13

Introducción al sistema operativo U.C.S.D.

En el presente capítulo se pretende introducir, de un modo práctico, las facilidades necesarias de U.C.S.D. Pascal; tan sólo las más importantes, para que aquellos lectores que posean ordenadores con este sistema operativo (APPLE II, TERAK- 8510a, TRS-80, IBM-PC, P-2000), comenzando la lectura por el presente capítulo y continuando con el orden natural del libro, aprendan también a manejar su máquina, al menos lo suficiente como para introducir y correr los programas que se exponen a lo largo del libro.

Nota preliminar

El siguiente capítulo presupone que poseemos dos unidades de disco y los disquets del sistema operativo U.C.S.D. Dichos disquets, denominados aquí MYDISK1: y MYDISK2:, deben contener los archivos que permiten trabajar en U.C.S.D. Pascal, caracterizados por llevar el prefijo SYSTEM:

```
SYSTEM.COMPILER
SYSTEM.PASCAL
SYSTEM.EDITOR
SYSTEM.FILER
SYSTEM.LIBRARY
...
```

Obviamente, cada vez que el lector se encuentre con el nombre del disquet del sistema, MYDISK1:, deberá sustituirlo por el nombre del disquet de su unidad número uno y cada vez que se encuentre con el

identificador MYDISK2: deberá sustituirlo por el nombre de su disquet que se halle en la unidad número dos. Generalmente dichos nombres admiten ser sustituidos por los identificadores #4: y #5:, respectivamente.

Una vez que nuestros disquets están en sus unidades respectivas, con la puerta cerrada, podemos ya hacer arrancar el sistema dando los interruptores (pulsando alguna tecla..., el modo de inicializar el sistema varía según máquinas). El ordenador ejecuta entonces el programa, "BOOTSTRAPLOADER", que permite hablar y entender el lenguaje Pascal.

Si todo ha marchado bien, junto a un mensaje de bienvenida, se verá en la parte superior de la pantalla el desplegado:

```
COMMAND: E(EDIT, R(UN, F(ILE, C(COMP, L(INK, X(ECUTE,
A(SSEM, D(EBUG? (1.1)
```

Apple II posee pantalla de sólo 40 caracteres, siendo las normales de 80 caracteres. Para visualizar el desplegado completo teclear CTRL-A, pulsando ambas teclas a un tiempo. Esto no es privativo de Apple II, en IBM-PC (NCI) el papel análogo a CTRL-A lo efectúa "?" (– ").

Algunas versiones, como la NCI de IBM-PC, antes del desplegado de COMMAND presentan la posibilidad de actualizar la fecha tecleando un número entre 1 y 31 para el día, un guión, las tres primeras letras del nombre del mes en inglés, un guión, los dos dígitos últimos del año. A continuación debe oprimirse Return (↵, Ret..). Si simplemente se oprime Return la fecha permanece tal como estaba. Si se oprime un espacio seguido de Return la fecha se incrementa un día... Después de actualizar la fecha el desplegado COMMAND aparece en la cabecera de la pantalla. De todas las letras que nos da a escoger el desplegado COMMAND, E,R,F,C,L,X,A,D, pulsemos una E, de E(DIT, es ahí donde iremos siempre que deseemos editar un nuevo trabajo

Bajo el desplegado de EDIT:

```
EDIT: A(DSJ, C(PY, D(LETE, F(IND, I(NSRT, J(UMP,
R(PLACE, Q(UIT, X(CHNG, Z(AP (1.1)
```

Nos parece la pregunta:

```
NO WORKFILE IS PRESENT. FILE? (RET FOR NO FILE,
ESC-RET TO EXIT)
```

Lo que nos dice es que no tenemos en el nivel EDIT ningún archivo de trabajo. Esto era evidente para nosotros ya que hemos ido a EDIT a editar nuestro primer trabajo. Nosda, además, varias opciones a elegir:

1) Responder a la pregunta FILE? con un nombre de archivo de cualquier trabajo anterior almacenado en alguno de nuestros disquets. Oprimiendo Return después, se visualizaría dicho trabajo en EDIT. (RETURN =↵, RET ...).
2) Oprimir la tecla Return para entrar en EDIT a editar un trabajo nuevo.
3) Teclear ESC-RETURN para salir del nivel EDIT y retornar al nivel COMMAND en el que nos hallábamos.

Insert

Escogemos la segunda opción y después de oprimir RETURN, de entre todas las posibilidades que nos presenta el nivel EDIT: A, C, D, F, I, J, R, Q, X, Z, escogemos teclear la I de I(NSRT: es ahí donde tendremos que ir siempre que queramos introducir, mediante teclado, un programa en el área de trabajo.

Tratemos ahora el teclado de nuestro ordenador como si fuera el de una máquina de escribir y tecleemos la frase:

```
HACE UN DIA HORROROSO
```

Supongamos que nos hemos equivocado y en realidad deseamos poner:

```
HACE UN DIA ESTUPENDO
```

Pulsemos entonces ← hasta que tengamos la frase: HACE UN DIA y añadamos luego ESTUPENDO.

Supongamos ahora que deseamos completar el mensaje de modo que ponga:

```
HACE UN DIA ESTUPENDO
PARA IR AL CAMPO
```

Para hacerlo, usar la tecla RETURN para cambiar de línea y luego teclear normalmente: PARA IR AL CAMPO.

Observemos ahora el desplegado de I(NSRT; aparece la posibilidad de teclear ETX, (es decir CTRL-C..., ver esquema final si carecemos de una tecla ETX). Si elegimos esta posibilidad nuestro trabajo se trasladará a la memoria de la computadora; si elegimos la posibilidad ESC, saldremos también al nivel E(DIT, pero la máquina no meterá en memoria nuestro mensaje y se perderá.

Tecleemos ESC. Nos encontramos en el nivel EDIT pero nuestro texto ha desaparecido. Volvamos de nuevo a I(NSRT tecleando una I e insertemos de nuevo nuestro mensaje. Salgamos de I(NSRT tecleando ETX. Hemos vuelto al nivel EDIT y nuestro texto sigue estando presente.

Supongamos que ahora queremos cambiar el texto:

```
HACE UN DIA ESTUPENDO
PARA IR AL CAMPO
```

por:

```
HACE UN DIA ESTUPENDO
PERO NO VOY A IR AL CAMPO
```

Delete

Mover dentro de EDIT el cursor hasta que esté sobre la P de PARA. Tecleemos una D de D(ELETE y a continuación pulsemos → cuatro veces seguidas. Salgamos de D(ELETE tecleando ETX. Nos hallamos de nuevo en EDIT. Está claro para qué sirve ir al nivel D(ELETE: para borrar.

Bien, mover ahora, dentro de EDIT, el cursor hasta que esté situado en el espacio anterior a la I de IR. Entrar en I(NSRT teclenado una I. Insertar PERO NO VOY A. Salir de INSERT teclenado ETX.

Antes de seguir practicar la redacción y modificación de diversos mensajes teniendo en cuenta además las siguientes facilidades:

a) Para mover el cursor dentro del nivel EDIT:

REPT	Tecla de repetición.
↓ (CTRL–L)	Mueve el cursor hacia abajo.
↑ (CTRL–O)	Mueve el cursor hacia arriba.
J E	El cursor salta al final del texto.
J B	El cursor salta al comienzo del texto.
P	El cursor salta una página.

