

# Apéndice



## Apéndice A

# Aspectos complementarios de la programación

En este capítulo se estudian algunos aspectos que, aunque no se pueden considerar esenciales para una introducción a la programación en Pascal, sí resultan interesantes como complemento a los temas ya estudiados.

En primer lugar se analiza la posibilidad de definir en Pascal subprogramas con parámetros que son, a su vez, subprogramas. Con esto se pueden conseguir programas con un alto nivel de flexibilidad. Finalmente, se estudia la utilización de variables (seudo)aleatorias, que resultan especialmente útiles para desarrollar programas en los que interviene el azar.

### A.1 Subprogramas como parámetros

La utilización de procedimientos y funciones como parámetros eleva a un nivel superior la potencia y versatilidad, ya conocida, de los propios subprogramas. Supongamos, por ejemplo, que disponemos de dos funciones  $f, g : \mathbb{R} \rightarrow \mathbb{R}$ , y queremos generar a partir de ellas una tercera que convierta cada  $x \in \mathbb{R}$  en el máximo entre  $f(x)$  y  $g(x)$ , como se muestra gráficamente en la figura A.1.

La dificultad del problema radica en que la función que deseamos definir no halla el máximo de dos números reales dados sino que, dadas dos funciones cualesquiera  $f$  y  $g$  y un número real  $x$ , halla  $f(x)$  y  $g(x)$  (aplicando las funciones al real) y obtiene el máximo de esos valores:

$$\text{MaxFuncs}(f, g, x) = \begin{cases} f(x) & \text{si } f(x) < g(x) \\ g(x) & \text{en otro caso} \end{cases}$$

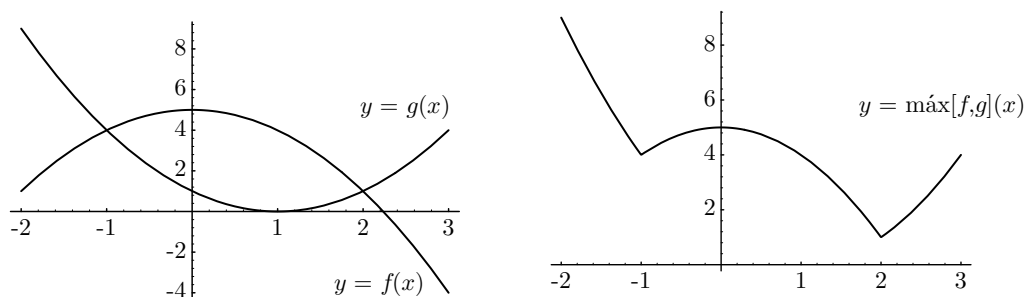


Figura A.1.

O sea, la función `MaxFuncs` responde al siguiente esquema:

$$\begin{aligned} \text{MaxFuncs} &: (\mathbb{R} \rightarrow \mathbb{R}) \times (\mathbb{R} \rightarrow \mathbb{R}) \times \mathbb{R} \rightarrow \mathbb{R} \\ \text{MaxFuncs} &(\quad f \quad, \quad g \quad, \quad x \quad) = \dots \end{aligned}$$

La novedad consiste en que, si se quiere implementar un programa que resuelva el problema, dos de los parámetros serían subprogramas (concretamente, funciones). Esto está permitido en Pascal y para ello se incluye en la lista de parámetros formales el encabezamiento completo del subprograma parámetro,

```
function MaxFuncs (function F (x: real): real;
                  function G (x: real): real; x: real): real;
  {Dev. el máximo de F(x) y G(x)}
begin
  if F(x) > G(x) then
    MaxFuncs := F(x)
  else
    MaxFuncs := G(x)
end; {MaxFuncs}
```

y en la llamada, como parámetro real, se coloca el identificador del subprograma que actúa como argumento:

```
y := MaxFuncs (Sin, Cos, Pi/4)
```

En la ejecución de la llamada, al igual que ocurre con cualquier tipo de parámetros, el subprograma ficticio se sustituye por el subprograma argumento, concretándose así su acción o resultado (dependiendo de si es un procedimiento o una función, respectivamente).

Cuando el parámetro de un subprograma es otro subprograma, la consistencia entre la llamada y la definición requiere que los subprogramas (parámetros)

formales (**F** y **G**) y los reales (**Sin** y **Cos**) tengan el mismo encabezamiento, esto es, igual número de parámetros y del mismo tipo. Además, en el caso de tratarse de funciones, el tipo del resultado de la función debe ser también el mismo.

La ganancia en flexibilidad es clara: no se ha definido la función máximo, punto a punto, de dos funciones fijas, sino de dos funciones cualesquiera (predefinidas o definidas por el usuario). Así, por ejemplo, si el usuario tiene definida una función **Cubo**, la siguiente es otra llamada válida:

```
WriteLn(MaxFuncs(Abs, Cubo, 2 * x + y))
```

En Pascal, los parámetros-subprograma sólo pueden ser datos de entrada (no de salida). Además, el paso de parámetros-subprograma no puede anidarse.

Al ser esta técnica un aspecto complementario de este libro, simplemente se añaden algunos ejemplos prácticos para mostrar su aplicación. Los ejemplos son el cálculo de la derivada de una función en un punto, la búsqueda dicotómica de una raíz de una función dentro de un cierto intervalo y la transformación de listas.

### A.1.1 Ejemplo 1: derivada

La derivada de una función<sup>1</sup> en un punto se puede hallar aproximadamente mediante la expresión

$$\frac{f(x + \Delta x) - f(x)}{\Delta x}$$

tomando un  $\Delta x$  suficientemente pequeño. El cálculo de la derivada se ilustra gráficamente en la figura A.2.

Como este cálculo depende de la función  $f$ , el subprograma en Pascal deberá incluir un parámetro al efecto:

```
function Derivada(function F(y: real): real; x: real): real;
  {PreC.: F debe ser continua y derivable en x}
  {Dev.  el valor aproximado de la derivada de de F en x}
  const
    DeltaX = 10E-6;
  begin
    Derivada:= (F(x+DeltaX) - F(x))/DeltaX
  end; {Derivada}
```

Si efectuamos una llamada a esta función pasándole la función **Sin**, y la abscisa **Pi/3**, obtenemos un valor aproximado, como era de esperar:

```
Derivada(Sin, Pi/3) ~> 4.9999571274E-01
```

---

<sup>1</sup>La función deberá ser continua y derivable en el punto considerado.

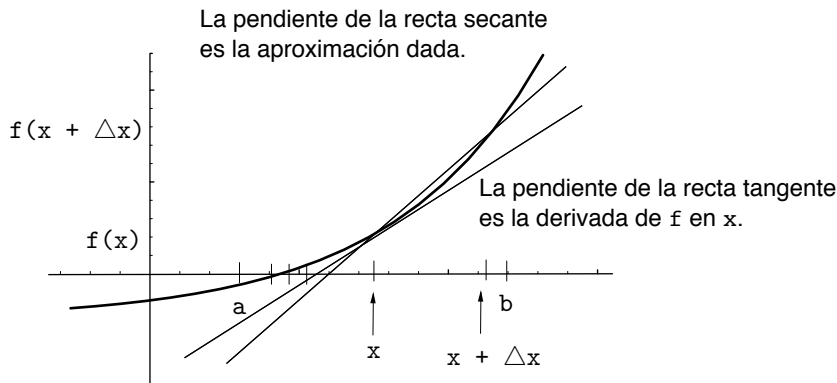


Figura A.2.

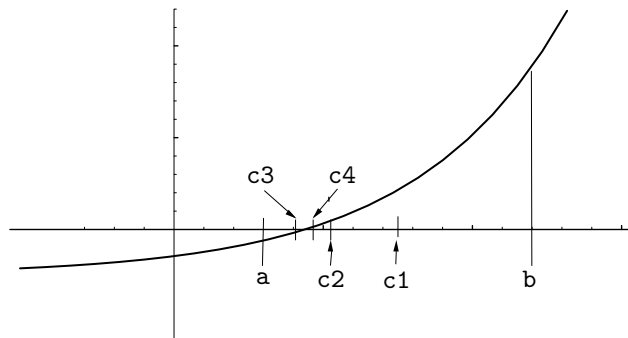


Figura A.3.

### A.1.2 Ejemplo 2: bipartición

Este método para calcular los ceros o raíces de una función se expuso en el apartado 6.5.1 para una función particular, y la generalización natural es definirlo en Pascal como un subprograma que opere con una función cualquiera, además de efectuar la búsqueda en un intervalo cualquiera.

Las condiciones exigidas son que la función debe ser continua en ese intervalo y tener distinto signo en sus extremos: digamos para simplificar que deberá ser negativa en el extremo izquierdo y positiva en el derecho, como se muestra en la figura A.3.

Este cálculo puede incluirse dentro de una función `CeroBipar` que reciba como parámetros la función de la que se calcula la raíz y los extremos del intervalo

en el que se busca ésta. La función devuelve la abscisa de la raíz.

Siguiendo las ideas expuestas en el apartado 6.5.1, una primera aproximación al diseño puede ser

```

izda:= extrIz;
dcha:= extrDc;
Reducir el intervalo
Dar el resultado

```

Recordemos que la tarea *reducir el intervalo* se puede refinar mediante un bucle **while**, utilizando una variable adicional `puntoMedio` para almacenar el valor del punto central del intervalo. `F` será el identificador del parámetro que recogerá la función de la que se busca la raíz. El subprograma final es el siguiente:

```

function CeroBipar(function F(x: real): real;
                  extrIz, extrDc: real): real;
{PreC.: F continua en [extrIz, extrDc],
  F(extrIz) < 0 < F(extrDc) }
{PostC.: F(CeroBipar(F, extrIz, extrDc)) ≈ 0 ,
  extrIz < CeroBipar(F, extrIz, extrDc) < extrDc }
const
  Epsilon= error permitido;
var
  puntoMedio: real;
begin
  {Reducción del intervalo}
  while (extrDc - extrIz) > 2 * Epsilon do begin
    {Inv.: F(extrIz) < 0 y F(extrDc) > 0 }
    puntoMedio:= (extrDc - extrIz) / 2;
    if F(puntoMedio) < 0 then
      {El cero se encuentra en [puntoMedio,extrDc]}
      extrIz:= puntoMedio
    else {El cero se encuentra en [extrIz,puntoMedio]}
      extrDc:= puntoMedio
    end; {while}
  CeroBipar:= puntoMedio
end. {CeroBipar}

```

Por ejemplo, si suponemos implementado en Pascal el polinomio  $p(x) = x^2 - 2$  mediante la función `P`, se obtendrá un cero del mismo en  $[0,2]$  efectuando la llamada

`CeroBipar(P, 0, 2)`

### A.1.3 Ejemplo 3: transformación de listas

Una transformación frecuente de las listas consiste en aplicar cierta función (genérica) a todos sus elementos. Una implementación en Pascal de esta transformación es casi directa desarrollando un procedimiento `AplicarALista` que tenga como argumentos la función a aplicar y la lista a transformar:

```

procedure AplicarALista(function F(x: real): real;
                        var lista: tListaR);
  {Efecto: aplica F a cada uno de los elementos de lista}
  var
    aux: tListaR;
begin
  aux:= lista;
  while aux <> nil do begin
    aux^.valor:= F(aux^.valor);
    aux:= aux^.siguiente
  end {while}
end; {AplicarALista}

```

Donde los elementos de la lista deben tener un tipo conocido y fijo (en el ejemplo, `real`),

```

type
  tListaR = ^tNodoR;
  tNodoR = record
    valor: real;
    sig: tListaR
  end; {tNodoR}

```

y éste debe ser el mismo antes y después de su transformación, por lo que la función convierte elementos de ese tipo en elementos del mismo. En este ejemplo la lista está formada por valores reales, por lo que también lo son el argumento y el resultado de la función.

Así, por ejemplo, la llamada `AplicarALista(Sqr, listaDeReales)` tendrá por efecto elevar al cuadrado todos los elementos de la lista `listaDeReales`.

## A.2 Variables aleatorias

A menudo es necesario construir programas en los que interviene el azar: este comportamiento indeterminista es inevitable con frecuencia (por ejemplo, en procesos de muestreo), y otras veces es deseable (en programas relacionados con simulación, juegos, criptografía, etc.)

En este apartado se introduce el uso del azar para resolver problemas algorítmicamente.<sup>2</sup> Empezaremos por resumir los mecanismos que ofrece Turbo Pascal para introducir el azar en nuestros programas, así como la forma de definir otros, sencillos y útiles; después se verá un método para simular variables aleatorias cualesquiera, para terminar dando algunos ejemplos de aplicación.

### A.2.1 Generación de números aleatorios en Turbo Pascal

La construcción de un buen generador de números pseudoaleatorios no es una tarea fácil; por eso y por sus importantes aplicaciones, casi todos los lenguajes de programación incluyen algún generador predefinido.

Para obtener los valores aleatorios en Turbo Pascal, se emplea la función `Random`, que genera un valor pseudoaleatorio y que se puede utilizar de dos modos distintos:

- Con un argumento entero y positivo<sup>3</sup>  $n$ , en cuyo caso el resultado es un número extraído uniformemente del conjunto  $\{0, \dots, n - 1\}$ :

```
Random(6) ~> 2
Random(6) ~> 5
Random(6) ~> 0
```

- Sin argumento, en cuyo caso el resultado es un real, extraído uniformemente del intervalo  $[0, \dots, 1)$ :

```
Random ~> 0.78593640172
Random ~> 0.04816725033
```

Cuando estas funciones se usan en un programa, se debe usar el procedimiento `Randomize` (sin argumentos), para “arrancar” la generación de números aleatorios, haciendo intervenir información cambiante, no controlada, como determinadas variables del sistema (el reloj), de modo que las distintas ejecuciones de ese programa funcionen de diferente modo.

