UNIVERSIDAD POLITECNICA DE CANARIAS

INGENIERIA TECNICA DE TELECOMUNICACIONES

PROYECTO FIN DE CARRERA

TITULO: MODULA-2 EN EL ENTORNO MS-DOS, APLICACION EN EL CONTROL DE ESTABILIDAD DE UNA CENTRAL ELECTRICA

AUTOR: Fco. Javier García Martel

TUTOR: Sebastian Suárez Gil

JUNIO 1989

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2009

UNIVERSIDAD POLITECNICA DE CANARIAS

INGENIERIA TECNICA DE TELECOMUNICACIONES

PROYECTO FIN DE CARRERA

TITULO: MODULA-2 EN EL ENTORNO MS-DOS, APLICACION EN EL CONTROL DE ESTABILIDAD DE UNA CENTRAL ELECTRICA

AUTOR: Fco. Javier García Martel

TUTOR: Sebastian Suárez Gil

JUNIO 1989

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

INDICE

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

PROLOGO

Este texto es el resultado del estudio profundo de los recursos del MODULA - 2, lenguaje evolucionado de su antecesor PASCAL, en un entorno de trabajo ampliamente extendido como es el IBM/PC, MS/DOS, incidiendo en los aspectos que diferencian a este nuevo lenguaje respecto a otros.

Esto supone un soporte didáctico para los lectores que quieran acercarse, no sólo al MODULA - 2 o a la programación de alto nivel en general, sino a aquellos que quieran profundizar en el conocimiento del MS/DOS y del IBM/PC o compatibles.

La obra se ha dividido en doce temas, presentados de una forma progresiva e ilustrados con más de 60 módulos ejemplos, que se han ejecutado con los compiladores MODULA - 2 para IBM/PC de LOGITECH versiones V 1.10 y V 2.0, de forma que un seguimiento paulatino ayudándose de un ordenador personal, mostrará al lector las características del lenguaje desde sus primeros elementos hasta el final.

Como colofón, se ha incluído en un apéndice final, una aplicación práctica cuyo objetivo, es la implementación de un sistema de control de estabilidad para una central eléctrica, añadiendo a nuestro entorno las tarjetas PCL-712, PCLD-782 y PCLD-785 de US. Technologies, lo cual convierte a nuestro PC compatible en una poderosa herramienta de adquisición de datos, así como de control.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

INTRODUCCION

Puede considerarse al MODULA - 2, como un lenguaje evolucionado directamente de su predecesor PASCAL, ambos desarrollados por el profesor Niklaus Wirth.

Si bien el PASCAL ha alcanzado un notable éxito, es éste paradógicamente, el que hizo reflexionar a Wirth a cerca de la creación de un nuevo lenguaje.

Hay que tener en cuenta que el PASCAL, fue pensado como un lenguaje didáctico e instructivo, diseñado para la ensañanza de programación en colegios y universidades. El hecho de que se obtuviese un lenguaje de alto nivel de propósito general, estructurado, y de fácil aprendizaje, hizo que la industria y los profesionales de la informática, se fijasen en él.

Sin embargo, este lenguaje pensado para los estudiantes, presentaba ciertas limitaciones para los informáticos y para grandes desarrollos de software, tales como la imposibilidad de compilación separada (lo que dificulta el trabajo en diferentes equipos de programación), la ejecución de procesos de forma concurrente o la programación de bajo nivel.

Estas carencias , se intentaron suplir de una forma algo anárquica por los diferentes fabricantes de compiladores, lo que supuso confusión en la nomenclatura de los diferentes lenguajes PASCAL, además ocasionaba que, programas escritos en una determinada versión, no fuesen transportables a otros compiladores.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

MODULA - 2 extiende el PASCAL en un aspecto ascendente (con el concepto de módulo y de concurrencia) y en un aspecto descendente (proporcionando recursos para el acceso al bajo nivel de máquina).

- El concepto de módulo y en especial la posibilidad de la compilación separada de éstos, facilita el mantenimiento del software, lo que permite, que se puedan desarrollar proyectos de gran tamaño que incluyan a varios programadores o equipos de programación.

Es más fácil en principio desarrollar y corregir 20 módulos de 2.000 lineas que un programa de 40.000 lineas, de forma que, si se hace una corrección en un módulo, no es necesaria la recompilación del programa, sino la del módulo en cuestión, lo que supone un ahorro de tiempo significativo, sobre todo para programas de gran tamaño.

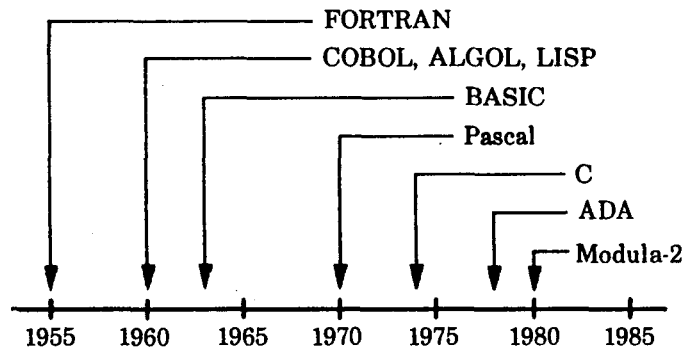
- MODULA - 2 desarrolla el concepto de concurrencia, permitiéndonos elaborar procesos que se ejecuten en pseudo concurrencia, es decir, compartiendo un único procesador entre ellos.

- El acceso al bajo nivel de máquina, permite la inserción de rutinas en código ensamblador que interactúan con el código compilado del lenguaje, también facilita el acceso a las funciones del sistema operativo a través de las interrupciones así como la manipulación de la memoria del ordenador mediante direccionamiento y acceso a ésta, lo que posibilita la modificación del contenido de los vectores de interrupción permitiendo crear manipuladores de interrupciones.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Con el MODULA - 2, por lo tanto, se ha conseguido un lenguaje de propósito general, modular, disciplinado, consistente y con una sintaxis sistemática que facilita el aprendizaje y una cómoda lectura del código para cualquier programador ajeno a él, combinando las mejores ideas de sus predecesores.

En la siguiente figura se pude ver la evolución en el tiempo de los principales lenguajes de programación.



En el apéndice final, se incluye la implementación de un sistema de control de estabilidad para una central eléctrica, añadiendo a nuestro entorno la tarjetas PCL-712, PCLD-782 y PCLD-785 de US. Technologies, cuyo manejo lo posibilita los recursos que M2 proporciona, convirtiendo a nuestro PC en un elemento capacitado para la adquisición de datos a la vez que los controla.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 1
INTRODUCCION AL LENGUAJE

1.1 FORMATO DE UN PROGRAMA MODULA - 2

Todos los Programas en MODULA - 2 comienzan con la palabra clave MODULE, seguida por el nombre que se le asigna al módulo y un punto y coma (;), esto constituye la cabecera del módulo.

```
MODULE NombreMod;                          Cabecera del MODULO

<lista de importaciones>                   Sección de declaración
<declaración de constantes>
<tipos de declaraciones>
<declaración de variables>
<declaraciones de procedimientos>

BEGIN                                      Cuerpo del programa
<Secuencia de sentencias>
END NombreMod.
```

A continuación se pone la sección de declaraciones, donde se declaran y definen los elementos usados en el programa, los elementos de esta sección son opciones y se pueden omitir dependiendo del tipo de programa.

La lista de importaciones dice al compilador los procedimientos a incluir en el programa, estos procedimientos externos que se encuentran en librerías estándar proporcionadas por el propio compilador, son importadas a través de la palabra reservada IMPORT. También el usuario puede implementar sus propias librerias e importar

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

procedimientos de éstas.

La declaración de las ctes. (CONST) se usa, para asignar nombres a constantes.

La declaración de tipos (TYPE), permite crear tipos de datos.

La sección de declaración de variables (VAR) se utiliza para declarar las variables antes de su uso.

La sección de declaración de procedimientos (PROCEDURE), es donde se definen las subrutinas que el programa utilizará posteriormente.

La palabra clave BEGIN señala el comienzo de la sección del módulo de programa, donde se localizan las sentencias, es esta parte la que se ejecuta cuando se corre el programa.

La última linea es donde se concluye el programa, consiste en la palabra reservada END, un identificador que debe coincidir con el nombre que se le asignó al módulo y un punto (.).

Seguidamente y a modo de ejemplo se incluye un programa sencillo en MODULA - 2 .

Modula-2/86 ejemplo1.MOD

```
1  MODULE Ejemplo1;
2    FROM Terminal IMPORT WriteString, WriteLn, Read;
3    VAR ch: CHAR;
4  BEGIN
5    WriteString('¡El programa funcionó! (pulse una tecla)');
6    WriteLn;
7    Read(ch);
8  END Ejemplo1.
```

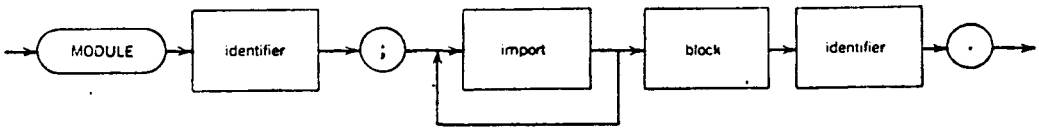
```
Modula-2/86 Compiler Version V 1.10

==)     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La primera linea del programa,señala el comienzo del mismo dándole nombre al programa, en nuestro caso ejemplo1.

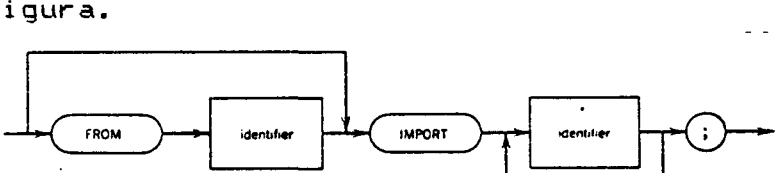
A continuación se muestra el esquema sintáctico de un módulo.

program module



En la linea siguiente se toman los procedimientos mencionados (Read,WriteString,WriteLn) de la librería estandar Terminal. Como vemos, estas sentencias no están implementadas dentro del propio lenguaje, siendo necesario importarlas de alguna librería exterior.

La importación de objetos de diferentes librerías, debe ajustarse a la estructura sintáctica que se muestra en la siguiente figura.

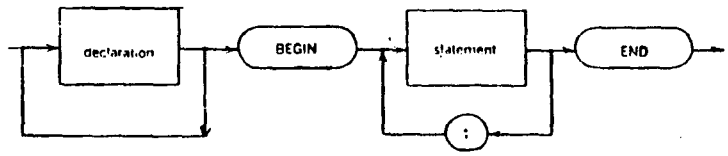


Por lo tanto hemos utilizado los procedimientos importados Read (que lee cualquier carácter del teclado, depositando dicho valor en la variable ch declarada previamente). El procedimiento WriteString, visualiza por la pantalla el contenido de la cadena que tiene como parámetro. WriteLn produce la visualización de una linea en blanco.

Antes de utilizar la variable ch en el cuerpo del programa es decir entre el BEGIN y el END, ha sido necesaria declararla previamente, en nuestro caso en la tercera linea del programa.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

En el siguiente diagrama sintáctico de un bloque se incluye la posible declaración de variables.



Dentro del cuerpo del programa es donde se utilizan las instrucciones. Los programas escritos en MODULA - 2 deben ajustarse a las especificaciones sintácticas de su estructura, en los capítulos siguientes se detallará la sintaxis correcta que deberán poseer las instrucciones.

1.2 SIMBOLOS DEL LENGUAJE

Todos aquellos elementos que el programador utiliza para escribir sus programas, y que constituyen el vocabulario de la gramática de MODULA - 2, se denominan símbolos del lenguaje, tales símbolos son: identificadores, números, cadenas o strings, operadores, delimitadores y comentarios.

1.2.1 IDENTIFICADORES Y NUMEROS

Los identificadores son aquellos nombres proporcionados por el usuario cuando la sintaxis de una sentencia o instrucción, así lo requiere. Ejemplo de identificadores son: Las variables,las etiquetas,los nombres de los procedimientos etc. En M2 un identificador, se forma por una secuecia de letras y dígitos, donde el primer carácter,debe ser siempre una letra .

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Ejemplos de identificadores válidos son:

```
i,dato,x1,empleados
I,Dato,X1,Empleados
```

Ejemplos de identificadores incorrectos son:

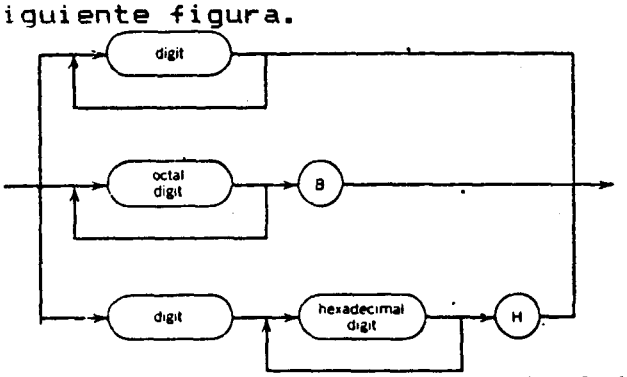
```
Ultimo nombre  (El blanco no está permitido)
7dias          (no puede comenzar con un digito)
dias-años      (no se permiten caracteres especiales)
```

Hay que tener en cuenta que para M2 las letras mayúsculas y minúsculas son diferentes, por lo tanto, sonido, Sonido y soNido , son identificadores distintos.

Los números de M2 pueden ser enteros o reales, se forman por una secuencia de dígitos sin permitir la presencia de espacios en blanco.

Los números reales, además, pueden contener un punto decimal y una parte fraccionaria. Para ellos puede usarse también una representación científica o en coma flotante, mediante la letra E, que significa 10 elevado a la parte exponente.

La estructura sintáctica de los números enteros, se muestra en la siguiente figura.



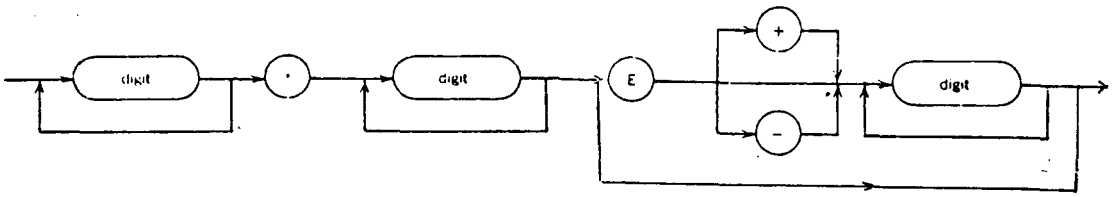
Si a un número válido, se le añade la letra B, será tomado como un número Octal, si se le añade la H, será tomado como un número Hexadecimal.

Se observa que un número Hexadecimal, debe empezar con

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

un **dígito decimal** del 0 al 9 por ejemplo: 0FFFFH.

La **estructura** sintáctica de los números reales, se muestra en la siguiente figura.



Ejemplos válidos de números son:

1988, 23, 1, 47.3, 4.3E-2, 0., 0.0, 1.E-6

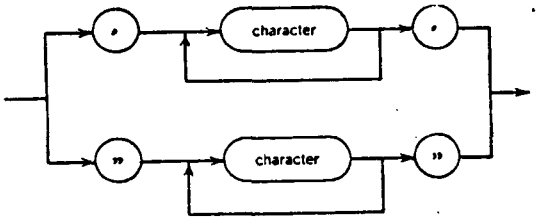
Ejemplos de números incorrectos son:

.5 (No empieza por un dígito)
-3.2 (No empieza por un dígito)
27,3 (No se permite la coma decimal)
42.Ea3 (Sólo se permite E)
1.123.456 (Hay dos puntos repetidos)

1.2.2 CADENAS OPERADORES Y DELIMITADORES

Una cadena o String se define como cualquier secuencia de caracteres encerrados entre comillas.

Su estructura sintáctica es la que se muestra en la figura.



Ejemplos válidos:	Ejemplos incorrectos:
'Miguel de Cervantes'	'Miguel de Cervantes"
"John's program"	'John's program'

Dado que las comillas se utilizan para delimitar la cadena, no está permitido el uso de comillas dentro de la

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

propia cadena.

Los operadores y delimitadores, son símbolos especiales con cometidos y funciones propias de M2. A continuación, se listan con su significado.

```
    +  Adición, unión de conjuntos.
    -  Sustracción, diferencia de conjuntos.
    *  Multiplicación,intersección de conjuntos.
    /  División real, diferencia de conjuntos.
   :=  Asignación.
    &  AND booleano.
    =  Igual.
<>,#   No igual.
    >  Mayor que.
    <  Menor que.
   <=  Menor o igual que.
   >=  Mayor o igual que.
    ^  Puntero.
    ()  Paréntesis.
    []  Paréntesis para matrices.
    {}  LLaves para conjuntos.
(* *)  Delimitadores de comentarios.
    ..  Delimitadores subrango.
    .  Punto.
    ,  Coma.
    ;  Punto y coma.
    :  Dos puntos.
    |  Delimitador de alternativas.
```

Estos operadores, serán estudiados con detalle en el transcurso de los próximos capítulos.

1.2.3 PALABRAS RESERVADAS Y COMENTARIOS

Las palabras reservadas, poseen un significado especial en M2 y no pueden ser utilizadas fuera de su contexto, además deberán ser escritas en mayúsculas.

En la siguiente lista, se muestran todas las palabras reservadas en M2.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

AND	LOOP
ARRAY	MOD
BEGIN	MODULE
BY	NOT
CASE	OF
CONST	OR
DEFINITION	POINTER
DIV	PROCEDURE
DO	QUALIFIED
ELSE	RECORD
ELSIF	REPEAT
END	RETURN
EXIT	SET
EXPORT	THEN
FOR	TO
FROM	TYPE
IF	UNTIL
IMPLEMENTATION	VAR
IMPORT	WHILE
IN	WITH

Los comentarios están formados por una cadena de caracteres encerrados entre los delimitadores de comentarios.

(* ejemplo de comentario *)

Estos son usados por el progrmador en cualquier lugar del programa para recordar o señalar, algún aspecto en concreto del mismo.

1.3 LA SENTENCIA DE ASIGNACION

Se entiende por sentencia de asignación, a aquella instrucción que llena de contenido una variable, con el cálculo de una determinada expresión.

Su sintaxis es: designador:= expresión;

Ejemplos válidos son: a:= b; y:= y+1; x3:= 25;

Al ejecutarse la sentencia, primero se evalúa la expresión a la derecha de ':=' y después el resultado se transfiere al designador.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A la hora de evaluar una expresión deben tenerse presente las siguientes consideraciones:

1) Cada variable de la expresión debe poseer un valor previo.

2) No puede haber dos operadores adyacentes. (a*-b no es correcto, debe escribirse a*(-b)).

3) Los operadores de multiplicación tienen mayor jerarquía que los de suma.

4) Cuando existen dudas en la evaluación de las expresiones, pueden utilizarse paréntesis (a+b*c puede escribirse a+(b*c)).

Los demás tipos de sentencias que componen el repertorio de M2 se estudiarán en los siguientes capítulos.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 2
TIPOS DE DATOS ELEMENTALES EN MODULA - 2, CTES. Y VARIABLES

2.1 TIPOS DE DATOS

Como vimos en el tema anterior, todas las variables deben ser declaradas antes de su uso. Declarar una variable, es precisamente asignarle un cierto tipo de dato.

MODULA - 2 dispone de seis tipos elementales definidos de forma estándar en el lenguaje, estos son: INTEGER, CARDINAL, REAL, CHAR, BOOLEAN y BITSET.

2.2 EL TIPO INTEGER

El tipo INTEGER, define los números enteros positivos y negativos. Dependiendo de la arquitectura interna del ordenador sobre el que esté trabajando el compilador de M2, los valores que puede tomar una variable entera, están comprendidos en el rango determinado por la expresión:

$$-2^{n-1} \text{ hasta } 2^{n-1}-1$$

Para 16 bits resulta: -2^{15} hasta $2^{15}-1$, es decir, el rango de números enteros con el que se puede trabajar con un ordenador de 16 bits, va desde -32768 hasta 32767.

En el caso en que una operación sobrepase este intervalo, se producirá un overflow, situación que deberá evitar el programador.

Las variables declaradas como INTEGER, son usadas

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

fundamentalmente por instrucciones del tipo aritmético, por tanto los operadores que se utilizan con datos de este tipo son: +(suma), -(diferencia), *(producto),DIV(división entera) y MOD(resto de una división).

Veamos algunos ejemplos con el operador DIV, y como el resultado es siempre un número entero.

29 DIV 7 =4, -36 DIV 5 =-7, 36 DIV (-5)=-7

Debemos considerar que la división por cero, no está definida y normalmente causa un error de programa.

El operador MOD obtiene el resto de una división de dos datos que deben ser positivos.

0 MOD 11 = 0	11 MOD 11 = 0
1 MOD 11 = 1	12 MOD 11 = 1
3 MOD 11 = 3	13 MOD 11 = 2
10 MOD 11 =10	19 MOD 11 = 8

M2 posee dos funciones cuyos argumentos pueden ser valores del tipo entero, estos son: ABS y ODD.

ABS(x), devuelve el valor absoluto de la expresión mencionada.

ODD(x), evalúa si x, es par o impar, TRUE si x es impar y FALSE si x es par.

2.3 EL TIPO CARDINAL

Este tipo es un subconjunto del tipo INTEGER y contiene únicamente los números enteros positivos y el cero.

Los valores que puede tomar una variable de este tipo van desde 0 hasta 2^n-1, para n=16, resulta un rango que va desde 0 hasta 65535.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Los operadores que pueden utilizar variables de este tipo son los mismos que los del tipo INTEGER.

El uso de operandos INTEGER y CARDINAL en una misma expresión, no está permitido.

M2 dispone de dos funciones incorporadas: INTEGER y CARDINAL

INTEGER(x), transforma la variable x inicialmente declarada como CARDINAL a INTEGER.

CARDINAL(y), transforma el valor y, inicialmente declarado como INTEGER a CARDINAL.

2.4 EL TIPO REAL

Para la utilización de expresiones aritméticas con números reales, se utilizan los mismos operadores que en los casos anteriores, con la salvedad de que deberá usarse la barra (/) división, en vez del operador DIV.

Como vimos en el capítulo anterior, los números reales permiten dos representaciones, una con un punto decimal y otra con notación científica o coma flotante.

ejemp: 2.3E2 = 230.0
2.3E-1 = 0.23

M2 dispone de dos funciones que transforman datos declarados como CARDINAL a REAL y viceversa.

TRUNC(x), devuelve la parte entera del número real x.

FLOAT(x), convierte cualquier número entero positivo a una representación en coma flotante.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

2.5 EL TIPO CHAR

Los datos del tipo CHAR pueden representar cualquiera de los símbolos del alfabeto ASCII, el juego de estos caracteres se muestra en la siguiente tabla.

First Digits	Last Digit 0	1	2	3	4	5	6	7
00	nul	soh	stx	etx	eot	enq	ack	bel
01	bs	tab	lf	vt	ff	cr	so	si
02	dle	dc1	dc2	dc3	dc4	nak	syn	etb
03	can	em	sub	esc	fs	gs	rs	us
04		!	"	#	$	%	&	'
05	(	)	*	+	,	-	.	/
06	0	1	2	3	4	5	6	7
07	8	9	:	;	<	=	>	?
10	@	A	B	C	D	E	F	G
11	H	I	J	K	L	M	N	O
12	P	Q	R	S	T	U	V	W
13	X	Y	Z	[	\	]	^	_
14	`	a	b	c	d	e	f	g
15	h	i	j	k	l	m	n	o
16	p	q	r	s	t	u	v	w
17	x	y	z	{	|	}	~	del

El valor octal de cada carácter, se obtiene uniendo los dígitos fila y columna donde aparece.

ejem: $A=101_{(8} = 8^2 + 8^0 = 65_{(10}$

En M2 los datos de tipo CHAR se escriben entre comillas.

Pueden utilizarse tres funciones para el manejo de datos de este tipo.

ORD(x): Devuelve el número de orden (CARDINAL) del símbolo mencionado en el argumento de la función.

CHR(x): Devuelve el carácter cuyo orden es el cardinal x

CAP(x): Hace aparecer el valor x con mayúsculas si está en minúsculas, de lo contrario mantiene la x.

2.6 LA FUNCION VAL

Como se ha visto en las funciones de conversión de tipos

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

descritas anteriormente, no hay forma de convertir una variable del tipo REAL a INTEGER, ni una del tipo CHAR a INTEGER.

CARDINAL<--->INTEGER	REAL<--->CARDINAL	CHAR<--->CARDINAL
INTEGER(x)	TRUNC(x)	ORD(x)
CARDINAL(x)	FLOAT(x)	CHR(x)

Sin embargo M2 contiene una función de conversión más estandar, llamada VAL, que permite convertir entre los tipos INTEGER, CHAR y CARDINAL. Su forma general es:

VAL (tipo al que se convierte, valor)

De esta manera, un dato c de tipo CARDINAL, se puede convertir en un dato de tipo INTEGER, de la siguiene forma:

I:=VAL(INTEGER,c) (también se podría utilizar INTEGER(c))

También para convertir un INTEGER a REAL, se podría utilizar VAL, pero en conjunción con FLOAT.

R:= FLOAT(VAL(CARDINAL,I))

Al final de este capítulo, se muestra un programa ejemplo, donde se detallan estos tipos de conversiones.

2.7 EL TIPO BOOLEAN

Una variable declarada del tipo BOOLEAN, sólo puede tomar en un momento dado uno de los dos posibles valores lógicos, o TRUE o bien FALSE.

Los operadores para el manejo de datos de este tipo son: AND(también se puede expresar con &), OR y NOT.

En el cuadro de la figura, se muestran las tablas de verdad de estos tres operadores.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

C_1	C_2	C_1 OR C_2	C_1 AND C_2	NOT C_1
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Para simplificar expresiones lógicas, se pueden aplicar las normas básicas del álgebra booleana, como son las leyes de Morgan.

