función yyparse se desencadena la ejecución del análisis sintáctico de todo el texto de la entrada. Generalmente, desde yyparse se llama reiteradamente a un analizador lexicográfico (que debe ser una función de nombre yylex), y la única llamada a yyparse se produce desde otra función de mayor nivel jerárquico (que suele ser una función main).

La función yyparse devuelve un valor de tipo entero representativo del resultado del análisis sintáctico efectuado.

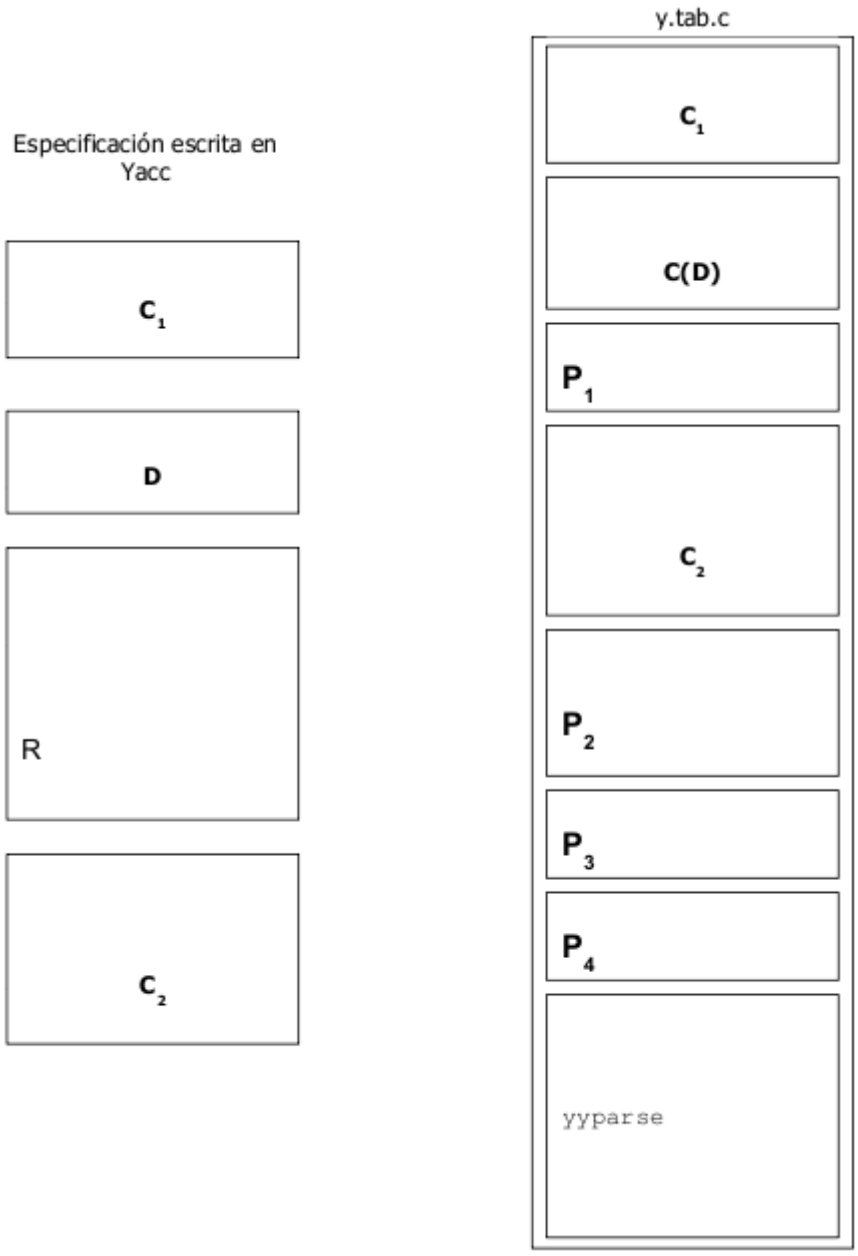
2. Forma general de una especificación YACC

Esquema general de una especificación Yacc

Una especificación Yacc está formada por 3 secciones, tal y como se ilustra en el siguiente esquema; la raya vertical de la izquierda representa el comienzo de las líneas del fichero texto de entrada, es decir la posición de la primera columna de cada línea. Puede observarse que es por completo análoga a la forma de una especificación Lex.