Como ejemplo de uso, damos el siguiente programa, que simula el lanzamiento de un dado y de una moneda:

---

<sup>2</sup>Hay obviamente una dificultad intrínseca para reproducir el azar (de naturaleza indeterminista) mediante algoritmos (que son procesos deterministas). De hecho, en la práctica sólo se consigue simular un comportamiento “aparentemente” aleatorio –*seudoaleatorio*– aunque esto es suficiente para un gran número de aplicaciones.

<sup>3</sup>En realidad, en Turbo Pascal debe ser de tipo `word` (véase el apartado B.3).

```

Program DadoYMoneda (output);
const
  N = 10; {núm. de lanzamientos}
var
  i, lanzamientos, caras, cruces: integer;
begin
  Randomize;
  for i:= 1 to N do {N lanzamientos de dado}
    WriteLn(Random(6)+1);
  for i:= 1 to N do {N lanzamientos de moneda}
    if Random < 0.5 then
      WriteLn('Cara')
    else
      WriteLn('Cruz')
  end. {DadoYMoneda}

```

Como puede imaginarse, la función `Random` se puede combinar con otras operaciones y funciones para lograr efectos más variados. Por ejemplo la expresión

$$\text{Random} * (\mathbf{b} - \mathbf{a}) + \mathbf{a}$$

produce el efecto de una variable aleatoria uniforme del intervalo  $[a, b)$ . De modo similar, la expresión

$$\text{Random}(\mathbf{b} - \mathbf{a} + 1) + \mathbf{a}$$

produce el efecto de una variable aleatoria uniforme del conjunto  $\{a, \dots, b\}$ .

## A.2.2 Simulación de variables aleatorias

Aunque las variables aleatorias uniformes bastan en un gran número de situaciones, muchas veces se requiere generar otras variables siguiendo otras funciones de distribución. En este apartado se introduce el método de la *transformada inversa*,<sup>4</sup> que permite generar variables aleatorias arbitrarias. En los ejercicios se describen sin justificar algunos métodos particulares.

En primer lugar se presenta este método para variables aleatorias continuas. Si  $F : \mathbb{R} \rightarrow [0, 1]$  es una función de distribución cualquiera, el método consiste en generar la variable aleatoria  $y \sim \text{Unif}[0, 1)$ , y hallar el  $x$  tal que  $F(x) = y$  (véase la figura A.4). En otras palabras, si  $y \sim \text{Unif}[0, 1)$ , entonces  $F^{-1}(y) \sim F$ .

Por ejemplo, supongamos que deseamos generar una *variable aleatoria uniforme* en el intervalo  $[a, b)$ . Como su función de distribución es,

$$F(x) = \begin{cases} 0 & \text{si } x \leq a \\ \frac{x-a}{b-a} & \text{si } a \leq x \leq b \\ 1 & \text{si } b < x \end{cases}$$

---

<sup>4</sup>También conocido como *look up-table*.

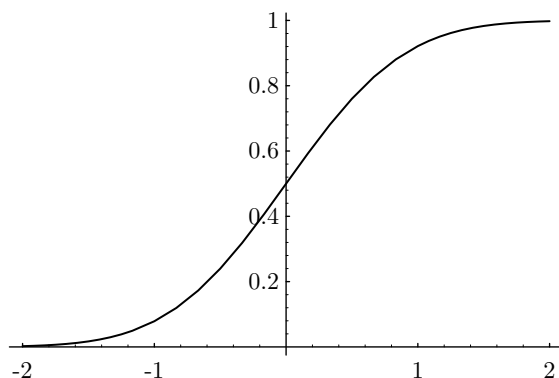


Figura A.4.

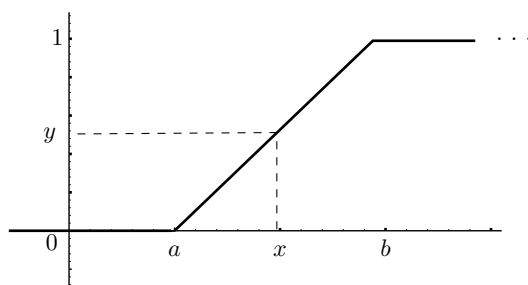


Figura A.5.

para  $y \in [0, 1)$  se tiene  $F^{-1}(y) = a + y * (b - a)$ . Por lo tanto, si se genera  $y \sim Unif[0, 1)$  (mediante  $y := \text{Random}$  simplemente, por ejemplo), basta con hacer  $x := a + y * (b - a)$  para obtener la variable aleatoria  $x \sim Unif[a, b)$ , como se ve en la figura A.5.

Naturalmente, no siempre es tan fácil invertir  $F$ , e incluso a veces es imposible hacerlo por medios analíticos. Sin embargo, la cantidad  $x = F^{-1}(y)$ , conocida  $y$ , siempre puede hallarse como el cero de la función  $F(x) - y$  (véase el apartado 6.5.3), de la que sabemos que es decreciente, por lo que es idóneo el método de bipartición (véase el apartado 6.5.1).

El método expuesto es sencillo y se adapta bien a variables discretas. Sea la función de probabilidad  $Prob(x = i) = p_i$ , para  $1 \leq i \leq n$ ; su función de distribución es  $P(k) = \sum_{i=1}^k p_i$  para  $1 \leq k \leq n$ , y expresa la probabilidad de que  $x \leq i$ . Entonces, el método consiste en generar la variable aleatoria  $y \sim Unif[0, 1)$ , y hallar el mínimo  $k$  tal que  $P(k) \geq y$  (véase la figura A.6).

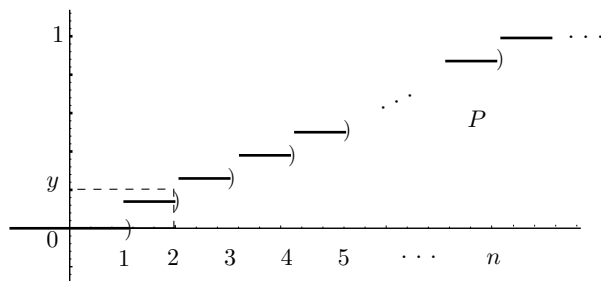


Figura A.6.

Por ejemplo, supongamos que se desea trucar un dado de forma que dé las cantidades  $1, \dots, 6$  con probabilidades  $0'15, 0'1, 0'15, 0'15, 0'3, 0'15$  respectivamente. Para ello, se hallan las cantidades  $P(k)$ , que resultan ser  $0'15, 0'25, 0'40, 0'55, 0'85, 1'00$ , respectivamente; luego se genera  $y \sim Unif[0, 1)$ , mediante  $y := \text{Random}$ , y finalmente, basta con hallar el  $\text{mín}\{k \text{ tal que } P(k) \geq y\}$ . Si las cantidades  $P(k)$  se almacenaron en un array, esta búsqueda se puede realizar por inspección de la tabla,<sup>5</sup> que tiene el siguiente contenido:

$Prob(x = i)$	$P(k)$	$i$
0'15	0'15	1
0'1	0'25	2
0'15	0'40	3
0'15	0'55	4
0'3	0'85	5
0'15	1'00	6

Si  $y := \text{Random}$  genera el valor  $0'75$ , por ejemplo, hay que buscar el menor  $P(k) \geq 0'75$ , y a partir de él localizar en la tabla de búsqueda la cara del dado que le corresponde (el cinco, en este caso).

La inspección se puede hacer secuencialmente o mediante búsqueda dicotómica (véase el apartado 15.1). El desarrollo se deja como ejercicio (véase el ejercicio 15).

### A.2.3 Ejemplos de aplicación

Las aplicaciones de los algoritmos no deterministas son múltiples y variadas. Se muestran en este apartado dos de ellas como botón de muestra, remitiendo

<sup>5</sup> Así se explica su denominación *look up-table* en inglés.

al lector interesado a los ejercicios, donde se indican otras, y a los comentarios bibliográficos.

Como ejemplo inicial, considérese que un supermercado desea hacer un sorteo entre sus clientes, de forma que sus probabilidades de ser premiados sean proporcionales al dinero gastado en la tienda. Trivialmente se ve que esta situación es un caso particular del dado trucado, explicado al final del apartado anterior, por lo que no requiere mayor explicación (véase el ejercicio 15).

Un ejemplo típico es el acto de barajar, que consiste sencillamente en

*Dado un array A de n elementos,  
Situarse en  $A_1$  uno cualquiera,  
escogido aleatoriamente entre  $A_1$  y  $A_n$ .  
...  
Situarse en  $A_{n-1}$  uno cualquiera,  
escogido aleatoriamente entre  $A_{n-1}$  y  $A_n$ .*

donde la elección entre  $A_{izda}$  y  $A_{dcha}$  se efectúa mediante la asignación

`posic:= variable aleatoria uniforme del conjunto {izda, ..., dcha}`

referida en el apartado A.2.1 y en el ejercicio 12c de este capítulo.

Finalmente, esbozamos el método del *acontecimiento crítico*, para simular una cola (una ventanilla de un banco, por ejemplo), donde llega un cliente cada  $\lambda_1$  unidades de tiempo y tarda en ser atendido  $\lambda_2$  unidades de tiempo, por término medio.<sup>6</sup>

Un paso intermedio del diseño de este programa puede ser el siguiente:

```
{Dar valores iniciales a los datos}
reloj:= 0;
longCola:= 0;
sigLlegada:= 0; {tiempo hasta la siguiente llegada}
sigSalida:= 0; {tiempo hasta la siguiente salida}
repetir sin parar
  if cola vacía o sigLlegada < sigSalida then begin
    {llegada de cliente}
    reloj:= reloj + sigLlegada;
    sigSalida:= sigSalida - sigLlegada;
    sigLlegada:= ExpNeg( $\lambda_1$ );
    longCola:= longCola + 1;
    EscribirSituacion(reloj, longCola)
  end {then}
  else begin
    {salida de cliente}
```

---

<sup>6</sup>El tiempo entre llegadas (resp. salidas) es una variable aleatoria exponencial negativa de parámetro  $\lambda_1$  que se supone desarrollada, `ExpNeg(lambda)` (véase el ejercicio 14).

```

reloj:= reloj + sigSalida;
sigLlegada:= sigLlegada - sigSalida;
sigSalida:= ExpNeg( $\lambda_2$ );
longCola:= longCola - 1;
EscribirSituacion(reloj, longCola)
end {else}
Fin repetir

```

### A.3 Ejercicios

1. Desarrolle un programa completo que utilice la función `MaxFun` que aparece en el texto, y trace la gráfica de la función `MaxFun(Sin, Cos, x)` en el fragmento del plano  $XY = [0, 3\pi] \times [-0'5, 1'5]$ .
2. Halle un cero del polinomio  $p(x) = x^2 - 2$  en  $[0, 2]$ , siguiendo el proceso indicado en el apartado A.1.2.
3. Aplique la función `Derivada` en un programa al polinomio del ejercicio anterior. Desarrolle un programa para comprobar la diferencia entre el resultado obtenido por el programa y el calculado de forma analítica,  $p'(x) = 2x$ , confeccionando una tabla con los valores de ambos, para diferentes valores de la variable  $x$  (por ejemplo,  $x \in \{0'0, 0'1, \dots, 1'0\}$ ) y de la constante `DeltaX` (por ejemplo,  $\{1, 0'5, 0'25, 0'125, 0'0625\}$ ).
4. Desarrolle un subprograma para hallar el cero de una función dada siguiendo el método de Newton-Raphson explicado en el apartado 6.5.2.
5. Desarrolle un subprograma que halle  $\sum_{i=\text{inf}}^{\text{sup}} a_i$ , para una sucesión cualquiera  $a : \mathbb{N} \rightarrow \mathbb{R}$ .  
Aplicáse al cálculo de  $\sum_{i=1}^n \frac{1}{i!}$ , siendo  $n$  un dato dado por el usuario.
6. Desarrolle un subprograma que halle  $\int_a^b f(x)dx$ , siguiendo los métodos siguientes:
  - (a) iterativamente, mediante la correspondiente descomposición en rectángulos (véase el ejercicio 11 del capítulo 6).
  - (b) recursivamente (véase el ejercicio 7 del capítulo 10).
7. Desarrolle una nueva versión del subprograma `AplicarALista` del apartado A.1.3, siguiendo una estrategia recursiva.
8. Construya un procedimiento `AplicarAArbol` que aplique una función a los elementos de un árbol, transformándolos, de la misma forma que `AplicarALista` lo hace con las listas.
9. Desarrolle un subprograma que filtre los datos de una lista según una función lógica, eliminando los que no cumplan ese test. Si, por ejemplo, se trata de una lista de enteros y el test es la función `Odd`, el efecto del filtro consiste en suprimir los pares.
10. Desarrolle una versión recursiva del subprograma del ejercicio anterior.

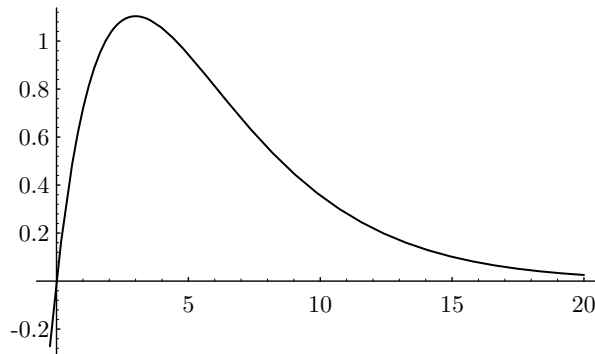


Figura A.7.