```
NOT (p OR q) = (NOT p) AND (NOT q)
NOT (p AND q) = (NOT p) OR (NOT q)
```

Las prioridades de los operadores a la hora de evaluar las expresiones lógicas son:

El operador NOT tiene mayor prioridad, después le sigue AND y por último OR.

2.8 EL TIPO BITSET

Cuando un dato, se declara del tipo BITSET, los valores que puede tomar tal dato, se encuentran comprendidos entre 0 y n-1, donde n es el número de bits de una palabra, generalmente n=16.

Algunos ejemplos de declaración de conjuntos son:

```
{ } Conjunto vacio
{0} El conjunto contiene sólo el cero
{1,2,3}
{3,2,1} Es el mismo conjunto que el anterior
```

Para simplificar la escritura de un conjunto con mayor número de elementos, se permite la notación "..",(llamada subrango).

ejem: {0..3,8..11,15} = {0,1,2,3,8,9,10,11,15}

Los operadores que M2 posee para el tratamiento de conjuntos son:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

+ (unión de conjuntos), - (diferencia de conjuntos), * (intersección de conjuntos), / (diferencia de conjuntos) y por último, el operador IN para detectar la pertenencia de un elemento a un conjunto (que equivale al ∈ matemático) . Para este operador, el resultado es TRUE (si el elemento pertenece al conjunto) o FALSE (en caso contrario). Para los restantes operadores el resultado es un conjunto.

El tipo BITSET se ha mencionado en este capítulo por ser un tipo de dato predefinido en M2, al igual que CARDINAL, REAL, etc. Pero será en el capítulo 7 donde se verá con más detalle, al estudiarlo en el contexto de los conjuntos.

2.9 LA DECLARACION DE VARIABLES

En M2 todas las variables deben declararse en la sección de declaración de variables antes de ser usadas.

El bloque de declaración de variables, comienza siempre con la palabra reservada VAR, seguida de todos los nombres de las variables que se desean declarar, cada una con su tipo asignado. La sintaxis de la declaración de variables sería:

```
VAR
    <lista de variables>:<tipo>;
```

Lista de variables, significa que varias variables del mismo tipo se pueden escribir seguidas, separadas por comas.

```
ejem: VAR
        i,j,k:CARDINAL;
        CambioDolar:REAL;
        letra:CHAR;
        varon:BOOLEAN;
        contador:INTEGER;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

2.10 DECLARACION DE CONSTANTES

En M2 también pueden declararse datos que posean un valor constante durante toda la ejecución del programa.

Una constante se declara en un programa, en la sección de declaración de constantes, siguiendo la siguiente forma general: CONST

```
        <nombre de la cte.> = <valor>;
```

ejemp: CONST

```
        largo = 10;
        ancho = 23;
        alto = 12;
```

El siguiente programa ilustra los tipos de conversiones estudiados en este capítulo, de su ejecución resulta:

```
1.30E+001    23A    65    10    1.01E+001      1
```

Modula-2/86 conversi.mod

```
 1  MODULE ConversionTipos;
 2     FROM InOut IMPORT Write, WriteLn, WriteCard, WriteInt;
 3     FROM RealInOut IMPORT WriteReal;
 4
 5     VAR
 6          ch: CHAR;
 7          x: INTEGER;
 8          c: CARDINAL;
 9          r: REAL;
10  BEGIN
11     ch:='A';
12     x:=-10;
13     c:=13;
14     r:=10.123;
15     WriteReal (FLOAT(c),5); (*Escribe 1.30E+001*)
16     WriteCard (c+VAL(CARDINAL,ABS(x)),4); (*Escribe 23*)
17     Write (CHR(65)); (*Escribe A*)
18     WriteCard(ORD(ch),4); (*Escribe 65*)
19     WriteCard(TRUNC(r),4); (*Escribe 10*)
20     WriteReal(r,5); (*Escribe 1.01E+001*)
21     WriteInt(VAL(INTEGER,TRUNC(1.123)),5); (*Escribe 1*)
22  END ConversionTipos.
```

```
Modula-2/86 Compiler Version V 1.10

==>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Los **parámetros** tales como 5,4 que se observan en los procedimientos WriteReal, WriteCard, WriteInt, determinan la longitud mínima del campo de salida que se visualiza en pantalla, en el siguiente capítulo, se verá con más detalle al estudiarse las librerías de Entrada/Salida y su procedimientos.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 3
PROCEDIMIENTOS DE ENTRADA Y SALIDA

3.1 INTRODUCCION

Se entiende por rutinas de Entrada/Salida a aquellos procedimientos utilizados por el programador para leer o escribir datos, hacia o desde el computador. El dispositivo de entrada de datos más utilizado es el teclado, mientras el dispositivo de salida es la pantalla o impresora.

Estas rutinas de E/S no están incluidas dentro del lenguaje M2 como sentencias ejecutables, sino como librerías standard incorporadas normalmente con el compilador.

Veamos a continuación algunos de los módulos de librería más utilizados, incluídas sus rutinas de E/S.

3.2 MODULO CardinalIO

Este módulo relaciona valores cardinales en forma decimal y hexadecimal, incluyendo los procedimientos ReadCardinal, WriteCardinal, ReadHex y WriteHex.

Procedimiento ReadCardinal

Su forma general es:

```
PROCEDURE ReadCardinal (VAR c:CARDINAL);
```

Este procedimiento va a leer un número decimal sin signo del terminal actual, almacenando dicho número en la variable

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

c. La lectura habrá finalizado cuando encontremos el carácter ESC (Escape), EOL (Fin de línea) o blanco.

Procedimiento WriteCardinal

Su forma general es :

```
PROCEDURE WriteCardinal (c:CARDINAL;w:CARDINAL);
```

Escribe un número cardinal en forma decimal en el terminal actual. La variable c es el valor a escribir en el terminal y la variable w contendrá la anchura del campo donde se escribirá. Es decir, si el número tiene 3 cifras significativas, entonces deberemos almacenar en w el valor 3 como anchura del campo. El valor de c puede superar la anchura del campo w, escribiendo en el terminal su valor correspondiente. Si el valor de c es menor que la anchura w, entonces el campo se rellenará con espacios en blanco a su izquierda.

Procedimiento ReadHex

Su forma general es :

```
PROCEDURE ReadHex (VAR c:CARDINAL);
```

Lee un número cardinal del terminal actual, en formato hexadecimal, almacenando su valor en la variable c. Posee las mismas características que el procedimiento ReadCardinal.

Procedimiento WriteHex

Su forma general es :

```
PROCEDURE WriteHex (c:CARDINAL;dig:CARDINAL);
```

Escribe un número cardinal en el terminal actual, en formato hexadecimal. La variable c es el valor a escribir,

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

mientras que dig será la anchura del campo donde se almacena la variable c. Posee las mismas características que el procedimiento WriteCardinal.

El siguiente módulo muestra, la utilización de diversos procedimientos importados de la librería CardinalIO.

Modula-2/86 libcard.MOD

```
 1 MODULE ModCardinal;
 2   FROM CardinalIO IMPORT ReadCardinal,WriteCardinal,ReadHex,WriteHex;
 3   VAR
 4     car,hex,arg:CARDINAL;
 5 BEGIN
 6    ReadCardinal(car);
 7    ReadCardinal(arg);
 8    WriteCardinal(car,arg);
 9
10    ReadHex(hex);
11    ReadCardinal(arg);
12    WriteHex(hex,arg);
13
14 END ModCardinal.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

3.3 MODULO InOut

Dentro de este módulo se van a incluir las rutinas o procedimientos stándard de E/S a alto nivel para el trabajo con ficheros secuenciales.

Procedimiento OpenInput

Su forma general es:

PROCEDURE OpenInput (extdef:ARRAY OF CHAR);

Este procedimiento aceptará un nombre de fichero del terminal actual y lo abrirá como de entrada. La variable

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

extdef nos definirá el tipo o extensión del fichero. Por ejemplo:

```
OpenInput ("mod");
```

Si el nombre del fichero leído no termina en '.', y no tiene extensión, entonces mod se le añade como extensión al fichero que abramos.

Cuando importamos el procedimiento OpenInput, también se debe importar la variable boolena 'Done', que nos indicará si se ha ejecutado correctamente el procedimiento.

A continuación se muestra un fragmento de código de programa donde se utiliza el procedimiento OpenInput en conjunción con la variable Done.

```
BEGIN
   WriteString ('Introducir nombre de fichero de entrada');
REPEAT
   OpenInput(' ');
UNTIL Done;
```

Procedimiento OpenOutput

Su forma general es:

```
PROCEDURE OpenOutput (extdef:ARRAY OF CHAR);
```

Nos aceptará un nombre de fichero del terminal actual y lo abrirá como salida. La variable extdef nos definirá el tipo o extensión del fichero. Si el nombre del fichero termina con un punto, la extensión se añade al final del nombre.

Respecto a la variable 'Done', ésta presenta e mismo efecto que en el procedimiento anterior, tal y como se puede ver a continuación.

```
BEGIN
   WriteString ("Introducir nombre al fichero de entrada: );
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
REPEAT
    OpenOutput (* *)
UNTIL Done;
```

Procedimiento CloseInput

Su forma general es:

PROCEDURE CloseInput;

Cierra el fichero de entrada actual y devuelve el control de la entrada al terminal.

Procedimiento CloseOutput

Su forma general es:

PROCEDURE CloseOutput;

Cierra el fichero de salida actual y devuelve el control de salida al terminal.

Procedimiento Read

Su forma general es:

PROCEDURE Read (VAR ch:CHAR);

Lee un carácter del terminal de entrada y almacena dicho carácter en la variable ch. La variable booleana 'Done' será TRUE, a menos que estemos en el final de un fichero.

Procedimiento ReadString

Su forma general es:

PROCEDURE ReadString (VAR s:ARRAY OF CHAR);

Lee una cadena alfanumérica del terminal de entrada y, almacena la cadena en la variable s, que equivale a una matriz de n caracteres, donde n será en todo momento la longitud en caracteres de la matriz s. La dimensión de la matriz s es arbitraria dependiendo de la longitud que le

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

introduzcamos.

Existe una variable 'termCH', donde se alberga el carácter que provoca la terminación de la lectura (ReadString termina con cualquier carácter ASCII ≤ SP=40C), dicha variable debe ser importada del módulo InOut. El siguiente ejemplo, nos permite leer una linea entera de texto hasta que encuentre el carácter retorno de carro (CR).

```
REPEAT
    ReadString(a)
UNTIL termCH=EOL; (*EOL = fin de linea*)
```

Procedimiento ReadInt

Su forma general es:

```
PROCEDURE ReadInt (VAR x:INTEGER);
```

Lee un número entero sin signo del terminal de entrada actual, almacenando dicho número en la variable x.

Procedimiento ReadWrd

Su forma general es:

```
PROCEDURE ReadWrd (VAR w:WORD);
```

Lee una palabra de memoria del terminal de entrada actual, almacenando dicho valor en la variable w. El concepto de WORD es absolutamente dependiente de la máquina que utilicemos, y nos permite definir el contenido de una posición direccionable de memoria.

La variable 'Done' tomará el valor TRUE si la palabra de memoria es leída correctamente.

Procedimiento ReadCard

Su forma general es:

```
PROCEDURE ReadCard (VAR x:CARDINAL);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Lee un número cardinal sin signo del terminal de entrada actual, almacenando dicho número en la variable x.

Procedimiento Write

Su forma general es:

```
PROCEDURE Write (ch:CHAR);
```

Escribe un carácter en el terminal de salida actual. La variable ch deberá contener previamente el carácter a escribir.

Procedimiento WriteLn

Su forma general es:

```
PROCEDURE WriteLn;
```

Escribe una línea en blanco en el terminal de salida actual.

Procedimiento WriteString

Su forma general es:

```
PROCEDURE WriteString (s:ARRAY OF CHAR);
```

Escribe una cadena s en el terminal de salida actual. La variable s deberá contener precisamente la cadena a escribir.

Procedimiento WriteInt

Su forma general es:

```
PROCEDURE WriteInt (x:INTEGER;n:CARDINAL);
```

Escribe el número entero x de n caracteres como mínimo en el terminal de salida actual. La variable x deberá contener un número entero de n caracteres antes de escribirse en el terminal. Si la longitud n es mayor que el número de cifras de x, la salida rellenará con espacios en blanco.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Procedimiento WriteCard

Su forma general es:

```
PROCEDURE WriteCard (x,n:CARDINAL);
```

Escribe el cardinal x sin signo en el terminal de salida actual, estando limitada su longitud como mínimo a n caracteres.

Presenta las mismas características que el procedimiento anterior.

Procedimiento WriteOct

Su forma general es:

```
PROCEDURE WriteOct (x,n:CARDINAL);
```

Escribe un número cardinal x en formato octal en el terminal de salida actual, sobre un campo de n caracteres de longitud mínima. Tiene las mismas características que el procedimiento WriteInt.

Procedimiento WriteHex

Su forma general es:

```
PROCEDURE WriteHex (x,n:CARDINAL);
```

Escribe un número cardinal x en formato hexadecimal en el terminal de salida actual, definido sobre un campo de n caracteres de longitud mínima.

Procedimiento WriteWrd

Su forma general es:

```
PROCEDURE WriteWrd (w:WORD);
```

Escribe la palabra w en el dispositivo de salida actual. El concepto de WORD, como mencionamos anteriormente, es

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

absolutamente dependiente de la máquina.

3.4 MODULO Keyboard

En este módulo vamos a tener todas aquellas rutinas relacionadas con el terminal de entrada de datos (teclado).

Procedimiento Read

Su forma general es:

```
PROCEDURE Read (VAR ch:CHAR);
```

Lee un carácter cualquiera del terminal de entrada o teclado, y almacena su valor en la variable de tipo carácter ch. Tecleando CTRL-C terminará la ejecución del programa actual.

Procedimiento KeyPressed

Su forma general es:

```
PROCEDURE KeyPressed ():BOOLEAN;
```

Este procedimiento comprueba si se ha pulsado algun carácter del teclado, y nos dará como resultado un valor del tipo booleano. Si es TRUE es que se ha pulsado alguna tecla, y si es FALSE es que está esperando que pulsemos una tecla.

3.5 MODULO RealInOut

Este módulo contiene aquellos procedimientos de E/S que tienen relación con valores reales.

Procedimiento ReadReal

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Su forma general es:

```
PROCEDURE ReadReal (VAR x:REAL);
```

Lee un número real x del terminal de entrada y almacena dicho valor en la variable x. La variable 'Done' será TRUE si se ha encontrado un número, y FALSE en caso contrario.

Procedimiento WriteReal

Su forma general es:

```
PROCEDURE WriteReal (x:REAL;n:CARDINAL);
```

Escribe un número real x, ajustado a la derecha, en el terminal de salida actual, definido sobre un campo de al menos n caracteres de longitud. Los caracteres que sobran de n se llenan con espacios en blanco a su izquierda.

Procedimiento WriteRealOct

Su forma general es:

```
PROCEDURE WriteRealOct
```

Escribe un número real x, en formato octal, en notación científica, es decir con exponente y mantisa en el terminal de salida actual.

3.6 MODULO Terminal

Este módulo va a contener todos aquellos procedimientos que tienen relación con la E/S desde el terminal especificado.

Procedimiento Read

Su forma general es:

```
PROCEDURE Read (VAR ch:CHAR);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Lee un carácter del terminal y almacena dicho valor en la variable ch.

Procedimiento KeyPressed

Su forma general es:

```
PROCEDURE KeyPressed ():BOOLEAN;
```

Comprueba si hay algún carácter para leer en el terminal. La función será TRUE si se pulsa alguna tecla y FALSE en caso contrario.

Procedimiento ReadAgain

Su forma general es:

```
PROCEDURE ReadAgain;
```

Anulará la última lectura realizada, y hará que se lea de nuevo el último carácter.

Procedimiento ReadString

Su forma general es:

```
PROCEDURE ReadString (VAR s:ARRAY OF CHAR);
```

Lee una línea desde el terminal y almacena dicha línea en la variable s.

Procedimiento Write

Su forma general es:

```
PROCEDURE Write (ch:CHAR);
```

Escribe una carácter ch en el terminal. Pudiendo ser tanto carácter alfanumérico como de control.

Procedimiento WriteString

Su forma general es:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

PROCEDURE WriteString (s:ARRAY OF CHAR);

Escribe una cadena s en el terminal actual.

Procedimiento WriteLn

Su forma general es:

PROCEDURE WriteLn;

Escribe una línea en blanco en el terminal actual.

Veamos por último un módulo ejemplo, donde se utilizan algunos de los procedimientos estudiados hasta el ahora.

Modula-2/86 cap3.MOD

```
 1 MODULE Ejemplos;
 2   FROM Terminal IMPORT KeyPressed, WriteLn;
 3   FROM RealInOut IMPORT ReadReal, WriteReal;
 4   FROM InOut IMPORT Read, Write, WriteString;
 5   VAR
 6    ch: CHAR;
 7    i: INTEGER;
 8    n1, n2, n3, resul: REAL;
 9    str: ARRAY[0..79] OF CHAR;
10 BEGIN
11   Write(14C); (*borra pantalla*)
12   WriteString ('Introduce un caracter: '); Read (ch);
13   Write (14C);
14   WriteString ('El carácter es: '); Write (ch); WriteLn;
15
16   WriteString ('Pulsa cualquier tecla para continuar');
17   LOOP
18     IF KeyPressed() THEN
19       Write (14C);
20       FOR i:=0 TO 79 DO
21         str [i]:=' ';
22       END;
23       i:=-1;
24       EXIT
25     END;
26   END;
27   WriteString ('Introducir un texto: ');
28   REPEAT
29     i:=i+1;
30     Read (ch);
31     Write (ch);
32     str[i]:=ch;
33   UNTIL ch = 36C;
34   Write (14C);
35   WriteString ('El texto es: '); WriteLn; WriteString (str); WriteLn;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
36
37    WriteString ('Pulsa cualquier tecla para continuar');
38    LOOP
39      IF KeyPressed() THEN EXIT END;
40    END;
41    Write(14C);
42    WriteString('Introduzca tres nºs reales'); WriteLn;
43    ReadReal(n1); WriteLn;
44    ReadReal(n2); WriteLn;
45    ReadReal(n3); WriteLn;
46    resul := (n1+n2+n3) / 3.0 ;
47    WriteString('La media aritmética es:'); WriteReal(resul,5);
48  END Ejemplos.
```

```
Modula-2/86 Compiler Version V 1.10

=>      0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 4
LAS ESTRUCTURAS DE CONTROL

4.1 INTRODUCCION

Las sentencias de control, en muchos de los casos, son la esencia de un lenguaje de computador. Estas sentencias gobiernan no sólo la forma en que se ejecutará un programa, sino la estructura global ideada por el programador.

Estas estructuras nos permiten la ejecución alternativa de una sentencia o grupo de sentencias, o bien la ejecución reiterada siguiendo la comprobación de una cierta condición impuesta. El lenguaje M2 tiene dos grupos de estructuras de control bien caracterizadas; el grupo de sentencias repetitivas y el de sentencias alternativas. Dentro de cada grupo tenemos una serie de sentencias ejecutables, las sentencias IF y CASE como estructuras alternativas, y FOR, WHILE/DO, REPEAT/UNTIL y LOOP/EXIT como estructuras repetitivas.

A continuación se presenta un estudio detallado de las mismas.

4.2 SENTENCIA IF

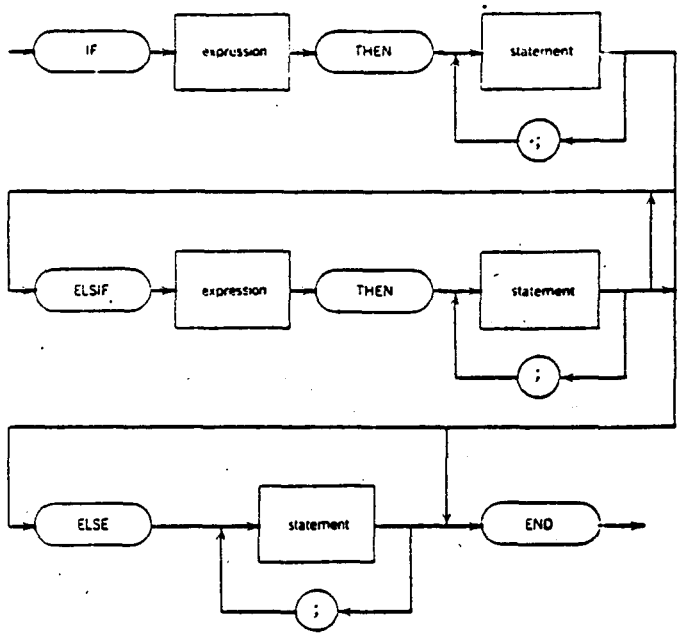
Consiste en una sentencia condicional que nos permite la elección de ejecutar un bloque de sentencias determinado, dependiendo de la evaluación de una condición o expresión boolena.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La forma general de la sentencia IF es:

```
IF <var. booleana1> THEN
  <bloque de sentencias A>
ELSIF <var. booleana2> THEN
  <bloque de sentencias B>
ELSE
  <bloque de sentencias C>
END;
```

El diagrama sintáctico de dicha estructura es:



Hemos de observar en este diagrama sintáctico como la variable booleana puede equivaler a una expresión cuyo resultado sea booleano.

En su forma más simple la sentencia IF posee la siguiente estructura:

```
IF <expresión booleana> THEN
   <bloque de sentencias>
END;
```

En primer lugar vamos a evaluar la expresión booleana; si es TRUE el programa pasa a ejecutar el bloque de sentencias, y si es FALSE, entonces saldremos de la estructura IF y el control pasará a ejecutar la siguiente

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

sentencia ejecutable dentro del programa.

Un ejemplo sería:

```
IF Z<100 THEN
  A:=19
END;
.............
```

Si queremos que nuestro programa cuando evalúe la expresión booleana nos permita elegir entre dos bloques de sentencias, deberemos utilizar el siguiente formato:

```
IF <expresión booleana> THEN
  <bloque de sentencias 1>
ELSE
  <bloque de sentencias 2>
END;
```

En este caso, si la expresión booleana es TRUE se ejecutará el <bloque de sentencias 1>, y si es FALSE lo hará el <bloque de sentencias 2>.

Un ejemplo de lo visto podría ser:

```
IF a THEN
   WriteString ('El resultado es TRUE');
ELSE
   WriteString ('El resultado es FALSE')
END;
```

Otro formato que nos permite esta sentencia alternativa es el IF...THEN...ELSIF...THEN...

La forma general de esta nueva estructura es:

```
IF <expresión 1> THEN <bloque de sentencias 1>
ELSIF <expresión 2> THEN <bloque de sentencias 2>
.....................................
ELSIF <expresión n> THEN <bloque de sentencias n>
END;
```

El análisis de esta estructura es: Si la expresión 1 es TRUE se ejecutará el bloque de sentencias 1, después de ejecutar dicho bloque, el control del programa pasará a la siguiente sentencia ejecutable que esté después de la

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

cláusula **END.** Si la expresión 1 es FALSE, entonces la cláusula **ELSIF** posterior evaluará la expresión 2; si es TRUE se ejecutará el bloque de sentencias 2, si es FALSE el programa evaluará la expresión 3 contenida en la cláusula ELSIF, realizando el mismo proceso descrito anteriormente, hasta llegar a la evaluación de la expresión n.

Un ejemplo sería:

```
IF a>2 THEN Write (a);
ELSIF a<1 THEN Write (a);
ELSIF a>0 THEN Write (a);
ELSE
  Write (a);
  WriteString ('es negativo')
END;
```

Observemos que en la última evaluación, si no se cumple la expresión a>0 no repetimos la cláusula **ELSIF**, sino que colocamos ELSE, pues no tenemos que evaluar más expresiones.

4.2.1 USO DE SENTENCIAS IF ANIDADAS

Una sentencia IF anidada es simplemente una sentencia IF que es el objeto de otra sentencia IF. Observemos el siguiente ejemplo:

```
IF C=110 THEN
  Write (C);
  IF ABS(D)>10 THEN E:=5
  ELSE E:=10
  END;
ELSE WriteString ('Error')
END;
```

Un análisis de este ejemplo es: El conjunto de sentencias donde se incluye la sentencia IF anidada se ejecutará si y sólo si el valor de C es igual a 110, si esto no ocurre se ejecutará la sentencia ELSE del final. La

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

expresión ABS es una función standard del lenguaje M2, que obtiene el valor absoluto del argumento.

El problema principal que se nos presenta en el uso de las sentencias IF anidadas, es definir el ámbito de cada IF es decir, donde comienza y donde acaba. Debemos de tener en cuenta que a cada IF que abrimos debe corresponderle una sentencia END que cierre dicha estructura. En nuestro ejemplo anterior vemos que la primera sentencia END corresponderá al IF interior, mientras que el último END corresponderá al IF exterior.

Siempre que utilicemos este tipo de estructuras debemos tener cuidado a la hora de presentar nuestro programa, pues una secuencia de código mal estructurado dificulta la lectura de éste y nos puede llevar a cometer errores en los anidamientos. Veamos el siguiente ejemplo:

```
IF A=B THEN
IF B=C THEN
A:=D;
ELSIF C=(B DIV A) THEN
C:=E;
ELSE C:=8
END;ELSE C:=5 END;
```

Sin duda en la lectura de esta porción de código, se pueden producir errores de interpretación al estar mal estructurado. Escrito nítidamente resultaría:

```
IF A=B THEN
  IF B=C THEN
    A:=D;
  ELSIF C=(B DIV A) THEN
    C:=E;
  ELSE
    C:=8
  END;
ELSE
  C:=5
END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

4.3 SENTENCIA CASE

Esta sentencia se utiliza cuando un programa requiere que se elija una alternativa entre varias, dependiendo del valor que tome una expresión o una variable de referencia.