C B	Copia a partir de la posición donde se encuentre el cursor el contenido del BUFFER (lo inmediato anterior borrado con DELETE).
C F	Copia en la posición del cursor el archivo cuyo nombre tecleemos seguido de un RETURN (p. ej.: MYDISK1: MIO.TEXT, RETURN) (practicar esta facilidad más adelante).

b) Para el borrado, dentro de DELETE:

→	Borra un caracter hacia adelante.
←	Borra el carácter inmediato anterior al cursor.
RETURN (↵)	Borra la línea a partir de la posición del cursor.
ETX (CTRL–C)	Se usa para salir de DELETE y volver a EDIT.

(*Ver en el esquema final otras opciones de teclado.*)

Nota: Las facilidades descritas suponen que el signo anterior al desplegado de EDIT es >. En caso contrario la dirección de los movimientos descritos puede ser justo la contraria. Dicho signo puede cambiarse teclenado > ó <. Tener en cuenta que los signos de las partes superiores de una tecla se obtienen oprimiendo dicha tecla junto con SHIFT (↑).

Quit

Si ya hemos practicado suficiente las facilidades anteriores nos preguntaremos ¿cómo abandonar el nivel EDIT?

Escojamos en el nivel EDIT la posibilidad Q(UIT. Tecleemos, pues, una Q. Nos aparece en pantalla:

```
QUIT:
   U(PDATE THE WORKFILE AND LEAVE
   E(XIT WITHOUT UPDATING
   R(ETURN TO EDITOR WITHOUT UPDAT
   W(RITE TO A FILENAME AND RETURN
   S(AVE WITH SAME NAME AND RETURN
```


Que traduciendo significa:

```
QUIT:
   U(Archivar el área de trabajo y salir
   E(Salir sin conservar
   R(Volver al nivel EDIT sin archivar
   W(Escribir en un archivo con el nombre que tecleemos
   S(Guardar en un archivo con el mismo nombre que nuestro
     archivo anterior
```

Tecleemos una E. Nos preguntan:

```
TROW AWAY CHANGES SINCE LAST UPDATE?
```

¿Deseamos destruir las últimas modificaciones hechas en EDIT?. Contestanto Y, de "yes", el contenido o las modificaciones hechas en nuestra visita a EDIT serán destruidas. Tecleando N, de "no", invalidamos la salida escogida, "EXIT", y tenemos la posibilidad de escoger de nuevo.

Comprobémoslo. Tecleemos E,Y. Nos hallaremos en el nivel COMMAND, regresemos a EDIT tecleando E. RETURN. Nuestro texto ha desaparecido. Tecleemos I de INSERT y volvamos a insertar nuestro antiguo mensaje:

```
HACE UN DIA ESTUPENDO
PARA IR AL CAMPO
```

Salgamos de INSERT tecleando ETX(CTRL–C ...). Salgamos de EDIT tecleando Q,U. Nos hallaremos en el nivel COMMAND. Intentemos ahora correr nuestro texto oprimiendo R de R(UN. (En algunas versiones como la NCI de IBM–PC, después de oprimir la R de "RUN", aparece la pregunta OUTPUT FILE FOR COMPILED LISTING? En ese caso y puesto que por el momento vamos a hacer caso omiso de dicha pregunta, para correr nuestro texto será necesario oprimir, después de la R de RUN, la tecla de RETURN(RET, ↵ ...).

El compilado dió error. Era de esperar ya que nuestro texto no tiene la forma de un programa, no está redactado en el lenguaje que nuestra máquina entiende, el PASCAL. Teclear un espacio para volver a COMMAND.

Filer

Desde COMMAND entremos en los archivos tecleando una F de F(ILER. Entonces nos aparece el desplegado:

Borremos nuestro SYSTEM.WRK.TEXT y volvamos a COMMAND tecleando la secuencia F,N,Y,Q. Vayamos a INSERT con la secuencia E,RETURN,I e insertemos el programa:

```
PROGRAM PEROGRULLO3;
CONST
  CH='*';
VAR
  I,J:INTEGER; C:CHAR; S:STRING;
BEGIN
  READ(I,C);
  READ(J,S);
  WRITE(I,CH,CH,S,CH,CH,C,CH,CH,J)
END.
```

Tras comprobar que está correctamente escrito salgamos de INSERT tecleando ETX. Salgamos de EDIT con Q,W,MYDISK1:MIO.TEXT, RETURN. Nos hallaremos en el nivel COMMAND. Vayamos a los archivos tecleando una F. Tecleemos luego una L. Respondamos a la pregunta DIR LISTING OF? tecleando MYDISK1:,RETURN. Observar que en el listado aparece MIO.TEXT. Tenemos dos copias de nuestro trabajo, la llamada SYSTEM.WRK.TEXT visualizable en EDIT y otra llamada MIO.TEXT almacenada en el disquet MYDISK1:. Corramos el programa saliendo del nivel FILER con la secuencia Q,R,(↵).

Tras el compilado, entremos por teclado los datos 3P9SOFISTICADO y oprimamos RETURN. El resultado será:

Salida en pantalla

```
3P9SOFISTICADO
3**SOFISTICADO**P**9
```

Observemos que inmediatamente después del nombre del programa hemos declarado un nuevo bloque, el bloque de las constantes. Este bloque del programa va encabezado por la declaración "CONST" y seguido por una serie de igualdades del tipo:

```
PI = 3.141592;
CH = '*'
```

Dichas igualdades sirven para poner nombre y declarar las constantes (cantidades fijas que no cambian de valor), que se usarán en el programa.

Justo después del bloque CONST, hemos declarado un bloque VAR. En dicho bloque se declaran las variables, indicando el tipo de valor que pueden tomar a lo largo del programa.

Las declaraciones:

```
I,J:INTEGER;        S:STRING;
C:CHAR;             A,B:BOOLEAN;
R,S:REAL;
```

indican que I,J pueden tomar valores enteros. S toma valores que tengan la forma de una tira de caracteres. C puede tomar valores en el conjunto de caracteres. A,B sólo pueden tomar los valores TRUE (cierto) y FALSE (falso). R,S toman valores en el conjunto de los números reales.

Observemos, por último, que las sentencias READ han leído los datos entrados por teclado, durante la corrida del programa, sobre una misma línea.

Borremos ahora nuestro SYSTEM.WRK.TEXT con la secuencia F,N,Y. Volvamos a COMMAND tecleando una Q y vayamos a EDIT con E.RETURN. Supongamos que estamos arrepentidos y deseamos tener en EDIT el trabajo recién borrado. Podemos optar por una de las dos posibilidades siguientes:

1) Saliendo de COMMAND teclear F,G (de GET), MYDISK1:MIO.TEXT,RETURN, con lo que se traslada al SYSTEM.WRK.TEXT una copia de MIO.TEXT dispuesta para ser corrida con RUN o contemplada en EDIT.
2) Ir a EDIT desde COMMAND y teclear, en respuesta a la pregunta FILE?, MYDISK1:MIO.TEXT oprimiendo RETURN después. El trabajo se hallará en EDIT dispuesto para ser contemplado o modificado. Para correrlo será necesaria una salida de EDIT diferente de Q,E para que no sea destruido el archivo SYSTEM.WRK.TEXT en nuestro regreso a COMMAND.

