Contenido

3.7 SINTAXIS DEL LENGUAJE TINY

3.7.1 Una gramática libre de contexto para TINY

La gramática para TINY está dada en BNF en la figura 3.6. A partir de ahí hacemos las siguientes observaciones. Un programa TINY es simplemente una secuencia de sentencias. Existen cinco clases de sentencias: sentencias if o condicionales, sentencias repeat o de repetición, sentencias read o de lectura, sentencias write o de escritura y sentencias assign o de asignación. Éstas tienen una sintaxis tipo Pascal, sólo que la sentencia if utiliza **end** como palabra clave de agrupación (de manera que no hay ambigüedad de else ambiguo en TINY) y que las sentencias if y las sentencias repeat permiten secuencias de sentencias como cuerpos, de modo que no son necesarios los corchetes o pares de **begin-end** (e incluso

Figura 3.6
Gramática del lenguaje TINY en BNF

programa → *secuencia-sent*
secuencia-sent → *secuencia-sent* **;** *sentencia* | *sentencia*
sentencia → *sent-if* | *sent-repeat* | *sent-assign* | *sent-read* | *sent-write*
sent-if → **if** *exp* **then** *secuencia-sent* **end**
| **if** *exp* **then** *secuencia-sent* **else** *secuencia-sent* **end**
sent-repeat → **repeat** *secuencia-sent* **until** *exp*
sent-assign → **identificador** **:=** *exp*
sent-read → **read** **identificador**
sent-write → **write** *exp*
exp → *exp-simple op-comparación exp-simple* | *exp-simple*
op-comparación → **<** | **=**
exp-simple → *exp-simple opsuma term* | *term*
addop → **+** | **-**
term → *term opmult factor* | *factor*
opmult → ***** | **/**
factor → **(** *exp* **)** | **número** | **identificador**

begin no es una palabra reservada en TINY). Las sentencias de entrada/salida comienzan mediante las palabras reservadas **read** y **write**. Una sentencia read de lectura puede leer sólo una variable a la vez, y una sentencia write de escritura puede escribir solamente una expresión a la vez.

Las expresiones TINY son de dos variedades: expresiones booleanas que utilizan los operadores de comparación = y < que aparecen en las pruebas de las sentencias if y repeat y expresiones aritméticas (denotadas por medio de *exp-simple* en la gramática) que incluyen los operadores enteros estándar +, -, * y / (este último representa la división entera, en ocasiones conocida como **div**). Las operaciones aritméticas son asociativas por la izquierda y tienen las precedencias habituales. Las operaciones de comparación, en contraste, son no asociativas: sólo se permite una operación de comparación por cada expresión sin paréntesis. Las operaciones de comparación también tienen menor precedencia que cualquiera de las operaciones aritméticas.

Los identificadores en TINY se refieren a variables enteras simples. No hay variables estructuradas, tales como arreglos o registros. Tampoco existen declaraciones de variable en TINY: una variable se declara de manera implícita al aparecer a la izquierda de una asignación. Además, existe sólo un ámbito (global), y no procedimientos o funciones (y, por lo tanto, tampoco llamadas a los mismos).

Una nota final acerca de la gramática de TINY. Las secuencias de sentencias *deben* contener signos de punto y coma separando las sentencias, y un signo de punto y coma después de la sentencia final en una secuencia de sentencias es ilegal. Esto se debe a que no hay sentencias vacías en TINY (a diferencia de Pascal y C). También escribimos la regla BNF para *secuencia-sent* como una regla recursiva por la izquierda, pero en realidad no importa la asociatividad de las secuencias de sentencias, porque el propósito es simplemente que se ejecuten en orden. De este modo, también podríamos simplemente haber escrito la regla *secuencia-sent* recursivamente a la derecha en su lugar. Esta observación también se representará en la estructura del árbol sintáctico de un programa TINY, donde las secuencias de sentencias estarán representadas por listas en lugar de árboles. Ahora volveremos a un análisis de esta estructura.

3.7.2 Estructura del árbol sintáctico para el compilador TINY

En TINY existen dos clases básicas de estructuras: sentencias y expresiones. Existen cinco clases de sentencias (sentencias if, sentencias repeat, sentencias assign, sentencias read y sentencias write) y tres clases de expresiones (expresiones de operador, expresiones de constante y expresiones de identificador). Por lo tanto, un nodo de árbol sintáctico se clasificará principalmente como si fuera una sentencia o expresión y en forma secundaria respecto a la clase de sentencia o expresión. Un nodo de árbol tendrá un máximo de tres estructuras hijo (sólo se necesitan las tres en el caso de una sentencia if con una parte else). Las sentencias serán secuenciadas mediante un campo de hermanos en lugar de utilizar un campo hijo.