Hay situaciones donde la utilización de una sentencia IF puede resultar engorrosa, como es cuando existen tantas alternativas como valores puede tomar una variable, tal es el caso del siguiente ejemplo:

```
IF CH='A' THEN X:=1
ELSIF CH:='B' THEN X:=2
ELSIF CH='C' THEN X:=3
ELSE X:=4
END;
```

La sentencia CASE nos permite que el valor de una expresión determine que bloque de sentencias será el ejecutable.

La forma general de la sentencia CASE es:

```
CASE <expresión> OF
   <etiqueta 1>:<bloque de sentencias 1> |
   <etiqueta 2>:<bloque de sentencias 2> |
   <etiqueta 3>:<bloque de sentencias 3> |
   ....................................
   <etiqueta N>:<bloque de sentencias N>
ELSE <bloque de sentencias>
END;
```

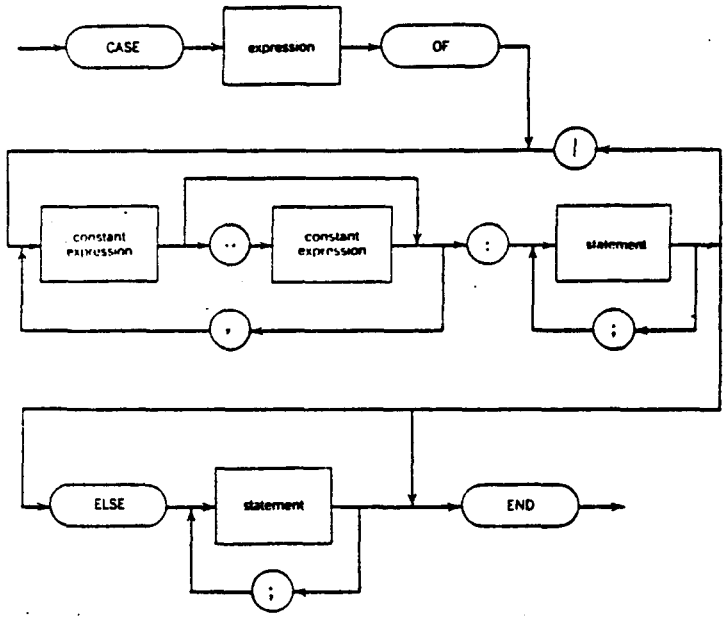
Las barras verticales que están después de cada bloque de sentencias, son necesarias en este tipo de estructura y su función es delimitar las alternativas. El tipo de datos de expresión y de etiqueta deben coincidir, pudiendo ser de cualquier tipo excepto REAL. ELSE podrá omitirse en el caso de que en la sentencia CASE, se hayan considerado todos los valores posibles de la variable de referencia.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La **ejecución** de la sentencia CASE implica en primer lugar la evaluación de la expresión, ejecutándose aquel bloque de sentencias cuya etiqueta coincida con el valor obtenido.

Cuando el resultado de la evaluación de la expresión no coincida con ninguna de las etiquetas, entonces se pasa a ejecutar las sentencias correspondientes a la alternativa ELSE.

En la siguiente figura se muestra el diagrama sintáctico de la sentencia CASE.



El **ejemplo** anterior implementado con una sentencia **CASE**, resultaría.

```
CASE CH OF
  'A': X:=1 ;
  'B': X:=2 ;
  'C': X:=3
ELSE X:=4
END;
```

Observemos que la barra delimitadora de **alternativas**, debe omitirse en el útimo caso, de lo contrario **obtendríamos** un error en la compilación del programa.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

4.4 SENTENCIA FOR

La sentencia FOR tiene el objetivo de permitir que un grupo de instrucciones se ejecuten un número reiterado de veces, mientras que una variable mencionada en la propia instrucción y que puede tomar valores entre dos límites, controla el número de repeticiones.

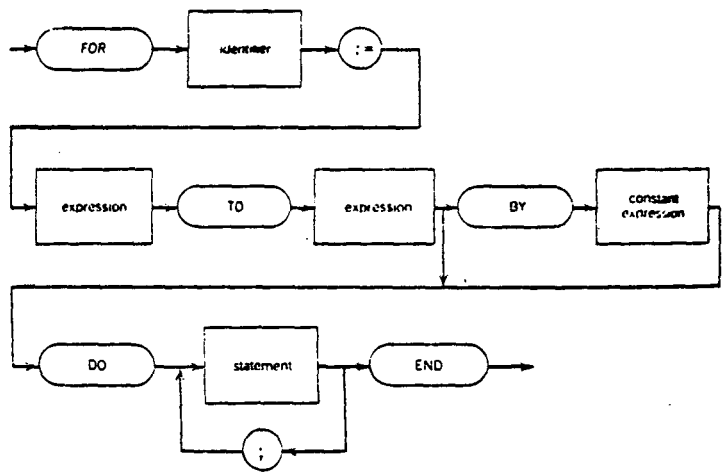
La forma general de la sentencia FOR es:

```
FOR <var. de referencia> := LimInf. TO LimSup. BY <incr> DO
    <bloque de sentencias>
END;
```

Los límites de la variable controlada podrán ser del tipo INTEGER o CARDINAL y ésta podrá incrementarse o decrementarse según sea positivo o negativo el valor del incremento. No obstante la inclusión de BY, es opcional y cuando éste se omite el incremento por defecto es 1.

El bloque de sentencias se repetirá hasta que el valor de la variable controlada supere el límite superior.

El diagrama sintáctico de la sentencia FOR es:



© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La siguiente porción de código de programa nos permite visualizar en pantalla los números impares comprendidos enre 1 y 100.

```
FOR X:=1 TO 100 BY 2 DO
   WriteInt (X,5)
END;
```

Un ejemplo similar al anterior pero decrementando la variable controlada es:

```
FOR X:=99 TO 1 BY -2 DO
   WriteInt (X,5)
END;
```

También podrá omitirse BY, en tal caso el incremento por defecto es 1.

```
FOR X:=1 TO 100 DO
   WriteInt (X,5)
END;
```

4.4.1 USO DE SENTENCIAS FOR ANIDADAS

Las sentencias FOR también permiten su anidamiento como se puede ver en el siguiente ejemplo.

```
FOR A:=1 TO 10 DO
   WriteString('Lenguaje');
   FOR B:=1 TO 100 DO
     WriteString ('MODULA');
   END;
END;
```

La ejecución de este código permite visualizar 1000 veces la palabra MODULA en pantalla y 10 Lenguaje, es decir una vez por cada 100 que se visualice MODULA.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

4.5 SENTENCIA WHILE

Esta sentencia se utiliza en situaciones donde se requiere la repetición de una instrucción o grupo de instrucciones, mientras se siga verificando la condición impuesta en la propia sentencia.

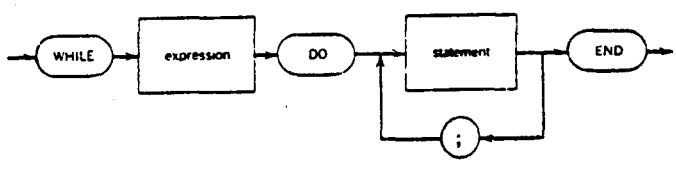
La forma general de la sentencia WHILE es:

```
WHILE <expresión> DO
   <bloque de sentencias>
END;
```

Cuando se ejecuta una sentencia WHILE, en primer lugar se evalúa la expresión booleana, si es TRUE, entonces se ejecutará el bloque de sentencias, para posteriormente volver a evaluar la expresión. Este proceso se reitera hasta que la expresión sea FALSE, en este caso, se ejecutará la siguiente instrucción en secuencia después de la WHILE.

Hay que tener en cuenta que la evaluación de la expresión, se realiza antes de que se ejecuten las sentencias que contiene, en el caso de que no se cumpliese la condición impuesta inicialmente, el bloque de sentencias no se ejecutaría nunca.

El diagrama sintáctico de la sentencia WHILE es:



© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Un ejemplo con este tipo de sentencia podría ser el siguiente.

```
ReadInt (X);
WHILE X<>10 DO
  WriteInt (num,5);
  WriteString ('el cuadrado es ');
  WriteInt (X*X,5);
  WriteLn;
  ReadInt (X)
END;
```

El bloque de sentencias se repetirá tantas veces como se cumpla la condición de X<>10. Si la variable controlada X toma el valor 10, la condición se hará FALSE y el proceso abandonará el lazo para pasar a ejecutar la siguiente sentencia del programa.

4.6 SENTENCIA REPEAT

Esta sentencia tiene una misión similar a la instrucción WHILE, es decir, repetir un número controlado de veces un grupo de sentencias, pero difieren fundamentalmente en que la sentencia REPEAT, nos garantiza que se realizará al menos una pasada a lo largo del bloque de sentencias contenidas en dicha estructura, ya que la evaluación de la expresión booleana se realiza al final del lazo.

Su forma general es:

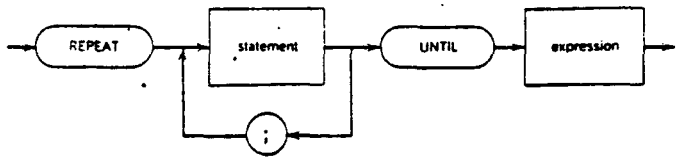
```
REPEAT
  <bloque de sentencias>
UNTIL <expresión>;
```

El término <bloque de sentencias> representa a una o más sentencias, y <expresión> se refiere a cualquier expresión booleana.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

El grupo de sentencias contenido en el lazo se repetirá hasta que la expresión booleana sea TRUE.

La siguiente figura representa el diagrama sintáctico de la sentencia REPEAT.



Veamos la siguiente porción de código como ejemplo de este tipo de sentencias.

```
REPEAT
  Read (CH);
  WriteCard (ORD(CH),5);
UNTIL CH='Q';
```

Las sentencias contenidas en el lazo se ejecutarán reiteradamente hasta que introduzcamos por teclado el carácter 'Q', que es la condición que se ha especificado en la propia instrucción.

A continuación se presenta un nuevo ejemplo, en el que se efectúa la misma secuencia de programa, pero expresada de dos formas diferentes.

```
J:=0;                    J:=0;
K:=3;                    K:=3;
WHILE K>0 DO             REPEAT
  J:=J+1;                  J:=J+1;
  K:=K-1;                  K:=K-1;
END;                     UNTIL K=0;
```

Si bien con la sentencia WHILE y con la REPEAT, se puede implementar un mismo propósito, sus diferencias hacen que una y otra se deriven a distintas aplicaciones.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

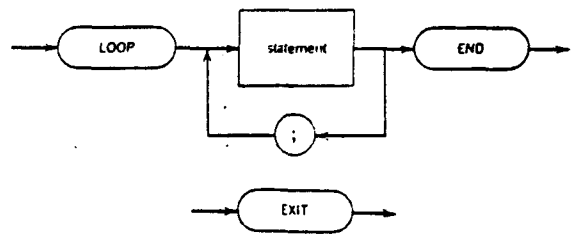
4.7 SENTENCIA LOOP

Esta sentencia posee características similares a las estudiadas anteriormente WHILE y REPEAT, con la diferencia de que en ésta, no se impone la verificación de una condición ni al principio ni al final, siendo la única forma de transferir el control al final del lazo a través de una pregunta (generalmente una sentencia IF), en cualquier lugar del bloque de instrucciones y en conjunción con la sentencia EXIT.

La forma general de este tipo de estructura es el siguiente:

```
LOOP
  <bloque de sentencias>
END;
```

Y su diagrama sintáctico sería:



Debe tenerse en cuenta, aunque no sea sintácticamente necesario, que una instrucción de este tipo debe contener al menos una sentencia EXIT, de lo contrario nuestro programa entraría en un lazo sin fin.

Estas consideraciones pueden observarse en el siguiente ejemplo:

```
LOOP
  ReadInt (X);
  WriteInt (X,5);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
      IF X=100 THEN
        EXIT
      END;
    END;
```

Con la sentencia LOOP es posible implementar todas las estructuras de repetición vistas hasta ahora, tal es el caso de los próximos ejemplos.

Sentencia WHILE

```
WHILE X=5 DO
  WriteString ('Hola')
END;
```

Equivalencia sentencia LOOP

```
LOOP
  IF NOT (X=5) THEN
    EXIT
  END;
  WriteString ('Hola')
END;
```

Sentencia REPEAT

```
REPEAT
  WriteString ('Hola')
UNTIL X=5;
```

Equivalencia sentencia LOOP

```
LOOP
  WriteString ('Hola')
  IF X=5 THEN
    EXIT
  END;
END;
```

Sentencia FOR

```
FOR X:=1 TO 5 BY 2 DO
  WriteInt (X,5)
END;
```

Equivalencia sentencia LOOP

```
X:=1
LOOP
  IF X>5 THEN
    EXIT
  END;
  WriteInt (X,5);
  INC (X,2)
END;
```

Sin embargo, al ser ésta una sentencia no estructurada, sólo deberemos usarla cuando cualquiera de las estructuras vistas anteriormente nos reduzca la eficiencia o legibilidad de nuestro programa.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A continuación se presenta un módulo de programa que incluye los códigos equivalentes vistos con anterioridad.

```
Modula-2/86                    comproba.MOD

 1  MODULE Comprobar;
 2  FROM InOut IMPORT ReadInt, WriteString, WriteInt, WriteLn, Write;
 3  VAR
 4    x:INTEGER;
 5  BEGIN
 6    Write(14C);
 7    x:=5;
 8    WHILE x=5 DO
 9      WriteString('Hola');
10      ReadInt (x)
11    END;
12    WriteLn;
13    x:=5;
14    LOOP
15      IF NOT (x=5) THEN EXIT END;
16      WriteString('Hola');
17      ReadInt (x)
18    END;
19
20    WriteLn;
21
22    REPEAT
23      ReadInt (x);
24      WriteString('Hola')
25    UNTIL x=5;
26    WriteLn;
27    LOOP
28      ReadInt (x);
29      WriteString('Hola');
30      IF x=5 THEN EXIT END
31    END;
32
33    WriteLn;
34
35    FOR x:=1 TO 5 BY 2 DO
36      WriteInt (x,2)
37    END;
38    WriteLn;
39    x:=1;
40    LOOP
41      IF x>5 THEN EXIT END;
42      WriteInt (x,2);
43      INC (x,2)
44    END;
45
46  END Comprobar.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 5
MATRICES Y CADENAS

5.1 INTRODUCCION

Se define como matriz a un grupo de datos, todos del mismo tipo, a los que se accede a través del indexado de una variable que los relaciona.

Los elementos que componen una matriz, se encuentran ordenados, no por su contenido, sino por la posición relativa que ocupan dentro de ella.

En M2 las matrices podrán ser de una o más dimensiones, la máxima dimensión que soporta va a depender del tipo de compilador utilizado.

La declaración de las matrices tendrá una configuración distinta dependiendo de la dimensión y del tipo de datos que almacene.

5.2 DECLARACION DE UNA MATRIZ UNI-DIMENSIONAL

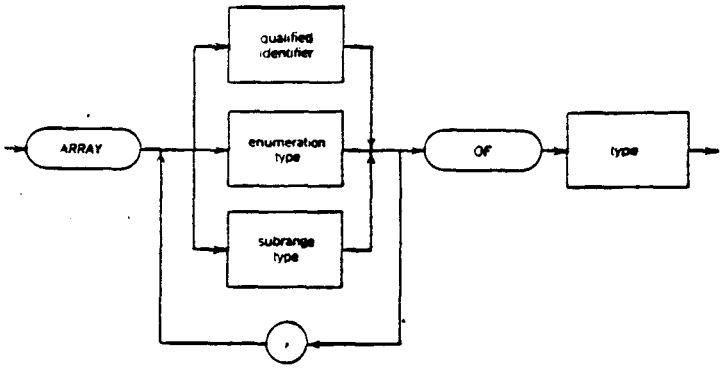
La forma general de declarar una matriz uni-dimensional es:

```
VAR
  <nombre de la matriz>:ARRAY [princip..fin] OF tipo;
```

La variable 'princip' representa el índice del primer elemento, y 'fin' el índice del último elemento. La longitud de la matriz viene expresada por 'fin - princip + 1'.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

El diagrama sintáctico de una matriz es:



Observemos como dentro del intervalo de la matriz podemos tener, tanto variables del tipo enumerado como del tipo subrango. Estos tipos de datos serán comentados en el capítulo 7.

El tamaño máximo de una matriz está condicionado por diversos factores: el computador utilizado, la implementación de M2, el tamaño de cada elemento etc..

Para acceder a uno de los elementos de una matriz unidimensional debemos mencionar el nombre de dicha matriz, y luego el índice que referencia al elemento concreto, encerrado entre corchetes.

nombre de matriz [índice]

5.3 DECLARACION DE UNA MATRIZ MULTI-DIMENSIONAL

El lenguaje M2 aporta dos métodos para la declaración de una matriz multi-dimensional; la estructura larga y la corta.

Para declarar una matriz multi-dimensional necesitamos

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

tantos índices como dimensiones tenga la matriz.

La forma general de declarar una matriz bi-dimensional utilizando el método de estructura larga es:

```
<nombre matriz>:ARRAY [princip1..fin1] OF
                ARRAY [princip2..fin2] OF tipo;
```

donde los términos 'princip1' y 'princip2' representan los índices del primer elemento de cada dimensión, y 'fin1' y 'fin2' los índices del último elemento de cada dimensión. Es decir, nos delimita el intervalo de trabajo de la matriz.

Para acceder a uno de los elementos de esta matriz bi-dimensional, escribimos el nombre de la matriz seguida por los índices que denotan la fila y columna del elemento al que hacemos referencia.

```
nommatriz [i,j]
```

La forma general de declarar una matriz tri-dimensional utilizando el método de estructura larga es:

```
<nombre matriz>:ARRAY [princip1..fin1] OF
                ARRAY [princip2..fin2] OF
                ARRAY [princip3..fin3] OF tipo;
```

Por último, veamos la declaración de una matriz de dimensión N:

```
<nombre matriz>:ARRAY [princip1..fin1] OF
                ARRAY [princip2..fin2] OF
                ......................
                ARRAY [principN..finN] OF tipo;
```

La cantidad de bytes que se necesitan para almacenar una matriz de dimensión N viene expresado por la siguiente expresión:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

L1*L2*L3*.....*LN*tamaño del dato

donde la variable L representa la longitud de las dimensiones de la matriz en cuestión. Un ejemplo práctico es:

```
s:ARRAY [1..8] OF
  ARRAY [1..10] OF
  ARRAY [1..3] OF INTEGER;
```

Esta matriz necesita 8*10*3*2 bytes, es decir 480 bytes para almacenar sus datos.

Anteriormente hemos visto que para la declaración de matrices necesitábamos colocar las palabras reservadas ARRAY....OF en cada una de las dimensiones. Para evitar esta reiteración, M2 nos proporciona el método de estructura corta, donde se va a suprimir la repetición de ARRAY....OF. Veamos un ejemplo práctico de este tipo de estructura.

```
s:ARRAY [1..8], [3..31], [1..11] OF CHAR;
```

5.4 CARACTERISTICAS DE LAS MATRICES

Podemos definir un tipo matriz dentro de la estructura TYPE de un programa, para luego definir una variable global con este tipo.

```
TYPE
    lista: ARRAY [1..10] OF INTEGER;

VAR
    numlista: lista;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

De igual modo, podremos definir una matriz dentro de la estructura VAR de forma directa:

```
VAR
  numlista: ARRAY [1..10] OF INTEGER;
```

En una matriz podemos utilizar índices negativos como intervalo de trabajo:

```
VAR
  clientes: ARRAY [-4..10] OF CARDINAL;
```

Por lo tanto, para acceder al quinto elemento de la matriz debemos expresarlo de la forma; clientes [0].

Cuando definimos los intervalos de una matriz de varias dimensiones, sus índices no tienen por que ser todos del mismo tipo.

```
VAR
  varios: ARRAY ['c'..'z'], [-2..8] OF BOOLEAN;
```

En la siguiente porción de código de programa, se muestra como declarar una matriz, así como referenciar los elementos de ésta asignándole los 80 primeros números enteros en conjunción con la sentencia LOOP estudiada en el capítulo anterior.

```
VAR
  mat:ARRAY [1..80]  OF CARDINAL;
  i:CARDINAL;
BEGIN
  i:=1;
  LOOP
    mat[i]:=i;
    INC (i);
    IF i>80 THEN
      EXIT
    END
  END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

5.5 ASIGNACION ENTRE MATRICES

Al tener dos matrices del mismo tipo, los elementos de una matriz pueden ser asignados a la otra matriz, sin necesidad de hacer este proceso de forma individual, es decir, asignando elemento a elemento.

```
VAR
   X,Y:ARRAY ['a'..'z'] OF INTEGER;
BEGIN
   X:=Y;
```

Esta operación iguala los elementos de las matrices X e Y.

Este concepto lo podemos extender para matrices con más de una dimensión;

```
VAR
  R,T: ARRAY [1..8], [1..4] OF CARDINAL;
```

Si hacemos la asignación:

```
R[1] := T[1];
```

se copian los elementos desde T [1,1] hasta T [1,4] en la matriz R.

5.6 CADENAS O STRINGS

Las cadenas son matrices de caracteres (CHAR), es decir poseen las mismas características de cualquier variable tipo matriz.

Todas las matrices de caracteres deberán tener como límite inicial el cero. El tamaño de una cadena vendrá determinada por la posición del carácter NULL ó nulo, cuyo código ASCII es el 0.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Un ejemplo de declaración de una cadena de 50 caracteres es:

```
VAR
  cad: ARRAY [0..49] OF CHAR;
```

Veamos la disposición de los caracteres en una cadena, tras una sentencia de asignación.