Get

Tomaremos la primera opción. Salgamos de EDIT con Q,E. Tecleemos desde COMMAND F,G,MYDISK1:MIO.TEXT, y oprimamos RETURN. Vayamos de nuevo a EDIT con Q,E. Nuestro MIO.TEXT se encuentra en EDIT tal y como esperábamos. Modifiquemos el texto para que ponga:

```
PROGRAM PEROGRULLO4;
CONST
  CH='*';
VAR
```

```
  I,J:INTEGER; C:CHAR; S:STRING;
BEGIN
  READLN(I,C);
  READLN(J,S);
  WRITE(I,CH,CH,S,CH,CH,C,CH,CH,J);
  WRITELN; WRITELN; WRITELN(C)
END.
```

Salgamos con Q,W,MYDISK2:MIISIMO.TEXT,RETURN y corramos el programa oprimiendo una R. El resultado de la corrida será:

Salida en pantalla:

```
120$
7BARBARA
120**BARBARA**$**7

$
```

Observemos que cada READLN lee en una línea diferente.

Borremos el archivo SYSTEM.WRK.TEXT y volvamos a EDIT con la secuencia F,N,Y,Q,E,RETURN. Entremos en INSERT y tecleemos el siguiente programa:

```
PROGRAM PEROGRULLO5;
CONST
  PI=3.141592;
VAR
  AREA,RADIO:REAL;
BEGIN
  WRITELN('TECLEA EL RADIO DE LA CIRCUNFERENCIA Y OPRIME RETURN');
  READLN(RADIO);
  AREA:=PI*RADIO*RADIO;
  WRITELN('EL AREA DEL CIRCULO DE RADIO=',RADIO,' ES:',AREA)
END.
```

Salir de EDIT con la secuencia Q,U. Correrlo con una R. El resultado fue, tecleando 10.5 y oprimiendo RETURN:

Salida en pantalla:

```
TECLEA EL RADIO DE LA CIRCUNFERENCIA Y OPRIME RETURN
 1.05000E1
EL AREA DEL CIRCULO DE RADIO= 1.05000E1 ES: 3.46361E2
```

Observar que el signo * es el que denota la operación producto.

Transfer

Bien, ¿y si quisiéramos llevarnos a casa copias sobre el papel o sobre un disquet nuestro (pongamos por caso BLANK:), de los programas archivados?

Tecleemos F para entrar en el nivel FILER, oprimamos luego una T. Nos sale la pregunta TRANSFER? (¿transferir?). Contestemos MYDISK1:MIO.TEXT y oprimamos RETURN. Aparece entonces TO WHERE? (¿a dónde?). Tecleemos PRINTER: y oprimamos RETURN. Si todo fue bien, y los interruptores de la impresora estaban dados, tendremos sobre el papel una copia de MIO.TEXT.

Repetir los pasos anteriores para pasar al papel copias de SYSTEM. WRK.TEXT y de MYDISK2:MIISIMO.TEXT, colocando dichos nombres en lugar de MYDISK1:MIO.TEXT.

Si al aparecer la pregunta TO WHERE? quitamos el disquet MYDISK2: de su unidad de disco y metemos nuestro disquet (previamente formateado y con nombre BLANK:), tecleando BLANK:MIPRIMER.TEXT y oprimiendo RETURN habremos realizado una copia de MIO.TEXT en BLANK, con nombre MIPRIMER.TEXT. Podemos comprobarlo haciendo el listado de BLANK: con la secuencia L,BLANK:,RETURN.

Sabemos cómo borrar el archivo SYSTEM.WRK.TEXT, pero, ¿cómo borrar otros archivos?

Si ya hemos terminado de pasar todos nuestros archivos a impresora y tenemos además un MIPRIMER.TEXT en BLANK:,

Remove

Con el disquet BLANK: en la segunda unidad de disco, dentro del nivel FILER, teclear una R de REMOVE. Aparece la pregunta REMOVE? Contestar tecleando BLANK:MIPRIMER.TEXT y oprimiendo RETURN a continuación. Nos preguntan de nuevo UPDATE DIRECTORY? Contestando Y, de "yes", MIPRIMER.TEXT será destruido y desaparecerá del listado del disquet. Contestando N, de "no", invalidaremos el haber entrado en REMOVE y nuestro archivo no será ya destruido.

Practicar esta facilidad, borrando MIO.TEXT de MYDISK1:; con el disquet MYDISK2: en la segunda unidad borrar también MIISIMO. TEXT.

Comprobar que todos los archivos anteriores han sido destruidos efectuando el listado de sus correspondientes disquets.

Con MYDISK1: y MYDISK2: de nuevo en las unidades uno y dos respectivamente, introducir en EDIT el siguiente programa:

```
PROGRAM ESFERA;
CONST
  PI=3.141592;
VAR
  UNIDADES:STRING; RADIO,VOLUMEN,AREA:REAL;
BEGIN
WRITELN('EL RADIO ,  POR FAVOR?');
READLN(RADIO);
WRITELN('EN QUE UNIDADES?');
READLN(UNIDADES);
VOLUMEN:=4/3*PI*RADIO*RADIO*RADIO;
AREA:=4*PI*RADIO*RADIO;
WRITELN;
WRITELN('VOLUMEN=',VOLUMEN,' ',UNIDADES,' AL CUBO');
WRITELN('AREA=',AREA,' ',UNIDADES,' AL CUADRADO')
END.
```

Salir del nivel EDIT tecleando Q,U. Correr el programa anterior con el comando R(UN del nivel COMMAND. Si todo funciona correctamente (si el compilado da error volver a EDIT tecleando una E y modificar el programa para que ponga exactamente lo mismo que el PROGRAM ESFERA anterior), nos saldrá en pantalla, introduciendo un radio igual a 3.5 y unidades igual a metros, el siguiente mensaje:

```
EL RADIO ,  POR FAVOR?
 3.50000
EN QUE UNIDADES?
METROS

VOLUMEN= 1.79594E2 METROS AL CUBO
AREA= 1.53938E2 METROS AL CUADRADO
```

Después de haber corrido el programa ESFERA vayamos al nivel FILER tecleando una F. Tecleemos L de LISTING: en el listado de MYDISK1: (RAM:), (teclear MYDISK1: (RAM:) y oprimir RETURN), aparecen SYSTEM.WRK.TEXT y SYSTEM.WRK.CODE. Son las versiones visualizable o imprimible y ejecutable del programa introducido en EDIT.

Execute

Volvamos de nuevo a COMMAND, tecleando una Q. Una vez en el nivel COMMAND teclear una X de EXECUTE. Nos sale la pregunta:

EXECUTE WHAT FILE? Respondamos tecleando "SYSTEM.WRK.", la terminación CODE no es necesario teclearla, la máquina la sobreentiende. Tendremos el mismo resultado que cuando corríamos el programa, sólo que ahora nos ahorramos el compilado; todos los programas CODE se suponen compilados, en caso contrario no son ejecutables.