11. Construya un procedimiento general de ordenación en el que el criterio por el que se ordenan los elementos sea un subprograma que se suministra como parámetro. El encabezamiento del procedimiento es

**type**

```
tVector = array [1..n] of tElemento;
procedure Ordenar(var v:tVector;
                   function EsAnterior(x, y:tElemento):boolean);
```

Con este procedimiento se consigue una gran flexibilidad, al poder ordenar el vector en la forma que más interese al usuario, con llamadas a `Ordenar` en las que `EsAnterior` se particularice con las relaciones de orden “<”, OrdenAlfabético, MejorCalificación, etc.

12. Defina funciones aleatorias para simular lo siguiente:
- Una variable aleatoria del intervalo  $[a, b]$ .
  - Una variable aleatoria del intervalo  $(a, b]$ .
  - Una variable aleatoria del conjunto  $\{a, \dots, b\}$ .
  - Un dado que da un seis con probabilidad  $1/2$ , y un número del uno al cinco con probabilidad uniforme.
  - Una moneda trucada, que cae de cara dos de cada tres veces.
13. Genere números naturales de forma que aparezca un 1 con probabilidad  $1/2$ , un 2 con probabilidad  $1/4$ , un 3 con probabilidad  $1/8$ , etc.
14. Empleando el método de la transformada inversa, genere una variable aleatoria exponencial negativa de parámetro  $\lambda$ , cuya función de densidad  $f$  viene dada por  $f(x) = \lambda e^{-\lambda x}$ , y que se representa gráficamente en la figura A.7).  
Se recuerda que, previamente, debe hallarse la función  $F$  de distribución correspondiente.

15. Mediante el método de la transformada inversa, desarrolle un dado trucado que dé las cantidades 1 a 6 con probabilidades arbitrarias dadas como dato.
16. Defina una función que genere una variable aleatoria a partir de una función de distribución arbitraria  $F$  definida sobre el intervalo  $[a, b]$ .

## A.4 Referencias bibliográficas

Este apartado se ha centrado en el uso del azar, suponiendo que las herramientas necesarias están incorporadas en nuestro lenguaje de programación, lo que es el caso habitual. Sin embargo, el desarrollo de buenos generadores de números aleatorios no es una tarea fácil, e incluso es frecuente encontrar en el mercado y en libros de texto generadores con defectos importantes. En [Yak77, PM88, Dan89, War92], entre otras referencias, se estudian los principios de su diseño, explicando el generador más difundido actualmente (el de Lehmer) y examinando algunos de los errores de diseño más frecuentes; en [War92, Sed88] se describen dos pruebas de calidad (el test espectral y el de la chi-cuadrado).

La simulación de variables aleatorias no uniformes se estudia en multitud de textos. El método de la transformada inversa se puede estudiar en [Yak77, Mor84, PV87], entre otros textos, junto con otros métodos generales (el del “rechazo” y el de la “composición”). Algunos métodos particulares para la simulación de la normal se pueden encontrar en las dos últimas referencias. En [Dew85a] se vincula la simulación de variables aleatorias con algunas aplicaciones de los algoritmos no deterministas a través de amenos ejemplos y situaciones. De esta última referencia se ha tomado el método del acontecimiento crítico, y otras aplicaciones prácticas de la simulación se pueden encontrar en [PV87].

Finalmente, debemos indicar que en el apartado 20.5 se introducen otras aplicaciones de los algoritmos no deterministas.

En este apéndice hemos tocado muy superficialmente la posibilidad de que los parámetros de los subprogramas sean subprogramas a su vez. Esta posibilidad es ampliamente explotada en otros modelos de programación como el funcional, por lo que el lector interesado puede consultar cualquier referencia sobre el mismo, por ejemplo [BW89]. Lo cierto es que, en programación imperativa, este mecanismo se usa en muy contadas ocasiones.

## Apéndice B

# El lenguaje Turbo Pascal

El lenguaje Turbo Pascal posee numerosas extensiones con respecto al lenguaje Pascal estándar, que, por una parte, le confieren una mayor potencia y capacidad, pero por otra merman la posibilidad de transportar sus programas a otros computadores.

Es interesante conocer estas extensiones por las siguientes razones:

- Porque amplían la capacidad para manejar otros tipos numéricos del lenguaje Pascal, superando las limitaciones de los tipos estándar y facilitando el intercambio de este tipo de valores con programas escritos en otros lenguajes.
- Porque existen en muchos otros lenguajes de programación, y por ello han pasado a ser algo admitido y utilizado, siendo un estándar de facto. Por ejemplo, el tipo cadena (*string*) con sus operaciones asociadas.
- Porque son imprescindibles para la utilización de ciertos tipos de datos del Pascal estándar, como ocurre con los archivos, en los que la conexión con su implementación física no está definida en Pascal.
- Porque permiten reforzar ciertas características deseables en todo lenguaje evolucionado. La modularidad de Pascal se refuerza mediante las unidades que nos permiten definir, por ejemplo, los tipos abstractos de datos (véase el capítulo 19), aunque con limitaciones con respecto a otros lenguajes como Modula2.

En los próximos apartados se explican brevemente las particularidades más interesantes de Turbo Pascal, siguiendo el orden en el que se han presentado en el texto los aspectos del lenguaje con los que se relacionan.

## B.1 Elementos léxicos

En primer lugar estudiaremos las diferencias más significativas en la forma de escribir los identificadores y ciertos símbolos especiales.

La longitud de los identificadores en Turbo Pascal sólo es significativa en sus 64 primeros caracteres, mientras que en Pascal son significativos todos los caracteres.<sup>1</sup>

En Turbo Pascal el signo @ es un operador diferente de ^, mientras que en Pascal se pueden usar indistintamente. El signo @ en Turbo Pascal permite que un puntero señale a una variable existente no creada como referente del puntero.

En Turbo Pascal un comentario debe comenzar y terminar con el mismo par de símbolos { y } o (\* y \*). En Pascal, un símbolo de un tipo puede cerrarse con el del otro.

## B.2 Estructura del programa

El encabezamiento del programa en Turbo Pascal es opcional, por lo que puede omitirse en su totalidad. Sin embargo, si se escribe la palabra **program** deberá ir acompañada del identificador de programa. Los nombres de archivo como **input** y **output** también son opcionales, pero si se escriben, deberá hacerse correctamente.

Las diferentes secciones de declaraciones y definiciones se abren con las palabras reservadas correspondientes como en Pascal estándar. No obstante, en Turbo Pascal se puede alterar el orden de las diferentes secciones y abrirlas repetidas veces.

## B.3 Datos numéricos enteros

Turbo Pascal introduce dos tipos numéricos naturales predefinidos llamados **byte** y **word**. Sus dominios incluyen solamente valores enteros positivos, siendo para el tipo **byte** {0, ..., 255} y para el tipo **word** {0, ..., 65535}.

Un valor perteneciente al tipo **byte** ocupa precisamente un byte en memoria (de aquí su nombre), mientras que uno del tipo **word** ocupa dos bytes.

El tipo entero **integer** se complementa con dos nuevos tipos predefinidos denominados **shortInt** y **longInt**. Sus dominios son enteros positivos y nega-

---

<sup>1</sup>Aunque en el *User Manual & Report* (segunda edición, pg. 9) [JW85] se dice textualmente: *Implementations of Standard Pascal will always recognize the first 8 characters of an identifier as significant.*

tivos, siendo para el tipo `shortInt`  $\{-128, \dots, 127\}$ , mientras que el de `longInt` es  $\{-214483648, \dots, 2147483647\}$ .

El tipo `shortInt` ocupa un byte, el tipo `integer` ocupa 2 y el `longInt` 4 bytes.

Estos tipos son especialmente útiles para realizar cálculos enteros con valores grandes, como el cálculo del factorial, para los que el tipo `integer` resulta muy limitado.

A estos tipos numéricos se les pueden aplicar los mismos operadores y operaciones que al tipo entero, e incluso se pueden asignar entre sí siempre que los valores asignados estén comprendidos dentro de los respectivos dominios.

Existe un tercer tipo llamado `comp` que es un híbrido entre entero y real: se almacena en la memoria como un entero (en complemento a dos), pero se escribe como un real (en notación científica). A pesar de su implementación como entero no es un tipo ordinal. Dado que no se le pueden aplicar las operaciones de tipo entero, lo consideraremos y utilizaremos como real. Se utiliza en aquellas aplicaciones que necesiten valores “grandes”, con precisión entera, pero donde no haya que aplicar operaciones enteras.

En Turbo Pascal la expresión `n mod m` se calcula como `n - (n div m) * m` y no produce error si `m` es negativo, mientras que en Pascal estándar no está definido si `m` es cero o negativo.

## B.4 Datos numéricos reales

Turbo Pascal dispone de tres tipos de datos reales (codificados en punto flotante) que complementan al tipo estándar `real`, denominados `single`, `double` y `extended`, además del ya comentado `comp`. Sus diferentes características se muestran en la siguiente tabla:

Tipo	Dominio	Cifras significativas	Ocupación de memoria
<code>single</code>	$\{\pm 1.5E - 45, \dots, \pm 3.4E38\}$	7 u 8	4
<code>double</code>	$\{\pm 5.05E - 324, \dots, \pm 1.7E308\}$	15 ó 16	8
<code>extended</code>	$\{\pm 1.9E - 4951, \dots, \pm 1.1E4932\}$	19 ó 20	10
<code>comp</code>	$\{-2^{63}, \dots, 2^{63} - 1\}$	19 ó 20	8
<code>real</code>	$\{\pm 2.9E - 39, \dots, \pm 1.7E38\}$	11 ó 12	6

Los tipos `single` y `double` cumplen el estándar IEEE 754, que es el más utilizado en representación en punto flotante, lo que los hace idóneos para el intercambio de datos reales con programas escritos en otros lenguajes. Es curioso que el tipo estándar `real` de Turbo Pascal no sea estándar en su codificación interna.

Los tipos reales adicionales, incluyendo el tipo `comp`, admiten todos los operadores y operaciones del tipo `real`, e incluso son asignables entre sí, dentro de sus respectivos dominios.<sup>2</sup>

## B.5 Cadenas de caracteres

Es muy frecuente construir programas que precisen cadenas de caracteres para formar nombres, frases, líneas de texto, etc.

En Pascal estándar, este tipo de datos hay que definirlo como un array de caracteres con una longitud fija. Si la secuencia de caracteres tiene una longitud menor que la longitud del array, la parte final de éste queda indefinido. Con el fin de evitar posibles errores, es conveniente almacenar la longitud utilizada del array para no acceder a la parte sin definir.

En Turbo Pascal existe un tipo de datos específico predefinido llamado *string*, que podemos traducir como cadena de caracteres. Este tipo es similar a un array de caracteres, pero su longitud es gestionada automáticamente por el compilador, hasta un cierto límite. Además Turbo Pascal dispone de las funciones y procedimientos necesarios para procesar las cadenas.

### B.5.1 Declaración de cadenas

En la declaración de una variable de cadena se define la longitud máxima de la cadena, lo que se conoce como su longitud física.

```
var
  cadena: string[20];
```

Con esta declaración la variable `cadena` podrá tener a lo sumo 20 caracteres, es decir, este tamaño es un límite máximo, ya que la cadena puede tener menos. Para saber cuántos tiene en realidad, junto con la cadena se guarda un índice que contiene la longitud real de la cadena, lo que se denomina su longitud lógica.

Si al leer la variable `cadena` con la instrucción:

```
ReadLn(cadena)
```

---

<sup>2</sup>Para su utilización debe estar activada la opción

```
[Options][Compiler][Numericprocessing][X]8087/80287
```

(véase el apartado C.3.3).

o asignar un valor con:

```
cadena:= 'Lupirino'
```

escribimos menos de 20 caracteres, el índice almacenará el número real de caracteres escritos. Si escribimos 20 o más, el índice valdría 20, pero en el último caso se perderían los caracteres posteriores al vigésimo, truncándose la cadena.

También es posible declarar su longitud máxima, en cuyo caso la cadena toma una longitud física de 255 caracteres. Por ejemplo:

```
var
  nombre: string;
```

Se puede acceder a los caracteres de una cadena por sus índices, como en un array, siendo el elemento de índice 0 el que almacena la longitud lógica de la cadena. Por ejemplo, mediante las siguientes instrucciones:

```
longitud:= Ord(cadena[0])
inicial:= cadena[1]
```

se asigna a la variable `longitud` la longitud de la cadena y el primer carácter de ésta a la variable `inicial`.

### B.5.2 Operadores de cadenas

Dos cadenas se pueden comparar entre sí con los operadores usuales de relación, considerándose “menor” (anterior) aquella cadena que precede a la otra por orden alfabético.<sup>3</sup> Es decir, la comparación se realiza carácter a carácter, hasta que una cadena difiera de la otra en uno, siendo menor aquella que tiene menor ordinal en el carácter diferenciador. Puede suceder que una de las dos cadenas termine sin haber encontrado un carácter distinto, en cuyo caso la cadena más corta se considera la menor.

Además de los operadores de relación, se puede aplicar el operador `+`, llamado de concatenación, que une dos cadenas para formar otra. Por ejemplo, haciendo:

```
cadena1:= 'Pepe';
cadena2:= 'Luis';
cadena:= cadena1 + cadena2
```

la variable `cadena` tendría el valor `'PepeLuis'`.

---

<sup>3</sup>Lamentablemente, el orden entre cadenas se basa en el código ASCII, por lo que no funciona del todo de acuerdo con el orden alfabético español (acentos, ñes). Además, las mayúsculas preceden a las minúsculas, por lo que no pueden compararse entre sí de acuerdo con el orden alfabético.