```
VAR
  cas:ARRAY [0..9] OF CHAR;
BEGIN
  cas:="Hola Pepe";
```

La variable 'cas' tendrá la siguiente forma:

0	1	2	3	4	5	6	7	8	9
H	o	l	a		P	e	p	e	NULL

Se observa que toda cadena debe terminar siempre con el carácter NULL, a no ser que el número de caracteres, ocupe el espacio total que hemos declarado para ella.

```
cas:="Hola Pilar";
```

En esta ocasión no existirá carácter NULL al final de la cadena, pues tiene 10 caracteres, que es la longitud máxima que la cadena 'cas' puede contener.

Veamos a continuación algunas de las funciones más utilizadas dentro del tratamiento de cadenas, incluidas todas ellas en el módulo Strings.

Procedimiento Assign

Su forma general es:

```
PROCEDURE Assign (VAR origen,destino:ARRAY OF CHAR);
```

Asignará el contenido de la cadena 'origen' en la cadena 'destino', por ejemplo:

```
Assign (S1,S2);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

almacena el contenido de la cadena S1 en la cadena S2.

Procedimiento Insert

Su forma general es:

```
PROCEDURE Insert(subcad:ARRAY OF CHAR;
                 cad:ARRAY OF CHAR;indice:CARDINAL);
```

Inserta la variable 'subcad' dentro de la cadena 'cad', partiendo de la posición 'índice'. Un ejemplo práctico sería:

```
subcad:="abcd";
cad:="**";
Insert (subcad,cad,1);
```

El resultado de esta llamada al procedimiento Insert es:

cad = "*abcd*"

Hay que tener en cuenta que la cadena destino, debe ser suficientemente larga para poder contener el resultado de la operación.

Si ocurriese que el valor de la variable 'índice' fuese mayor o igual a la longitud de la cadena, entonces la subcadena se añadiría al final de la misma, es decir;

```
Insert (subcad,cad,3);
```

daría como resultado cad = "**abcd".

Procedimiento Delete

Su forma general es:

```
PROCEDURE Delete (VAR cad:ARRAY OF CHAR;
                      indice:CARDINAL;
                      num:CARDINAL);
```

Borra una serie de caracteres 'num' de una cadena 'cad' a partir de la posición 'índice' que especificamos, por ejemplo:

```
cad1:="Esto es un ejemplo";
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
Delete (cad1,5,3);
```

Esta función Delete dará como resultado que en la cadena cad1 quede almacenada la frase: "Esto un ejemplo".

Si el cardinal 'índice' fuese mayor que la longitud de la cadena, entonces no ocurriría nada.

Procedimiento Pos

Su forma general es:

```
PROCEDURE Pos (subcad,cad:ARRAY OF CHAR);CARDINAL;
```

Devuelve la posición donde empieza la subcadena 'subcad' dentro de la cadena 'cad'. Por ejemplo:

```
subcad:="gato";
cad:="perrogatorатón";
WriteCard (Pos(subcad,cad),5);
```

Este código haría visualizar el número 5 en la pantalla, pues es la posición donde se halla la palabra 'gato' dentro de la cadena 'cad'.

Procedimiento Copy

Su forma general es:

```
PROCEDURE Copy (cad:ARRAY OF CHAR;
                indice:CARDINAL;
                num:CARDINAL;
                VAR res:ARRAY OF CHAR);
```

Copia una subcadena de 'num' caracteres a partir de la posición 'índice', obtenida de la cadena 'cad', almacenando el resultado en 'res'. Por ejemplo;

```
s1:="Esto es un caso";
Copy (s1,5,2,s2);
```

Da como resultado en s2 el valor "es".

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Procedimiento Concat

Su forma general es:

```
PROCEDURE Concat (c1,c2:ARRAY OF CHAR;
                  VAR res:ARRAY OF CHAR);
```

El procedimiento Concat une 2 cadenas c1 y c2, almacenando el resultado en 'res'. Por ejemplo;

```
s1:= "Hombre";
s2:= " alto";
Concat (s1,s2,s3);
```

Esta llamada a Concat dará como resultado que en la cadena s3 se almacene "Hombre alto".

Procedimiento Length

Su forma general es:

```
PROCEDURE Length (VAR cad:ARRAY OF CHAR):CARDINAL;
```

Devuelve la longitud de la cadena 'cad', en número de caracteres. Por ejemplo;

```
s1:="Hola";
WriteCard (Length (s1),3);
```

Con este pequeño código, se visualizará en pantalla el valor 4, que corresponde a la longitud de la cadena s1.

Procedimiento CompareStr

Su forma general es:

```
PROCEDURE CompareStr (c1,c2:ARRAY OF CHAR):INTEGER;
```

Compara las cadenas c1 y c2, dando como resultado un número entero.

Si las cadenas c1 y c2 son exactamente iguales dará como resultado 0, si c1 es menor que c2 el resultado será -1, y si c1 es mayor que c2 el resultado será 1.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
c1:="Hola";
c2:="Lola";
WriteInt (CompareStr(c1,c2),5);
```

Con la ejecución de este código, se visualizaría en pantalla el valor -1, pues el carácter 'H' es menor que 'L', según el orden ASCII.

El siguiente módulo de programa, es un ejemplo en el que se muestran diversas formas de manipulación de cadenas.

Modula-2/86 cadenas.MOD

```
 1 MODULE Cadenas;
 2 FROM Strings IMPORT Assign, Concat;
 3 FROM InOut IMPORT WriteString, WriteInt, WriteLn;
 4 TYPE
 5   SieteCar = ARRAY[0..6] OF CHAR;
 6 VAR
 7   S1,S2: SieteCar;
 8   Indice: CARDINAL;
 9   Letras: ARRAY[0..12] OF CHAR;
10   Numeros: ARRAY[0..5] OF CHAR;
11 BEGIN
12    Letras := 'ABCDEFGHIJKL';          (* Copia constantes a variable *)
13    WriteString('Array('); WriteString(Letras); WriteString(') - ');
14    FOR Indice := 0 TO HIGH(Letras) DO
15       WriteInt(ORD(Letras[Indice]),4);
16    END;
17    WriteLn;
18    Numeros := '12345'; Assign(Numeros,Letras); (*Asigna variable a variable*)
19    WriteString('Array('); WriteString(Letras); WriteString(') - ');
20    FOR Indice := 0 TO HIGH(Letras) DO
21       WriteInt(ORD(Letras[Indice]),4);
22    END;
23    WriteLn;
24    S1 := 'Cosa'; S2 := 'Bonita'; Concat(S1,S2,Letras); (*Concatena variables a variable*)
25    WriteString('Array('); WriteString(Letras); WriteString(') - ');
26    FOR Indice := 0 TO HIGH(Letras) DO
27       WriteInt(ORD(Letras[Indice]),4);
28    END;
29    WriteLn;
30    S1 := S2; (* Asigna variable a variable *)
31    Concat(Letras,Numeros,Letras);    (* Concatena una variable en otra *)
32    WriteString('Array('); WriteString(Letras); WriteString(') - ');
33    FOR Indice := 0 TO HIGH(Letras) DO
34       WriteInt(ORD(Letras[Indice]),4);
35    END;
36    WriteLn;
37    Concat(Numeros,Letras,Letras);
38    WriteString('Array('); WriteString(Letras); WriteString(') - ');
39    FOR Indice := 0 TO HIGH(Letras) DO
40       WriteInt(ORD(Letras[Indice]),4);
41    END;
42 END Cadenas.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 6
LOS PROCEDIMIENTOS

6.1 INTRODUCCION

Hasta ahora todos los programas que hemos visto, constaban de una sección de declaraciones y del cuerpo principal del programa, aunque esta forma de diseñar programas es correcta para aprender los conceptos básicos de M2, no es adecuada para programas de un mayor tamaño, pues una de las carácteristicas más importantes para el desarrollo de cualquier programa, es la posibilidad de fragmentarlo, es decir, diseñarlo estructuradamente, descompuesto en varias unidades lógicas o subprogramas. M2 nos permite este tipo de programación a través de los procedimientos o subrutinas.

En los siguientes apartados, veremos los tipos de procedimientos de M2, que son:

Los procedimientos estándar implementados en el propio lenguaje.

Los procedimientos de usuario que crea el programador.

Los procedimientos de librerías que necesitan ser importados de estas por el programador.

6.2 LOS PROCEDIMIENTOS ESTANDAR

M2 dispone de varios procredimientos predefinidos en el propio lenguaje, y que por lo tanto no necesitan ser importados de una librería, a este tipo de procedimientos se

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

les denomina estándar. A continuación se muestra una lista con algunos de estos procedimientos:

```
ABS(x)    ; Devuelve el valor absoluto de x.
CAP(ch)   ; Devuelve la letra mayúscula equivalente de ch.
CHR(x)    ; Devuelve el carácter que representa el número
            Cardinal ASCII x.
DEC(x)    ; Decrementa x en 1.
DEC(x,n)  ; Decrementa x en n.
EXCL(s,i) ; Excluye el elemento i del conjunto s.
FLOAT(x)  ; Convierte el Cardinal x en un número Real.
HALT      ; Para la ejecución del programa.
HIGH(a)   ; Devuelve límite superior de una matriz.
INC(x)    ; Incrementa x en 1.
INC(x,n)  ; Incrementa x en n.
INCL(s,i) ; Incluye el elemento i en el conjunto s.
ODD(x)    ; Devuelve TRUE si x es impar.
ORD(x)    ; Devuelve el número ordinal de x, cuando éste
            está dentro de un conjunto numerado.
TRUNC(x)  ; Devuelve la parte entera de un número Real.
VAL(t,n)  ; Convierte n a su equivalente del tipo t.
```

Algunos de estos procedimientos los hemos visto anteriormente, otros que no están en esta lista, los veremos en capítulos posteriores en un contexto más propicio para su comprensión.

Veamos un ejemplo con el procedimiento estándar CAP. Este se utiliza para devolver la mayúscula de cualquier letra, si la letra es mayúscula o el carácter que se introduce en el procedimiento no es letra, no ocurre ningún cambio.

Modula-2/86 cap.mod

```
 1 MODULE PruebaCAP;
 2   FROM InOut IMPORT Read,Write;
 3   VAR
 4     ch:CHAR;
 5 BEGIN
 6   Write(14C);(*borra la pantalla*)
 7   REPEAT
 8     Read(ch);
 9     ch:=CAP(ch);
10     Write(ch);
11   UNTIL ch='$';
12 END PruebaCAP
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

6.3 LOS PROCEDIMIENTOS DE USUARIO

Los procedimientos de usuario, son aquellos que declara y utiliza el propio programador y que no forman parte de librerías estándar implementadas en M2.

Todos los procedimientos en M2 tienen la siguiente forma general:

```
PROCEDURE <nom.proc> (lista de parámetros);
          <declaración de ctes.>
          <declaración de tipos>      Sección de declaración
          <declaración de variables>
          <declaración de procedimientos>
BEGIN
     <cuerpo de sentencias>           Código del procedimiento
END  <nom.proc>;
```

Si el procedimiento tiene parámetros, estos deben ser incluidos en la lista de parámetros, en caso contrario la lista y sus paréntesis se omiten. Los procedimientos deben terminar con el nombre del procedimiento seguido de punto y coma (;), como se puede ver, la forma de un procedimiento es muy similar a la de módulo, que se vio en la página 1, con la simple excepción de que un procedimiento no puede importar a otro, aunque si es posible definir un procedimiento dentro de otro.

Los procedimientos se declaran justo antes del cuerpo principal del programa, es decir justo antes del BEGIN, tal y como se observa en la forma general de un módulo.

Hay tres tipos fundamentales de procedimientos que puede diseñar un programador; Procedimietos con y sin parámetros y procedimientos que devuelven un valor (procedimientos función),cada uno de los cuales será examinado a continuación.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

6.3.1 PROCEDIMIENTOS SIN PARAMETROS

Como primer ejemplo, se crearán tres procedimientos muy simples, llamados P1,P2,P3 que imprimen su nombre en la pantalla cuando se ejecutan.

```
PROCEDURE P1;
BEGIN
 WriteString('P1');
 WriteLn;
END P1;
```

```
PROCEDURE P2;
BEGIN
 WriteString('P2');
 WriteLn;
END P2;
```

```
PROCEDURE P3;
BEGIN
 WriteString('P3')
 WritLn;
END P3;
```

Una vez definidos los procedimintos se ponen en la sección de declaración de procedimientos, y para que se ejecuten sólo será necesrio llamarlos por su nombre dentro del cuerpo del programa.

Modula-2/86 sinpar.MOD

```
 1  MODULE Proc;
 2    FROM InOut IMPORT WriteString,WriteLn;
 3
 4    PROCEDURE P1;
 5    BEGIN
 6      WriteString('Este es P1');
 7      WriteLn;
 8    END P1;
 9
10    PROCEDURE P2;
11    BEGIN
12      WriteString('Este es P2');
13      WriteLn;
14    END P2;
15
16    PROCEDURE P3;
17    BEGIN
18      WriteString('Este es P3');
19      WriteLn;
20    END P3;
21
22  BEGIN
23    P1;(*se ejecuta P1*)
24    P2;(*se ejecuta P2*)
25    P3;(*se ejecuta P3*)
26  END Proc.
```

```
Modula-2/86 Compiler Version V 1.10

==>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Otro ejemplo de procedimiento sin parámetros, sería el multiplica del siguiente programa. Este procedimiento toma dos datos enteros positivos por teclado, y los guarda en una variable llamada resultado. Sin embargo el objetivo de este ejemplo, es comprender la diferencia entre variables locales y globales.

```
Modula-2/86                          globloc.mod

 1  MODULE Proc;
 2    FROM InOut IMPORT ReadCard,WriteCard,WriteString,WriteLn;
 3    VAR
 4      resultado:CARDINAL;(*variable global*)
 5
 6    PROCEDURE Multiplica;
 7      VAR
 8        a,b:CARDINAL;(*variables locales*)
 9    BEGIN
10      WriteString('introduzca a:');ReadCard(a);
11      WriteLn;
12      WriteString('introduzca b:');ReadCard(b);
13      WriteLn;
14      resultado:=a*b;
15    END Multiplica;
16
17  BEGIN
18    Multiplica;
19    WriteString('resultado=');
20    WriteCard(resultado,5)
21  END Proc.
  ----------------------------------------

   Modula-2/86 Compiler Version V 1.10

   ==>    0 Error(s) found
```

Una variable global, es aquella que es conocida a través de todo el módulo, pudiendo ser usada por cualquier procedimieto dentro del mismo y en el cuerpo principal del programa. En nuestro caso, resulatado, sería la variable global.

Por el contrario una variable local es aquella que es para uso interno del procedimiento en que está declarada exclusivamente. En nuestro caso serían a y b.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Una variable global existe durante todo el tiempo que el programa se está ejecutando, mientras que una variable local existe sólo mientras el procedimiento en el cual se declaró, se está ejecutando. Esto significa que una variable local no mantiene su valor entre llamadas a procedimientos.

Debido a que las variables locales son conocidas sólo en los procedimientos en los cuales están declaradas, es posible tener variables locales y globales con el mismo nombre, sin que halla conflicto, también es posible tener la misma variable en más de un procedimiento.

6.3.2 PROCEDIMIENTOS CON PARAMETROS

En los ejemplos que hemos visto hasta ahora, ninguna información se pasaba explicitamente al procedimiento. Sin embargo muchos procedimientos requieren que se les pase información desde la rutina que llama, al procedimiento. Esto se puede hacer en teoria a través de las variables globales, pero esto presenta dos problemas, primero, que al tenerse varias variables globales en un programa, pueden ocurrir conflictos y cambios involuntarios del valor de estas, y segundo, que usando variables globales se reduce la generalidad de una rutina.

En un procedimiento con parámetros, estos, pueden ser de entrada y de E/S al mismo. Los parámetros se especifican en la lista de parámetros que se detalló en la forma general de un procedimiento. Un parámetro es una variable local especial que se usa para pasar información a un procedimiento.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La lista de parámetros,tiene la siguiente forma general:

(Var1:<tipo>;Var2:<tipo>;....;VarN:<tipo>)

donde, de Var1 a VarN, van los nombres de los parámetros y tipo, es el TYPE de cada parámetro. Los parámetros del mismo tipo se pueden listar juntos, separados por comas (,), y los diferentes tipos están separados por punto y coma (;).

Las variables de Var1 a VarN, son llamadas parámetros formales, es decir son aquellos que se definen en el propio procedimiento, en el cual deben especificarse con su tipo.

Los parámetros formales, que no estén precedidos de la palabra clave VAR, se refieren a parámetros de un solo sentido, es decir de entrada al procedimiento, y se dice que el parámetro es de modo 'valor'.

Si van precedidos de la palabra VAR, se refieren a parámetros entrada/salida, es decir, que un parámetro puede entrar a un procedimieto con un valor y salir con otro modificado, en este caso se llaman parámetros de modo 'variable'.

En el siguiente ejemplo se muestra un programa que llama al procedimiento Maximo, que determina el máximo valor entre tres números enteros.

```
Modula-2/86                          compar.MOD

 1  MODULE Proc;
 2    FROM InOut IMPORT ReadInt,WriteInt,WriteLn,Write,
 3                      WriteString;
 4    VAR
 5      x,y,z,m:INTEGER;
 6
 7    PROCEDURE Maximo(a,b,c:INTEGER; VAR max:INTEGER);
 8    BEGIN
 9      IF a>b THEN max:=a ELSE max:=b END;
10      IF c>max THEN max:=c END;
11    END Maximo;
12
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
13  BEGIN
14    Write(14C);(*limpia la pantalla*)
15    WriteString('Introduzca un nº entero x:');ReadInt(x); WriteLn;
16    WriteString('Introduzca un nº entero y:');ReadInt(y); WriteLn;
17    WriteString('Introduzca un nº entero z:');ReadInt(z); WriteLn;
18    Maximo(x,y,z,w);(*llamada al procedimiento*)
19    WriteString('El máximo es:'); WriteInt(w,5)
20  END Proc.

-----------------------------------------

Modula-2/86 Compiler Version V 1.10

==>     0 Error(s) found
```

Se observa en la lista de parámetros del procedimiento (linea 7 del programa) que hay tres parámetros de entrada al procedimiento, que son: a, b y c, y un parámetro de entrada/salida ,max, que va precedido de la palabra VAR.

Vimos que en el ejemplo de procedimientos sin parámetros se invocaba a estos simplemente mencionando su nombre. En el caso de que halla parámetros, hemos de llamar al procedimiento (linea 18) con parámetros del mismo tipo que los definidos en la lista de parámetros formales.

A los parámetros que se mencionan en la llamada a los procedimientos, se les llama parámetros actuales. Otra llamada a este procedimientos, podría ser: Maximo(1,6,-9,w).

6.3.3 LOS PROCEDIMIENTOS FUNCION

Los procedimientos función sirven para escribir expresiones, cuyos operandos son el resultado de la evaluación de dichos procedimientos.

La única diferencia respecto a la declaración de los procedimientos estudiados hasta ahora, es que debe incluirse

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

el tipo de resultado en la evaluación del procedimiento función, después de la lista de parámetros y separado por (:).

El valor que se devuelve al programa principal después de la llamada, debe mancionarse con la instrucción RETURN, dentro del cuerpo del procedimiento función.

Convirtiendo el procedimiento Maximo del apartado anterior en un procedimiento función, resultaría:

```
PROCEDURE Maximo(a,b,c:INTEGER):INTEGER;
  VAR
    max:INTEGER;
BEGIN
  IF a>b THEN max:=a ELSE max:=b END;
  IF c>max THEN max:=c END;
  RETURN max;
END Maximo;
```

LLamadas a este procedimiento podrían ser:

```
w:=Maximo(x,y,z);
w:=Maximo(x,y,z)+7-2;
w:=(Maximo(9,y,5)*2)+5;
```

Una vez asignados estos valores a w, se podría visulaizar el resultado, tal y como se hizo anteriormente, con WriteInt(w,5);.

También se podría escribir: WriteInt(Maximo(x,y,z),5);.

Si un procedimiento función que se cree, no tiene argumento, es decir, parámetros valor o variables, incluso así, los paréntesis son necesarios, tanto en la declaración como en la llamada.

Dentro del cuerpo de un procedimientos función pueden existir, tantas sentencias RETURN como sea necesario. En cualquier caso, también pueden aparecer sentencias RETURN dentro de los otros tipos de procedimientos, lo que señalaría el final del procedimiento, sin necesidad de continuar hasta el END final.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

6.4 LOS PROCEDIMIENTOS DE LIBRERIA

Además de los procedimientos de E/S que vimos en el capítulo 3, M2 dispone de librerías especiales, tal es el caso de la librería MathLibo, que contiene procedimientos definidos para operaciones matemáticas.

```
PROCEDURE sqrt(x:REAL):REAL;
```
Devuelve la raiz cuadrada de un número x positivo.

```
PROCEDURE exp(x:REAL):REAL;
```
Devuelve e^x, donde e=2.7182181.

```
PROCEDURE ln(x:REAL):REAL;
```
Devuelve el logaritmo natural (en base e=2.7182181) de un número x>0.

```
PROCEDURE sin(x:REAL):REAL;
```
Devuelve el seno de un ángulo x, expresado en radianes.

```
PROCEDURE cos(x:REAL):REAL;
```
Devuelve el coseno de un ángulo x, expresado en radianes.

```
PROCEDURE arctan(x:REAL):REAL;
```
Devuelve el arcotangente de un ángulo x,expresado en radianes.

```
PROCEDURE real(x:INTEGER):REAL;
```
Realiza la conversión de tipos desde INTEGER a REAL.

```
PROCEDURE entier(x:REAL):INTEGER
```
Devuelve la parte entera de x.

Como puede observarse todos los procedimientos de esta librería son del tipo función.

Tambien M2 dispone de procedimientos de conversión, es decir, que convierten cadenas (strings)a números y viceversa. Seguidamente se detallan los procedimientos incluidos en el módulo NumberConversion.

```
PROCEDURE StringToCard(str:ARRAY OF CHAR; VAR num:CARDINAL;
                                          VAR done:BOOLEAN);
```
Convierte la cadena str a un número Cardinal que deposita en num. La variable done contendrá TRUE si se ha realizado la conversión con éxito.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
PROCEDURE StringToInt(str:ARRAY OF CHAR; VAR num:INTEGER;
                                         VAR done:BOOLEAN);
```

Conversión análoga a la anterior, pero en este caso es un número del tipo INTEGER, el que se desposita en num.

```
PROCEDURE StringToNum(str:ARRAY OF CHAR; base:BASE;
                                         VAR num:CARDINAL;
                                         VAR done:BOOLEAN);
```

Convierte la cadena str que contiene un número en una determinada base (octal, por ejemplo) a un número CARDINAL que se depositará en num. La variable done contendrá TRUE si se ha realizado con éxito la conversión.

```
PROCEDURE CardToString(num:CARDINAL; VAR str:ARRAY OF CHAR;
                                     anchura:CARDINAL);
```

Convierte el número CARDINAL num en una cadena str definida con una longitud de anchura caracteres.

```
PROCEDURE IntToString(num:INTEGER; VAR str:ARRAY OF CHAR;
                                   anchura:CARDINAL);
```

Convierte el número INTEGER num en una cadena str definida con una longitud de anchura caracteres.

```
PROCEDURE NumToString(num:CARDINAL; base:BASE;
                                    VAR str:ARRAY OF CHAR;
                                    anchura:CARDINAL);
```

Convierte el número CARDINAL num en una cadena str de tantos caracteres como se especifica en la anchura y que es la representación del número en la base que se especifica.

Otro módulo de librería, RealConv, nos permite hacer conversiones de números reales a cadenas y viceversa, a través de los procedimientos RealToString y StringToReal.