Save

Si deseamos almacenar copias tanto de texto como ejecutable de nuestro programa esfera, podemos usar la facilidad TRANSFER del siguiente modo:

```
TRANSFER?  SYSTEM.WRK.TEXT     oprimir RETURN
TO WHERE?  SAR:ESFERA.TEXT     oprimir RETURN
```

Tecleando de nuevo en el nivel FILER, T de TRANSFER:

```
TRANSFER?  SYSTEM.WRK.CODE     oprimir RETURN
TO WHERE?  SAR:ESFERA.CODE     oprimir RETURN
```

(Suponiendo, p. ej., SAR: en la unidad número dos)

Este sistema es realmente más largo de lo necesario. Dentro del nivel FILER existe la facilidad S(AVE que guarda a un tiempo las versiones de texto y de código del archivo de trabajo (caso de que haya sido compilado y existan las dos versiones, en otro caso sólo guarda la versión de texto), sin necesidad de teclear las terminaciones TEXT y CODE que son añadidas automáticamente. Por ejemplo, supongamos que deseamos pasar SYSTEM.WRK en sus dos versiones al disquet SAR:. Tecleando una S de SAVE en el nivel FILER, tendremos:

```
SAVE AS?  SAR:ESFERA  oprimir RETURN
```

A continuación sale un mensaje según el cual nuestro SYSTEM.WRK.TEXT y nuestro SYSTEM.WRK.CODE se hallan ahora en el disquet SAR:, con nombres ESFERA.TEXT y ESFERA.CODE, respectivamente.

Vayamos al nivel COMMAND tecleando una Q. La secuencia, X, SAR:ESFERA, RETURN nos da, mediante la facilidad EXECUTE, la posibilidad de ejecutar el programa ESFERA sin necesidad de meterlo en EDIT, siempre y cuando exista la versión de código de ESFERA en el disquet SAR:.

Date

Dentro del nivel FILER, D(ATE, sirve para actualizar la fecha. Tecleando una D dentro del nivel FILER, aparece el mensaje:

```
DATE SET  1..31 - JAN..DEC  -  00..99
TODAY IS 25- JUL- 84
NEW DATE?
```

Ateniéndonos al formato de la primera línea, tecleamos entonces un número entre 1 y 31 para el día del mes; las tres primeras letras del mes en inglés; los dos dígitos últimos del año en que nos encontramos. Las tres cosas separadas por guiones. Oprimimos luego RETURN. Los archivos que creemos mientras la fecha no sea modificada de nuevo, tendrán en el listado la fecha introducida. (En IBM—PC esta posibilidad se presenta nada más poner en marcha el sistema.)

Bad blocks

Dentro del nivel FILER teclear una B, de B(AD BLOCKS. Nos sale la pregunta:

```
BAD BLOCK SCAN OF WHAT VOL?
```

Que nos pregunta qué volumen debe revisarse para ver si existe algún bloque dañado, por huellas, raspaduras, polvo, ... Tecleando el nombre del disco a revisar y oprimiendo RETURN, nos sale el mensaje:

```
SCAN FOR 272 BLOCKS? (Y/N)
```

A este mensaje el usuario debe responder tecleando una Y, de "yes", o una N, de "no", y oprimir RETURN.

En el primer caso se explorará el disco completo, en el segundo tendremos que teclear luego el número de los bloques que deseamos explorar.

Al final aparecerá un mensaje del tipo:

```
0 BAD BLOCKS
```

en cuyo caso no hay ningún bloque dañado.

También puede aparecer un mensaje del tipo:

```
BLOCK 226 IS BAD
BLOCK 232 IS BAD        (/)
```

esto nos informa que los bloques 226 y 232 están en mal estado.

Examine

El comando eX(amine se activa desde el nivel FILER tecleando una "X". EXAMINE, permite marcar los bloques dañados de modo que el usuario no pueda escribir en ellos.

Supongamos que con el comando del nivel FILER, "B(adblocks", hemos obtenido la información (/). Al invocar el comando eX(amine nos aparecerá entonces un mensaje del tipo:

```
FILE(S) ENDANGERED:
MIO.TEXT  225  238        (X)
FIX THEM?
```

El sistema nos avisa que los bloques a examinar forman parte del fichero MIO.TEXT y nos pregunta si fija los posibles bloques dañados.

Si respondemos "N", la orden queda anulada. Si respondemos "Y", los bloques 226 y 232 habrán sido fijados y el archivo MIO.TEXT destruido, en cambio en el directorio del disquet, aparecerán como:

```
BAD.00226.BAD
BAD.00232.BAD
```

Si no existe ningún archivo en los bloques dañados, entonces en lugar del desplegado (X), al invocar eX(amine, nos aparecerá:

```
EXAMINE BLOCKS ON WHAT VOL?
```

respondemos MYDISK1: y oprimimos RETURN.

Nos preguntan entonces:

```
BLOCK RANGE?
```

respondemos 226-232 y oprimimos RETURN.

Junto a otras informaciones sale la pregunta:

MARK BAD BLOCKS? (Y/N)

Si se responde con una "N" se anulará la orden. Si se responde con una "Y", los bloques dañados quedarán marcados y aparecerá finalmente el mensaje:

BAD BLOCKS MARKED.

Change

El comando C(hange), que se activa desde el nivel FILER tecleando una C, permite cambiar de nombre a los ficheros y a los discos sin más que responder al mensaje:

CHANGE WHAT FILE?

el nombre del disquet o del archivo completos. Ejemplos:

MYDISK: para cambiar el nombre de MYDISK:
MYDISK: MIO.TEXT para cambiar el nombre de MIO.TEXT

oprimiendo RETURN, sale el mensaje:

CHANGE TO WHAT?

El usuario debe responder con los nuevos nombres, del disquet, o del fichero, según se trate.

Krunch

El comando K(runch se activa desde el nivel FILER oprimiendo una K. Permite compactar el disco cuyo nombre se teclee, poniendo a nuestra disposición el máximo número de bloques sin usar, para escribir archivos en ellos. Este comando debe llamarse siempre que se efectúe el borrado, mediante REMOVE, de algún fichero, ya que nos permitirá usar el hueco que ha dejado, de un modo no fragmentado, formando un solo bloque con el resto del espacio no usado.

Identificadores de volumen en U.C.S.D. Pascal

Con el fin de que el sistema pueda distinguir entre identificador de volumen e identificador de fichero, los primeros deben acabar en ":".

U.C.S.D. permite identificar de este modo no sólo a los discos, sino también a los diversos dispositivos periféricos.

CONSOLE: "pantalla"
PRINTER: "impresora"
REMOUT; REMIN: } "terminales remotos o línea de teléfono"

Tecleando una V, de V(olumen, en el nivel FILER, se visualiza una lista de identificadores de volumen disponibles. Los números que aparecen en la izquierda de la lista son los números lógicos de las unidades de E/S. Cualquier unidad, puede ser referenciada por "#n:", donde n es el número lógico situado a la izquierda del identificador de volumen correspondiente.

Generalmente Console: puede ser referenciado por #1:

El disquet de la unidad uno por #4:
El disquet de la unidad dos por #5:
Printer: por #6:

Opciones del compilador U.C.S.D. Pascal

Existen ciertas instrucciones que hacen que el compilador trabaje de un modo o de otro; por ejemplo, poniendo la instrucción (* $G + *) se puede usar la sentencia GOTO dentro de un programa sin que el compilado dé error. El estado normal de la máquina es el (* $G – *), aun cuando no se haya explicitado dicha declaración; por tanto, sin la sentencia (* SG + *), si usamos GOTO, el compilado dará error.