### B.5.3 Funciones de cadenas

Turbo Pascal proporciona un conjunto de funciones predefinidas para procesar las cadenas. Estas funciones se presentan a continuación:<sup>4</sup>

- **Concat:**  $\mathcal{S} \times \mathcal{S} \times \dots \times \mathcal{S} \rightarrow \mathcal{S}$ .

Concatena las cadenas argumento para producir una cadena resultado. Produce el mismo resultado que el operador +.

- **Length:**  $\mathcal{S} \rightarrow \mathcal{Z}$ .

Halla la longitud lógica (entero) de la cadena argumento. La función **Length** equivale al ordinal del carácter 0 de la cadena argumento.

- **Pos** ( $\mathcal{S}_1, \mathcal{S}_2$ )  $\rightarrow \mathcal{Z}$ .

Indica la posición (un entero) de la primera cadena dentro de la segunda, o el valor 0 si no se encuentra. Esta función es especialmente útil para buscar un texto dentro de otro.

- **Copy**(s: **string**; z1, z2: **integer**): **string**;

Extrae de **s** una subcadena formada por **z2** caracteres a partir del **z1**-ésimo (incluido)

Veamos algunos ejemplos, con sus salidas:

```
var
cadena1, cadena2, cadena3:  string[40];
...
cadena1:= 'Alg.';
cadena2:= ' y estr. de datos';
cadena3:= Concat(cadena1, cadena2);
WriteLn(cadena3);
WriteLn(Length(cadena3));
WriteLn(Pos(cadena2, cadena3));
WriteLn(Copy(cadena3, 8 ,4))
```

Alg. y estr. de datos
21
5
estr

### B.5.4 Procedimientos de cadenas

Como complemento de las funciones predefinidas, Turbo Pascal también dispone de un conjunto de procedimientos de gran utilidad para el manejo de cadenas. Estos procedimientos son los siguientes:

<sup>4</sup>Dada la diversidad de tipos de algunos procedimientos y funciones de cadenas, se ha optado por dar sus encabezamientos en lugar de su definición funcional.

- **Delete**(var s: string; z1, z2: integer)

Borra z2 caracteres de la cadena s a partir del z1-ésimo (incluido). Al utilizarlo, la cadena reduce su longitud en el número de caracteres eliminados.

- **Insert**(s1: string; var s2: string; z: integer)

Inserta en s2 la cadena s1 a partir de la posición z. El procedimiento Insert, por el contrario, aumenta la longitud de la cadena en el número de caracteres insertados.

- **Str**(r: real ; var s: string)

Convierte el valor real r (también puede ser un entero z) en la cadena s. Str convierte un valor numérico en su representación como cadena de caracteres, lo que permite, por ejemplo, medir la longitud en caracteres de un número. Se utiliza también en aplicaciones gráficas, donde es obligatorio escribir cadenas.

- **Val**(s: string; var r: real; var z: integer)

Convierte la cadena s en el valor real r (también puede ser un entero) y devuelve un código entero z, que es 0 si se puede hacer la conversión, y en caso contrario señala la posición del error. Este procedimiento es quizás el más interesante de todos, al efectuar la conversión de una cadena formada por dígitos y aquellos símbolos permitidos para formar un número (tales como la letra E mayúscula o minúscula y los símbolos punto, más y menos) en su valor numérico. Si por error la cadena no tiene forma correcta de número, se señala la posición del carácter causante del fallo, lo que permite corregirlo. De esta forma se puede robustecer el proceso de introducción de números, que es una de las causas principales de errores de ejecución.

Veamos algunos ejemplos, tomando como punto de partida las asignaciones del ejemplo anterior.

Delete(cadena3, 11, 14);	
WriteLn(cadena3);	Algoritmos de datos
Insert(' simples', cadena3, 20);	
WriteLn(cadena3);	Algoritmos de datos simples
valNum1:= 123;	
Str(valNum1, cadena1);	
WriteLn(cadena1);	123
Val('12A', valNum1, error);	
WriteLn(valNum1,' ',error);	0 3
Val(cadena1, valNum1, error);	
WriteLn(valNum1,' ',error)	123 0

## B.6 Tipos de datos estructurados

Las particularidades de Turbo Pascal en cuanto a estos tipos de datos son escasas: en lo referente a arrays, se debe señalar que los procedimientos `Pack` y `Unpack` no están definidos en Turbo Pascal.

## B.7 Instrucciones estructuradas

Las diferencias con Pascal en cuanto a instrucciones estructuradas se limitan al tratamiento de la instrucción `case`, que en Turbo Pascal presenta tres variaciones, que son las siguientes:

- En Turbo Pascal el valor de la expresión selectora de un `case` puede no coincidir con alguna de sus constantes sin que se produzca error, como sucedería en Pascal estándar. En este caso, en Turbo Pascal continúa la ejecución del programa en la instrucción que sigue a `case`.
- La instrucción `case` en Turbo Pascal puede disponer de una parte `else` que se ejecuta cuando la expresión selectora no coincide con ninguna de sus constantes.
- Dentro de las constantes de la instrucción `case`, Turbo Pascal permite la definición de intervalos, lo que no está permitido en Pascal estándar.

A continuación se muestra un pequeño ejemplo de las diferencias anteriores y su sintaxis.

```
program ClasificacionDeCaracteres (input, output);
var
  car: char;
begin
  Write('Introduzca un carácter: ');
  ReadLn(car);
  case car of
    'a'..'z':WriteLn('Es minúscula');
    'A'..'Z':WriteLn('Es mayúscula');
    '0'..'9':WriteLn('Es número')
    else WriteLn('Es un símbolo')
  end {case}
end. {ClasificacionDeCaracteres}
```

## B.8 Paso de subprogramas como parámetros

Turbo Pascal difiere de Pascal estándar en la forma de realizar la declaración y paso de subprogramas como parámetros. En Turbo Pascal los subprogramas deben ser obligatoriamente de un tipo con nombre para poder ser pasados como parámetros.

Por ejemplo, en la definición de la función *Derivada* (véase el apartado A.1.1) se utilizaba como parámetro una función de argumento real y resultado también real. El tipo de esta función se define en Turbo Pascal como se muestra a continuación:

```
type
  tMatFun = function (x: real): real;
```

Para un procedimiento con dos parámetros enteros se escribiría:

```
type
  tProcInt = procedure (a, b: integer);
```

Los identificadores utilizados en estos encabezamientos se utilizan a efectos de la definición del tipo sin que tengan ninguna repercusión sobre el resto del programa.

Para declarar la función como parámetro formal, la declaramos del tipo `tMatFun` dentro del encabezamiento de la función ejemplo. De esta forma:

```
function Derivada (Fun: tMatFun; x: real): real;
  const
    DeltaX = 10E-6;
  begin
    Derivada:= (Fun(x + DeltaX) - Fun(x))/DeltaX
  end; {Derivada}
```

Dentro del programa principal efectuamos la llamada a la función *Derivada* pasándole cualquier función definida por el usuario que encaje con el tipo `tMatFun`. Suponiendo definida una función *Potencia*, una posible llamada sería:

```
WriteLn('La derivada es: ', Derivada(Potencia, x))
```

La utilización de subprogramas como parámetros requiere, dentro del esquema de gestión de memoria de Turbo Pascal, la realización de llamadas fuera del segmento de memoria donde reside el programa. Estas *llamadas lejanas* se activan marcando la opción

[F10] [Options] [Compiler] [Code generation] [X] [Force far calls]

dentro del entorno integrado (véase el apartado C.3.3). También se puede hacer desde el programa fuente utilizando las denominadas *directrices de compilación*, que son unas marcas que delimitan la parte del programa que debe compilarse con esta u otras funciones activadas. En el caso de las llamadas lejanas, la directriz de activación es `{F+}` y la desactivación es `{F-}`.

## B.9 Archivos

Para poder trabajar con archivos en Turbo Pascal es necesario relacionar el archivo externo, existente en el dispositivo de E/S y nombrado según el sistema operativo, con la variable de tipo archivo definido en nuestro programa. Una vez establecida esta relación se puede proceder a utilizar las instrucciones `Reset` o `ReWrite` y posteriormente las de lectura o escritura respectivamente, que tendrán efecto sobre el archivo externo relacionado.

Esta conexión se establece mediante el procedimiento `Assign` de Turbo Pascal que tiene como parámetros el identificador de la variable archivo y una cadena que representa el nombre del archivo externo expresado en el lenguaje de comandos del sistema operativo. Por ejemplo, la instrucción:

```
var
  archivo: text;
  ...
  Assign(archivo, 'C:\CARTAS\CARGEST.DOC')
```

vincula la variable `archivo` con el archivo llamado `CARGEST.DOC` existente en la unidad `C:`, directorio `CARTAS`.<sup>5</sup>

La otra instrucción adicional de Turbo Pascal para el proceso de archivos es `Close`:

```
Close(archivo)
```

que sirve para cerrar un archivo una vez que se termina de procesarlo. Al ejecutarse la instrucción `Close`, aquellas operaciones de lectura o escritura que pudieran quedar pendientes son completadas, quedando el archivo externo actualizado. A continuación el archivo argumento es cerrado y se pierde la conexión establecida por `Assign`.

---

<sup>5</sup>Si se omite la unidad se tomará la actual, al igual que si se omiten los directorios. En cualquier caso deberá figurar un nombre de archivo, no estando permitido el uso de comodines (véase el apartado A.1.2 de [PAO94]).

En Turbo Pascal las variables de archivo no tienen asociada la variable intermedia (cursor) definida por el operador `^` propia de Pascal estándar. Al escribir el símbolo `^` detrás de un variable de archivo se produce un error. Esta diferencia es importante, pues impide que en Turbo Pascal se pueda inspeccionar una componente del archivo sin leerla. Por esta razón ciertos programas que emplean este tipo de acceso deben ser modificados.

Los procedimientos `Get` y `Put`, para el manejo de archivos, tampoco están definidos en Turbo Pascal. Aquellos programas que los utilizan deben modificarse.

En Turbo Pascal la lectura de una marca de fin de línea, en un archivo de texto, devuelve el carácter ASCII 13 (retorno de carro), y si continúa la lectura, el carácter ASCII 10 (alimentación de línea). En Pascal estándar la lectura de una marca de fin de línea se realiza como la de un único carácter y devuelve un espacio en blanco.

Turbo Pascal dispone de numerosas extensiones para el tratamiento de archivos. Algunas realizan llamadas a las funciones del sistema operativo permitiendo, por ejemplo, cambiar el directorio de trabajo, borrar un archivo, etc. Otras permiten efectuar un acceso directo a las componentes de los archivos mejorando el tratamiento secuencial de Pascal estándar. También se permiten ficheros sin tipo para realizar operaciones a bajo nivel. Para su estudio remitimos a la bibliografía complementaria.

## B.10 Memoria dinámica

Las diferencias en cuanto al manejo de memoria dinámica residen en que en Turbo Pascal los procedimientos `New` y `Dispose` sólo reciben una variable de tipo puntero, mientras que en Pascal estándar se permiten parámetros adicionales. Recordemos, además, que en Turbo Pascal el operador `@` tiene un significado diferente de `^` (véase el apartado B.1).

## B.11 Unidades

Las unidades consisten en conjuntos de objetos, tales como constantes, tipos, variables, procedimientos y funciones que pueden ser definidos o declarados e incluso iniciados en la propia unidad. Estos objetos normalmente están relacionados entre sí y se orientan a la resolución de ciertos problemas o tareas.

Con las unidades se puede ampliar el repertorio de instrucciones del lenguaje de una forma modular, agrupándolas por acciones, sin tener que mostrar cómo se realizan estas acciones, que quedan ocultas en una parte privada.

Hay dos tipos de unidades en Turbo Pascal, aunque ambas son utilizadas de idéntica forma. Estos dos tipos son:

- Unidades predefinidas, que se dedican a tareas concretas como, por ejemplo, la interacción con el sistema operativo, el tratamiento de la pantalla de texto o la creación de gráficos.
- Unidades definidas por el programador, para resolver otros problemas no previstos en las unidades predefinidas.

Cada unidad es compilada por separado y es incorporada a un programa mediante una llamada a la misma, realizada al comienzo del programa (antes de las definiciones y declaraciones), en una cláusula

`uses unidad`

Las unidades se incorporan a los programas, que, de esta forma, pueden acceder a los objetos que forman la unidad y utilizarlos en la resolución de dichos problemas o tareas.

Las unidades son una forma apropiada (en Turbo Pascal) para construir bibliotecas de subprogramas para realizar cálculos o procesos concretos. También se utilizan para la definición de tipos abstractos de datos (véase el capítulo 19).

### B.11.1 Unidades predefinidas de Turbo Pascal

El lenguaje Turbo Pascal incorpora un conjunto de unidades que le dan una mayor potencia y flexibilidad. Son las siguientes:

- **System:** Esta unidad incluye todas las instrucciones predefinidas de Pascal estandar. Es incorporada de forma automática en todos los programas, por lo que no es necesario nombrarla en la cláusula **uses**.
- **DOS:** En esta unidad se pueden encontrar los equivalentes en Pascal de las principales llamadas al sistema operativo.
- **Crt:** Contiene funciones y procedimientos para trabajar con la pantalla de texto.
- **Printer:** Es una pequeña unidad que facilita el trabajo con la impresora. En ella se trata a la impresora como un archivo de texto llamado **lst**. Un procedimiento **Write** o **WriteLn** que se dirija al archivo **lst**, tendrá como efecto el envío de la salida a la impresora. Veamos un ejemplo:

```
uses printer;  
...  
WriteLn (1st, 'texto')
```

- **Graph3**: Es una unidad para la compatibilidad con los gráficos de tortuga<sup>6</sup> de la versión 3.0 de Turbo Pascal. Depende de la unidad **Crt**, por lo que ésta debe ser llamada previamente.
- **Turbo3**: Es una unidad para compatibilidad con ciertas instrucciones de la versión 3.0. Al igual que **Graph3** también depende de **Crt**.
- **Graph**: Es la unidad donde se definen las rutinas gráficas necesarias para usar la pantalla en los modos gráficos de alta resolución.