```
PROCEDURE RealToString(r:REAL; digits,ancho:INTEGER;
                       VAR str:ARRAY OF CHAR;
                       VAR okay:BOOLEAN);
```

Este procedimiento convierte un número real a una notación estándar, o científica, donde r, es el número real a ser representado, digit, es el número de dígitos que se situarán a la derecha del punto decimal, ancho, es la anchura máxima de la representación de salida, str, es la matriz de caracteres que contendrá la salida, y okay, es una variable boolana que nos indica si la conversión se ha hecho correctamente o no.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La clave para elegir una representación estándar o científica, es digits, si digits es ≥ 0, obtendremos una notación estándar, en caso contrario, será científica.

En caso de que la representación no quepa en ancho, entonces str se devuelve vacía, y okay es puesta a FALSE.

El mínimo requerido para ancho es:

Si digits < 0, entonces ancho debe ser ≥ ABS(digits) +8
Si digits ≥ 0, entonces ancho debe ser ≥ ABS(digts) +2 +antes

donde antes, es el número de dígitos antes del punto decimal de r en notación estándar, por ejemplo:

r=123.456 ---> antes=3
r=0.012 ---> antes=1

Veamos un programa ejemplo, con éste procedimiento, donde se visualiza un mismo número en notación estándar y científica.

Modula-2/86 converea.MOD

```
1  MODULE PruebaConversion;
2         FROM InOut IMPORT WriteString,WriteLn;
3         FROM RealConversions IMPORT RealToString;
4         VAR
5           str: ARRAY [0..30] OF CHAR;
6           resultado:BOOLEAN;
7  BEGIN
8    (* notación standard *)
9  (*Convierte 123.23 a 123.230*)
10
11 (*RealToString ( r ,dig,ancho,str,resultado)*)
12   RealToString(123.23, 3,  10,  str,resultado);
13   IF resultado THEN WriteString(str);WriteLn END;
14
15
16   (* notación científica *)
17 (*Convierte 123.23 a 1.2E+002*)
18
19 (*RealToString ( r ,dig,ancho,str,resultado)*)
20   RealToString(123.23,-1,  12,  str,resultado);
21   IF resultado THEN WriteString(str); END
22
23 END PruebaConversion.
```


© Del documento, los autores. Digitalización realizada por ULPGC. Bibliotecas Universitaria, 2006

```
Modula-2/86 Compiler Version V 1.10

=)    0 Error(s) found
```

Como se observa, resultado, debe inspeccionarse después de la llamada a RealToString, para verificar si la conversión se ha realizado con éxito.

```
PROCEDURE StringToReal(str:ARRAY OF CHAR;
                       VAR r:REAL;
                       VAR okay:BOOLEAN;)
```

Este procedimiento, convierte una matriz de caracteres a un número real, donde str, es la matriz a ser convertida, r, es la variable real que contendrá el resultado y okay es la variable booleana que nos dice si la conversión ha sido correcta.

Los caracteres en blanco de la cadena son pasados por alto, y los caracteres de control y el de espacio, se consideran, para señalar el final de la cadena.

A continuación se muestra un programa ejemplo que contiene el procedimiento gets, donde el parámetro de E/S es una cadena.Este programa tiene como objetivo tomar una cadena del teclado y una vez leída, visualizarla por pantalla. El procedimiento gets sirve para suplir ciertas limitaciones, que nos produce el ReadString, pues la lectura de una cadena con este procedimiento siempre termina con un carácter ASCII <= al space, por lo tanto gets permite que sean leídas lineas enteras de texto, en vez de una sola palabra de una vez.

Como se observa en la cadena del procedimiento, en esta no se especifican sus límites. A una declaración de este tipo se le denomina matriz abierta.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
Modula-2/86                    toma.MOD

 1 MODULE TomaCadena;
 2 FROM InOut IMPORT Read, Write, WriteString, EOL;
 3 VAR
 4   cadena: ARRAY [0..79] OF CHAR;
 5 PROCEDURE gets (VAR a:ARRAY OF CHAR);
 6 CONST
 7   BS = 8;
 8 VAR
 9   ch: CHAR;
10   i: CARDINAL;
11 BEGIN
12    i:=0;
13    REPEAT
14       LOOP (* Acepta el CR y los caracteres españoles *)
15         Read (ch);
16         IF (ORD (ch)>31) AND (ORD (ch)<127) OR  (ORD (ch)=8) AND (160) OR
17            (ORD (ch)=30)  OR (ORD (ch)>159) AND (ORD (ch)<169) OR
18            (ORD (ch)=130) OR (ORD(ch)=129)
19         THEN
20            EXIT
21         END;
22       END; (* loop *)
23       Write (ch);
24       IF ORD(ch)=8B THEN i:=i-1; Write(40C); Write(177C)
25       ELSIF (ch#EOL) AND (i<HIGH(a)) THEN
26         a[i]:=ch;
27         i:=i+1;
28       END;
29    UNTIL (ch=EOL) OR (i=HIGH(a));
30    IF i=HIGH(a) THEN Write(EOL) END;
31    a[i]:=CHR(0);
32 END gets;
33 BEGIN
34   gets(cadena);
35   WriteString(cadena);
36 END TomaCadena.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

La finalidad de la declaración de una matriz abierta en un procedimiento, es crear un procedimiento de propósito general que tome matrices como argumento.

Todas las matrices abiertas asumen que el límite inferior es 0, y el límite superior se puede determinar usando el procedimiento estándar HIGH.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 7

DECLARACION DE TIPOS

7.1 INTRODUCCION

En el capítulo 2, se estudiaron los tipos de datos elementales ya predefinidos en M2 (INTEGER, CARDINAL, CHAR, REAL, BOOLEAN y BITSET), pero además de estos, él programador puede utilizar otros tipos de datos creados por el mismo, a tenor de sus necesidades. Los tipos definibles por el usuario pueden ser:

Tipo array, tipo enumeración, tipo subrango, tipo procedure, tipo set, tipo record y tipo pointer. Además puede renombrar los tipos de datos estándar.

Todos los tipos definidos por el usuario, deben ser declarados en la sección de declaración de tipos del módulo.

La forma general de una declaración de tipo es

```
TYPE <nombre> = <type>;
```

Donde <nombre> es el nuevo tipo y type es uno de los tipos de datos ya predefinidos (Cardinal, Real, etc.) o bien un tipo definido previamente, por ejemplo:

```
TYPE
  Indice = CARDINAL;
  Balance = REAL;
  Contador = Indice; --->En este caso Indice es un tipo
                         definido previamente.
```

Una vez definidos los nuevos tipos, la siguiente declaración de variables es correcta.

```
VAR
  i,j,k:Indice;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
principio,fin:Indice;
haber,debe:Balance;
cont:contador;
```

Aquí i,j,k, principio, fin y cont son del tipo CARDINAL, debe y haber, del tipo REAL.

En cualquier caso, una variable puede declararse con un tipo determinado, haciendo uso del bloque de definición de tipos (TYPE), como en el caso anterior, o bien, en el propio bloque de declaración de variables (VAR) como se muestra a continuación.

```
VAR
  i,j,k:CARDINAL;
  principio, fin:CARDINAL;
  haber,debe:REAL;
  cont:CARDINAL;
```

El programador puede elegir cualquiera de las dos formas, si bien la primera tiene una ventaja adicional, pues es más fácil cambiar un tipo en la sección de declaración de tipos,que cambiar los tipos a las variables del mismo tipo en la sección de variables, por ejemplo; si en el caso anterior se necesitasen valores negativos para i,j,k , para principio, fin y cont, habría que cambiar tres veces CARDINAL por INTEGER, mientras que en el ejemplo que le precedía, sólo cambiando una vez CARDINAL por INTEGER, es decir, Indice=CARDINAL; por Indice=INTEGER; , estaría resuelta la corrección, lo que le da mayor flexiblidad y facilidad para cambios de este tipo.

Sin embargo, la auténtica ventaja de declarar el tipo de una variable en el bloque de definición de tipos, no es el hecho de renombrar tipos ya predefinidos, sino el crear nuevos tipos de datos, que es lo que veremos a continuación.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

7.2 EL TIPO ARRAY

Si bien este tipo de datos se vio ya en el capítulo 5, pues era inevitable a la hora de declarar matrices, en este caso lo mencionamos, como eslabón para introducirnos en otros tipos de datos definidos por el usuario.

La forma más usual de definir un string o cadena de caracteres en M2, es a través del tipo ARRAY.

A continuación se muestra un ejemplo, que define un nuevo tipo llamado Cadena80, que se utiliza para declarar variables cadenas, como nombre, calle y ciudad.

```
TYPE
  Cadena80 = ARRAY [0..79] OF CHAR;

VAR
  nombre:Cadena80;
  calle:Cadena80;
  ciudad:Cadena80;
```

Es muy común, dimensionar un string con 80 caracteres, pues son los que ocupa una línea de pantalla. De esta forma, cualquier variable del tipo Cadena80, puede ser usada con funciones como; ReadString o WriteString, por ejemplo: ReadString(nombre),WriteString(ciudad).

7.3 EL TIPO ENUMERADO

Uno de los tipos de datos más importantes en M2, es éste, pues permite crear un tipo de dato especificando los valores que puede tomar. Su forma general es:

```
TYPE
   <nombre> = (<N1>,<N2>,<N3>,....,<Nn>);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

donde **nombre**,es el nombre del nuevo tipo, y desde N1 a Nn son los valores que puede tomar una variable de este tipo.

Por ejemplo:

```
TYPE
  fruta=(Manzana,Naranja,Pera,Limon);
  ciudad=(Madrid,Barcelona,Bilbao);
  lenguaje=(Cobol,Pascal,Modula,Fortran);
  tipodia=(Lun,Mar,Mier,Jue,Vier,Sab,Dom);
```

También se pueden declarar variables con tipo enumerado,

```
VAR
  color:(rojo,amarillo,verde,azul);
  dia:tipodia;
  residencia:ciudad;
```

en este caso, día y residencia han sido declaradas con los tipos anteriores.

7.3.1 ORD Y VAL EN EL TIPO ENUMERADO

Los valores definidos por una variable del tipo enumerado, tienen asociados un número Cardinal, que indica el orden en que han sido escritos estos valores, por lo tanto en el caso que vimos anteriormente; manzana < naranja < pera <limon.

En el capítulo anterior se mencionaron estos procedimientos estándar, y ser vió ,que el uso más común de ORD era devolver el valor ASCII de un carácter.

En los tipos enumerados, se utiliza para que devuelva la posición relativa de un elemento, sabiendo que el primer elemento posee la posición 0, por ejemplo, en relación a los ejemplos anteriores resulta:

```
ORD(manzana)=0
ORD(Modula)=2
ORD(Mar)=1
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

En el caso de VAL, vimos que este procedimiento se utilizaba para convertir un dato de un tipo a otro.

En el tipo enumerado, se utiliza para determinar el elemento que se encuentra en un determinado orden, por ejemplo: VAL 'fruta,3)=limon .

7.3.2 INC Y DEC EN EL TIPO ENUMERADO

Ya se ha visto que los procedimientos estandar INC y DEC tenían la siguiente forma general, donde var se incrementaba

```
INC(<var>,<cantidad>);
DEC(<var>,<cantidad>);
```

o decrementaba según el valor cantidad, si este valor se omitía, cantidad era igual a 1.

En los tipos enumerados, también pueden utilizarse INC y DEC para asignar a una variable el sucesor o antecesor del valor actual.

El siguiente ejemplo nos sirve para aclarar esta diferencia.

```
TYPE
   fruta=(manzana,naranja,pera,limon);

VAR
   e:fruta;

BEGIN
   e:=manzana;
   INC(e); (*e=naranja*)
   INC(e,2); (*e=limon*)
   DEC(e,3); (*e=manzana*)
END
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

7.4 EL TIPO SUBRANGO

El tipo subrango se usa para restringir el rango de un tipo de dato predefinido, o el de un tipo enumerado definido previamente.

La forma general de un tipo subrango es:

```
TYPE
  <nombre>=[C1..C2];
```

Donde 'nombre', es el nombre del tipo subrango y C1,C2, son los límites inferior y superior del intervalo.

```
TYPE
  letras=['a'..'z'];
  año=[1..365];
  indice=[-100..100];
```

En este ejemplo, las cotas de letras son del tipo char, las de año del tipo cardinal y las de índice del tipo entero, lo que no está permitido, es que halla cotas del tipo real.

Cuando un límite inferior es negativo, se asume que las cotas son del tipo entero, en caso contrario se asume que son del tipo cardinal.

Se puede declarar un subrango de un tipo enumerado definido previamente, por ejemplo:

```
TYPE
  categoria=(director,subdirector,jefe,ayudante,auxiliar);
  yuppi=[director..jefe];
```

El siguiente programa ejemplo utiliza los tipos enumerado y subrango, y nos sirve para determinar los días laborables de cualquier mes de Enero, si le introducimos el primer día de semana que cae en 1 de Enero.

Modula-2/86 enumer.MOD

```
1 MODULE Dias;
2   FROM InOut IMPORT WriteLn,WriteString,ReadCard,WriteCard;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 3    TYPE
 4      Dia=(Lun,Mar,Mier,Jue,Vier,Sab,Dom);
 5      DiaSemana=[Lun..Vier];
 6      Mes=[1..31];
 7    VAR
 8      Enero: ARRAY[1..31] OF Dia;
 9      i,entrada: Mes;
10      j: DiaSemana;
11  BEGIN
12    WriteString('¿Qué dia de semana es el 1 de Enero? ');
13    ReadCard(entrada); WriteLn; (*Lun=1,Mar=2,....,Dom=7*)
14    FOR i:=1 TO 31 DO
15      Enero[i]:=VAL(Dia,(i+entrada-2) MOD 7)
16    END;
17    (*Sólo Visualiza las fechas de los dias laborables*)
18    WriteString('Los dias laborables del mes de Enero son:'); WriteLn;
19    FOR i:=1 TO 31 DO
20      FOR j:=Lun TO Vier DO
21        IF Enero[i]=j THEN WriteCard(i,3) END
22      END
23    END
24  END Dias.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

7.5 EL TIPO PROCEDIMIENTO

Una particularidad de M2 es declarar procedimientos de forma que, pueden llegar a ser tipos de determinadas variables, si bien esta consideración, no es extendible a procedimientos predefinidos en el propio lenguaje, como pueden ser INC o DEC por ejemplo.

Para realizar una declaración de este tipo, basta con especificar el número y el tipo de cada uno de los parámetros implicados.

Sean las declaraciones del tipo procedimiento;

```
TYPE
   funcion=PROCEDURE(REAL):REAL;
   procedi=PROCEDURE(CARDINAL,CARDINAL);
   variabl=PROCEDURE(VAR CHAR);
```

y las declaraciones de variables dentro del mismo programa:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
VAR
  C:funcion;
  W:procedi;
  R:variabl;
```

Dentro del cuerpo de instrucciones, podría escribirse;

```
BEGIN
   C:=Cos;
   W:=WriteCard;
   R:=Read;
```

de manera que resulta posible realizar llamadas a los procedimientos Cos, WriteCard y Read, de la forma:

C(x) en vez de Cos(x), W(x,5) en vez de WriteCard(x,5) y R(ch) en vez de Read(ch).

7.6 EL TIPO CONJUNTO

M2 es uno de los pocos lenguajes de propósito general, que sorporta datos de este tipo.

Los conjuntos se declaran en este lenguaje mediante la sentencia SET, su forma general es;

```
TYPE
  <nombrec.>=SET OF <type>;
```

donde nombrec. es el nombre del conjunto y type, sólo puede ser un tipo enumerado o subrango, por ejemplo:

```
TYPE
  conjunto=SET OF [0..2];
```

Las variables que se declaran con este tipo (que es un tipo conjunto), pueden tomar todos aquellos valores que puedan formarse con los elementos que se definieron en el tipo, es decir, con 0,1 y 2, por ejemplo:

```
VAR
  x:conjunto;
```

En este caso, los valores que puede tomar x son:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

{},{0},{1},{2},{0,1},{0,2},{1,2},{0,1,2}. En general 2^n, donde n es el número de elementos del conjunto base, en este caso 3.

Ya vimos en el capítulo segundo, el tipo estándar BITSET, esto puede especificarse ahora de mejor forma,

```
TYPE
  BITSET = SET OF [0..n-1];
```

donde n es la longitud de la palabra que utiliza el computador en cuestión.

Una vez visto, como se pueden definir conjuntos, veamos la forma general para asignar datos de tipo conjunto a una variable.

<varc.>:=<nombrec.>{conjunto};

En este caso nombrec. es el nombre que se dió a un tipo conjuto y varc. es una variable del mismo tipo.
por ejemplo: x:=conjunto{1,3}, en cambio, la asignación x:={1,3} sería incorrecta.

Para la declaración de conjuntos constantes se utilizará el mismo procedimiento, que para declarar constantes de otro tipo.

```
CONST
    Vacio={};
    impares={1,3,5,7,9};
```

En la declaración de conjuntos variables, también se pueden utilizar procedimientos habituales para variables.

```
TYPE
  EdadLaboral=[18..65];
  TipoEdad=SET OF EdadLaboral;
  Dia=(Lun,Mar,Mier,Jue,Vier,Sab,Dom);
  TipoDia=SET OF Dia;
  Meses=(Ene,Feb,Mar,Abr,May,Jun,Jul,Ago,Sep,Oct,Nov,Dic);
  TipoMes=SET of Meses;
  Verano=SET  OF  [Jul..Sep];-->Tipo subrango a partir de
                               un tipo enumerado.
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
VAR
  EdadCasarse,EdadLaboral:TipoEdad;
  DiasLaborables,Descanso:TipoDia;
  Trabajo:TipoMes;
  Vacaciones:Verano;
```

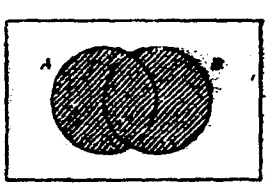
Asignaciones de valores a las variables anteriores, podrían ser:

```
EdadCasarse:=TipoEdad{20..40};
DiasLaborables:=TipoDia{Lun..Vier};
Descanso:=TipoDia{Sab,Dom};
Trabajo:=TipoMes{Ene..Jul,Sep..Dic};
Vacaciones:=Verano{Ago};
```

M2 dispone de varios operadores para el tratamiento de conjuntos, algunos de los cuales ya han sido comentados someramente, estos son:

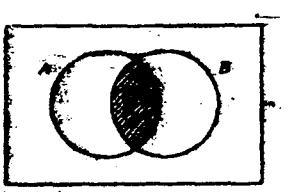
+ Union de conjuntos
* Intersección de conjuntos
- Diferencia de conjuntos
/ Diferencia simétrica de conjuntos
IN Pertenencia a un conjunto
= Igualdad entre conjuntos
<>,# Desigualdad entre conjuntos
<=,>= Inclusión entre conjuntos

Recordando con los siguientes diagramas de Venn el cometido de los cuatro primeros operadores, veamos algunos ejemplos de operaciones con estos:



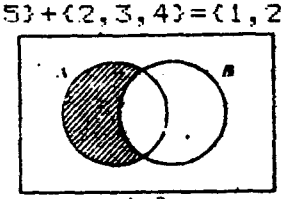
A+B

{}+{}={}
{1,2}+{}={1,2}
{1,3,5}+{2,3,4}={1,2,3,4,5}



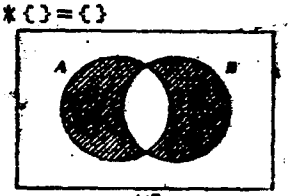
A*B

{1,3,5}*{2,3,4}={3}
{1..10,12}*{8..15}={8,9,10,12}
{1,2,3}*{}={}



A-B

{1,3,5}-{2,3,4}={1,5}
{1,2,3}-{}={1,2,3}
{2,3,4,6,7,8}-{1..10}={}



A/B

{1,3,5}/{2,3,4}={1,2,3,4,5}
{1,2,3}/{}={1,2,3}
{2,4,6}/{1..4,7}={1,3,6,7}

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Para denotar la pertenencia a un conjunto, se utiliza el operador IN, que tiene el mismo significado que el $\in$ algebraico.

```
3 IN {1,3,5}=TRUE
4 IN {1,3,5}=FALSE
(2+4) IN {1..10}=TRUE
```

El operador =, se usa para denotar la igualdad entre conjuntos, los operadores <>,# la desiguldad.

>=,<= se usan para las inclusiones entre conjuntos, por ejemplo, para incluir el conjunto A en B, se escribe A<=B.

Además de estos operadores, M2 dispone de dos procedimientos estándar INCL(S,i) y EXCL(S,i) que incluyen y excluyen respectivamente el elemento i del conjunto S.

El siguiente programa muestra un ejemplo simple de tratamiento de conjuntos, con algunas operaciones propias de estos. Consiste en un pequeño juego donde se pretende adivinar que países pertenecen al conjunto del Benelux, dentro de un conjunto de ocho países.

Modula-2/86 conjunto.MOD

```
 1  MODULE Benex;
 2    FROM InOut IMPORT Write,Read,WriteLn,WriteString;
 3    TYPE
 4      Estados=(Belgica,Holanda,Luxemburgo,Dinamarca,Austria,Suiza,Italia,Finlandia);
 5      Paises=SET OF Estados;
 6    VAR
 7      Benelux:Paises;
 8      Solucion:Paises;
 9      ch:CHAR;
10  BEGIN
11    Benelux:=Paises{Belgica,Holanda,Luxemburgo};
12    Solucion:=Paises{};
13    WriteString('Entre los siguientes paises,¿Cuáles cree que corresponden al Benelux?:');
14    WriteLn;WriteLn;
15    WriteString('Suiza,Italia,Belgica,Dinamarca,Luxemburgo,Austria,Holanda,Finlandia');
16    WriteLn;WriteString('q para salir');WriteLn;
17    REPEAT
18      Read(ch);Write(ch);
19      CASE ch OF
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
20          'b':INCL(Solucion,Belgica);
21          'h':INCL(Solucion,Holanda);
22          'l':INCL(Solucion,Luxemburgo);
23          'd':INCL(Solucion,Dinamarca);
24          'a':INCL(Solucion,Austria);
25          's':INCL(Solucion,Suiza);
26          'i':INCL(Solucion,Italia);
27          'f':INCL(Solucion,Finlandia)
28        ELSE
29        END;
30        WriteLn;
31      UNTIL ch='q';
32      IF Benelux<>Solucion THEN WriteString('Solución incorrecta! ') END;
33      IF Austria IN Solucion THEN WriteString('Austria no pertenece');
34      ELSIF Italia IN Solucion THEN WriteString('Italia no pertenece');
35      ELSIF Dinamarca IN Solucion THEN WriteString('Dinamarca no pertenece');
36      ELSIF Suiza IN Solucion THEN WriteString('Suiza no pertenece');
37      ELSIF Finlandia IN Solucion THEN WriteString('Finlandia no pertenece');
38      ELSE
39      WriteString('Solución correcta')
40      END;
41  END Benex.
```

```
Modula-2/86 Compiler Version V 1.10

==>     0 Error(s) found
```

7.7 EL TIPO REGISTRO

Un registro es un grupo de variables relacionadas lógicamente entre ellas y aunadas por otra variable, los elementos que constituyen un registro, pueden ser de tipos diferentes.

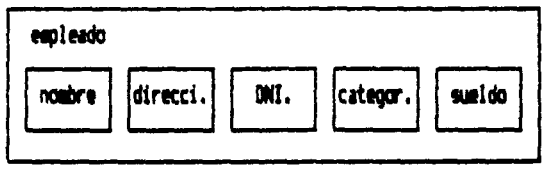
La declaración de un registro es, un tipo de dato que puede definir el usuario en la sección TYPE, del módulo o de un procedimiento, de la siguiente forma;

```
TYPE
  NombreReg = RECORD
                var1:tipo;
                .... ....
                varN:tipo;
              END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

donde **NombreReg**, es el nombre del registro, y de var1 a varN, están los nombres de las variables individuales que forman el registro, tipo, es el tipo de las variables.

El registro de los datos de un empleado como el de la figura;



podría declararse como sigue:

```
TYPE
   RegEmpleado=RECORD
                  nombre:ARRAY [0..35] OF CHAR;
                  direccion: ARRAY [0..40] OF CHAR;
                  DNI:CARDINAL;
                  categoria:(jefe,auxiliar,directivo);
                  sueldo:REAL;
                  END;
```

Para crear una variable del tipo RECORD, ésta se debe declarar en la sección de declaración de variables, en nuestro caso:

```
VAR
  empleado:RegEmpleado;
```

La forma general para el acceso a un campo individual de un registro, es la siguiente;

<NombreReg>.<NombreCampo>

donde **NombreReg** es el nombre de la variable RECORD, al punto (.), se le llama operador selector de campo y, **NombreCampo** es el nombre especificado del campo al cual se accede.

Asignaciones a los campos de este registro podrían ser:

```
empleado.nombre[15]:='X';
empleado.DNI:=65535;
empleado.categoria:=jefe;
empleado.sueldo:=68000.;
```

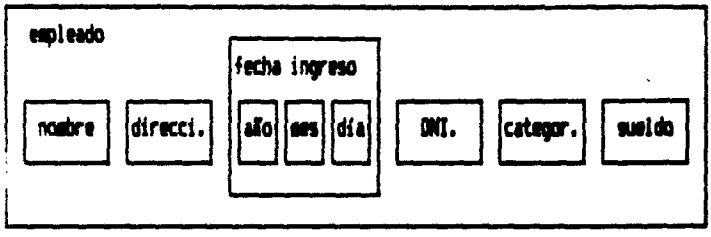
Los campos de los registros, se pueden usar, donde quiera que se use una variable simple, por ejemplo:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
WriteCard(empleado.DNI,6); ReadString(empleado.nombre);
```

También puede definirse un tipo registro dentro de otro, lo que constituye un anidamiento de registros.