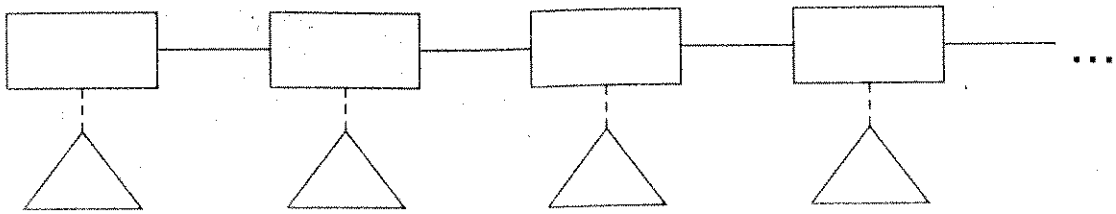
Los atributos que deben mantenerse en los nodos de árbol son como sigue (aparte de los campos ya mencionados). Cada clase de nodo de expresión necesita un atributo especial. Un nodo de constante necesita un campo para la constante entera que representa. Un nodo de identificador necesita un campo para contener el nombre del identificador. Y un nodo de operador necesita un campo para contener el nombre del operador. Los nodos de sentencia generalmente no necesitan atributos (aparte del de la clase de nodo que son). Sin embargo, por simplicidad, en el caso de las sentencias assign y read conservaremos el nombre de la variable que se está asignando o leyendo justo en el nodo de sentencia mismo (más que como un nodo hijo de expresión).

La estructura de nodo de árbol que acabamos de describir se puede obtener mediante las declaraciones en C dadas en la figura 3.7, que se repiten del listado correspondiente al archivo `globals.h` en el apéndice B (líneas 198-217). Advierta que utilizamos uniones en estas declaraciones para ayudar a conservar el espacio. Éstas también nos ayudarán a recordar cuáles atributos van con cada clase de nodo. Dos atributos adicionales que aún no hemos mencionado están presentes en la declaración. El primero es el atributo de contabilidad `lineno`; éste nos permite imprimir los números de líneas del código fuente con errores que puedan ocurrir en etapas posteriores de la traducción. El segundo es el campo `type`, el cual después necesitaremos para verificar el tipo de las expresiones (y solamente expresiones). Está declarado para ser del tipo enumerado `ExpType`; esto se estudiará con todo detalle en el capítulo 6.

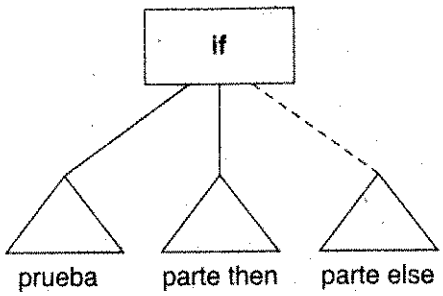
Figura 3.7
Declaraciones en C para un nodo de árbol sintáctico TINY

```
typedef enum {StmtK,ExpK} NodeKind;
typedef enum {IfK,RepeatK,AssignK,ReadK,WriteK}
             StmtKind;
typedef enum {OpK,ConstK,IdK} ExpKind;

/* ExpType se usa para verificación de tipo */
typedef enum {Void,Integer,Boolean} ExpType;

#define MAXCHILDREN 3

typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp;} kind;
     union { TokenType op;
             int val;
             char * name; } attr;
     ExpType type; /* para verificación de tipo de expresiones
*/
   } TreeNode;
```

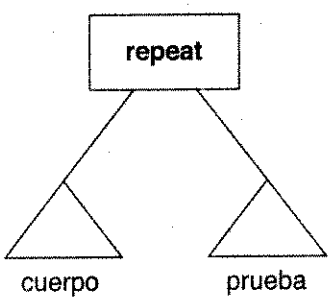
Ahora queremos dar una descripción visual de la estructura del árbol sintáctico y mostrar visualmente este árbol para un programa de muestra. Para hacer esto utilizamos cajas rectangulares para indicar nodos de sentencia y cajas redondas u ovales para indicar nodos de expresión. La clase de sentencia o expresión se proporcionará como una etiqueta dentro de la caja, mientras que los atributos adicionales también se enumerarán allí entre paréntesis. Los apuntadores hermanos se dibujarán a la derecha de las cajas de nodo, mientras que los apuntadores hijos se dibujarán abajo de las cajas. También indicaremos otras estructuras de árbol que no se especifican en los diagramas mediante triángulos y líneas punteadas para indicar estructuras que puedan o no aparecer. Una secuencia de sentencias conectadas mediante campos de hermanos se vería entonces como se muestra a continuación (los subárboles potenciales están indicados mediante las líneas punteadas y los triángulos).