En lo que sigue se explica la correspondencia que hay entre las distintas partes de una especificación de entrada y la estructura del correspondiente fichero de salida. En esas explicaciones se hace mención de la siguiente figura que refleja la relación entre la entrada y la salida del traductor Yacc.



La codificación de una especificación Yacc no es tan rígida como la de una especificación Lex. Podemos decir que la entrada a Yacc se escribe con un formato libre. Sin embargo, la costumbre y el estilo típico de las especificaciones Yacc indican que:

1. El separador de secciones %% y los delimitadores %{ y %} se coloquen en la primera columna de una línea.

2. Los símbolos de la parte izquierda de las producciones se empiecen a escribir a partir de la primera columna de una línea.

Una especificación Yacc ha de tener al menos una producción en la sección de reglas. Las secciones de definiciones y de rutinas pueden estar vacías; en caso de ausencia de la sección de rutinas puede suprimirse el separador %% que marca el final de la sección de reglas. El separador %% entre las secciones de definiciones y de reglas siempre ha de estar presente, aunque no exista la sección de definiciones.

Sección de definiciones. Código de definiciones

Se trata de código escrito en C y delimitado por los símbolos %{ y %}; no es necesario que estos delimitadores se pongan a partir de la primera columna de una línea, pero es la costumbre.

El código C de esta sección suele estar formado por definiciones globales de objetos que se usan en otras partes del código generado. Este código se traslada literalmente al fichero de salida, al principio de código generado.

En el esquema que relaciona la especificación y el fichero generado, este código está representado por C1.

Sección de definiciones. Definiciones de los símbolos

En la sección de definiciones también se incluyen diversas definiciones relativas a los símbolos terminales y no terminales de la gramática sintáctica especificada con posterioridad en la sección de reglas. Cada símbolo terminal se declara poniendo su nombre y un valor numérico asociado a él. Esta asociación se define con objeto de fijar el valor empleado en la comunicación entre los analizadores lexicográfico y sintáctico. En esta declaración se emplea la palabra reservada %token. Por ejemplo, las declaraciones

```
%token nombre1 n1
%token nombre2 n2
```

definen nombre1 y nombre2 como nombres de símbolos terminales de la gramática sintáctica, y asocian el valor n1 al símbolo nombre1 y el valor n2 al símbolo nombre2. Los valores asociados mediante la declaración %token deben de concordar con los valores que realmente devuelve el analizador lexicográfico (valores que se habrán definido para las piezas sintácticas en la especificación de Lex).

El símbolo (no terminal) inicial de la gramática se declara mediante la palabra reservada %start; así la declaración

```
%start nombre
```

indica que se considere nombre como el símbolo inicial de la gramática.

Sección de reglas

Es la parte fundamental de una especificación Yacc; en ella se pone la gramática que define la sintaxis del lenguaje que se pretende analizar con el programa generado.

Las reglas de la gramática se escriben con una notación parecida a la notación BNF-No Ampliada; a continuación se describe la notación de Yacc, con comentarios comparativos con la bien conocida notación BNF.

1. Nombres de los símbolos de la gramática.En la notación de Yacc no se emplean los caracteres < y > para delimitar los nombres de los símbolos no terminales; ya se ha comentado antes cómo se escriben y cómo se distinguen entre sí los símbolos terminales y no terminales.

2. Separación entre las dos partes de una regla. En Yacc la separación entre la parte izquierda y la parte derecha de una regla se indica mediante el carácter dos puntos; en la notación BNF esta separación está representada mediante la secuencia de tres caracteres ::=.

3. Reglas con la misma parte izquierda.
   Para indicar que varias reglas consecutivas tienen la misma parte izquierda, tanto en Yacc como en BNF se emplea el carácter |.

4. Parte derecha que es la palabra vacía.
   En la notación de Yacc la palabra vacía se representa mediante la ausencia de la parte derecha, o sea, para indicar la presencia de la palabra vacía no se pone símbolo alguno (precisamente la ausencia de símbolo es la representación de la palabra vacía). En la notación BNF la palabra vacía suele representarse con la letra griega ε.