Los contenidos particulares de cada una de ellas pueden consultarse en la bibliografía complementaria.

### B.11.2 Unidades definidas por el usuario

De la misma forma que al escribir nuevos procedimientos y funciones un programador amplía las ya existentes en el lenguaje y puede utilizarlas en su programa, se pueden escribir nuevas unidades, y añadirlas a las existentes en Turbo Pascal. Una vez compiladas se pueden incorporar a aquellos programas que las necesiten sólo con nombrarlas en la cláusula **uses**.

Una de las principales aplicaciones de las unidades definidas por el usuario es la creación de nuevos tipos de datos complejos, cuyas definiciones y operaciones asociadas a los mismos son incluidas en una unidad. Estos tipos abstractos de datos pueden incorporarse en la forma descrita anteriormente a aquellos programas que los precisen (véase el capítulo 19).

Las unidades tienen dos partes: una pública, llamada **interface**, donde se definen los objetos que la unidad ofrece para que puedan ser usados por los programas, y otra privada llamada **implementation**, donde se concretan y desarrollan los objetos mencionados en la parte pública y donde se definen otros objetos locales privados que quedan ocultos y a los que no es posible acceder desde los programas que utilicen la unidad.

Para escribir una unidad tenemos que conocer su estructura y ésta es:

---

<sup>6</sup>Los gráficos de tortuga fueron desarrollados por Seymour Papert en el MIT dentro del lenguaje Logo y consisten en un paradigma de una tortuga que se desplaza un cierto número de pasos en una dirección o que gira un cierto ángulo, y que va dejando un rastro que forma el gráfico.

**unit** *nombre de la unidad;*  
**interface**  
    **uses** *lista de unidades;*  
    *definiciones y declaraciones públicas;*  
**implementation**  
    *definiciones y declaraciones privadas;*  
    *procedimientos y funciones;*  
**begin**  
    *código de iniciación*  
**end.**

En primer lugar aparece la palabra reservada **unit**, seguida por el nombre de la unidad, de forma similar al nombre de un programa.

La palabra reservada **interface** abre las definiciones y declaraciones públicas. Si la unidad en cuestión depende de otras unidades, debe situarse en primer lugar la cláusula **uses** seguida por la lista de unidades necesitadas. A continuación se deben definir constantes, tipos, y declarar variables, y los encabezamientos de procedimientos y funciones que serán visibles al programa que utilice la unidad. Los cuerpos de los procedimientos y funciones declarados no se incluyen aquí.

La palabra reservada **implementation** inicia la parte privada, en la que deben desarrollarse los procedimientos y funciones cuyos encabezamientos se han declarado en la parte **interface**. Para ello deben repetirse en esta parte los encabezamientos, si bien pueden abreviarse eliminando sus parámetros si los tienen.

La parte **implementation** puede completarse con otros objetos enteramente privados, incluso otros procedimientos y funciones, que pueden ser utilizados por los públicos pero que no queremos que sean visibles. Estos subprogramas deberán tener su encabezamiento completo. Todos los objetos definidos o declarados en la parte de **interface** son visibles en **implementation**.

Después de esta parte se puede ubicar lo que se denomina código de iniciación, que consiste en un conjunto de instrucciones para dar valores iniciales a aquellas estructuras variables utilizadas por la propia unidad, en este caso se coloca un **begin**, como se mostró en el esquema anterior.

En el capítulo 19 pueden encontrarse ejemplos de definición y utilización de unidades dedicadas a tipos abstractos de datos.

Una vez escrita y depurada la unidad, ésta se compila dando lugar a un archivo con extensión TPU. Cuando se compila un programa que contiene una cláusula **uses** seguida por el nombre de la unidad, el compilador busca el archivo \*.TPU correspondiente, agrega sus definiciones y declaraciones a las del propio programa, y enlaza el código de la unidad y del programa. Dado que la unidad ha sido compilada previamente, la conexión entre ambos es bastante rápida.

### B.11.3 Modularidad incompleta de Turbo Pascal

La utilización de unidades en Turbo Pascal refuerza los aspectos modulares del lenguaje Pascal estándar siendo equivalentes, con pequeñas limitaciones, a los módulos existentes en otros lenguajes.

Las unidades permiten solucionar ciertos problemas de jerarquía modular como, por ejemplo, las llamadas a subprogramas desde otros varios, lo que obligaba a situar los subprogramas llamados por encima de su verdadero nivel para hacerlos accesibles a dos o más subprogramas diferentes. La solución a este problema se alcanza incorporando una unidad con los subprogramas llamados.

Las unidades tiene una modularidad de acciones completa, al estar separadas las partes pública y privada de los subprogramas, lo que les permite alcanzar una verdadera ocultación de la información. Sin embargo, la modularidad de los datos no es completa, al no permitir mencionar públicamente tipos con una implementación privada (oculta) como en otros lenguajes, por ejemplo, Modula2 o Ada.

Por ello, cuando utilizamos las unidades de Turbo Pascal para la definición de tipos abstractos de datos, su declaración y definición tienen que estar en la parte pública **interface**.



## Apéndice C

# El entorno integrado de desarrollo

Turbo Pascal<sup>®</sup> es un producto comercial desarrollado por la empresa Borland International, Inc., cuyo uso está muy extendido.<sup>1</sup>

Turbo Pascal ha ido evolucionando a lo largo del tiempo de acuerdo con las necesidades del mercado. Esta evolución se ha concretado en la aparición de sucesivas versiones del producto. Los principales saltos cualitativos se producen en el paso de la versión 3.0 a la 4.0, al introducirse un entorno integrado de desarrollo, y en el paso de la versión 5.0 a la 5.5 permitiendo algunas características de la programación orientada a objetos. La evolución posterior tiende a completar las posibilidades del entorno y facilitar su utilización (uso del ratón en la versión 6.0, resaltado de palabras reservadas en la 7.0, etcetera). La versión más reciente en el momento de escribir estas líneas es la 7.01, que contiene la ayuda traducida al castellano. Además, existe una versión para Windows<sup>®</sup> denominada Delphi<sup>®</sup>.

El contenido de este apéndice corresponde a la versión 7.0, si bien se puede aplicar con muy pequeñas variaciones desde la versión 4.0.

### C.1 Descripción del entorno

Turbo Pascal no es sólo un compilador de un lenguaje de programación, sino un completo entorno integrado de desarrollo compuesto por todos los componentes necesarios para desarrollar programas, entre otros:

- Un potente *editor*, que permite escribir y modificar programas (y texto en general), con la posibilidad de cortar, copiar, pegar, buscar y reemplazar texto.

---

<sup>1</sup>El uso legítimo de Turbo Pascal requiere la correspondiente licencia.

- Un *compilador* del lenguaje Turbo Pascal que cumple, salvo pequeñas excepciones, la sintaxis y semántica de Pascal estándar. Existe la posibilidad de compilar en memoria o en disco. La primera opción permite alcanzar una gran velocidad de compilación, mientras que la segunda se utiliza para crear los programas ejecutables.
- Un *depurador* que permite realizar un seguimiento de los programas, ejecutándolos paso a paso, deteniendo la ejecución del programa e inspeccionando sus objetos.
- Una *ayuda* a la que se puede acceder desde el entorno, que permite la consulta rápida de la sintaxis y semántica de Turbo Pascal.
- Desde el entorno se puede acceder al DOS, para realizar tareas propias del sistema operativo, sin tener que abandonar el trabajo en curso.

Este entorno está controlado por menús, es decir, el programador puede elegir en cualquier momento entre una serie de opciones organizadas jerárquicamente. Así, en algunos casos, al escoger una opción se abre un submenú que muestra las nuevas (sub)opciones disponibles.

El entorno está basado en ventanas que pueden estar asociadas a programas (pudiendo trabajar con varios a la vez, transfiriendo información de unos a otros), mensajes u operaciones.

Pero para conocerlo, lo mejor es practicar, y eso mismo es lo que proponemos en el apartado siguiente.

## C.2 Desarrollo completo de un programa en Turbo Pascal

En este apartado vamos a describir la forma adecuada y eficiente para escribir, almacenar y modificar un programa, para compilarlo y ejecutarlo y para depurarlo.

### C.2.1 Arranque del entorno

En primer lugar tenemos que arrancar el entorno, para ello pasamos al directorio donde se encuentre, por ejemplo PASCAL. En la versión 7.0, el compilador se encuentra dentro del directorio \BIN que a su vez está dentro del directorio \PASCAL. Hacemos:

```
C:\> CD PASCAL ↵
```

Figura C.1.

o

```
C:\> CD PASCAL\BIN ↵
```

para la versión 7.0

A continuación arrancamos el entorno tecleando TURBO:

```
C:\PASCAL\BIN> TURBO ↵
```

Aparece la pantalla inicial, mostrada en la figura C.1. La línea superior es el menú, es decir, el conjunto de opciones que se pueden ejecutar. En el centro aparece la ventana de edición, con un nombre de archivo por defecto, y debajo una línea que muestra los atajos disponibles, o sea, aquellas teclas que nos permiten efectuar ciertas acciones con una o dos pulsaciones.

Para acceder a las opciones del menú se pulsa [F10] y a continuación su inicial (o la letra resaltada en su caso). Para salir de la barra de menús y editar (crear o modificar) nuestro programa pulsamos [ESC], entonces el cursor pasa a la parte interior de la ventana de edición, que es donde vamos a escribir nuestros programas. Todas las operaciones pueden realizarse igualmente utilizando el ratón.

Los números de la esquina inferior izquierda expresan la fila y columna en que se encuentra el cursor. El número de la esquina superior derecha expresa la ventana que está activa. Turbo Pascal puede tener varios programas abiertos

a la vez en distintas ventanas. También se utilizan ventanas para realizar el seguimiento y depuración de nuestros programas, y para el envío de mensajes.

Inicialmente Turbo Pascal asigna un nombre al fichero de trabajo, que para la ventana 1 es `NONAME00.PAS`

### C.2.2 Edición del programa fuente

Se realiza escribiendo las sucesivas líneas del programa, terminando cada línea pulsando la tecla `↵`. El alineamiento se realiza la primera vez con la tecla de tabulación y a partir de entonces se hace automáticamente en cada salto de línea.

Para corregir o modificar texto se utilizan las teclas habituales:

teclas de cursor	Para moverse por la ventana.
[DEL] o [SUPR]	Borra la letra que tiene el cursor debajo.
tecla de Retroceso	Borra retrocediendo hacia la izquierda.
[INSERT]	Elige el modo de inserción
[F10]	Sale de la ventana de edición.

Si se elige el modo de inserción, lo que escribamos va desplazando el texto escrito a la derecha desde la posición de inserción (si es que lo hay). En el modo de no inserción, el texto que escribamos ocupa el lugar del texto escrito, por lo que éste último se pierde.

Utilizando las teclas citadas anteriormente hemos escrito el programa para el cálculo recursivo del factorial (véase el apartado 10.1), que aparece en la figura C.2.

Dentro del menú principal existe una opción EDIT (Editar) que ofrece algunas posibilidades complementarias del editor que permiten copiar y pegar fragmentos de texto. Sus opciones trabajan sobre bloques de texto que hay que marcar pulsando la tecla de mayúsculas y las del cursor.

Existe un almacenamiento temporal de texto llamado el portapapeles (en inglés *clipboard*) donde se guardan los bloques de texto cortados (CUT) o copiados (COPY) hasta que se peguen (PASTE) en otro sitio o se corte o copie un nuevo bloque.

### C.2.3 Grabar el programa fuente y seguir editando

Para no perder el trabajo que se va realizando, es necesario ir grabando el texto escrito periódicamente. Para ello se utiliza la opción SAVE del menú EDIT, o bien la tecla [F2]. La primera vez que se hace esto, Turbo Pascal

Figura C.2.

muestra la ventana de la figura C.3 (aunque con otra lista de archivos, con toda probabilidad) en la que se nos invita a asignar un nombre al programa. Para movernos por las distintas opciones de la ventana usamos la tecla [TABULADOR], salimos con [ESC] y elegimos con la tecla ↵.

Supongamos que queremos llamar a nuestro archivo **FACT** y queremos grabarlo en el directorio raíz de la unidad **A**. Entramos en el apartado **SAVE FILE AS** y escribimos el nombre que queremos darle:

```
A:\FACT
```

No es necesario darle extensión, ya que por defecto se le asigna **.PAS**. Si no ponemos unidad y directorio, Turbo Pascal toma los que tiene asignados como directorio de trabajo. Al pulsar la tecla ↵ se graba el programa.

A partir de este momento, cada vez que queramos actualizar en el disco el archivo con el programa, usando el mismo nombre, bastará con pulsar [F2]. Se recomienda actualizar el programa cada diez o veinte minutos y, en cualquier caso, siempre antes de compilar, para evitar la pérdida accidental de parte de nuestro trabajo. Al hacerlo, se graba una copia del programa primitivo con la extensión **.BAK** y de la nueva versión con la extensión **.PAS**.