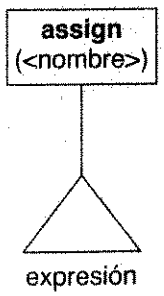
Una sentencia if (con potencialmente tres hijos) tendrá un aspecto como el siguiente.



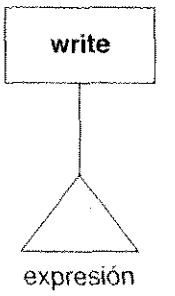
Una sentencia repeat tendrá dos hijos. La primera es la secuencia de sentencias que representan su cuerpo; la segunda es la expresión de prueba:



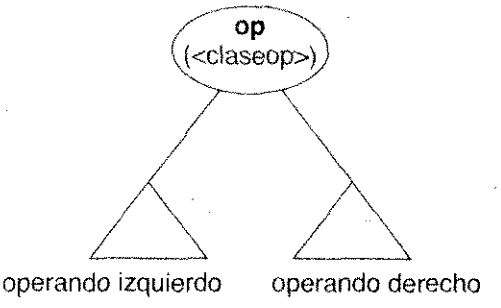
Una sentencia de asignación ("assign") tiene un hijo que representa la expresión cuyo valor se asigna (el nombre de la variable que se asigna se conserva en el nodo de sentencia):



Una sentencia de escritura ("write") también tiene un hijo, que representa la expresión de la que se escribirá el valor:



Una expresión de operador tiene dos hijos, que representan las expresiones de operando izquierdo y derecho:



Todos los otros nodos (sentencias read, expresiones de identificador y expresiones de constante) son nodos hoja.

Finalmente estamos listos para mostrar el árbol de un programa TINY. El programa de muestra del capítulo 1 que calcula el factorial de un entero se repite en la figura 3.8. Su árbol sintáctico se muestra en la figura 3.9.

Figura 3.8
Programa de muestra en el lenguaje TINY

```
{ Programa de muestra
en lenguaje TINY-
calcula el factorial
}
read x; { entrada de un entero }
if 0 < x then { no calcula si x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact { salida factorial de x }
end
```

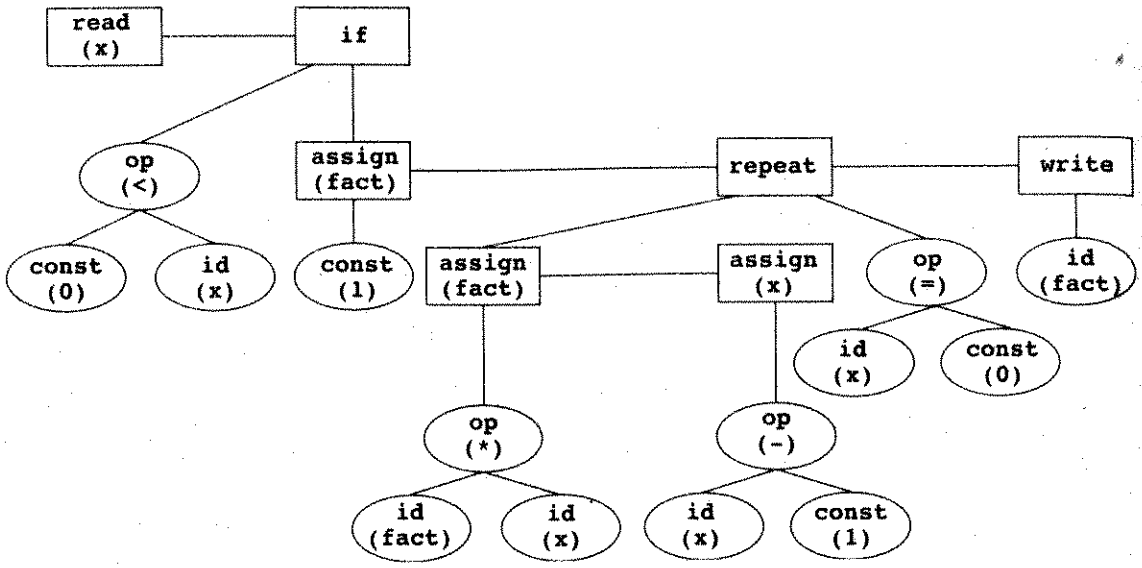


Figura 3.9
Árbol sintáctico para el programa TINY de la figura 3.8