5. Separación entre los símbolos de la parte derecha.
   Para indicar la separación entre dos símbolos consecutivos de la parte derecha de una regla, en la notación de Yacc ha de ponerse al menos un espacio en blanco (o un tabulador, o incluso un final de línea). Se puede decir que así ocurre también en la notación BNF.

6. Marca de final de regla.
   En una especificación Yacc, no es preciso indicar explícitamente el punto donde termina una regla (detrás del último símbolo de la parte derecha); no obstante, para favorecer la legibilidad de la gramática, en el estilo habitual de escritura se pone un punto y coma para indicar el final de una regla, o bien un punto y coma detrás de la última regla de una secuencia de reglas consecutivas que tienen la misma parte izquierda.
   En la notación BNF no se marca el final de las reglas.
   Así pues, el carácter punto y coma es un metasímbolo de la notación Yacc.

7. Inexistencia de una notación ampliada.
   La notación BNF tiene dos variantes: Ampliada y No-Ampliada; en la ampliada se usan metasímbolos para simplificar la escritura. En la notación de Yacc no son de uso tales metasímbolos. Se podría resumir

esta carencia de Yacc diciendo que una gramática de entrada a Yacc ha de escribirse en notación no ampliada.

El traductor Yacc, a partir de las reglas de la gramática, obtiene las funciones de transición y de operación relativas a un autómata en el que apoya el algoritmo reconocedor (el analizador sintáctico); estas funciones, implementadas mediante tablas, se trasladan al fichero generado.

La función yyparse, que realiza el análisis del texto de entrada, es un algoritmo único e invariable para todos los analizadores que se generan; desde esta función se consultan las tablas generadas en cada caso. Puede decirse que el analizador sintáctico generado por Yacc está formado por un algoritmo y unas estructuras de datos. El algoritmo es el mismo para todas las gramáticas, lo que cambia según el lenguaje que se analiza son las estructuras de datos.

En el esquema que relaciona la entrada y la salida del traductor Yacc puede apreciarse que el código de la función yyparse ocupa la parte final del fichero generado.

Sección de rutinas

En esta sección se coloca código escrito en C, que se traslada literalmente al fichero generado. Usualmente se ponen aquí rutinas de apoyo para el tratamiento semántico. Son funciones a las que se llama desde las acciones que pueden asociarse a las reglas de la gramática (esta posibilidad de asociación no se estudia aquí).

También se puede aprovechar esta sección para incorporar la función principal main desde la que se produce la llamada al analizador sintáctico generado: la función yyparse; así puede verse en el Ejemplo 1 expuesto en el capítulo inicial.

En la figura que ilustra la relación entre la entrada y la salida del traductor Yacc, el código de esta sección está representado mediante C2.

En esa misma figura las partes de la salida indicadas mediante P1, P2, P3 y P4 no interesan en lo que aquí se expone.

3 Generación de Analizadores Sintácticos

Comunicación entre los analizadores lexicográfico y sintáctico

Cada vez que el analizador sintáctico llama al analizador lexicográfico espera que le devuelva la siguiente pieza sintáctica encontrada en el texto que se analiza.

Cuando estos analizadores han sido generados por Yacc y Lex, se tiene que desde yyparse se realiza una llamada a yylex; la función yylex devuelve una representación numérica de la pieza sintáctica encontrada; por lo tanto, los valores numéricos asociados a las piezas han de ser comunes en ambos analizadores: han de estar asignados los mismos valores en ambas especificaciones, la de Lex y la de Yacc.

A veces también se precisa conocer desde el analizador sintáctico el lexema de la pieza sintáctica recibida.

En el fichero lex.yy.c generado por Lex aparecen declaradas las variables globales yytext e yyleng cuyo contenido ya se ha explicado anteriormente.

Dado que el fichero y.tab.c generado por Yacc se compila (proceso de compilación y montaje) junto con el fichero lex.yy.c, se tiene que esas variables globales también son accesibles desde el código del analizador sintáctico.

Si se realizan tareas de análisis exclusivamente sintácticas, no será preciso consultar los lexemas de las piezas; esa consulta se requiere desde las rutinas del análisis semántico (en el caso de que estuviesen incorporadas a la especificación de entrada a Yacc).