La tecla [F2] es un atajo de la opción **FILE** (Archivo) del menú principal donde se encuentran aquellas opciones necesarias para crear, almacenar e imprimir archivos, salir al DOS y terminar una sesión de trabajo. Por su interés las hemos resumido a continuación:

Figura C.3.

NEW	Abre una nueva ventana de trabajo.
OPEN... (F3)	Lee un archivo.
SAVE (F2)	Almacena el texto en un archivo con el nombre actual.
SAVE AS...	Almacena el texto en un archivo con un nuevo nombre.
SAVE ALL	Almacena en archivos todos los textos del entorno.
CHANGE DIR...	Cambia el directorio de trabajo.
PRINT	Imprime el programa activo.
PRINTER SETUP...	Configura la salida para diferentes impresoras.
DOS SHELL	Sale al DOS (se vuelve al entorno escribiendo EXIT).
EXIT (ALT+X)	Finaliza la ejecución de Turbo Pascal.

La diferencia entre las opciones NEW, OPEN y SAVE es que la primera sirve para abrir una nueva ventana de edición, la segunda para abrir una ventana con un archivo que fue creado con anterioridad, y que es leído desde el disco, y la tercera para guardar un archivo. Para cambiar el nombre del programa con el que estamos trabajando, por ejemplo, cuando se van a introducir cambios que no se sabe si serán definitivos, usamos SAVE AS.

### C.2.4 Compilación

En esta fase del desarrollo del programa vamos a realizar la traducción de nuestro programa fuente en Pascal (usualmente almacenado en un archivo con extensión PAS) a un programa objeto, ejecutable, que al compilarse en el disco tendrá la extensión EXE (véase el apartado 5.3 de [PAO94]).

Activamos el menú `COMPILE` haciendo:

`[Alt] + [C]` o bien `[F10] [C]` y después pulsamos `[C]`

o, de otro modo, más cómodamente

`[Alt] + [F9]`

Si el programa se compila con éxito, aparece el mensaje:

`Compile successful: Press any key`

si no, habrá que corregir los errores que vayan apareciendo. Éstos son mostrados por el compilador mediante mensajes de error en la ventana de edición.

Veamos un ejemplo: si por un descuido olvidamos declarar la variable global `n`, al intentar leerla, se produce un error:

`Error 3: Unknown identifier.` (identificador desconocido)

situándose el cursor en la línea en que se ha detectado el error, debajo de la instrucción `ReadLn(n)`.

El compilador no siempre es capaz de señalar la posición del error con precisión, por lo que en ciertos casos hay que indagar su origen por encima de donde se señala.

El menú `COMPILE` consta de varias opciones interesantes que resumimos a continuación:

<code>COMPILE</code> (ALT+F9)	Compila el programa fuente en curso
<code>MAKE</code> (F9)	Compila los archivos de programa modificados
<code>BUILD</code>	Compila todos los archivos de programa
<code>DESTINATION MEMORY</code>	Permite elegir destino, memoria o disco
<code>PRIMARY FILE...</code>	Define el programa primario para <code>MAKE</code> y <code>BUILD</code>
<code>CLEAR PRIMARY FILE</code>	Borra el programa primario
<code>INFORMATION...</code>	Muestra información sobre la compilación en curso

Con `MAKE` y `BUILD` se compila el archivo de programa y en su caso aquellos otros archivos de programa que dependan de él o a los que haga referencia, como, por ejemplo, las unidades definidas por el programador (véase el apartado B.11.2).

Durante la depuración, la compilación se puede efectuar almacenando el programa ejecutable en memoria RAM, lo que permite una mayor velocidad.

Una vez que el programa esté totalmente depurado puede compilarse en disco, creándose el archivo ejecutable. Para esto hay que cambiar el destino de la compilación de la siguiente forma: abrimos el menú `COMPILE` y pulsamos `[D]` activándose la orden `DESTINATION` que tiene dos opciones, `MEMORY` y `DISK`. Seleccionando `DISK` las posteriores compilaciones se dirigirán siempre a disco.

El programa objeto ejecutable tendrá el mismo nombre que el archivo en el que se encuentre el programa fuente, pero con la extensión `.EXE`.

Una vez que hemos compilado (en memoria o en disco) el programa con éxito ya se puede ejecutar.

### C.2.5 Ejecución

Para ejecutar el programa se activa el menú `RUN` con `[ALT] + [R]` o `[F10]` `[R]` y se selecciona la orden `RUN` volviendo a pulsar `[R]`, o directamente con `[CTRL] + [F9]`.

Desde el entorno integrado, al terminar el programa se vuelve a la ventana de edición sin tiempo para ver las salidas. Podemos ver la ventana de salida tecleando `[ALT] + [F5]`.

Veamos un ejemplo de ejecución del programa `Fact`:

```
Turbo Pascal Version 7.0 Copyright (c) 1983,92 Borland International
Escriba un número natural pequeño: 5
El factorial de 5 es 120
```

### C.2.6 Depuración

Durante el proceso de compilación y ejecución de un programa es frecuente que se originen errores que no sean fáciles de corregir. Para ayudar al programador en este caso, el depurador integrado de Turbo Pascal permite analizar el funcionamiento de nuestro programa, ejecutarlo paso a paso, examinar y modificar variables y fijar puntos de ruptura (en los que se detiene la ejecución del programa de forma condicional o incondicional, permitiendo inspeccionar el estado del mismo).

En primer lugar, tenemos que activar el depurador desde el menú de opciones haciendo `[F10]` `[OPTIONS]` `[DEBUGGER]` y marcando la opción `INTEGRATED`.

Para que el compilador genere la información necesaria para el depurador, hemos de asegurarnos de que la opción `DEBUG INFORMATION` está marcada, y si tenemos objetos locales, comprobar también la opción `LOCAL SYMBOLS` dentro de las opciones de la pantalla de opciones del depurador: `[F10]` `[OPTIONS]` `[COMPILER]` `[DEBUGGING]` (véase el apartado C.3.3).

Figura C.4.

A continuación recompilamos el programa y, al ejecutarlo, el depurador asume el control del programa.

Para ejecutar el programa paso a paso, sin entrar en las llamadas a la función, elegimos la opción [F10] [RUN] [STEP OVER] o simplemente pulsamos repetidamente [F8], viendo cómo las líneas del programa se van iluminando y ejecutando sucesivamente. La pantalla alterna entre la ventana de edición y la de salida. Podemos ver que las llamadas recursivas a la función se resuelven en un solo paso.

Para ejecutar paso a paso, pero entrando en las llamadas a subprogramas, elegimos la opción [F10] [RUN] [TRACE INTO] o pulsamos repetidamente [F7], viendo cómo las líneas de la función se iluminan al irse ejecutando. En el ejemplo del factorial, la función se repite varias veces debido a las llamadas recursivas. Hay que tener cuidado y no pasarse del final del programa, ya que en tal caso se vuelve a comenzar.

La ejecución paso a paso tiene su complemento perfecto con la inspección de constantes, variables y parámetros, que nos permitirá examinar los valores que tienen estos objetos tras la ejecución de cada instrucción. Para ello hay que abrir una ventana de inspección llamada WATCHES, en la que, en nuestro ejemplo, colocaremos los identificadores `n` y `num`. Hacemos [F10] [DEBUG] [WATCH], con lo que se abre la ventana, y después [F10] [DEBUG] [ADD WATCH] o simplemente [CTRL]+[F7], lo que permite introducir los identificadores que se deseen.

Si se ejecuta el programa paso a paso, se verán los valores adoptados en cada momento por los objetos incluidos en la ventana de inspección. En el ejemplo, al comenzar el programa tanto `n` como `num` son etiquetados como identificadores desconocidos (`Unknown identifier`), para cambiar posteriormente, adoptando `n` el valor introducido por el usuario, y reduciéndose `num` en cada llamada recursiva.

En la figura C.4 vemos la apariencia de la ventana WATCHES en un punto intermedio del proceso, donde las sucesivas llamadas recursivas han ido reduciendo el valor del argumento de la función hasta alcanzar el caso base.

Figura C.5.

Una opción interesante en casos como éste es la de `CALL STACK` (llamada a la pila) que permite ver la sucesión de llamadas realizadas. Recordemos que los objetos locales de las llamadas a subprogramas son almacenados en una estructura de datos del tipo pila (véase el apartado 17.2.3). Esta opción se activa haciendo `[F10] [DEBUG] [CALL STACK]`.

En la figura C.5 se muestra la serie de llamadas producidas para calcular el factorial de 4, empezando por el propio programa, dentro de la ventana `CALL STACK`.

El depurador permite también detener la ejecución de un programa para la inspección de objetos sin necesidad de ejecutar paso a paso. Esto se hace estableciendo un punto de ruptura `BREAKPOINT`.

En el ejemplo, fijaremos dicho punto al final de la función, de forma condicional para un valor de `num = 0`. De esta forma se efectuarán las sucesivas llamadas a la función, deteniéndose el programa cuando la condición se haga verdadera.

Haciendo `[F10] [DEBUG] [BREAKPOINTS]`, aparece la ventana de edición de los puntos de ruptura en la que, eligiendo la opción `[EDIT]`, podremos escribir el número de línea y la condición. A continuación salimos de la ventana y ejecutamos el programa. Éste se detiene en el punto de ruptura y muestra un mensaje, lo que nos permite inspeccionar los valores de sus variables y parámetros, así como las llamadas existentes en la pila.

### C.2.7 Salida de Turbo Pascal

Para terminar una sesión de trabajo con el entorno integrado de desarrollo, se teclea `[F10] [FILE] [EXIT]`. Es importante no olvidarse de actualizar el programa en el disco si hemos hecho cambios.

## C.3 Otros menús y opciones

En este apartado se hace un breve resumen de otros menús y opciones interesantes de Turbo Pascal que no se han expuesto en el apartado anterior.

La explicación completa de todos los menús y opciones de Turbo Pascal rebasa los objetivos de este apéndice, por lo que debe acudir a la bibliografía complementaria.

### C.3.1 Search (Búsqueda)

Las opciones correspondientes a este menú se utilizan para buscar y sustituir caracteres, palabras o frases, y para buscar líneas, errores y procedimientos o funciones. Son un complemento del propio editor de texto de Turbo Pascal.

### C.3.2 Tools (Herramientas)

El menú TOOLS permite la utilización simultánea de Turbo Pascal y de otros programas complementarios no integrados en el mismo. Estos otros programas pueden servir, por ejemplo, para realizar búsquedas de texto, programar en lenguaje ensamblador, usar un programa de depuración más potente o para analizar y optimizar el funcionamiento de los programas.

### C.3.3 Options (Opciones)

El menú OPTIONS es uno de los más importantes para el correcto funcionamiento del entorno, pues permite configurar muchos de sus parámetros, que de ser incorrectos dan lugar a errores que impiden la compilación.

Las opciones se seleccionan con la tecla de tabulación y se activan o desactivan pulsando la tecla de espacio (aparece una cruz o quedan en blanco).

Dentro del menú de opciones nos interesan especialmente algunas de las correspondientes al compilador (incluidas en el submenú COMPILER), que expone a continuación:

- [ ] FORCE FAR CALLS

Permite llamadas fuera del segmento actual de instrucciones. Debe marcarse al usar procedimientos y funciones como parámetros (véase el apartado A.1).

- [ ] RANGE CHECKING

Comprueba si los índices de arrays y cadenas se encuentran dentro de sus límites, y si las asignaciones a variables de tipo escalar no están fuera de

sus intervalos declarados. Debe activarse cuando puedan aparecer errores de este tipo.

- [ ] OVERFLOW CHECKING

Comprueba errores de desbordamiento después de efectuar las siguientes operaciones: +, -, \*, Abs, Sqr, Succ y Pred. Debe activarse cuando puedan aparecer errores de este tipo.

- [ ] COMPLETE BOOLEAN EVAL

Realiza la evaluación completa de las expresiones booleanas sin optimizarlas, es decir, la evaluación del circuito largo (véase el apartado 3.5) en caso de estar activada y del circuito corto en caso contrario.

- [ ] DEBUG INFORMATION

Genera información de depuración imprescindible para ejecutar el programa paso a paso y para fijar puntos de ruptura. Debe activarse para depurar.

- [ ] LOCAL SYMBOLS

Genera información de los identificadores locales necesaria para examinar y modificar las variables locales de un subprograma y para ver las llamadas producidas hasta llegar a un determinado subprograma con la opción DEBUG CALL STACK. Debe activarse para depurar dentro de un subprograma.

- [ ] 8087/80287

Genera código para el coprocesador numérico y debe estar activada para poder utilizar los tipos reales de Turbo Pascal.

- MEMORY SIZES

Fija los tamaños de memoria de la pila y el montículo, que son dos estructuras necesarias para el funcionamiento de los programas. El tamaño de la pila (STACK SIZE) puede ser insuficiente en programas (normalmente recursivos) con gran número de llamadas a subprogramas, como la función de Ackermann (véase el apartado 10.3.3). En dichos casos puede aumentarse su tamaño hasta un valor máximo de 65535.

- DEBUGGER

En esta opción se fijan ciertos parámetros del depurador, entre otros, si se usa el depurador integrado o independiente.

- DIRECTORIES

Aquí se fijan los directorios de trabajo de los diferentes archivos que se utilizan en Turbo Pascal. Para situar más de un directorio en la lista, se separan mediante punto y coma.

- ENVIRONMENT

Abre un submenú donde se pueden fijar muchos de los parámetros de funcionamiento del entorno integrado, parámetros de pantalla, del editor, del ratón, al arrancar, colores, etc. Esta configuración se guarda en el archivo `TURBO.TP`

### C.3.4 Window (Ventana)

El menú WINDOW abre un submenú para el control de las ventanas del entorno integrado. Se puede elegir su disposición, modificar el tamaño y posición y elegir la ventana activa, entre otras opciones.