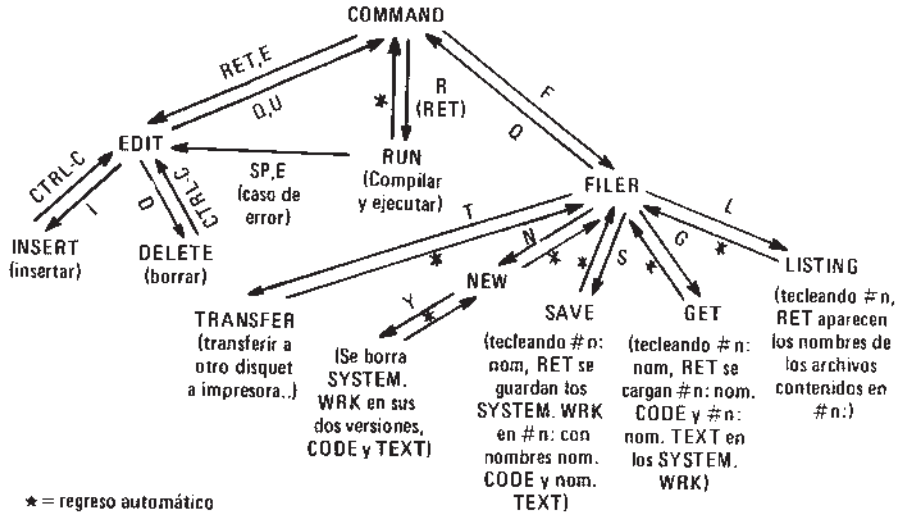
Las opciones del compilador se declaran mediante sentencias del tipo (* $ opción + *) ó bien (* $ opción – *). Aparte de la opción GOTO, se usan en el presente libro las opciones: (* $S + *), para desdoblamiento de memoria en programas largos, el estado normal si no se declara es (* $S – *). (* $L– *), para que se forme un archivo con el listado del programa. En ausencia de dicha declaración, el compilador se halla en el estado (* $L + *).

Al compilador se le puede ordenar que no genere el código que comprueba el resultado de una operación E/S, suministrándole la instrucción (* $I – *); entonces, aunque haya una operación errónea, el compilador no nos lo dirá. La sentencia (* $I – *) puede ser anulada a lo largo del programa en el momento que se dé la opción al compilador (* $I + *).

El uso de (* $I – *) va ligado a una función U.C.S.D., la función "IORESULT", que está permitida con dicha opción del compilador y que da como resultado un número igual a cero si la operación anterior no dio error y distinto de cero si es errónea. Esto permite usar en los programas operaciones que pueden ser o no erróneas y según el resultado de IORESULT bifurcar o no a otras sentencias del programa. Ver el programa "ACTUALIZA" del capítulo de ficheros.

Se verán en el capítulo dedicado a la confección de unidades U.C.S.D. otras opciones del compilador relativas al uso de ficheros de inclusión.

ESQUEMA DEL SISTEMA OPERATIVO MINIMO



TECLADOS DE ALGUNAS MAQUINAS QUE USAN U.C.S.D. PASCAL

Tecla usual	Significado	Apple II	IBM-PC (NCI)	TRS-80	TERAK 8510	Terminal ADM3-A	Terminal HAZELTINE 1500	Terminal SORCIO, 120
ESC	Escape	ESC	ESC	CTRL/DA + B	ESC	ESC	ESC	ESC
RET	Cambio de línea; RET del desplegado.	RET	←	ENTER	RET	RETURN	RETURN	RETURN
DEL	Borrar línea anterior en Insert.	CTRL-X	DEL	CTRL/DA + U	DEL	RUBOUT	SHIFT + DEL	RUBOUT
ETX	"Enter text"	CTRL-C	CTRL-C	CTRL/DA + C	ETX	CTRL-C	CTRL-C	CTRL-C
TAB	Avanza el cursor 10 caracteres en Edit.	CTRL-I	⇌	CTRL/DA + I	TAB ó CTRL-I	CTRL-I	TAB	TAB
EOF	Fin de entrada de texto en input.	CTRL-C	CTRL-C	CTRL/DA + C	ETX	CTRL-C	CTRL-C	CTRL-C
CTRL-S	Parar	CTRL-S	CTRL-S	CTRL/DA + B	DC3 ó CTRL-S	CTRL-S	CTRL-S	CTRL-S
BREAK	Romper	CTRL-∂	CTRL-↑-BREAK	CTRL/DA + B	CTRL + 9	CTRL + B	BREAK o CTRL + ∂	BREAK
↑	Avanzar el cursor en Edit hacia arriba.	CTRL-O	↑	↑	↑	↑	CTRL-L	↑
↓	Idem hacia abajo	CTRL-L	↓	↓	↓	↓	CTRL-K	↓

TECLADOS DE ALGUNAS MAQUINAS QUE USAN U.C.S.D. PASCAL (Continuación)

Tecla usual	Significado	Apple II	IBM-PC (NCI)	TRS-80	TERAK 8510	Terminal ADM3-A	Terminal HAZELTINE 1500	Terminal SORC IQ, 120
←	Idem a la izquierda.	↓	↓	←	↓	←	BACKSPACE	↓
→	Idem a la derecha.	↑	↑	→	→	→	CTRL-P	→
]	Escribe "]"	SHIFT-M	]	SHIFT →				
[	Escribe "["	CTRL-K	[	SHIFT ←				

Nota: El fabricante puede en cualquier momento cambiar el teclado presente, en cualquiera de los modelos.

14

Procedimientos y funciones segment. Ficheros de inclusión. Unidades

Procedimientos y funciones segment

Los procedimientos y funciones segment no son diferentes al resto de los procedimientos y funciones, sus declaraciones son idénticas, sólo varía su encabezamiento que debe ir precedido del vocablo "SEGMENT".

Ejemplos:

```
SEGMENT FUNCTION FF(X,A,B,C:REAL);
BEGIN
      _____
      _____
      _____
END;

SEGMENT PROCEDURE EJES(K:REAL);
BEGIN
      _____
      _____
      _____
END;
```

Los procedimientos y/o funciones declarados de este modo no ocupan lugar en memoria más que cuando se están corriendo. La ventaja de su utilización es la posibilidad de redactar programas largos en la memoria disponible.

Ficheros de inclusión

Puede desearse tener una porción de programa almacenada en un fichero de texto (por ejemplo, por ser una secuencia de sentencias común a varios programas), de modo que el texto almacenado sea incluido en un lugar determinado del programa fuente, con tan sólo poner en dicho sitio una sentencia de inclusión del fichero.

Supongamos que tal fichero se denomina "AUXFICHERO.TEXT", entonces el programa:

```
PROGRAM EJEMPLIFICANDO;
VAR A,B,C,D: INTEGER;
BEGIN
(* $I AUXFICHERO.TEXT *)
```

sería tratado como si el texto del archivo, "AUXFICHERO.TEXT" se encontrara justo en el lugar que ocupa la orden al compilador "(* $I AUXFICHERO.TEXT *)".

Debe aclararse que U.C.S.D. Pascal no permite trabajar con ficheros de inclusión que a su vez contengan a otros ficheros mediante la opción (* $I ... *).

Además, debe ponerse también de manifiesto que este mecanismo requiere un área de memoria considerable (buffer). Puede no ser posible su uso en microordenadores de menos de 48k.