Manera simplificada de asociar valores a las piezas sintácticas

Hasta ahora, se ha considerado una manera de indicar los valores asociados a las piezas sintácticas que requiere la definición explícita de:

1. Los valores que devuelve la función yylex (en la especificación de Lex).
2. Los valores asociados a los símbolos terminales (en la especificación Yacc),

siendo obligada la coincidencia de los nombres y valores en ambas especificaciones.

En este punto se describe otra manera de asociar los valores a las piezas; esta nueva manera es más cómoda (se precisa escribir menos) y más segura (es menos propensa a que se cometan errores en la definición).

En la práctica es esta nueva manera la que conviene emplear siempre; la manera considerada hasta ahora sólo ha servido para favorecer el entendimiento de los conceptos expuestos.

La nueva manera simplificada conlleva cambios en ambas especificaciones.

Clasificación de las piezas sintácticas

Para empezar la descripción de los cambios, se establece una clasificación de las piezas sintácticas. Las piezas sintácticas (símbolos terminales de la gramática sintáctica) se consideran clasificadas en dos grupos: piezas sintácticas anónimas y piezas sintácticas nominales.

Las piezas sintácticas anónimas son las que se corresponden con un lexema que siempre es de longitud 1 (siempre está formado por un único carácter). Pongamos un ejemplo.

El punto y coma (;), la coma (,), el operador aritmético aditivo (+). Las piezas sintácticas nominales son las que se corresponden con lexemas que siempre son o que pueden ser de longitud mayor que 1(el símbolo de asignación :=). El operador de relación menor o igual (<=), los identificadores.

Las piezas anónimas se denomina así porque no se les asigna un nombre; el valor numérico asociado es el valor ordinal (en el alfabeto ASCII) del carácter (único) que constituye su lexema.

Las piezas nominales reciben este calificativo porque se les asigna un nombre. En este caso, la asociación entre nombre y valor ha de indicarse explícitamente (el valor asociado ha de ser mayor que 256 para que no pueda confundirse con el valor) ordinal de ninguno de los caracteres del alfabeto.

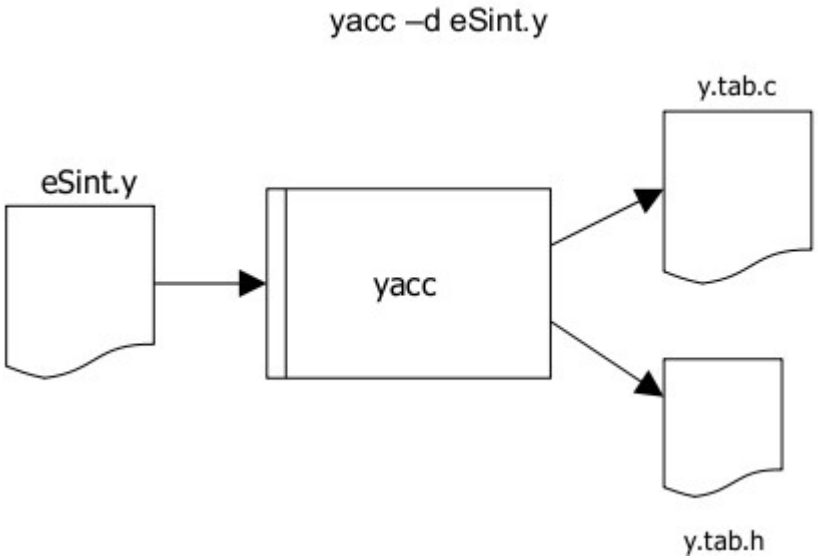
Modificaciones en la especificación de Yacc

En la sección de definiciones se declaran los nombres de las piezas sintácticas nominales mediante la palabra reservada %token. No es necesario declarar los valores numéricos asociados a los nombres de los símbolos terminales. Si no se declaran (y conviene no hacerlo) el traductor Yacc asocia automáticamente valores, empezando por el número 257 y prosiguiendo con valores consecutivos, en el orden en que estén colocados los nombres de los símbolos en la declaración.