### C.3.5 Help (Ayuda)

El entorno integrado de Turbo Pascal dispone de una extensa ayuda integrada que incluye la explicación de todos sus mandatos, los del lenguaje Pascal (con las extensiones de Turbo Pascal) y otros aspectos como unidades y directivas de compilación.

El texto de la ayuda tiene formato de *Hipertexto*, es decir, ciertas palabras aparecen resaltadas, y basta con situar el cursor sobre ellas y pulsar la tecla `↵` para acceder a información concreta sobre el concepto indicado.

La ayuda de Turbo Pascal es dependiente del contexto, o sea, se abre en el apartado correspondiente al mandato con el que estamos trabajando o en la palabra señalada por el cursor. No obstante, si queremos buscar otros temas, podemos acudir al menú de ayuda.

Además, la ayuda dispone de información sobre los mensajes de error, sobre el uso (sintaxis y semántica) de los identificadores predefinidos, palabras reservadas, etc., constituyendo un verdadero manual del usuario.

## C.4 Ejercicios

1. Invitamos al lector a que utilice el entorno de Turbo Pascal para desarrollar gradualmente los programas que se han propuesto como ejercicios en los sucesivos capítulos del libro.

2. Utilice los tipos numéricos enteros de Turbo Pascal para el cálculo del factorial. Determine cuál es el número mayor del que se puede calcular el factorial, en los tipos `byte`, `integer`, `word`, `shortint` y `longint`.
3. Calcule el valor de  $\pi$  y del número  $e$ , con la máxima precisión posible utilizando el tipo real `extended` y la suma de las series que se presentan a continuación:
 
$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$
4. Utilice las cadenas de caracteres para comprobar si una frase dada forma un palíndromo.
5. Escriba un procedimiento que haga más robusta la entrada de valores numéricos enteros utilizando el procedimiento `Val`. Para ello el procedimiento leerá el número como una cadena de caracteres. Si se produce algún error en la introducción deberá mostrar la parte correcta y pedir el resto del número.
6. Compile los programas ejemplo con paso de subprogramas como parámetros (veáse el apartado A.3) en Turbo Pascal.
7. Escriba un programa en Turbo Pascal que pida la unidad, directorio, nombre y extensión de un archivo de texto, lo abra para escritura y lea repetidamente una cadena desde teclado y la escriba en dicho archivo, añadiendo un salto de línea al final de la misma, finalizando la introducción cuando se escriba una cadena vacía.
8. Escriba un programa en Turbo Pascal que pida la unidad, directorio, nombre y extensión de un archivo de texto existente y muestre su contenido en pantalla, incluyendo los saltos de línea. Asimismo, al terminar la lectura del archivo, el programa mostrará un resumen indicando el total de caracteres leídos, cuántos son respectivamente mayúsculas, minúsculas, números u otros símbolos, y cuántos saltos de línea tenía.

## C.5 Referencias bibliográficas

La explicación más directa, sobre el funcionamiento del entorno y del lenguaje Turbo Pascal la podemos encontrar en su propia ayuda interactiva, donde acudiremos por su inmediatez para solventar aquellas dudas que pueden aparecer mientras utilizamos el entorno. No obstante, no es recomendable utilizar la ayuda como texto para el aprendizaje de Turbo Pascal, siendo conveniente acudir a otras obras estructuradas de una forma más pedagógica.

Para utilizar un compilador tan completo y extenso como Turbo Pascal no hay nada mejor que disponer de los manuales originales [Bor92b], [Bor92d], [Bor92a] y [Bor92c].

El libro [ON93] está concebido como un completo manual de referencia de Turbo Pascal cubriendo todos los aspectos del entorno, con ejemplos completos.

El texto [Joy90] es un manual de programación en Turbo Pascal que ofrece una visión completa del mismo en sus versiones 4.0, 5.0 y 5.5.

El libro [CGL<sup>+</sup>94] está orientado a un primer curso de programación estructurada y orientada a objetos utilizando como lenguaje el Turbo Pascal y cuenta con numerosos ejercicios.

## Bibliografía

- [AA78] S. Alagíc y M.A. Arbib. *The design of well-structured and correct programs*. Springer Verlag, 1978.
- [AHU88] A. V. Aho, J. E. Hopcroft, y J. Ullman. *Estructuras de datos y algoritmos*. Addison-Wesley Iberoamericana, 1988.
- [AM88] F. Alonso Amo y A. Morales Lozano. *Técnicas de programación*. Paraninfo, 1988.
- [Arn94] D. Arnow. Teaching programming to liberal arts students: using loop invariants. *SIGCSE Bulletin of the ACM*, 3:141–144, 1994.
- [Bar87] J. G. P. Barnes. *Programación en Ada*. Díaz de Santos, 1987.
- [BB90] G. Brassard y P. Bratley. *Algorítmica (concepción y análisis)*. Masson, 1990.
- [BB97] G. Brassard y P. Bratley. *Fundamentos de algoritmia*. Prentice-Hall, 1997.
- [Ben86] J. Bentley. *Programming pearls*. Addison-Wesley Publishing Company, 1986.
- [Bie93] M. J. Biernat. Teaching tools for data structures and algorithms. *SIGCSE Bulletin of the ACM*, 25(4):9–11, Dic. 1993.
- [BKR91] L. Banachowski, A. Kreczmar, y W. Rytter. *Analysis of algorithms and data structures*. Addison-Wesley, 1991.
- [Bor92a] Borland International Inc. *Turbo Pascal 7.0 Library Reference*, 1992.
- [Bor92b] Borland International Inc. *Turbo Pascal 7.0 Programmer's Guide*, 1992.
- [Bor92c] Borland International Inc. *Turbo Pascal 7.0 TurboVision Guide*, 1992.

- [Bor92d] Borland International Inc. *Turbo Pascal 7.0 User's Guide*, 1992.
- [BR75] J. R. Bitner y M. Reingold. Backtrack programming techniques. *Communications of the ACM*, 18(11):651–655, Nov. 1975.
- [BW89] R. Bird y P. Wadler. *Introduction to functional programming*. Prentice Hall International, 1989.
- [CCM<sup>+</sup>93] J. Castro, F. Cucker, F. Messeguer, A. Rubio, Ll. Solano, y B. Valles. *Curso de programación*. McGraw-Hill, 1993.
- [CGL<sup>+</sup>94] J. M. Cueva Lovelle, M. P. A. García Fuente, B. López Pérez, M. C. Luengo Díez, y M. Alonso Requejo. *Introducción a la programación estructurada y orientada a objetos con Pascal*. Editado por los autores, 1994.
- [CK76] L. Chang y J. F. Korsh. Canonical coin changing and greedy solutions. *Journal of the ACM*, 23(3):418–422, Jul. 1976.
- [CM] W. Collins y T. McMillan. Implementing abstract data types in Turbo Pascal.
- [CMM87] M. Collado, R. Morales, y J. J. Moreno. *Estructuras de datos. Realización en Pascal*. Díaz de Santos, S. A., 1987.
- [Col88] W.J. Collins. The trouble with for-loop invariants. *SIGCSE Bulletin of the ACM*, pages 1–4, 1988.
- [Dan89] R. L. Danilowicz. Demonstrating the dangers of pseudo-random numbers. *SIGCSE bulletin of the ACM*, 21(2):46–48, Jun. 1989.
- [dat73] Datamation. Número especial dedicado a la programación estructurada, Dic. 1973.
- [DCG<sup>+</sup>89] P. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. B. Tucker, A. J. Turner, y P. R. Young. Computing as a discipline. *Communications of the ACM*, 32(1):9–23, 1989.
- [DDH72] O. J. Dahl, E. W. Dijkstra, y C. A. R. Hoare. *Structured Programming*. Academic Press Ltd., 1972.
- [Dew85a] A. K. Dewney. Cinco piezas sencillas para un bucle y generador de números aleatorios. *Investigación y Ciencia*, 105:94–99, Jun. 1985.
- [Dew85b] A. K. Dewney. Ying y yang: recurrencia o iteración, la torre de hanoi y las argollas chinas. *Investigación y Ciencia*, 100:102–107, En. 1985.

- [Dij68] E.W. Dijkstra. Goto statement considered harmful. *Communications of the ACM*, 11(3), Mar. 1968.
- [DL89] N. Dale y S. Lilly. *Pascal y estructuras de datos*. McGraw-Hill, 1989.
- [DM84] B. P. Demidovich y I. A. Maron. *Cálculo Numérico Fundamental*. Paraninfo, Madrid, 1984.
- [DW89] N. Dale y C. Weems. *Pascal*. McGraw-Hill, 1989.
- [ES85] G. G. Early y D. F. Stanat. Chinese rings and recursion. *SIGCSE Bulletin of the ACM*, 17(4), 1985.
- [For82] G. Ford. A framework for teaching recursion. *SIGCSE Bulletin of the ACM*, 14(2):32–39, July 1982.
- [Fru84] D. Frutos. Tecnología de la programación. Apuntes de curso. Manuscrito, 1984.
- [FS87] G. Fernández y F. Sáez. *Fundamentos de Informática*. Alianza Editorial. Madrid, 1987.
- [GGSV93] J. Galve, J. C. González, A. Sánchez, y J. A. Velázquez. *Algorítmica. Diseño y análisis de algoritmos funcionales e imperativos*. Rama, 1993.
- [GL86] L. Goldschlager y A. Lister. *Introducción moderna a la Ciencia de la Computación con un enfoque algorítmico*. Prentice-Hall hispanoamericana. S.A. Méjico, 1986.
- [GT86] N. E. Gibbs y A. B. Tucker. A model curriculum for a liberal arts degree in computer science. *Communications of the ACM*, 29(3), march 1986.
- [Har89] R. Harrison. *Abstract data types in Modula-2*. John Wiley and sons, 1989.
- [Hay84] B. Hayes. Altibajos de los números pedrisco. A la búsqueda del algoritmo general. *Investigación y Ciencia*, 90:110–115, Mar. 1984.
- [Hig93] T.F. Higginbotham. The integer square root of n via a binary search. *SIGCSE Bulletin of the ACM*, 23(4), Dic. 1993.
- [HS90] E. Horowitz y S. Sahni. *Fundamentals of data structures in Pascal*. Computer Science Press, 3 edición, 1990.

- [Joy90] L. Joyanes Aguilar. *Programación en Turbo Pascal Versiones 4.0, 5.0 y 5.5*. Mc Graw-Hill, 1990.
- [JW85] K. Jensen y N. Wirth. *Pascal user manual and report. Revised for the ISO*. Springer Verlag, 1985.
- [KR86] B.W. Kernighan y D.M. Ritchie. *El lenguaje de programación C*. Prentice-Hall, 1986.
- [KSW85] E. B. Koffman, D. Stemple, y C. E. Wardle. Recommended curriculum for cs2, 1984. *Communications of the ACM*, 28(8), aug. 1985.
- [LG86] B. Liskov y J. Guttag. *Abstraction and interpretation in program development*. MIT Press, 1986.
- [Mar86] J. J. Martin. *Data types and structures*. Prentice Hall, 1986.
- [McC73] D. D. McCracken. Revolution in programming: an overview. *Datamation*, Dic. 1973.
- [Mor84] B. J. T. Morgan. *Elements of simulation*. Chapman and Hall, 1984.
- [MSPF95] Ó. Martín-Sánchez y C. Pareja-Flores. A gentle introduction to algorithm complexity for CS1 with nine variations on a theme by Fibonacci. *SIGCSE bulletin of the ACM*, 27(2):49–56, Jun. 1995.
- [ON93] S. K. O'Brien y S. Nameroff. *Turbo Pascal 7, Manual de referencia*. Osborne Mc Graw-Hill, 1993.
- [PAO94] C. Pareja, A. Andeyro, y M. Ojeda. *Introducción a la Informática (I). Aspectos generales*. Editorial Complutense. Madrid, 1994.
- [PJ88] M. Page-Jones. *The practical guide to structured design*. Prentice-Hall, 1988.
- [PM88] S. K. Park y K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, Oct. 1988.
- [Pn93] R. Peña. *Diseño de programas. Formalismo y abstracción*. Prentice Hall, 1993.
- [Pre93] R. S. Pressman. *Ingeniería del Software. Un enfoque práctico*. McGraw-Hill, 1993.
- [PV87] L. Pardo y T. Valdés. *Simulación. Aplicaciones prácticas en la empresa*. Díaz de Santos, S. A., 1987.

- [RN88] L. Råde y R. D. Nelson. *Adventures with your computer*. Penguin Books Ltd., Middlesex, Gran Bretaña, 1988.
- [Sal93] W. I. Salmon. *Introducción a la computación con Turbo Pascal*. Addison-Wesley Iberoamericana, 1993.
- [Sed88] R. Sedgewick. *Algorithms*. Addison-Wesley, 1988.
- [Str84] B. Stroustrup. *The C++ programming language*. Addison Wesley, 1984.
- [Tam92] W. C. Tam. Teaching loop invariants to beginners by examples. *SIGCSE Bulletin of the ACM*, pages 92–96, 1992.
- [Tur91] A. J. Turner. Computing curricula (a summary of the ACM/IEEE-CS joint curriculum task force report). *Communications of the ACM*, 34(6):69–84, 1991.
- [War92] J. S. Warford. Good pedagogical random number generators. *Communications of the ACM*, pages 142–146, 1992.
- [Wei95] M. A. Weiss. *Estructuras de datos y algoritmos*. Addison-Wesley Iberoamericana, 1995.
- [Wie88] S. Wiedenbeck. Learning recursion as a concept and as a programming technique. *Communications of the ACM*, 1988.
- [Wil89] H. S. Wilf. *Algorithms et complexité*. Masson, 1989.
- [Wir86] N. Wirth. *Algoritmos + Estructuras de datos = Programas*. Ediciones del Castillo. Madrid, 1986.
- [Wir93] N. Wirth. Recollections about the development of Pascal. *ACM Sigplan Notices*, 28(3):1–19, 1993.
- [Wri75] J. W. Wright. The change making problem. *Journal of the ACM*, 22(1):125–128, Jan. 1975.
- [Yak77] S. J. Yakowitz. *Computational probability and simulation*. Addison-Wesley Publishing Company, 1977.