Unidades

Una unidad es un conjunto de procedimientos y/o funciones que pueden utilizarse como si se declararan en el programa fuente. De hecho, se hallan *compilados* y almacenados en lugar aparte (son como ficheros de inclusión "compilados", lo que permite ahorrar, en el compilado del programa fuente, el tiempo de la compilación del fichero de inclusión). Así como los ficheros de inclusión son ficheros de texto ordinario, las unidades poseen declaraciones específicas y su contenido puede dividirse en las siguientes partes:

1) INTERFACE: Contiene las declaraciones de tipo VAR, CONST, TYPE, cabeceras de procedimientos y/o funciones "públicas", esto es, usables, "llamables", en el programa fuente que hospedará la unidad.
2) IMPLEMENTACION: Contiene la parte "privada" de la unidad: declaración de variables locales a los procedimientos y funciones de la unidad, contenido de los procedimientos y funciones de la unidad, cuya cabecera debe ahora omitir cualquier declaración de parámetros.
3) Una última parte que se ejecuta previamente a cualquier procedimiento o función de la unidad.

Las unidades deben llevar también la orden al compilador (* $S + *).

En ordenadores con suficiente memoria puede no ser necesaria la declaración (* $S + *).

El llamado de las unidades tiene la forma USES (*$U UNIDAD.CODE *), donde UNIDAD es el nombre de la unidad.

Veamos, a continuación, cómo convertir los procedimientos del programa OPERPOL en una unidad de nombre POLUFF. Tener en cuenta que a los procedimientos y funciones ya conocidos se ha añadido una función, "FUCTION VALOR (X:VECTOR; Z:REAL): REAL;", que halla el valor del polinomio X, en X = Z. La comprensión, de la redacción de dicha función, no tiene ninguna dificultad considerando:

$$X[1] + X[2] * Z + X[3] * Z^2 + \ldots +$$
$$+ X[n] * Z^{n-1} + X[n+1] * Z^n =$$
$$= (\ldots((X[n+1] * Z + X[n]) * Z +$$
$$+ X[n-1]) * Z \ldots) * Z + X[1].$$

(ξ)

```
(*$S+*)
UNIT POLUFF;
  INTERFACE (*PARTE PUBLICA DEL PROGRAMA*)
    TYPE VECTOR=ARRAY[1..20] OF REAL;
    PROCEDURE LEE(VAR X:VECTOR);
    FUNCTION GRADO(X:VECTOR):INTEGER;
    FUNCTION VALOR(X:VECTOR; Z:REAL):REAL;
    PROCEDURE OP( P1,P2:VECTOR; CH:CHAR; VAR R:VECTOR);
IMPLEMENTATION (*PARTE PRIVADA DEL PROGRAMA*)
  VAR
    S,T:REAL; PI,PG,PK,K,G,GRP1,GRP2:INTEGER; V:VECTOR;
PROCEDURE LEE;(*NO DECLARAR AQUI  PARAMETROS*)
BEGIN
  WRITELN('TECLEA EL POLINOMIO');
  G:=0;
  WHILE NOT EOLN DO
  BEGIN
    READ(S);
    G:=G+1;
    X[G]:=S
  END;
```

```
  READLN;
  WRITELN;
  FOR G:=G+1 TO 20 DO X[G]:=0
END;
FUNCTION GRADO;
BEGIN
  K:=20;
  REPEAT
  K:=K-1
  UNTIL (X[K+1]<>0) OR (K=0);
  GRADO:=K
END;
FUNCTION VALOR;
BEGIN
  K:=GRADO(X); T:=Z*X[K+1]+X[K];
  REPEAT
    K:=K-1;
    T:=Z*T+X[K]
  UNTIL K=1;
  VALOR:=T
END;
PROCEDURE OP;
BEGIN
  GRP1:=GRADO(P1); GRP2:=GRADO(P2);
  CASE CH OF
  '+': FOR PG:=1 TO 20 DO
         R[PG]:=P1[PG]+P2[PG];
  '-': FOR PG:=1 TO 20 DO
         R[PG]:=P1[PG]-P2[PG];
  '*': BEGIN
         IF GRP1 + GRP2 <=19 THEN
           BEGIN
             FOR PG:= 1 TO GRP2+1 DO
              FOR PI:= 0 TO GRP1 DO R[PG+PI]:=0;
             FOR PG:=1 TO GRP2+1 DO
              FOR PI:= 0 TO GRP1 DO
                R[PG+PI]:=R[PG+PI]+P1[PI+1]*P2[PG];
             PK:=GRP1+GRP2;
             IF PK<19 THEN
               BEGIN
                 PI:=1;
               REPEAT
                 PI:=PI+1;
                 R[PK+PI]:=0
               UNTIL PK+PI=20
             END;
         END;
    END;
  'D','M': IF GRP1>=GRP2 THEN
           BEGIN
             PK:=GRP1-GRP2;
             PI:=0; V:=P1;
             REPEAT
               R[PK-PI+1]:=V[GRP1+1-PI]/P2[GRP2+1];
               FOR PG:=PK-PI+1 TO GRP1+1-PI DO
                 V[PG]:=V[PG]-R[PK+1-PI]*P2[PG-PK+PI];
```



```
               PI:=PI+1
             UNTIL PI=PK+1;
             IF CH='D' THEN
               FOR PG:=PK+2 TO 20 DO
                  R[PG]:=0;
             IF CH='M' THEN R:=V;
           END
           ELSE
           BEGIN
             IF CH='M' THEN R:=P1;
             IF CH='D' THEN FOR PG:=1 TO 20 DO
               R[PG]:=0;
             END;
    END;(*CASE*)
 END;
BEGIN
END.
```

Veamos ahora cómo confeccionar dos programas, equivalentes a OPERPOL y MULIDIVPOL, hospedando en ellos la unidad POLUFF.

Tener en cuenta que dichos programas deben ser corridos mediante la secuencia: C, L, X, correspondiente a compiler, linker y execute.

En versiones anteriores a la IV.0, la declaración USES necesita usar explícitamente el Linker para hospedar la unidad. En la versión IV.0 esto no es necesario y puede sustituirse la secuencia C,L,X por la R de "RUN".

(ξ)

```
(*$S+*)
PROGRAM OPERPOL;
USES (*$U MYDISK1:POLUFF.CODE*) POLUFF;
VAR
 OPER:CHAR; P,Q,D:VECTOR; J:INTEGER;
BEGIN
LEE(P); WRITELN('TECLEA OPERACION');
READLN(OPER); WRITELN;
LEE(Q);
OP(P,Q,OPER,D);
FOR J:=1 TO 20 DO WRITELN(D[J]);
END.
```

(ξ)

```
(*$S+*)
PROGRAM MULIDIVPOL;
USE(*$U MYDISK1:POLUFF.CODE*) POLUFF;
PROCEDURE MCD(P1,P2:VECTOR;  VAR MD:VECTOR);
VAR
  RESTO:VECTOR; I,GRP1,GRP2,GR:INTEGER;
BEGIN
  GRP1:=GRADO(P1); GRP2:=GRADO(P2);
  IF GRP1>=GRP2 THEN
  BEGIN
    OP(P1,P2,'M',RESTO);
```

```
    GR:=GRADO(RESTO);
    IF (GR=0) AND (RESTO[1]=0) THEN
    BEGIN
      (*TOMAMOS M.C.D. CON COEF. DE MAYOR GRADO
        IGUAL A UNO*)
      FOR I:=1 TO GRP2+1 DO
       P2[I]:=P2[I]/P2[GRP2+1];
      MD:=P2
    END
    ELSE
      BEGIN
        MCD(P2,RESTO,MD)
      END
  END
  ELSE
  MCD(P2,P1,MD)
END;
PROCEDURE MCM(P1,P2:VECTOR; VAR M:VECTOR);
VAR
  I,GRW:INTEGER; V,W:VECTOR;
BEGIN
  MCD(P1,P2,W); OP(P1,W,'D',V); OP(V,P2,'*',W);
  (*HACEMOS QUE COEF. DE TERM. GRADO MAYOR SEA UNO*)
  GRW:=GRADO(W); FOR I:=1 TO 20 DO W[I]:=W[I]/W[GRW+1];
  M:=W
END;
BEGIN
LEE(P);
LEE(Q);
MCD(P,Q,D); MCM(P,Q,N);
WRITELN('  MCD              MCM');
FOR J:=1 TO 20 DO WRITELN(D[J],'     ',N[J]);
WRITELN(GRADO(P),'/',GRADO(Q))
END.
```

La unidad Poluff, puede ayudarnos a confeccionar otros muchos programas además de los ya conocidos.