La forma de declarar un tipo registro como el del ejemplo anterior, pero con otro campo adicional llamado fecha ingreso, que a su vez contiene los campos año, mes y día tal y como se muestra en la figura;



sería:

```
TYPE
   fecha=RECORD
         ano:[1900..2050];
         mes:[1..12];
         dia:[1..31];
         END;

   RegEmpleado=RECORD
               nombre:ARRAY [0..35] OF CHAR;
               direccion: ARRAY [0..40] OF CHAR;
               fechaingreso:fecha;
               DNI:CARDINAL;
               categoria:(jefe,auxiliar,directivo);
               sueldo:REAL;
               END;
```

o bien:

```
TYPE
   RegEmpleado=RECORD
               nombre:ARRAY [0..35] OF CHAR;
               direccion: ARRAY [0..40] OF CHAR;
               fechaingreso:RECORD
                            año:[1900..2050];
                            mes:[1..12];
                            día:[1..31];
                            END;
               DNI:CARDINAL;
               categoria:(jefe,auxiliar,directivo);
               sueldo:REAL;
               END;
```

Si la variable empleado, la declaramos con el tipo

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

anterior, es decir;

```
VAR
  empleado:RegEmpleado;
```

la siguiente asignación, sería correcta:

```
empleado.fechaingreso.año:=1972;
```

En M2, también pueden utilizarse conjuntamente las declaraciones de registros y matrices, lo que permite organizar gran cantidad de información, por ejemplo:

```
VAR
  departamento:ARRAY[0..99] OF RegEmpleado;
```

Esta declaración de variable, nos permite tener 100 registros del mismo tipo que en el ejemplo anterior, y si queremos hacer una asignación similar, al elemento 20 de la matriz departamento, resultaría:

```
departamento[20].fechaingreso.año:=1972;
```

En M2 es muy fácil pasar un registro a un procedimiento, debido a que los registros pueden ser usados como parámetros. Veamos el siguiente programa ejemplo que consiste en llenar de contenido los campos de un registro, para después llamar a un procedimiento que, visualiza por pantalla los datos asignados a los diversos campos.

Modula-2/86 regpro.MOD

```
1  MODULE RegAProc;
2    FROM InOut IMPORT ReadCard,WriteCard,WriteString,WriteLn,Write;
3    FROM RealInOut IMPORT WriteReal;
4  TYPE
5    fecha=RECORD
6            ano:[1900..2050];
7            mes:[1..12];
8            dia:[1..31];
9    END;
10   RegEmpleado=RECORD
11                 nombre:ARRAY [0..35] OF CHAR;
12                 direccion:ARRAY [0..40] OF CHAR;
13                 FechaIngreso:fecha;
14                 DNI:CARDINAL;
15                 categoria:(jefe,auxiliar,directivo);
16                 sueldo:REAL;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
17    END;
18  VAR
19     empleado:RegEmpleado;
20  PROCEDURE visualizaReg(C:RegEmpleado);
21  BEGIN
22    WriteString('Nombre: '); WriteString(C.nombre); WriteLn;
23    WriteString('Direccion: '); WriteString(C.direccion); WriteLn;
24    WriteString('Fecha de Ingreso: '); WriteCard(C.FechaIngreso.dia,2);
25    WriteString('-'); WriteCard(C.FechaIngreso.mes,2);
26    WriteString('-'); WriteCard(C.FechaIngreso.ano,2); WriteLn;
27    WriteString('DNI: '); WriteCard(C.DNI,6); WriteLn;
28    WriteString('Categoria: ');
29    CASE C.categoria OF
30       jefe: WriteString('Jefe') |
31       auxiliar: WriteString('Auxiliar')|
32       directivo: WriteString('Directivo')|
33    END;
34    WriteLn;
35    WriteString('Sueldo: ');WriteReal(C.sueldo,8);WriteLn;
36  END visualizaReg;
37  BEGIN
38  (*asignaciones para pasar información a los campos de la variable RECORD empleado*)
39     empleado.nombre:= 'Wirth Niklaus';
40     empleado.direccion:= "MODULA - 2";
41     empleado.FechaIngreso.dia:=7;
42     empleado.FechaIngreso.mes:=4;
43     empleado.FechaIngreso.ano:=1980;
44     empleado.DNI:=559335;
45     empleado.categoria:=jefe;
46     empleado.sueldo:=68000.;
47     visualizaReg(empleado); (*llamada al procedimiento*)
48  END RegAProc.
```

```
Modula-2/86 Compiler Version V 1.10
=)     0 Error(s) found
```

Como se observa, el parámetro formal C, del procedimiento, se declara del tipo RegEmpleado, y cuando se llama al procedimiento, el argumento (empleado) también es del mismo tipo, con lo que se consigue pasar el registro con todos sus campos al procedimiento visulizaReg.

Ha sido posible pasar el registro al procedimiento, definiendo primero un tipo RECORD y luego una variable de ese tipo, sin embargo, no hubiese sido factible si se hubiese hecho una declaración directa, es decir, declarando la variable con su tipo directamente en la sección VAR.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

7.7.1 LOS REGISTRO VARIANTES

Hasta ahora los registros que se han visto, tenían un número fijo de campos, sin embargo, es posible que el número de éstos, cambie dinámicamente acomodándose a las diferentes necesidades. Los registros variantes o variables, permiten que un registro disponga de un número u otro de campos dependiendo del valor que tome uno de ellos, a éste, se le llama campo discriminante.

Supongamos que queremos diseñar un registro que contenga tres campos fijos, que son; número, nombre y categoría, éste último puede tomar dos valores; subalterno o jefe, y dependiendo que tome uno u otro valor, el registro tendrá uno o dos campos más respectivamente, si categoría=subalterno, el registro contiene un campo más, llamado salario y si categoría=jefe, el registro tendrá dos campos más, llamados; sueldo y dietas. Como se observa, categoría sería el campo discriminante.

La declaración de tal tipo sería:

```
TYPE
   tipocat=(subalterno,jefe);
   empleado=RECORD
              numero:CARDINAL;
              nombre:ARRAY[0..40] OF CHAR;
              categoria:tipocat;
                CASE categoria:tipocat OF
                     jefe:sueldo:CARDINAL;
                          dietas:CARDINAL|
                     subalterno:salario:CARDINAL
                END;(*del CASE*)
              END; (*del RECORD*)
```

La forma general de la porción variante de un registro es;

```
CASE <campdisc>:<tipo> OF
     <const.1>:<listavar.1>|
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
                    ..... .....
                    <const.N:<listavar.N>
              END;
```

donde campdisc (campo discriminante), es la variable que se usa para determinar, que campo se selecciona, tipo, es el tipo de campdisc, los listavar.i, son los campos variables (con sus tipos), que tomará el registro en caso de que campdisc tome el valor const.i.

7.7.2 LA SENTENCIA WITH

A veces el escribir continuamente el nombre de una variable registro y el punto (.) selector de campo, puede llegar a ser tedioso. M2 nos ayuda a solucionar este pequeño inconveniente con la sentencia WITH, que nos sirve para especificar el registro con que estamos trabajando. Su forma general es:

```
WITH <nombre.reg> DO
     <cuerpo del código>
END;
```

nombre.reg es el registro en uso, de forma que, una vez la sentencia WITH se ha implementado, todo lo que esté en el 'cuerpo del código', no necesita usar el nombre del registro.

Por ejemplo, en el último programa ejemplo, a partir de la linea 42, se podría escribir:

```
41  (*asignaciones para pasar información a los campos de la variable RECORD empleado*)
42    WITH empleado DO
43      nombre:= 'Wirth Niklaus';
44      direccion:= "MODULA - 2";
45      FechaIngreso.dia:=7;
46      FechaIngreso.mes:=4;
47      FechaIngreso.ano:=1980;
48      DNI:=55935;
49      categoria:=jefe;
50      sueldo:=68000.;
51    END;
52      visualizaReg(empleado); (*llamada al procedimiento*)
53  END RegAProc.
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Para evitar denominaciones extensas, también se podrían utilizar sentencias WITH anidadas, y el mismo ejemplo anterior, quedaría:

```
41  (Asignaciones para pasar información a los campos de la variable RECORD empleado#)
42    WITH empleado DO
43      nombre:= 'Wirth Niklaus';
44      direccion:= "MODULA - 2";
45      WITH FechaIngreso DO
46        dia:=7;
47        mes:=4;
48        ano:=1980;
49      END;
50      DNI:=55935;
51      categoria:=jefe;
52      sueldo:=68000.;
53    END;
54      visualizaReg(empleado); (#llamada al procedimiento#)
55  END RegAProc.
```

7.8 EL TIPO PUNTERO

La mayoría de los programas ejemplos que hemos visto hasta ahora, tenían un número fijo de variables, lo cual, es normal para programas que trabajan con una pequeña cantidad de información, sin embargo, se puede dar el caso de programas que manejen grandes cantidades de datos, y que el número de estos pueda disminuir o crecer a medida que el programa se ejecuta. Es debido a estas estructuras de datos dinámicas, por lo que M2 incorpora el tipo puntero, pués facilita el trabajo con ellas. Ejemplos de tales estructuras podrían ser las listas o las pilas.

Un puntero es una variable que contiene una dirección de memoria, que apunta a la zona donde se guarda información.

7.8.1 DECLARACION DE PUNTEROS

Para declarar una variable puntero, se requiere que se

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

defina primero, el tipo puntero en la sección TYPE del programa, la forma general es;

```
TYPE
   <TipoPuntero> = POINTER TO <tipo>;
```

donde TipoPuntero,es el nombre del nuevo tipo, y tipo, es el tipo de datos al que el puntero podrá apuntar. Veamos la siguiente declaración ejemplo:

```
TYPE
  tipoP = POINTER TO CARDINAL;
VAR
  P:tipoP;
```

La variable P, no contendrá por si misma un valor cardinal, sino la dirección de una región de memoria que pueda contener un dato de este tipo, es decir, P se puede usar para apuntar a un valor cardinal.

7.8.2 ASIGNACIONES, NEW Y DISPOSE

La asignación de un valor a un lugar de memoria, se hace a través del operador ^, por ejemplo; P^:=130; . En este caso 130 se situará en la posición de memoria que indique el puntero P, pero, ¿ cuál es la dirección a la que apunta P ?, ¿ en qué lugar de memoria será alojado el 130 ?. La respuesta, es que nos hace falta asignarle al puntero P una dirección, que apunte a la zona de memoria donde se va a situar el 130, la única forma de conseguirlo, es a través del procedimiento NEW, cuya forma general es;

```
NEW(<VarPuntero>);
```

donde VarPuntero es una variable de tipo puntero, por lo

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

tanto, si sería correcto escribir:

```
NEW(P);
P^:=130;
```

Con la instrucción NEW(P), le estamos asignando al puntero P la dirección de memoria de una región, de un tamaño suficinete para albergar un dato del tipo cardinal. Nosotros no sabremos (ni nos interesa) explícitamente, cual es esa dirección, aunque si sabemos que se encuentra en una zona de memoria denominada Heap, reservada para estos usos.

En las siguientes declaraciones de puntero que apuntan a tipos algo más complejos, se observa, que primero debe ser definido un tipo y luego el tipo puntero que apunta al tipo definido previamente.

```
TYPE
   str=ARRAY [0..79] OF CHAR;
   PuntStr=POINTER TO str;
VAR
   S:PuntStr;

TYPE
   Reg=RECORD
         I:INTEGER;
         S:ARRAY [0..9] OF CHAR;
       END;
   PuntReg:POINTER TO Reg;
VAR
   Punt:PuntReg;
```

Es importante hacer notar que en el primer caso, después de un NEW(S), el puntero S apuntará a una región de memoria, de 80 bytes de largo, (un dato de tipo CHAR ocupa un byte de memoria) pues una llamada a NEW, prepara un espacio de memoria suficiente para albergar los datos a los que apunta el puntero. En el segundo caso serían 12 bytes, (un dato de tipo INTEGER, ocupa dos bytes de memoria).

Cuando se desea liberar la memoria que se utilizó para

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

albergar un dato, se llama al procedimiento DISPOSE, que además asigna a la variable puntero un valor llamado NIL, (un puntero de este tipo, con contenido NIL, es un puntero que apuñta a ninguna parte) su forma general es;

DISPOSE (<VarPuntero>);

donde VarPuntero, es una variable de tipo puntero.

Las llamadas a los procedimientos estándar, NEW y DISPOSE, llaman a su vez a los procedimientos Allocate y Deallocate de la librería Storage, por lo tanto, la siguiente linea de importación debe estar incluida siempre en cualquier programa que use NEW y DISPOSE.

FROM Storage IMPORT ALLOCATE,DEALLOCATE;

El siguiente programa ejemplo, nos ilustra lo visto hasta ahora.

```
Modula-2/86                    puntero.MOD

 1  MODULE EjemploPuntero;
 2    FROM InOut IMPORT WriteInt;
 3    FROM Storage IMPORT ALLOCATE,DEALLOCATE;
 4    TYPE
 5      tipoP = POINTER TO INTEGER;
 6    VAR
 7      P:tipoP;
 8  BEGIN
 9    NEW(P);
10    P^:=-100;
11    WriteInt(P^,5);
12    DISPOSE(P);
13  END EjemploPuntero

-------------------------------------------------

Modula-2/86 Compiler Version V 1.10

==>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

7.8.3 UTILIZACION PRACTICA DE LOS PUNTEROS

En el ejemplo anterior se vió como alojar un dato en el Heap de memoria, utilizando un puntero, sin embargo, ésto raramente se hace, pues apenas se ha obtenido alguna ventaja de la utilización de un tipo puntero

La verdadera ventaja, radica en la utilización de los punteros en estructuras dinámicas de datos, donde éstos disminuyen o aumentan a medida que el programa se ejecuta, un ejemplo de ésto, sería una lista encadenada, que consiste en una lista, generalmente de registros, donde cada uno de ellos contiene un campo, que nos dice la dirección donde se va a albergar el siguiente registro de la lista.

A continuación, como ejemplo de una lista encadenada, se muestra una pequeña base de datos, que toma nombres por teclado y los relaciona con un número. Conviene destacar que se crea un puntero, que apunta a un registro con tres campos que son; número, nombre y próximo, donde próximo, es una variable de tipo puntero que apunta a registros del mismo tipo, y sirve para guardar la dirección del próximo registro, en este caso se declara el puntero y después el registro al que apunta.

Modula-2/86 listaen.MOD

```
1  MODULE Listaencadenada;
2    FROM InOut IMPORT WriteInt,Write,WriteLn,ReadString,WriteString;
3    FROM Strings IMPORT CompareStr;
4    FROM Storage IMPORT ALLOCATE,DEALLOCATE;
5  TYPE
6    TipoP = POINTER TO Reg;(*Se declara primero el puntero *)
7    Reg = RECORD            (*y después el objeto al que apunta*)
8            numero:INTEGER;
9            nombre: ARRAY [0..79] OF CHAR;
10           proximo:TipoP;
11         END;
12 VAR
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
13   P,P1,principio:TipoP;
14   I:INTEGER;
15 BEGIN
16   Write(14C);
17   NEW(P);
18   principio:=P;(*Se salva la posición inicial*)
19   I:=0;
20   REPEAT
21     WriteString('Introduzca un nombre: ');
22     ReadString(P^.nombre);WriteLn;
23     P^.numero:=I;
24     I:=I+1;
25     P1:=P; (*Asignaciones entre punteros del mismo tipo, están permitidas*)
26     NEW(P);
27     P1^.proximo:=P;
28   UNTIL(CompareStr(P1^.nombre,'')=0);(*Si no toma un nombre,en la toma de datos*)
29   P1^.proximo:=NIL;                  (*El último próximo de la lista,apunta a ninguna parte*)
30   P:=principio;
31   WHILE P^.proximo # NIL DO
32     WriteString(P^.nombre);
33     WriteInt(P^.numero,5);
34     WriteLn;
35     P:=P^.proximo
36   END;
37 END Listaencadenada.
```

```
Modula-2/86 Compiler Version V 1.10

    =>    0 Error(s) found
```

7.8.4 DESBORDAMIENTO DEL HEAP

Como ya sabemos, con la instrucción NEW, se asigna un lugar de memoria a los datos que direcciona el puntero, sin embargo, puede suceder que después de ésta, no quede espacio en el Heap para albergar ningún dato más y puede, que con la instrucción NEW estemos colisionando con la zona Stack de memoria, lo cual afectaría fatalmente a la ejecución del programa, por lo tanto, conviene saber antes de la ejecución de un NEW, si hay espacio disponible en el Heap. Previniendo esto, M2 proporciona un procedimiento en la librería Storage, llamado Available,que describimos con su lista de parámetros:

```
Available (nbytes:CARDINAL):BOOLEAN;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Este procedimiento nos determina, si un número de bytes determinado, podría ser alojado (respuesta TRUE), o no (respuesta FALSE). Por lo tanto, antes de alojar un dato de dos bytes de tamaño, se podría escribir;

IF Available(2) THEN NEW(P) END;

de esta forma, se pueden evitar errores de ejecución.

Sin embargo, a menudo, no se sabe con exactitud el tamaño de una variable, sobre todo en el caso de los registros, en este caso, se puede determinar con el procedimiento TSIZE, importado de la librería SYSTEM, (que veremos en un próximo capítulo) que devuelve el número de bytes necesarios para guardar una variable de cualquier tipo.

PROCEDURE TSIZE(t:cualquier tipo):CARDINAL;

El siguiente programa ejemplo, nos determina, si hay espacio disponible en el Heap, utilizando los procedimientos Available y TSIZE conjuntamente.

```
Modula-2/86                    heap.MOD

 1 MODULE PruebaHeap;
 2   FROM InOut IMPORT WriteString;
 3   FROM Storage IMPORT ALLOCATE,DEALLOCATE,Available;
 4   FROM SYSTEM IMPORT TSIZE;
 5 TYPE
 6   TipoReg = RECORD
 7               R:REAL;
 8               B:BOOLEAN;
 9   END;
10   TipoP =POINTER TO TipoReg;
11 VAR
12   P:TipoP;
13 BEGIN
14   IF Available (TSIZE(TipoP)) THEN
15     WriteString('Hay espacio disponible');NEW(P)
16   ELSE
17     WriteString('Fuera de memoria');P:=NIL
18   END;
19 END PruebaHeap.

Modula-2/86 Compiler Version V 1.10
=>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 8
LOS MODULOS

8.1 INTRODUCCION

Hasta ahora, hemos estado trabajando con módulos de programa, que importaban procedimientos especiales (ReadCard, WriteCard, etc.) de librerías, (InOut, Storage, etc.) proporcionadas por el propio compilador.

Sin embargo, M2 nos da la posibilidad de crear nuestras propias librerías o módulos de librería y utilizarlas de forma análoga, a las que hemos visto hasta ahora.

Para construir librerías, M2 proporciona dos herramientas; los módulos de definición y los módulos de implementación, que veremos más adelante.

El hecho de poder construir módulos de librerías, permite la partición de un programa en módulos, lo que da una gran flexibilidad sobre todo, a la hora de realizar grandes proyectos de software, por lo que se puede repartir el trabajo por bloques a diversos equipos de programadores.

Otra ventaja adicional, es la compilación separada, es decir, cada equipo puede trabajar independientemente y posteriormente, linkarse todos las piezas del programa, esto posee la ventaja añadida, de que siempre es posible modificar la parte de implementación de cualquier módulo, sin necesidad de recompilar los demás, ni los programas que importen procedimientos de tal módulo, sólo habría que linkar, con el ahorro en tiempo de compilación que esto supone.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

8.2 CREACION DE UN MODULO DE LIBRERIA

Crear una librería en M2 consta de dos pasos, el primero consiste en definir que es lo que queremos hacer, lo cual se realiza en el DEFINITION MODULE, y el segundo, en implementar como lo vamos a hacer, que se realiza en el IMPLEMENTATION MODULE. Los dos pasos, se basan en módulos distintos, pero relacionados.

8.2.1 MODULOS DE DEFINICION

El contenido de un módulo de definición, consta fundamentalmente de un conjunto de declaraciones de objetos, (constantes, variables, tipos, procedimientos) para exportar.

Su forma general es:

```
DEFINITION MODULE <nombremod>;
<lista de importaciones>;
EXPORT QUALIFIED <lista de identificadores>;
<declaración de objetos>;
END<nombremod>.
```

La lista de importaciones, es igual a la de los módulos de programa vistos hasta ahora.

La lista de identificadores, es donde se especifican exactamente que objetos serán conocidos fuera de la librería.

Generalmente, en la sección de declaración de objetos, sólo se definen los nombres de los procedimientos con sus parámetros, en el caso de que tipos, constantes o variables se necesiten por otras rutinas, es aquí también donde se definen.

Supongamos, que queremos definir un módulo que trabaje

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

con vectores de diez elementos, multiplicándolos y visualizándolos. Su módulo de definición quedaría como sigue.

```
Modula-2/86                    vectors.def

 1  DEFINITION MODULE Vectors;
 2     EXPORT QUALIFIED Vector,MultVect,VisuaVect;
 3     TYPE
 4       Vector = ARRAY [1..10] OF REAL;
 5     PROCEDURE MultVect(V1,V2:Vector;VAR V3:Vector);
 6     PROCEDURE VisuaVect(V:Vector);
 7  END Vectors.

 ------------------------------------

 Modula-2/86 Compiler Version V 1.10

 ==>     0 Error(s) found
```

En el caso de que fuese un registro lo que se exporta, los campos del registro también serían exportados.

```
Reg:RECORD
      campo1:CHAR;
      campo2:CAHR;
```

En este caso, si se exporta Reg, también se exportan campo1 y campo2.

8.2.2 LOS MODULOS DE IMPLEMENTACION

Como se puede deducir de lo visto anteriormente, un módulo de definición por sí mismo, no es ejecutable, pues sólo describe lo que se va a hacer. El como se va a hacer, es decir, la implementación de los objetos definidos en el DEFINITION MODULE, es lo que debe constar en el módulo de implementación.

En si mismo, un módulo de implementación es exactamente igual a un módulo de programa, excepto en dos aspectos; El primero, es que la palabra clave IMPLEMENTATION, precede a la

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

palabra MODULE en la primera linea, y segundo, que el cuerpo principal del programa es opcional (BEGIN).

Seguidamente se incluye el módulo de implementación de Vectors, cuyo módulo de definición se vio en el apartado anterior, además de un programa ejemplo de utilización del módulo de libreia Vectors.