# Índice alfabético

- Abs, 30, 32
- abstracción, 193
- abstracción de datos, 428, 431
- Ackermann, función de, 219
- acontecimiento crítico, 487
- agrupamiento, 135
- alcance de un identificador, 180
- aleatoria (variable), 482
- aleatorios (números), 483
- algoritmo, 3, 4
  - complejidad de un, 15
  - comprobación de un, 14
  - corrección de un, 14
  - formalmente, 8
  - informalmente, 6
  - verificación de un, 14
- algoritmos
  - de programación dinámica, 455
  - de vuelta atrás, 462
  - de *backtracking*, 462
  - devoradores, 450
  - divide y vencerás, 453
  - probabilistas, 468
- ámbito
  - de validez, 175
  - reglas de, 178
- anchura (recorrido en), 383
- and**, 36
- anidamiento
  - de bucles, 96
  - de instrucciones de selección, 90
- anillo, 388
- apuntador, 336
- árbol, 377
  - binario, 377
    - de búsqueda, 379
    - de decisión, 389
    - de juego, 389
    - general, 389
    - hoja de un, 378
    - n*-ario, 389
    - nodo hijo en un, 378
    - nodo padre en un, 378
    - raíz de un, 378
    - recorrido de un, 378
- archivo, 285
  - con tipo, 287
  - creación, 289
  - de texto, 294
  - escritura de un, 289
  - externo, 500
  - lectura de un, 291
- ArcTan**, 32
- array**, 253
- ASCII, 35
- aserción, 14
- asignación (instrucción), 52
- Assign**, 500
- autodocumentación, 69
- B**, 36
- búsqueda
  - binaria, 304
  - dicotómica, 149
  - en archivos, 320
  - en archivos arbitrarios, 321
  - en archivos ordenados, 321
  - en arrays, 301

- secuencial, 148, 302
- secuencial ordenada, 304
- backtracking*, 462
- begin**, 62
- biblioteca de subprogramas, 181
- bloque, 175
- Bolzano, teorema de, 113
- boolean**, 36
- bottom-up*, 139, 202
- bucle
  - índice de un, 101
  - cuerpo del, 94, 98
  - instrucciones, 94
  - postprobado, 98, 100
  - preprobado, 95
- burbuja
  - algoritmo, 329
- byte**, 492
- C**, 35
- cabecera, 59
- cabeza de una lista, 352
- cadenas
  - funciones, 496
  - operadores, 495
- campo, 272
- selector, 276
- case**, 92, 133, 498
- caso
  - base, 213
  - recurrente, 213
- char**, 35
- chi-cuadrado, 490
- Chr**, 36
- ciclo de vida, 18
- circuito
  - corto, 39
  - largo, 39
- circunflejo ( $\hat{\ }$ ), 337
- clave
  - en la búsqueda, 321
  - en la ordenación, 321
- Close**, 500
- código reutilizable, 201
- cola, 370
- coma flotante, 493
- comp**, 493
- compilación en Turbo Pascal, 512
- complejidad, 15
  - cálculo, 408
  - comportamiento asintótico, 402
  - de un algoritmo, 396
  - de un programa, 408
  - en el caso medio, 399
  - en el mejor caso, 399
  - en el peor caso, 399
  - en espacio, 400
  - en tiempo, 396
- composición, 85
- comprobación, 14
- computabilidad, 11
- Concat**, 496
- conjuntos, 244
- const**, 61
- constante, 52
  - anónima, 52
  - con nombre, 52
- conversión de tipos, 34, 36, 58
- Copy**, 496
- corrección, 14
  - de un programa, 73
  - parcial, 14
  - total, 15
- Cos**, 32
- coste
  - de ejecución, 396
  - de un programa, 396
- cuerpo
  - de un bucle, 94
  - de un programa, 62
  - de un subprograma, 169
- cursor, 288, 501
- dato, 28

- declaración, 240
  - global, 175
  - local, 161
- definición, 60
  - de subprograma, 161
- Delete, 497
- depuración, 10, 14, 514
  - de un programa, 74
- descripción, 240
- desigualdad de conjuntos, 246
- diagrama, 129
  - BJ, 131
  - de Böhm y Jacopini, 131
  - de flujo, 17, 125
  - limpio, 129
  - privilegiado, 131
  - propio, 129
- diagramas equivalentes, 135
- diferencia, 245
- diferenciación finita, 146
- dimensión, 255
- directrices de compilación, 500
- diseño
  - ascendente, 139, 202
  - descendente, 71, 134, 139
  - con instrucciones estructuradas, 141
  - con subprogramas, 193
- Dispose, 339
- div**, 28
- divide y vencerás, 453
- do**, 94
- double**, 493
- downto**, 101
  
- efectos laterales, 182
- eficiencia, 11
- else**, 89
- encabezamiento, 59
  - de un programa, 59
  - de un subprograma, 169
- end**, 62
  
- enumerado, 235
- EoF, 96, 288
- EoLn, 96, 295
- escritura
  - (instrucción), 54
  - con formato, 55
- especificación, 4, 78
- estado, 8, 74
  - final, 9
  - inicial, 9
- estructura de datos, 233
- estructura jerárquica, 193
- estructurada (programación), 85
- Exp**, 32
- expresión, 31
- extended**, 493
  
- factorial, 212
- False**, 36
- Fibonacci, sucesión de, 216
- FIFO, 370
- file**, 287
- fin
  - de archivo, 57, 285, 288
  - de línea, 58, 294
- for**, 100
- formato de salida de datos, 55
- forward**, 223
- función, 159
  - binaria, 30
  - infija, 30
  - interna, 30
  - monaria, 30
  - prefija, 30
  - recursiva, 212
- function**, 159
  
- Get**, 291, 501
- global, declaración, 175
- goto**, 49
  
- hipertexto, 519

- hoja de un árbol, 378
- Horner, regla de, 357
  
- identificador, 49, 52
  - ámbito, 175
  - alcance, 180
  - global, 175
  - local, 175
  - oculto, 175
  - predefinido, 49
  - visible, 175
- if**, 88, 89, 132
- igualdad de conjuntos, 246
- implementation**, 503
- in**, 246
- inclusión, 246
- independencia de subprogramas, 193
- indeterminista
  - algoritmo, 482
  - comportamiento, 482
- índice, 257
  - de un array, 255
  - de un bucle, 101
- ingeniería del *software*, 18
- inorden*, recorrido en, 379
- input**, 58, 492
- Insert**, 497
- instrucción, 52
  - de selección, 88
  - de asignación, 52
  - de escritura, 54
  - de lectura de datos, 57
  - de repetición, 94
  - iterativa, 94
- integer**, 28
- interface**, 503
- interfaz, 10, 192, 193
- intersección, 245
- invariante
  - de representación, 444
  - de un bucle, 14, 111
  
- inversión, 136
- iteración (instrucción), 94
  
- label**, 49
- lectura (instrucción), 57
- Length**, 496
- LIFO, 362
- lista, 352
  - cabeza de una, 352
  - circular, 388
  - de doble enlace, 387
  - doblemente enlazada, 387
  - enlazada, 352
- literal, 51
- llamada a un subprograma, 161, 170
- llamadas lejanas, 499, 517
- Ln**, 32
- local, declaración, 161
- longInt**, 493
- look up-table*, 484
  
- matriz, 260, 263
- MaxInt**, 28
- memoria dinámica, 335
- menú, 93
- Merge Sort*, 316
  - complejidad, 415
- mod**, 28, 493
- modelo
  - de von Neumann, 10
  - secuencial, 10
- modularidad, 190
- módulo, 189
  
- New**, 338
- Newton-Raphson, método de, 115
- nil**, 343
- nodo
  - de una lista, 352
  - hijo, 378
  - padre, 378
- not**, 36

- notación
  - científica, 32
  - exponencial, 32
  - O* mayúscula, 404
  - $\Omega$  mayúscula, 405
  - polaca inversa, 369
  - postfija, 369
  - $\Theta$  mayúscula, 405
- objeto
  - global, 200
  - no local, 200
- ocultación de la información, 193
- Odd*, 38
- of**, 49
- O* mayúscula, 404
- $\Omega$  mayúscula, 405
- or**, 36
- Ord*, 36
- ordenación
  - burbuja, 329
  - de archivos, 322
  - de arrays, 306
  - inserción directa, 309
  - intercambio directo, 310, 414
    - complejidad, 414
  - Merge Sort*, 316
    - complejidad, 415
  - ordenación rápida, 312
  - por mezcla, 316, 322, 415
    - complejidad, 415
  - Quick Sort*, 312, 414
    - complejidad, 414
  - selección directa, 307
- ordinales, 236
- organigrama, 125
- output**, 57, 492
- overflow*, 30
- Pack**, 498
- packed**, 49
- palíndromo, 389
- palabra reservada, 48
- parámetro, 161
  - de formato, 55
  - ficticio, 165
  - formal, 165
  - paso de, 170
  - por referencia, 166
  - por variable, 166
  - por dirección, 166
  - por valor, 166
  - real, 165
- pedrisco, números, 13, 119
- pertenencia, 246
- pila, 362
  - recursiva, 215
- pointer*, 336
- Pos**, 496
- postcondición, 78
- postorden*, recorrido en, 379
- precedencia, 35
- precondición, 78
- Pred**, 30, 36
- preorden*, recorrido en, 378
- principio
  - de autonomía de subprogramas, 181
  - de máxima localidad, 181
- procedimiento, 159
- procedure**, 159
- profundidad, recorrido en, 378
- Program**, 60
- programa estructurado, 134
- programación
  - con subprogramas, 157
  - dinámica, 455
  - estructurada, 85
- puntero, 336
- Put**, 289, 501
- Quick Sort*, 312
  - complejidad, 414

- $\mathcal{R}$ , 32
- raíz de un árbol, 378
- Random, 374, 483
- Randomize, 483
- Read, 57
- ReadLn, 57
- real, 32
- record**, 272
- recursión, 211
  - cruzada, 222
  - mutua, 222
- refinamiento por pasos sucesivos, 73
- registro, 271
  - con variantes, 276
    - campo selector, 276
    - parte fija, 276
- repeat**, 98, 133
- repetición, 86
- Reset, 291
- ReWrite, 289
- Round, 34
- símbolo
  - de decisión, 126
  - de entrada de datos, 126
  - de procesamiento, 126
  - de salida de datos, 126
  - terminal, 125
- salto de línea ( $\leftarrow$ ), 57
- secuencial, 285
- segmentación, 190
- selección, 85
  - instrucción, 88
- selector, 92
- set**, 244
- seudoaleatoria (variable), 482
- seudoaleatorios (números), 483
- seudocódigo, 17
- shortInt**, 493
- simulación
  - de colas
    - acontecimiento crítico, 487
    - de variables aleatorias, 484
- Sin**, 32
- single**, 493
- sobrecarga, 34, 37
- Sqr**, 30, 32
- SqRt**, 32
- Str**, 497
- string**, 494
- subprograma
  - definición de un, 161
  - llamada a un, 161, 170
  - tabla de activación de un, 215
- subrango, 238
- subrutina, 161
- Succ**, 30, 36
- sucesiones de recurrencia
  - de primer orden, 419
  - de orden superior, 421
- tabla
  - de activación, 215
  - de seguimiento, 515
- tamaño de los datos, 397
- teorema de Bolzano, 113
- test espectral, 490
- text**, 294
- then**, 88
- $\Theta$  mayúscula, 405
- tipo abstracto de datos, 428
  - corrección, 443–446
  - especificación, 440–441
  - implementación, 434, 441–443
  - invariante de representación, 444
- tipo de datos, 28, 52
  - anónimo, 242
  - básico, 28
  - con nombre, 240
  - enumerado, 235
  - estándar, 28
  - estructurado, 233
  - ordinal, 39
  - predefinido, 28

**to**, 101  
*top-down*, 139  
torres de Hanoi, 216  
transformada inversa, 484  
transición, 8  
    función de, 9  
trazado de un programa, 74  
**True**, 36  
**Trunc**, 34  
Turbo Pascal  
    entorno, 507  
    lenguaje, 491  
**type**, 240  
  
unidades, 435, 501  
    **implementation**, 503  
    **interface**, 503  
union, 245  
**unit**, 504  
Unpack, 498  
**until**, 98  
**uses**, 504  
  
**Val**, 497  
valor, 52  
variable, 52  
    aleatoria, 482  
    continua, 484  
    exponencial negativa, 489  
    simulación de, 484  
    uniforme, 484  
    de control, 101  
    puntero, 336  
    referida, 336  
    seudoaleatoria, 482  
variantes, registro con, 276  
vector, 260, 261  
vectores paralelos, 318  
verificación, 14, 106  
visibilidad de un identificador, 180  
vuelta atrás, 462  
  
**while**, 94, 132

**with**, 275, 278  
**word**, 492  
**Write**, 54  
**WriteLn**, 54  
  
**Z**, 28