Supongamos, por ejemplo, que deseamos hallar un polinomio de raíces, r1, r2, r3, ..., rn. Donde ri $\in$ R, i $\in \{1, ..., n\}$ y n $\leqslant$ 19. Obviamente, la solución más sencilla a dicho problema es el polinomio:

$$(X - r1)\,(X - r2)\,(X - r3)\,...\,(X - rn)$$

Solución que es sencillamente el producto de n polinomios Vi, i $\in \{1, ..., n\}$ de la forma:

$$Vi\,[1] = -ri; \quad Vi\,[2] = 1; \quad Vi\,[J] = 0; \quad J \in \{3, ..., 20\}$$

Con estas consideraciones, es fácil entender el proceso repetitivo que, sustancialmente, constituye el siguiente programa:

(ξ)

```
PROGRAM POLRAICES;
(*CONSTRUYE UN POLINOMIO DADAS SUS RAICES*)
USES (*$U MYDISK1:POLUFF.CODE*) POLUFF;
VAR
  RAIZ:ARRAY[1..19] OF REAL;
  V,S,L:ARRAY[1..20] OF REAL;
  I,J,K:INTEGER;
BEGIN
  I:=1;
  S[1]:=1; FOR K:=2 TO 20 DO S[K]:=0;
  WRITELN('ESCRIBE LAS RAICES Y OPRIME RETURN');
  REPEAT
    READ(RAIZ[I]);
    V[1]:=-1*RAIZ[I]; V[2]:=1;
    FOR J:=3 TO 20 DO V[J]:=0;
    OP(S,V,'*',L);
    S:=L; I:=I+1
  UNTIL (EOLN) OR (I>19);
  READLN;
  WRITELN('RESULTADO=');
  FOR I:= 1 TO 20 DO WRITELN (S[I])
END.
```

Recordemos ahora la fórmula del polinomio de interpolación de Lagrange, que halla el polinomio de menor grado que pasa por los puntos (X1, Y1), (X2, Y2), ..., (Xn, Yn); Xi, Yi $\in$ R, i $\in \{1, ..., n\}$.

$$P = \sum_{k=1}^{n} Yk \cdot \prod_{\substack{j=1 \\ j \neq k}}^{n} \frac{(X - Xj)}{(Xk - Xj)} = \sum_{k=1}^{n} \frac{Yk}{Pk\,(Xk)} \cdot Pk$$

Donde Pk es el polinomio de raíces Xi, i $\in \{1, ..., n\}$, i $\neq$ k.

Esto nos sugiere la idea de transformar POLRAICES en un procedimiento, para confeccionar un nuevo programa que halle el polinomio de interpolación de una serie de puntos dados. Ya que trabajamos a lo más con polinomios de grado 19, tendremos que trabajar con a lo más 19 puntos, para hallar el polinomio de interpolación que pase por dichos puntos.

(ξ)

```
PROGRAM INTERPOLACION;
(*CONSTRUYE EL POLINOMIO DE INTERPOLACION DE LAGRANGE*)
USES (*$U MYDISK1:POLUFF.CODE*) POLUFF;
TYPE RAIZ= ARRAY [1..19] OF REAL;
VAR
  V,S,L,A,T,SOLUCION,B,Q,X,Y:VECTOR;
  R:RAIZ;
  P:REAL; N,NU,I,J,K:INTEGER;
PROCEDURE POLRAICES(RZ:RAIZ; VAR W:VECTOR; NUM:INTEGER);
BEGIN
  I:=1;
  S[1]:=1; FOR K:=2 TO 20 DO S[K]:=0;
  REPEAT
    V[1]:=-1*RZ[I]; V[2]:=1;
    FOR K:=3 TO 20 DO V[K]:=0;
```

```
      OP(S,V,'*',L);
      S:=L; I:=I+1
    UNTIL I>NUM;
    W:=S
  END;
  BEGIN
    WRITELN('NUMERO DE PUNTOS?');
    READLN(N); NU:=N-1;
    WRITELN('TECLEA SUS COORDENADAS');
    FOR I:= 1 TO N DO READLN(X[I],Y[I]);
    FOR I:= 1 TO 20 DO SOLUCION[I]:=0;
    J:=1;
    REPEAT
      FOR I:= 1 TO N DO
        BEGIN
          IF I<J THEN R[I]:=X[I];
          IF I>J THEN R[I-1]:=X[I]
        END;
      POLRAICES(R,T,NU);
      Q[1]:=Y[J]/VALOR(T,X[J]);
      FOR K:= 2 TO 20 DO Q[K]:=0;
      OP(T,Q,'*',A);
      OP(A,SOLUCION,'+',B);
      SOLUCION:=B;
      J:=J+1
    UNTIL J=N+1;
    WRITELN('EL POLINOMIO DE INTERPOLACION ES:');
    FOR J:=1 TO N DO WRITELN(SOLUCION[J])
  END.
```

Unidades intrínsecas

Cada vez que se hospede una unidad en un programa, ejecutar el programa requiere el proceso C,L,X. Obviamente, sería más cómodo, en lugar de la combinación C,L,X, usar, como en programas anteriores, el comando RUN. Esto es posible si la unidad se convierte en una unidad "intrínseca", esto es, almacenada en SYSTEM LIBRARY.

Supongamos que deseamos introducir UNIT POLUFF en SYSTEM. LIBRARY, para ello necesitamos un disquet que posea las rutinas U.C.S.D. Pascal, de LIBRARY.CODE. Supongamos que dicho disquet es el MYDISK3:.

Cambiemos la cabecera de la unidad POLUFF para que quede como:

```
UNIT POLUFF; INTRINSIC CODE 25 DATA 26;
```

La versión que se expone de SISTEM.LIBRARY es la II.1. La última versión, IV.0, no necesita declaraciones especiales en la cabecera de sus unidades intrínsecas, estas deben construirse de modo totalmente análogo a las unidades regulares.

La declaración DATA sólo es necesaria en las unidades que han declarado variables en INTERFACE, que no son parámetros de procedimientos o funciones públicas. Como su nombre indica, almacena datos usables en el programa. La utilidad de los números 25 y 26 se verá más adelante.