Modula-2/86 vectors.MOD

```
 1 IMPLEMENTATION MODULE Vectors;
 2   FROM RealInOut IMPORT WriteReal;
 3   FROM InOut IMPORT Write;
 4
 5 PROCEDURE MultVect(V1,V2:Vector;VAR V3:Vector);
 6   VAR
 7     i:[1..10];
 8   BEGIN
 9     FOR i:=1 TO 10 DO
10       V3[i]:=V1[i]*V2[i];
11     END;
12 END MultVect;
13
14 PROCEDURE VisuaVect(V:Vector);
15   VAR
16     i:[1..10];
17   BEGIN
18     Write('[');
19     FOR i:= 1 TO 10 DO
20       WriteReal(V[i],5);
21       IF i<10 THEN Write(',') END;
22     END;
23     Write(']');
24 END VisuaVect;
25
26 END Vectors.
```

Modula-2/86 Compiler Version V 1.10

=> 0 Error(s) found

Modula-2/86 llamavect.MOD

```
 1 MODULE LLamaVectors;
 2   FROM Vectors IMPORT Vector,MultVect,VisuaVect;
 3   FROM RealInOut IMPORT WriteReal,ReadReal;
 4   FROM InOut IMPORT WriteLn,Write;
 5   VAR
 6     A,B,C:Vector;j:CARDINAL;
 7 BEGIN
 8   FOR j:=1 TO 10 DO
 9     ReadReal(A[j]);Write('*');
10     ReadReal(B[j]);Write(',')
11   END;
12   MultVect(A,B,C);WriteLn;
13   VisuaVect(C)
14 END LLamaVectors.
```

Modula-2/86 Compiler Version V 1.10

=> 0 Error(s) found

En la introducción a este tema, destacamos la ventaja que supone la compilación separada de los módulos. Supongamos que en la linea 18 y 23 del módulo de implementación, sustituimos tanto [como] por !, y lo volvemos a compilar, en el caso de

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

que quisiéramos hacer correr el programa llamavect, utilizando el procedimiento VisualizaVect. (modificado) de la librería Vectors, no habría necesidad de recompilar llamavect, sino linkarlo. Por supuesto, que tanto las correciones que se han hecho, como las dimensiones del progrma, son irrelevantes, pero nos puede dar una idea de las ventajas de la compilación separada.

Anteriormente se dijo, que en un módulo de implementación, el cuerpo principal del programa era opcional y es cierto, generalmente, la única razón por la que se pondría un cuerpo de programa, es para ejecutar algunas inicializaciones necesarias para las rutinas de las librerías, tal y como se verá en el siguiente ejemplo, donde creamos la librería Pila con los procedimientos Push y Pop, que sirven para el manejo de datos en forma de pila (LIFO).

Modula-2/86 pila.MOD

```
1  IMPLEMENTATION MODULE Pila;
2    FROM InOut IMPORT WriteString,WriteLn;
3    CONST
4      tamanopila=10;
5    VAR
6      pila:ARRAY [0..tamanopila] OF CHAR;
7      apunt:CARDINAL;
8
9    PROCEDURE Push(a:CHAR);(*devuleve mensaje, si la pila está llena*)
10     BEGIN
11       IF (apunt+1)>tamanopila THEN
12       WriteString('La pila está llena');WriteLn; RETURN
13       ELSE INC(apunt);
14       END;
15       pila [apunt]:=a;
16       RETURN
17       END Push;
18
19   PROCEDURE Pop(VAR b:CHAR);(*devuleve mensaje, si la pila está vacia*)
20     BEGIN
21       IF apunt=0 THEN WriteString('La pila está vacía');WriteLn; RETURN END;
22       b:=pila [apunt];
23       DEC (apunt);
24       RETURN
```

Modula-2/86 pila.def

```
1  DEFINITION MODULE Pila;
2    EXPORT QUALIFIED Push,Pop;
3    PROCEDURE Push(a:CHAR);
4    PROCEDURE Pop(VAR b:CHAR);
5  END Pila.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
25        END Pop;
26
27  BEGIN (*Cuerpo de programa, para inicializaciones necesarias*)
28     apunt:=0;
29     WriteString('Apuntador de pila incicializado.');WriteLn
30  END Pila.
```

```
Modula-2/86 Compiler Version V 1.10

==>     0 Error(s) found
```

En este caso el cuerpo principal del módulo de implementación, es utilizado para inicializar el apuntador de la pila, y las sentencia WriteString, para enviar un mensaje cuando esto se produzca.

En el siguiente programa usapila, se muestra una utilización de la librería Pila.

Modula-2/86 usapila.MOD

```
 1  MODULE UsaPila;
 2    FROM InOut IMPORT Write,WriteLn;
 3    FROM Pila IMPORT Push,Pop;
 4  VAR
 5    x:CARDINAL;
 6    varsalida:CHAR;
 7  BEGIN
 8    Push('2'); Push('-'); Push('A'); Push('L'); Push('U'); Push('D');
 9    Push('O'); Push('M');
10    FOR x:=1 TO 8 DO
11      Pop(varsalida);
12      Write(varsalida);
13    END;
14    WriteLn
15  END UsaPila.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

De la ejecución del programa , resulta en primer lugar, la visualización del mensaje 'Apuntador de pila inicializado' y posteriormente MODULA-2, esto quiere decir, que el cuerpo

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

principal del IMPLEMENTATION MODULE se ejecuta antes que el del módulo usapila, concretamente, cuando se realiza la importación de los objetos de la librería Pila.

8.3 IMPORTACION CUALIFICADA

Hasta ahora todas las listas de importaciones, han tenido la siguiente forma general:

FROM <librería> IMPORT <lista de procedimientos>;

Pero puede ocurrir, que halla conflicto entre elementos que con un mismo nombre, estén en diferentes librerías, como es el caso del WriteLn en el siguiente ejemplo.

```
Modula-2/86                          import.MOD

  1 MODULE Importacion;
  2    FROM InOut IMPORT WriteString,WriteLn;
  3    FROM Terminal IMPORT WriteLn;
!!!!!                                  ^ 70
! 70: identifier specified twice in importlist

  4 BEGIN
  5   WriteString('¡Esto es una prueba!');
  6   WriteLn;
  7   WriteLn;
  8 END Importacion.
 ------------------------------------------------

  Modula-2/86 Compiler Version V 1.10

  =>    1 Error(s) found
```

Sin embargo M2, nos proporciona otro sistema de importaciones, para referinos al elemento concreto de cada librería. Este consiste en sustituir la lista de importaciones por IMPORT(librería), y a la hora de llamar al procedimiento en cuestión, cualificarlo, es decir, darle la forma: <Nombre módulo>.<Identificador>, de forma que el

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

conflicto que supuso el error en la compilación anterior, se podría solucionar de la siguiente forma.

```
Modula-2/86                          Import2.MOD

 1 MODULE Importacion;
 2   IMPORT InOut;
 3   IMPORT Terminal;
 4 BEGIN
 5   InOut.WriteString("¡Esto es una prueba!");
 6   Terminal.WriteLn;
 7   InOut.WriteLn
 8 END Importacion.
```

```
Modula-2/86 Compiler Version V 1.10

:=>   0 Error(s) found
```

8.4 LOS MODULOS LOCALES

Se entiende por módulos locales, a aquellos que se definen de forma anidada dentro de otros módulos, su forma general es la siguiente.

```
MODULE <Nombre del módulo>;
IMPORT<Lista de elementos importados>;
EXPORT (QUALIFIED) <Lista de elementos exportados>;
<Declaración de constantes>;
<Declaración de tipos>;
<Declaración de variables>;
<Declaración de procedimientos>;
BEGIN
<Secuencia de sentencias>;
END <Nombre del módulo>;
```

Se observa que difiere fundamentalmente, de un módulo de programa, en la segunda y tercera linea. Los objetos declarados dentro del módulo, son sólamente visibles o utilizables por tal módulo. La única forma de trasiego de información de un módulo local con el exterior, es a través del IMPORT y el EXPORT, de forma que, un módulo local, sólo puede importar elementos del módulo exterior a él y al exportar, lo hace al módulo exterior a él.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Los módulos locales, no pueden ser compilados por separado.

Veamos el siguiente programa ejemplo, que contiene un módulo local,que a su vez tiene un procedimiento interno, que se usa para sumar números obtenidos por teclado.

```
Modula-2/86                    modloc.MOD

 1 MODULE ModLoc;
 2   FROM InOut IMPORT Write,WriteString,WriteCard,ReadCard,WriteLn;
 3
 4         MODULE ModuloLocal;
 5            IMPORT WriteString,WriteLn,ReadCard,Write;
 6            EXPORT cont,sum;
 7            VAR
 8              cont:CARDINAL;
 9
10            PROCEDURE sum;(*Procedimiento dentro del módulo local*)
11              VAR
12                c:CARDINAL;
13            BEGIN
14              REPEAT
15                WriteString('Introduzca un número: ');
16                ReadCard(c);WriteLn;
17                cont:=cont+c;
18              UNTIL c=0;
19            END sum;
20
21         BEGIN
22           Write(14C);(*Limpia la pantalla*)
23           cont:=0; (*Se inicializa el ModuloLocal*)
24         END ModuloLocal;
25
26 BEGIN
27   sum;WriteLn;
28   WriteString('La suma es: '); WriteCard(cont,5)
29 END ModLoc.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

El uso principal de un módulo local, es evitar la limitación que supone en algunos casos, la destrucción de las variables internas de los procedimientos, una vez se ha

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

concluido la ejecución de estos.

Cuando se llama a un procedimiento, una vez se sale de estos, el estado de sus variables internas se pierde, son irrecuperables, mientras que en un módulo local, se pueden recuperar desde el módulo exterior a él.

Como se puede observar, en la sentencia EXPORT del programa ejemplo, no se incluyó el QUALIFIED, esto ocurre generalmente en la mayoría de los módulos locales, sin embargo, se utiliza en ciertos casos, para evitar conflictos de objetos con la misma denominación.

Supongamos el siguiente caso;

```
MODULE M1;
  VAR
    A,B:CARDINAL;
    .
    .
  MODULE M2;
    EXPORT A,C;
    VAR
      A,C:CARDINAL;
      .
      .
  END M2;

BEGIN
  .
  .
  A:=100; (*¿A qué A se refiere?*)
END M1.
```

En este caso hay conflicto entre las aes de M1 y M2, la forma de solucionarlo, sería sustituir la linea EXPORT A,C; por EXPORT QUALIFIED A,C; , de forma que con esta sentencia cualificaríamos la variable A y nos referiríamos a ella de la forma M2.A=100 (*Esta es la A de M2*), y para referirnos a la A de M1, sería; A:=200;

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

8.5 MODULOS EJEMPLO

El siguiente módulo de librería nos permite utilizar los procedimientos descritos en el módulo de definición, ahora bien, para que estos funcionen correctamente, es preciso configurar el sistema operativo con los comandos ansi (en caso de que estemos utilizando el MS-DOS),de lo contrario lo que se visulizaría por pantalla, sería una secuencia de caracteres que no entendería el sistema operativo.

```
Modula-2/86                    util.def

 1  DEFINITION MODULE Util;
 2
 3        EXPORT QUALIFIED Cls,Cll,PxPy,Cuu,Cud,Bs,Sp,Vi,Neg,Tilt,Vv;
 4
 5        PROCEDURE Cls; (*Borra la Pantalla*)
 6
 7        PROCEDURE Cll; (*Borra una linea*)
 8
 9        PROCEDURE PxPy(Px,Py:CARDINAL); (*Posiciona el cursor respecto al origen*)
10
11        PROCEDURE Cuu(u:CARDINAL); (*Mueve el cursor u posiciones hacia arriba*)
12
13        PROCEDURE Cud(d:CARDINAL); (*Mueve el cursor d posiciones hacia abajo*)
14
15        PROCEDURE Bs(b:CARDINAL); (*Mueve el cursor b posiciones hacia atrás*)
16
17        PROCEDURE Sp(s:CARDINAL); (*Mueve el cursor s posiones hacia adelante*)
18
19        PROCEDURE Vi; (*Enciende modo video inverso*)
20
21        PROCEDURE Neg; (*Enciende modo negrillas*)
22
23        PROCEDURE Tilt; (*Enciende modo titilar*)
24
25        PROCEDURE Vv; (*Cancela  modos*)
26
27  END Util.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 1  IMPLEMENTATION MODULE Util;
 2
 3          FROM InOut IMPORT Write,WriteCard;
 4
 5                    (*Borrado de la pantalla*) (*Cls*)
 6          PROCEDURE Cls;
 7          BEGIN
 8          Write(14C); (* ESC [ 2 J *)
 9          END Cls;
10
11                    (*Borrado de linea*) (*Cll*)
12          PROCEDURE Cll;
13          BEGIN
14          Write(33C);Write(133C);Write(113C); (* ESC [ K *)
15          END Cll;
16
17   (*Posicionamiento del cursor respecto al origen*) (*PxPy(Px,Py:CARDINAL)*)
18          PROCEDURE PxPy (Px, Py : CARDINAL);
19            BEGIN
20            IF Px > 80 THEN Px:=80 END;
21            IF Py > 25 THEN Py:=25 END;
22                 (* ESC [ Py ; Px H *)
23  Write(33C);Write(133C);WriteCard(Py,0);Write(73C);WriteCard(Px,0);Write(110C);
24          END PxPy;
25
26                    (*Cursor arriba*) (*Cau(u:CARDINAL)*)
27          PROCEDURE Cau(u:CARDINAL);
28          VAR
29          x,y:CARDINAL;
30          BEGIN
31          IF (u<=25) AND (u>0) THEN x:=u ELSE
32          IF u=0 THEN x:=1 ELSE x:=25 END;
33          END;
34          FOR y:=1 TO x DO
35          Write(33C);Write(133C);Write(60C);Write(101C) (* ESC [ 0 A *)
36          END;
37          END Cau;
38
39                    (*Cursor abajo*) (*Cad(d:CARDINAL)*)
40          PROCEDURE Cad(d:CARDINAL);
41          VAR
42          x,y:CARDINAL;
43          BEGIN
44          IF (d<=25) AND (d>0) THEN x:=d ELSE
45          IF d=0 THEN x:=1 ELSE x:=25 END;
46          END;
47          FOR y:=1 TO x DO
48          Write(33C);Write(133C);Write(60C);Write(102C) (* ESC [ 0 B *)
49          END;
50          END Cad;
51
52                    (*Cursor atrás*) (*Ba(b:CARDINAL)*)
53          PROCEDURE Ba(b:CARDINAL);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
54          VAR
55          x,y:CARDINAL;
56          BEGIN
57          IF (b<=80) AND (b#0) THEN x:=b ELSE
58          IF b=0 THEN x:=1 ELSE x:=80 END;
59          END;
60          FOR y:=1 TO x DO
61          Write(33C);Write(133C);Write(60C);Write(104C) (* ESC [ 0 D *)
62          END;
63          END Be;
64
65                  (*Cursor adelante*) (*Sp(s:CARDINAL)*)
66          PROCEDURE Sp(s:CARDINAL);
67          VAR
68          x,y:CARDINAL;
69          BEGIN
70          IF (s<=80) AND (s#0) THEN x:=s ELSE
71          IF s=0 THEN x:=1 ELSE x:=80 END;
72          END;
73          FOR y:=1 TO x DO
74          Write(33C);Write(133C);Write(60C);Write(103C) (* ESC [ 0 C *)
75          END;
76          END Sp;
77
78                  (*Encender video inverso*) (*Vi*)
79          PROCEDURE Vi;
80          BEGIN
81          Write(33C);Write(133C);WriteCard(7,0);Write(155C);(*ESC [ 7 m*)
82          END Vi;
83
84                  (*Encender negrillas*) (*Neg*)
85          PROCEDURE Neg;
86          BEGIN
87          Write(33C);Write(133C);WriteCard(1,0);Write(155C);(*ESC [ 1 m*)
88          END Neg;
89
90                  (*Encender Titilar*) (*Tilt*)
91          PROCEDURE Tilt;
92          BEGIN
93          Write(33C);Write(133C);WriteCard(5,0);Write(155C);(*ESC [ 5 m*)
94          END Tilt;
95
96                (*Cancelar video inverso y otros modos*) (*Vv*)
97          PROCEDURE Vv;
98          BEGIN
99          Write(33C);Write(133C);WriteCard(0,0);Write(155C);(*ESC [ 0 m*)
100         END Vv;
101
102 END Util.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Este otro módulo de librería, nos permite dibujar lineas de longitud n en diferentes direcciones.

Modula-2/86 lineas.def

```
1  DEFINITION MODULE Lineas;
2    EXPORT QUALIFIED Line, Line0;
3    PROCEDURE Line(d,n:CARDINAL);(*Dibuja una linea de longitud n, en la  dirección d (ángulo=45 X d grados *)
4    PROCEDURE Line0(d,n:CARDINAL);
5  END Lineas.
```

Modula-2/86 Compiler Version V 1.10

=> 0 Error(s) found

Modula-2/86 lineas.MOD

```
1  IMPLEMENTATION MODULE Lineas;
2    FROM Util IMPORT Cuu,Cud,Bs,Sp;
3    FROM InOut IMPORT Write;
4
5    PROCEDURE Line(d,n:CARDINAL);
6    BEGIN
7      CASE d OF
8        0:Cero(n,'_')|
9        1:Uno(n)|
10       2:Dos(n,'|')|
11       3:Tres(n)|
12       4:Cuatro(n,'_')|
13       5:Cinco(n)|
14       6:Seis(n,'|')|
15       7:Siete(n)
16     ELSE
17       Cero(n,'-')
18     END
19   END Line;
20
21   PROCEDURE Line0(d,n:CARDINAL);
22   BEGIN
23     CASE d OF
24       0:Cero(n,'_')|
25       2:Dos(n,'|')|
26       4:Cuatro(n,'_')|
27       6:Seis(n,'|')
28     ELSE
29       Cero(n,'_')
30     END
31   END Line0;
32
33   PROCEDURE Cero(x:CARDINAL;ch:CHAR);
34   VAR
35     y:CARDINAL;
36   BEGIN
37     FOR y:=1 TO x DO
38       Write(ch)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
39      END;
40    END Cero;
41
42    PROCEDURE Uno(x:CARDINAL);
43    VAR
44      y:CARDINAL;
45    BEGIN
46      FOR y:=1 TO x DO
47        Write(57C);Cuu(1)
48      END;
49    END Uno;
50
51    PROCEDURE Dos(x:CARDINAL;ch:CHAR);
52    VAR
53      y:CARDINAL;
54    BEGIN
55      FOR y:=1 TO x DO
56        Write(ch);Cuu(1);Bs(1)
57      END;
58    END Dos;
59
60    PROCEDURE Tres(x:CARDINAL);
61    VAR
62      y:CARDINAL;
63    BEGIN
64      FOR y:=1 TO x DO
65        Write(134C);Cuu(1);Bs(2)
66      END;
67    END Tres;
68
69    PROCEDURE Cuatro(x:CARDINAL;ch:CHAR);
70    VAR
71      y:CARDINAL;
72    BEGIN
73      FOR y:=1 TO x DO
74        Write(ch);Bs(2)
75      END;
76    END Cuatro;
77
78    PROCEDURE Cinco(x:CARDINAL);
79    VAR
80      y:CARDINAL;
81    BEGIN
82      FOR y:=1 TO x DO
83        Write(57C);Cud(1);Bs(2)
84      END;
85    END Cinco;
86
87    PROCEDURE Seis(x:CARDINAL;ch:CHAR);
88    VAR
89      y:CARDINAL;
90    BEGIN
91      FOR y:=1 TO x DO
92        Write(ch);Cud(1);Bs(1)
93      END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

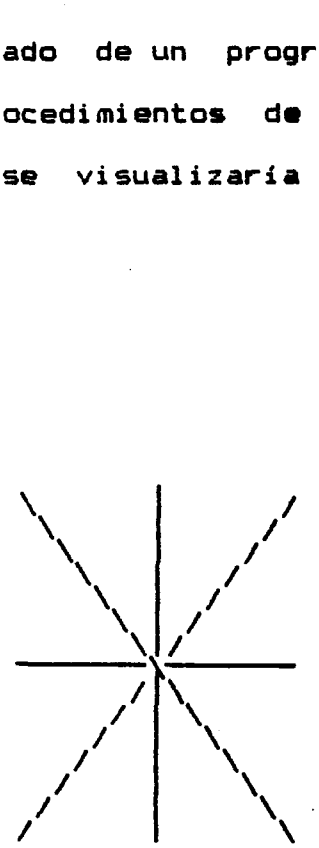
```
 94   END Seis;
 95
 96   PROCEDURE Siete(x:CARDINAL);
 97   VAR
 98     y:CARDINAL;
 99   BEGIN
100     FOR y:=1 TO x DO
101       Write(134C);Cud(1)
102     END;
103   END Siete;
104
105 END Lineas.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

A continuación se detalla el listado de un programa ejemplo, que utiliza algunos de los procedimientos de los módulos anteriores, y el resultado que se visualizaría en pantalla.

Modula-2/86 utiline.MOD

```
 1  MODULE PruebaUtiLineas;
 2    FROM Lineas IMPORT Line;
 3    FROM Util IMPORT Cls,PxPy,Neg,Vv;
 4  BEGIN
 5    Cls;
 6    Neg;
 7
 8    PxPy(40,20);Line(0,8);
 9    PxPy(40,20);Line(1,8);
10    PxPy(40,20);Line(2,8);
11    PxPy(40,20);Line(3,8);
12    PxPy(40,20);Line(4,8);
13    PxPy(40,20);Line(5,8);
14    PxPy(40,20);Line(6,8);
15    PxPy(40,20);Line(7,8);
16
17    Vv
18
19  END PruebaUtiLineas.
```



```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 9
LOS FICHEROS DE DISCO

9.1 INTRODUCCION

Para la creación y manejo de ficheros de disco, M2 proporciona dos formas de acceso a éstos; secuencial y aleatoria, cada una sustentada por su propio conjunto de rutinas.

A las rutinas de acceso aleatorio, se les suele llamar también de bajo nivel, pues manejan tipos de datos cercanos al hardware, como son bytes, words y caracteres.

A las de acceso secuencial, se les suele llamar de alto nivel. Para acceder a un fichero secuencial, se pueden utilizar las mismas rutinas de E/S (del módulo InOut y RealInOut) que hemos estado utilizando hasta ahora, tales como, ReadString, Read, Write, etc., además de otras que veremos posteriormente.

9.2 ACCESO A LOS FICHEROS DE FORMA SECUENCIAL

Se dice que se accede a un fichero de forma secuencial, cuando se lee o escribe (no simultaneamente) en un fichero, de forma que la posición actual indica el siguiente elemento a leer o escribir.

En un fichero secuencial, todos los elementos son del mismo tipo y generalmente, el número de ellos no es conocido.

Para el manejo de ficheros secuenciales se utilizan dos

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

tipos de rutinas, las rutinas de alto nivel de E/S, que se encuentran en los módulos de librería InOut y RealInOut:

Read: Lee un carácter.
ReadString: Lee una cadena.
ReadInt: Lee un entero
ReadCard: Lee un cardinal.
ReadReal: Lee un número en coma flotante.
Write: Escribe un carácter.
WriteLn: Escribe una linea en blanco.
WriteString: Escribe una cadena.
WriteInt: Escribe un entero.
WriteCard: Escribe un cardinal.
WriteOct: Escribe un cardinal en formato octal.
WriteHex: Escribe un cardinal en formato hexadecimal.
WriteReal: Escribe un número en coma flotante.

Y las rutinas de redireccionamiento que se encuentran en InOut:

OpenInput: Lee en el terminal el nombre de un fichero y abre éste a la lectura.
OpenOutput: Lee en el terminal el nombre de un fichero y abre éste a la escritura.
CloseInput: Cierra el fichero de entrada.
CloseOutput: Cierra el fichero de salida.

Hasta ahora las rutinas de E/S de alto nivel las hemos usado en la consola, es decir, toma de datos por teclado y salida por pantalla, sin embargo, con las rutinas de redireccionamiento, podemos seleccionar desde donde tomar datos y a donde mandarlos (consola, impresora, unidades de disco).

Cuando se llama a OpenInput, se visualiza por pantalla el mensaje 'in >', solicitando el nombre de un fichero para abrir a la lectura, en el caso de OpenOutput sería, 'out >', esperando el nombre del fichero a abrir para la escritura.

Cuando se llama a CloseInput o a CloseOutput, se cierran los ficheros de entrada y salida respectivamente.

Aunque ya se vio en el capítulo 3, vamos a reiterar el

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

significado del 'extdef' de la forma general de OpenInput (<extdef>), y de OpenOutput (<extdef>).

Si el nombre del fichero leído, no termina en '.', y no tiene extensión, entonces extdef se le añade como extensión.

Extdef, es un string que contiene la extensión deseada del fichero, si el string, es de tamaño cero, no se añade ninguna extensión.

Otra particularidad del OpenInput y OpenOutput, es que asignan a la variable booleana Done (que debe ser importada de InOut) el valor, TRUE o FALSE, en el caso de que la llamada a estas rutinas se halla realizado con éxito o no.

Un aspecto también importante de la variable Done, es la detección del final de un fichero, cuando se alcanza el final de éste, la variable Done toma el valor FALSE.

Un ejemplo clarificador de lo visto hasta ahora, podría ser el siguiente programa ejemplo, que nos hace una copia de un fichero, a la vez que nos visualiza la copia por pantalla.