En la gramática sintáctica de la sección de reglas un símbolo terminal anónimo se escribe poniendo entre apóstrofos el propio carácter que constituye el lexema.

Modificaciones en la ejecución del comando Yacc

Se ejecuta el comando de llamada al traductor Yacc con la opción –d (de define). Con ella se obtiene como resultado auxiliar un fichero de nombre y.tab.h, tal y como se ilustra en la siguiente figura.



En el fichero y.tab.h se tienen grabadas las definiciones en código C correspondientes a los símbolos terminales nominales incluidos en la sección de definiciones de la especificación traducida. Estas definiciones son las que asocian los nombres a sus valores numéricos.

Modificaciones en la especificación de Lex

Para declarar los valores de las piezas sintácticas nominales en la sección de definiciones es más cómodo y más seguro incluir el contenido del fichero y.tab.h, en vez de poner la secuencia de declaraciones #define equivalente.

Así pues, en la sección de definiciones, en vez de poner:

```
%{
#define   nombre1   n1
#define   nombre2   n2

. . . .   .

. . . .   .
#define   nombreq   nq
%}
```

se pone:

```
%{
#include    “y.tab.h“
%}
```

Valor devuelto en el caso de un carácter incorrecto

En los ejemplos vistos hasta ahora se ha empleado un único valor numérico (especial) representativo de la situación errónea provocada por la presencia de cualquier carácter que no pertenezca al alfabeto del lenguaje analizado.

Este mismo efecto se consigue si lo que se devuelve es el valor ordinal del carácter incorrecto encontrado (cualquiera que sea ese carácter); al proceder de esta manera, se devuelve un valor que, en ningún caso, puede coincidir con los valores que desde el analizador sintáctico se espera como representativo de alguna de las piezas sintácticas del lenguaje. Y se produce un error.

Debe de apreciarse que, al indicar que el analizador lexicográfico actúe de este modo, lo que se está haciendo es transformar en sintáctico un error que se ha encontrado en el nivel del análisis lexicográfico.

Resultados del análisis. Función principal

La función yyparse es de tipo entero; devuelve un valor numérico, según cuál haya sido el resultado del análisis léxico-sintáctico efectuado. El resultado devuelto es:

0 si no se ha encontrado error alguno en el texto analizado,

1 si se ha encontrado algún error (de las distintas clases que pueden detectarse).

También puede devolver otros valores en situaciones que no interesan aquí.

Una vez que yyparse ha devuelto el control, se puede aprovechar el valor devuelto para realizar alguna operación, según el resultado del análisis; así, si no se ha encontrado error alguno, se puede mostrar un mensaje indicativo de tal circunstancia.

La llamada a yyparse suele hacerse desde la función principal main, aunque no es obligado hacerlo desde ahí. Una posible versión de la función principal para realizar esta tarea es:

```
main () {
   if ( yyparse () == 0 ) {
      printf ("\nAnalisis Lexico-Sintactico terminado.\n");
      printf ("No se ha encontrado error alguno.\n\n");
   }
}
```

Tratamiento de errores

El analizador sintáctico generado por Yacc detecta diversas situaciones de error que pueden presentarse; como ya se ha comentado la función yyparse devuelve (para las situaciones que aquí se consideran) el valor 1 si se encuentra un error.

El analizador generado realiza un tratamiento de errores simple: cuando se encuentra un error se provoca la terminación del análisis; así se considera aquí. No obstante, se pueden incluir en la especificación de entrada rutinas dedicadas a la recuperación de errores; la ejecución de esas rutinas puede asociarse a distintas situaciones que susceptibles de presentarse durante el proceso de análisis.

En la librería de Yacc se tiene una función, de nombre yyerror, a la que se llama cuando se encuentra un error; la cabecera de esta función es:

```
yyerror (const char *errmsg)
```

el parámetro de la función es un literal indicativo de la clase de error que se ha encontrado. La ejecución de esta función hace que el mensaje comunicado se muestre por pantalla (o por el fichero de salida que se tenga como pre-definido).

La función yyerror se puede redefinir; para ello, ha de incluirse en la especificación de entrada a Yacc (en la sección de rutinas) una nueva definición de la función que tenga ese mismo nombre y el parámetro del tipo adecuado.