Supongamos ahora que en la unidad uno se halla el disco del sistema, MYDISK1:, (en el volumen #4:), que posee en el listado de su directorio a SYSTEM.LIBRARY. Supongamos también que en la unidad 2, (volumen #5:), tenemos MYDISK3: en cuyo listado figura LYBRARY.CODE. Desde el nivel COMMAND tecleemos una X de EXECUTE, aparecerá el mensaje:

```
EXECUTE WHAT FILE?
```

tecleemos:

```
MYDISK3:LIBRARY
```

saldrá el mensaje:

```
PASCAL SYSTEM LIBRARIAN
OUTPUT CODE FILE →
```

Supongamos que deseamos trasladar SYSTEM.LIBRARY a MIODISK:NEW.LIBRARY, poniendo el disquet en cualquier unidad que no sea la que posee el disco del sistema (volumen #4:), teclearemos:

```
MIODISK:NEW.LIBRARY
```

(Perfectamente la respuesta podría haber sido MYDISK1:SYSTEM. LIBRARY.)

Nos preguntan ahora:

```
LINK CODE FILE →
```

Puesto que deseamos pasar las rutinas de SYSTEM.LIBRARY a NEW. LIBRARY, la respuesta adecuada será:

```
MYDISK1:SYSTEM.LIBRARY
```

(De modo abreviado podía haberse tecleado un *.)

En estos momentos aparecerá en pantalla el mensaje:

```
SLOT TO LINK AND SPACE, = FOR ALL, ? FOR SELECT.
N(EW FILE, Q(UIT, A(BORT
LINK CODE FILE – – MYDISK1: SYSTEM.LIBRARY

 0–(30) LONGINTI      2452      8–      0
 1–(31) PASCALIO      1238      9–      0
 2–(29) TRANSCEND     1168     10–      0
 3–(27) TRADUFF        662     11–      0
 4–                      0     12–      0
 5–                      0     13–      0
 6–                      0     14–      0
 7–                      0     15–      0
OUTPUT CODE FILE – – MIODISK:NEW.LIBRARY
```

Tecleando un = y un espacio se copian todos los items en NEW. LIBRARY. (En versiones más recientes se sustituye = por E, de "Every".)

Tecleando un número de 1 a 15 (correspondiente a las columnas de la izquierda) se copia el slot correspondiente en NEW.LIBRARY, sin más que teclear un espacio a continuación.

Tecleando ?, sale la pregunta COPY SLOT O?, tecleando un espacio, se copia en NEW.LIBRARY el slot indicado.

Tecleando una A se aborta todo lo realizado.

Tecleando una N, nos sale la pregunta:

```
LINK CODE FILE – →
```

Supongamos que la unidad nueva que deseamos añadir es POLUFF. Tecleamos entonces:

```
MYDISK1:POLUFF y un espacio.
```

Sale entonces el mensaje:

```
LINK CODE FILE – → MYDISK: POLUFF.CODE
 0–      0      8–                    0
 1–      0      9–                    0
 2–      0     10–(25) POLUFF      1280
 3–      0     11–(26) DATA         102
 4–      0     12–                   0
 5–      0     13–                   0
 6–      0     14–                   0
 7–      0     15–                   0
```


Supongamos además que no deseamos que ocupe el lugar 10 sino el 6, entonces tecleamos un 10. En la pantalla se visualiza el mensaje:

```
10
SLOT TO LINK INTO?
```

Contestaremos tecleando un 6 y un espacio. Análogamente podemos copiar el slot 11 en el lugar 7.

Supongamos que ya hemos copiado en NEW.LIBRARY todos los slots y nuevas unidades deseados. Tecleemos una Q de Q(UIT, aparece el mensaje:

```
NOTICE?
```

Esto nos ofrece la oportunidad de poner una nota de copyright en nuestra librería, por ejemplo:

```
COPYRIGHT 1984 POLUFFLIBRARY BY ME
```

Finalmente, oprimiendo RETURN nos encontraremos de nuevo en el nivel COMMAND.

Entonces (si lo deseamos), podemos borrar del disquet MYDISK1: nuestro antiguo SISTEM.LIBRARY, y transferir la librería más completa, NEW.LIBRARY, a MYDISK1:. Luego, podemos cambiar el nombre de NEW.LIBRARY de nuevo a SYSTEM.LIBRARY, usando para ello la facilidad C(HANGE. Antes de usar la nueva librería el sistema debe ser reinicializado.

A partir de ahora, con la unidad POLUFF convertida en una unidad intrínseca, la sentencia USES (* $U MYDISK1: POLUFF.CODE *) POLUFF; debe omitir la orden al compilador, quedando reducida a USES POLUFF;.

Los programas que utilizan unidades intrínsecas no necesitan ser ejecutados mediante la secuencia, C (Compiler), L (Linker), X (Execute), basta con correrlos, "oprimir R en el nivel COMMAND".

El siguiente programa ejemplifica una llamada a la unidad POLUFF, ya metida en librería:

(ξ)

```
PROGRAM RUFFINI;
(*SACA LAS RAICES ENTERAS DE UN POLINOMIO
COMPRENDIDAS ENTRE -A[1]-1 Y A[1]+1, SIENDO A[1]
EL TERMINO INDEPENDIENTE DEL POLINOMIO*)
USES POLUFF;
VAR
```

```
  V:REAL; I,J,K:INTEGER; POL:VECTOR;
BEGIN
  REPEAT
    LEE(POL);
    WRITELN('RAICES ENTERAS ENTRE -',ABS(POL[1])+1:1,
    ' Y ',ABS(POL[1])+1:1,':');
    I:=ABS(TRUNC(POL[1]))+1;
    V:=VALOR(POL,0);
    IF V=0 THEN WRITELN('0');
    FOR K:= 1 TO I DO
    BEGIN
      V:=VALOR(POL,K);
      IF V=0 THEN WRITELN(K);
      J:=K*(-1);
      V:=VALOR(POL,J);
      IF V=0 THEN WRITELN(J)
    END;
  UNTIL EOF
END.
```

Los programas de este libro han sido redactados en un ordenador APPLEII; dicho ordenador tiene implementadas las funciones trascendentes SIN, COS, ATAN, LOG, LN, EXP, SQRT, en una unidad intrínseca llamada TRANSCEND. Esto explica por qué dicha declaración ha aparecido en programas anteriores a este capítulo.

A continuación se añaden dos capítulos que ejemplifican el uso de otras dos unidades intrínsecas de APPLEII: APPLESTUFF, responsable de la generación de números aleatorios y sonido, y, por otro lado, TURTLEGRAPHICS responsable de la graficación en pantalla.

APENDICE I

Sonido y números aleatorios en Apple II

• Los programas de este capítulo sólo son válidos para APPLE II, los usuarios de otras máquinas deben traducirlos al lenguaje propio de su máquina si desean usarlos.

Si en nuestro programa, justo tras la declaración del encabezamiento, hemos declarado previamente, "USES APPLESTUFF;", entonces, NOTE(a,b);, donde a,b:INTEGER;, hace sonar un pitido de tono relacionado con a y duración indicada por b.

Programas ejemplo de uso del procedimiento NOTE:

•

```
PROGRAM SONIDO1;
USES APPLESTUFF;
VAR
  TONO,DURACION:INTEGER;
BEGIN
  DURACION:=100;
  FOR TONO:=20 TO 32 DO
    NOTE(TONO,DURACION)
END.
```

•

```
PROGRAM PIANOLA;
(*IMITA EL TECLADO DE UN PIANO*)
USES APPLESTUFF;
CONST
  DURACION=70;
VAR
  NOTA:CHAR; T(*YONO*):INTEGER;
BEGIN
  READ(NOTA);
  WHILE NOTA<>'*' DO
  BEGIN
```