Conviene detenerse en el papel que juega Done, tanto para la detección del final del fichero, como para la verificación de que la apertura de los ficheros, se ha realizado con éxito.

Module-2/3S copiar.MOD

```
1  MODULE Copiador;
2     FROM InOut IMPORT WriteLn, Read, Write, WriteString,Done,
3          OpenOutput, CloseOutput, OpenInput, CloseInput;
4
5     IMPORT Terminal;
6
7     VAR
8        ch:CHAR;
9
10 BEGIN
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
11
12    WriteString ('Introduzca el nombre del fichero de entrada : ');
13    REPEAT
14      OpenInput ('xxx');
15    UNTIL Done; (* Solicita el nombre de un fichero, hasta que éste sea válido *)
16
17    WriteLn;
18
19    WriteString ('Introduzca el nombre del fichero de salida : ');
20    REPEAT
21      OpenOutput ('xxx');
22    UNTIL Done; (* Solicita el nombre de un fichero, hasta que éste sea válido *)
23
24    REPEAT
25      Read(ch);           (* Lee del fichero de entrada*)
26      Terminal.Write(ch); (* Escribe en pantalla *)
27      Write(ch);          (* Escribe en el fichero de salida *)
28    UNTIL NOT Done; (* Done = FALSE cuando se detecta el final del fichero *)
29
30    CloseInput;
31    CloseOutput;
32
33  END Copiador.
```

```
Modula-2/96 Compiler Version V 1.10

==\     0 Error(s) found
```

La presencia de los procedimientos Write de la librería Terminal y Write de InOut, nos sirve para aclarar las diferentes funciones que pueden realizar procedimientos, que hasta ahora nos podían parecer que desempeñaban una misma tarea. El Terminal.Write escribe los caracteres en pantalla, aunque el fichero de salida esté abierto.

Supongamos que a la hora de ejecutar el programa, respondemos a la petición de in > con copiar.mod y a la de out > con copiar.bis, una vez finalizada la ejecución, tendremos un fichero idéntico a copiar.mod, pero con el nombre copiar.bis.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Otras respuestas al out >, podrían ser:

out > b:copiar.bis --> En este caso, se crea un fichero idéntico a copiar.mod , con el nombre copiar.bis en la unidad de disco b:.

out > copiar --> Nos crea un fichero idéntico a copiar.mod con el nombre copiar.xxx.

out > prn -->Nos lista por impresora el fichero copiar.mod.

Otra respuesta al in >, podría ser:

in > b:copiar.mod --> En este caso, se buscaría en la unidad de disco b: el fichero copiar.mod, si no lo encuentra, nos vuelve a solicitar el nombre de un fichero de entrada, presentándonos de nuevo in > .

Un nuevo programa ejemplo, con mayor utilidad, es el enfatiza.mod, con el cual, se han realizado los listados de los programas ejemplo vistos hasta ahora.

Este programa, imprime los programas Modula-2 de forma condensada y enfatizando las palabras reservadas del lenguaje con una doble impresión.

Cabe destacar la utilización del módulo de librería ASCII, éste exporta las constantes simbólicas de los caracteres ASCII no imprimibles, es decir:

```
nul = 00C;   soh = 01C;   stx = 02C;   etx = 03C;
eot = 04C;   enq = 05C;   ack = 06C;   bel = 07C;
bs  = 10C;   ht  = 11C;   lf  = 12C;   vt  = 13C;
ff  = 14C;   cr  = 15C;   so  = 16C;   si  = 17C;
dle = 20C;   dc1 = 21C;   dc2 = 22C;   dc3 = 23C;
dc4 = 24C;   nak = 25C;   syn = 26C;   etb = 27C;
can = 30C;   em  = 31C;   sub = 32C;   esc = 33C;
fs  = 34C;   gs  = 35C;   rs  = 36C;   us  = 37C;
del = 177C;
```

Además de las vistas, también puede exportar EOL, que es una constante no ASCII, que tiene la ventaja de usar sólo un carácter para especificar el final de linea, en vez de cr y lf, (En nuestro caso EOL lo hemos importado de InOut) EOL =36C.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 1
 2 MODULE Enfatizar;
 3 (* Lista los programas Modula-2, de forma condensada y
 4    enfatizando las palabras reservadas con una doble impresión*)
 5 FROM InOut IMPORT Done, EOL, OpenInput, OpenOutput,Read,Write,
 6                   CloseInput, CloseOutput;
 7 FROM ASCII IMPORT esc,lf,si,dc2;
 8 CONST
 9     n = 40;       (* numero de palabras reservadas *)
10     lineaspag = 55; (* lineas por página *)
11 TYPE
12     letras = ARRAY [0 .. 15] OF CHAR;
13 VAR
14     c : CHAR;
15     x,i, k, l, m, r : CARDINAL;
16     contalin : CARDINAL; (* Contador de lineas *)
17     iden : ARRAY [0 .. 16] OF CHAR;
18     clave : ARRAY [1 .. n] OF letras;
19
20
21 PROCEDURE Copiar;
22 BEGIN
23   Write (c);
24   IF c=EOL THEN
25     contalin:=(contalin MOD lineaspag) +1;
26     IF contalin=1 THEN
27       FOR x:=1 TO 19  DO
28         Write(lf)
29       END
30     END
31   END;
32   Read (c)
33 END Copiar;
34
35
36 PROCEDURE Tabla;
37 BEGIN
38   clave [ 1] := 'AND          ';
39   clave [ 2] := 'ARRAY        ';
40   clave [ 3] := 'BEGIN        ';
41   clave [ 4] := 'BY           ';
42   clave [ 5] := 'CASE         ';
43   clave [ 6] := 'CONST        ';
44   clave [ 7] := 'DEFINITION   ';
45   clave [ 8] := 'DIV          ';
46   clave [ 9] := 'DO           ';
47   clave [10] := 'ELSE         ';
48   clave [11] := 'ELSIF        ';
49   clave [12] := 'END          ';
50   clave [13] := 'EXIT         ';
51   clave [14] := 'EXPORT       ';
52   clave [15] := 'FOR          ';
53   clave [16] := 'FROM         ';
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
54    clave [17] := 'IF            ';
55    clave [18] := 'IMPLEMENTATION ';
56    clave [19] := 'IMPORT        ';
57    clave [20] := 'IN            ';
58    clave [21] := 'LOOP          ';
59    clave [22] := 'MOD           ';
60    clave [23] := 'MODULE        ';
61    clave [24] := 'NOT           ';
62    clave [25] := 'OF            ';
63    clave [26] := 'OR            ';
64    clave [27] := 'POINTER       ';
65    clave [28] := 'PROCEDURE     ';
66    clave [29] := 'QUALIFIED     ';
67    clave [30] := 'RECORD        ';
68    clave [31] := 'REPEAT        ';
69    clave [32] := 'RETURN        ';
70    clave [33] := 'SET           ';
71    clave [34] := 'THEN          ';
72    clave [35] := 'TO            ';
73    clave [36] := 'TYPE          ';
74    clave [37] := 'UNTIL         ';
75    clave [38] := 'VAR           ';
76    clave [39] := 'WHILE         ';
77    clave [40] := 'WITH          ';
78  END Tabla;
79
80
81  PROCEDURE Identificador () : BOOLEAN;
82  BEGIN
83    l := 1; r := n; iden [k] := ' ';
84    REPEAT (* busqueda binaria *)
85      m := (l+r) DIV 2;
86      i := 0;
87      WHILE (iden [i] = clave [m,i]) AND
88            (iden [i] # ' ') DO i := i+1; END;
89      IF iden [i] <= clave [m,i] THEN r := m-1; END;
90      IF iden [i] >= clave [m,i] THEN l := m+1; END;
91    UNTIL l > r;
92    RETURN l=r+1; (* TRUE=identificador, FALSE=palabra reservada *)
93  END Identificador;
94
95
96  BEGIN
97    Tabla; contalin := 1;
98
99    REPEAT
100    OpenInput ('')
101   UNTIL Done;
102
103   OpenOutput ('');
104   IF NOT Done THEN (* Se para la ejecución *)
105     HALT
106   END;
107
108   Write(esc); Write('M'); (* Modo de impresión Elite*)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
109    Write(si); (*Modo comprimido*)
110
111   Read (c);
112    REPEAT
113      IF (c >= 'A') AND (c <= 'Z') THEN
114        k := 0;
115        REPEAT
116          iden [k] := c;
117          k := k+1;
118          Read (c)
119        UNTIL (c < 'A') OR (c > 'Z');
120
121        IF Identificador () THEN (* no se enfatiza *)
122          FOR i := 0 TO k-1 DO
123             Write (iden [i])
124          END;
125        ELSE (* es palabra reservada *)
126          Write (esc); Write ('G'); (* para enfatizar con doble impresión *)
127          FOR i := 0 TO k-1 DO
128             Write (iden [i])
129          END;
130          Write (esc); Write ('H'); (* Se cancela la doble impresion  *)
131        END
132      END;
133      Copiar
134    UNTIL NOT Done;
135
136    Write(esc); Write('P'); (* Cancela el modo elite *)
137    Write(dc2); (* Cancela el modo comprimido *)
138
139    CloseInput;(* Se cierran los ficheros *)
140    CloseOutput
141
142  END Enfatizar.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

9.2.1 LIMITACIONES DE LOS FICHEROS SECUENCIALES

Hay dos razones fundamentales que limitan el uso de los ficheros secuenciales, en programas comerciales que utilicen M2.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La primera, es que la visualización de los mensajes in > y out > no son controlables y aunque adecuados para utilidades del programador, no lo son para programas comerciales, donde se está controlando continuamente la pantalla.

La segunda desventaja, es que el acceso a un fichero secuencial de un gran programa, que trabaja con miles de bytes de información,toma bastante tiempo, pues, para obtener información de la mitad del fichero, toda la información anterior debe ser leída previamente.

9.3 ACCESO A FICHEROS DE FORMA ALEATORIA

Las rutinas para acceder a ficheros de forma aleatoria, se encuentran en el módulo de librería FileSystem y se muestran a continuación.

FICHEROS DE TEXTO Y BINARIOS
Create: Crea un fichero temporal. **Close:** Cierra un fichero. **Lookup:** Crea un fichero permanente. **Rename:** Renombra un fichero. **Delete:** Borra un fichero. **SetRead:** Abre un fichero a la lectura. **SetWrite:** Abre un fichero a la escritura. **SetModify:** Abre un fichero a la lectura o escritura. **SetOpen:** Abre un fichero inactivo. **Reset:** Posiciona al principio el puntero de un fichero. **SetPos:** Posiciona el puntero en una posición especifica. **GetPos:** Obtiene la posición actual del puntero. **Length:** Obtiene la longitud en bytes de un fichero.
FICHEROS DE TEXTO
ReadChar: Lee un carácter de un fichero. **WriteChar:** Escribe un carácter en un fichero.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

FICHEROS BINARIOS
ReadWord: Lee una palabra de un fichero. **WriteWord:** Escribe un palabra en un fichero. **ReadByte:** Lee un byte de un fichero. **WriteByte:** Escribe un byte en un fichero. **ReadNBytes:** Lee el número especificado de bytes. **WriteNBytes:** Escribe el número especificado de bytes.

Como se puede observar, hay tres grupos de rutinas, uno para los fichero de texto, es decir, aquellos que sólo contienen caracteres ASCII imprimibles, otro para ficheros binarios, que contienen información no imprimible y un tercer grupo que se utiliza para manejar los dos anteriores.

El estudio de los ficheros binarios, se realizará en el próximo capitulo.

También en esta misma librería, se encuentra definido File, que es un tipo registro exportable. File, se usa para crear descriptores de ficheros, es decir, variables del tipo File, cuya función, es contener información a cerca de los ficheros con que estemos trabajando.

La mayoría de los campos del tipo File, son irrelevantes para el programador, pues los utilizan los procedimientos internos de la librería FileSystem, sin embargo, si es importante el manejo de eof y res.

```
TYPE
  File=RECORD
          eof:BOOLAN;
          res:(done,notdone,notsupported,callerror,
               unknownmedium,unknownfile,paramerror,
               toomanyfiles,eom);
          . . . . . .
          . . . . . .
          campos irrelevantes
          . . . . . .
          . . . . . .
       END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Como se observa, el campo eof, es del tipo booleano y nos indica si el final del fichero se ha alcanzado o no. El campo res, es del tipo enumerado y cada uno de sus posibles valores, nos determina el resultado de alguna operación en concreto. A continuación, se muestran los posibles valores de res y su significado.

done: Se ha realizado con éxito una rutina.
notdone: No se ha terminado con éxito una rutina.
notsupported: De uso interno de FileSystem.
callerror: Acceso impropio a un fichero (ejem: intentar escribir en un fichero abierto para la lectura).
unknownmedium: La unidad de disco que se ha especificado no existe.
unknownfile: El fichero que se especificó para borrar no existe.
paramerror: Error en los parámetros (ejem: un nombre de fichero no válido).
toomanyfiles: Se ha excedido el número de ficheros abiertos (sólo doce ficheros pueden estár abiertos al mismo tiempo).
eom: No queda espacio en el disco.

Hay que tener en cuenta, que todas las respuestas al campo res, se encuentran definidas dentro del módulo FileSystem, en el tipo enumerado Response, por lo tanto, siempre que se utilicen rutinas de bajo nivel, habrá que incluir junto con File, en la lista de importaciones, Response.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

9.3.1 LECTURA Y ESCRITURA EN FICHEROS ALEATORIOS

Una vez vistas las herramientas necesarias para el manejo de ficheros aleatorios, estamos en disposición de leer y escribir en ellos, sin embargo, dos pasos previos son necesarios; localizar el fichero y abrirlo a la lectura o escritura.

Para localizar un fichero, se requiere el uso de Lookup, cuya forma general es:

Lookup (<VarFile>,<NombreFichero>,<NuevoFichero>);

VarFile, es una variable del tipo File, que debe ser definida previamente y que utilizaremos como descriptor de fichero, NombreFichero, es la cadena que debe contener un nombre válido de fichero y NuevoFichero es una opción booleana para determinar si un fichero no existente debe se creado (NuevoFichero=TRUE), o no (NuevoFichero=FALSE).

En el caso de que la localización de nombre fichero, se haya realizado con éxito, el campo res del descriptor de fichero VarFile, toma el valor done.

El siguiente fragmento nos ilustra la forma de localizar un fichero de entrada:

```
REPEAT
  WriteString('Introduzca el nombre de un fichero de entrada')
  ReadString(nombrefichero);WriteLn;
  Lookup(F1,nombrefichero,FALSE);(*Si no existe nombrefichero,
                                   repite el lazo solicitando
UNTIL F1.res=done;                 un nuevo nombrefichero*)
```

Generalmente, cuando NuevoFichero es FALSE, estamos localizando un fichero de entrada y si es TRUE, de salida.

Una vez encontrado el fichero, el descriptor de fichero

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

le es asignado (F1 en nuestro caso).

El segundo paso previo a la lectura o escritura, es abrir el fichero, usando SetRead, SetWrite o SetModify, cuyas formas generales son:

```
SetRead(<VarFile>);
SetWrite(<VarFile>);
SetModify(<VarFile>);
```

Por ejemplo, con la sentencia SetRead(F1), estamos abriendo a la lectura, sólamente, el fichero que localizamos anteriormente.

SetWrite abre un fichero como salida, sólo a la escritura, SetModify, abre un fichero tanto para entrada como para salida, aceptando operaciones de lectura o escritura.

También es posible utilizar un fichero inactivo, es decir, abierto pero no activo, a través de SetOpen(<VarFile>). Después de una llamada a SetOpen, sólo se podrá acceder a ese fichero con una de las rutinas anteriores.

Siempre que se realice con éxito cualquiera de estos procedimientos, el campo res del registro descriptor de fichero, toma el valor done.

Para leer caracteres de un fichero, se utiliza ReadChar(<VarFile>,<ch>), cuando la lectura se realiza con éxito, res toma el valor done, pero cuando se alcanza el final del fichero, un carácter null (ASCII 0) se deposita en ch y eof toma el valor TRUE, significando que el final del fichero, se ha alcanzado.

En el siguiente fragmento, se observa como leer caracteres de un fichero y visualizarlos en pantalla, con

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

excepción del carácter que indica el final del fichero.

```
REPEAT
  ReadChar(F1,ch);
  IF ch # CHR(0) THEN Write(ch) END;
UNTIL F1.eof;
```

El procedimiento WriteChar(<VarFile>,<ch>), escribe caracteres en un fichero, si se realiza correctamente, res toma el valor done, en caso de no quedar espacio en el fichero de disco de salida, res toma el valor eom.

Una vez finalizadas las operaciones de lectura escritura con los ficheros, estos deben ser cerrados con el procedimiento Close(<VarFile>).

A continuación se muestra un programa ejemplo, que nos lista un archivo por impresora en letra cursiva, y paralelamente nos lo visualiza por pantalla.

```
Modula-2/86                    italica.MOD

 1  MODULE FichSecuen;
 2    FROM InOut IMPORT ReadString, WriteString, WriteLn,Write;
 3    FROM FileSystem IMPORT File, Response, Lookup, SetWrite, SetRead,
 4                           Close, ReadChar, WriteChar;
 5
 6    VAR
 7      ch: CHAR;
 8      F1,F2: File;
 9      fnombre: ARRAY [0..30] OF CHAR;
10
11  BEGIN
12    Write(14C);(*Limpia la pantalla*)
13    REPEAT
14      WriteString ('Introduzca el nombre del fichero de entrada: ');
15      ReadString(fnombre);WriteLn;
16      Lookup(F1,fnombre,FALSE);
17    UNTIL F1.res = done;
18
19    Lookup(F2,'prn',TRUE);
20
21    SetRead(F1);
22    SetWrite(F2);
23
24    WriteChar(F2,33C); WriteChar(F2,64C);(*Activa el modo cursiva*)
25
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
26    REPEAT
27      ReadChar(F1,ch);
28      IF ch<>CHR(0) THEN Write(ch); WriteChar(F2,ch) END;
29    UNTIL F1.eof OR (F2.res=eos);
30
31    WriteChar(F2,33C); WriteChar(F2,65C);(*Cancela el modo cursiva*)
32
33    Close(F1);Close(F2)
34
35  END FichSecuen.
```

```
Modula-2/86 Compiler Version V 1.10

=>      0 Error(s) found
```

De lo visto hasta ahora, se ha podido observar, las ventajas de las rutinas de bajo nivel, frente a las de alto nivel, en cuanto al control de los mensajes por pantalla y a la posibilidad de tener, hasta doce ficheros abiertos a la vez.

Sin embargo, la gran ventaja de las rutinas de bajo nivel, está en el acceso aleatorio.

La librería FileSystem, nos proporciona rutinas como SetPos,GetPos o Length, para posicionar el puntero de fichero en cualquier lugar de éste y hacer un acceso aleatorio.

Un puntero de fichero, es una variable interna, que mantiene la pista, de la posición actual de las rutinas en un fichero. Se puede pensar en un puntero de fichero, como, en una punta de flecha que apunta al carácter actual, avanzando una posición, cada vez que se realice una operación de lectura o escritura.

SetPos se utiliza para posicionar el puntero, en cualquier lugar del fichero, su forma general es:

SetPos (<VarFile>,<alto>,<bajo>)

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Donde alto y bajo, son valores de tipo Cardinal y la posición del puntero que resulta de la llamada a este procedimiento, sería; posición = (alto * (2^{16})) + bajo.

Hay que tener en cuenta que la posición, vendrá expresada con los valores alto y bajo, esto es debido a que a menudo, la dimensión de un fichero, sobrepasa los 65535 bytes y M2 no soporta valores cardinales mayores, por lo tanto, son dos números, los que se utilizan para representar la posición del fichero, de esta forma, la posición máxima donde se podría ubicar el puntero, sería:

$$posición = (65535 * (2^{16})) + 65535 = 2^{32}$$

Para posicionar el puntero en ficheros menores de 65535 bytes, hacemos alto = 0, por ejemplo; SetPos(F1,0,10134).

También se puede situar el puntero al principio del fichero. Teniendo en cuenta que el primer byte de un fichero es el cero, resultaría; SetPos(F1,0,0).

Por el contrario GetPos, obtiene la posición del puntero dentro de un fichero.

GetPos(<VarFile>,<alto>,<bajo>).

Length devuelve la longitud en bytes de un fichero:

Length (<VarFile>,<alto>,<bajo>)

Las definiciones de estos procedimientos serían:

```
PROCEDURE SetPos (VAR f:File; alto,bajo:CARDINAL);
PROCEDURE GetPos (VAR f:File; VAR alto,bajo:CARDINAL);
PROCEDURE Length (VAR f:File; VAR alto,bajo:CARDINAL);
```

A continuación, se muestra un programa ejemplo, que utiliza SetPos y Length, permitiéndonos imprimir porciones de un fichero.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 1  MODULE Copia;
 2    FROM InOut IMPORT ReadString, WriteString, WriteLn,Read;
 3    FROM FileSystem IMPORT File, Response, Lookup, SetWrite,SetRead,
 4         Close,ReadChar,WriteChar,Length,SetPos;
 5    FROM Util IMPORT PxPy,Cls,Cll,Vi,Vv,Neg,Es;
 6    VAR
 7      h,l:CARDINAL;
 8      ch: CHAR;
 9      F1,F2: File;
10      fnombre: ARRAY [0..30] OF CHAR;
11  (* ------------------------------------------------------------ *)
12    PROCEDURE Pantalla;
13      BEGIN
14        Cls;
15        PxPy(18,5); Neg; WriteString('¿Qué porción de fichero quiere imprimir?: ');Vv;
16        PxPy(28,10); Vi; WriteString('0 : desde  0 al  25 % '); Vv;
17        PxPy(28,11); Vi; WriteString('1 : desde  0 al  50 % '); Vv;
18        PxPy(28,12); Vi; WriteString('2 : desde  0 al  75 % '); Vv;
19        PxPy(28,13); Vi; WriteString('3 : desde  0 al 100 % '); Vv;
20        PxPy(28,14); Vi; WriteString('4 : desde 25 al  50 % '); Vv;
21        PxPy(28,15); Vi; WriteString('5 : desde 25 al  75 % '); Vv;
22        PxPy(28,16); Vi; WriteString('6 : desde 25 al 100 % '); Vv;
23        PxPy(28,17); Vi; WriteString('7 : desde 50 al  75 % '); Vv;
24        PxPy(28,18); Vi; WriteString('8 : desde 50 al 100 % '); Vv;
25        PxPy(28,19); Vi; WriteString('9 : desde 75 al 100 % '); Vv;
26        PxPy(28,20); Vi; WriteString('Introduzca su elección '); Vv;Es(1);
27        Read(ch);
28    END Pantalla;
29  (* ------------------------------------------------------------ *)
30    PROCEDURE Abrir;
31      BEGIN
32        SetRead(F1);
33        SetWrite(F2);
34    END Abrir;
35  (* ------------------------------------------------------------ *)
36    PROCEDURE Cerrar;
37      BEGIN
38        Close(F1);
39        Close(F2);
40    END Cerrar;
41  (* ------------------------------------------------------------ *)
42    PROCEDURE Cero(x:REAL);
43      VAR
44        Puntero,Posicion:REAL;
45      BEGIN
46        Length(F1,h,l);
47        Posicion:=(FLOAT(h)*2.E16 + FLOAT(l)) * x;
48        Puntero:=0.;
49        REPEAT
50          ReadChar(F1,ch);
51          IF ch#CHR(0) THEN WriteChar(F2,ch) END;
52          Puntero:=Puntero + 1.;
53        UNTIL (F1.eof) OR (F2.res = eos) OR (Puntero >= Posicion);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
54    END Cero;
55  (* ------------------------------------------------------------ *)
56    PROCEDURE Uno(x,y:REAL);
57      VAR
58        Final,Posicion:REAL;
59      BEGIN
60        Length(F1,h,l);
61        Posicion:=(FLOAT(h)*2.E16 + FLOAT (l)) * x;
62        Final:=(FLOAT(h)*2.E16 + FLOAT (l)) * y;
63        IF Posicion >65535.
64           THEN
65             h:=TRUNC((Posicion-65535.)/2.E16);
66             l:=65535
67           ELSE
68             h:=0;
69             l:=TRUNC(Posicion);
70        END;
71        SetPos(F1,h,l);
72        REPEAT
73          ReadChar(F1,ch);
74          IF ch#CHR(0) THEN WriteChar(F2,ch) END;
75          Posicion:=Posicion + 1.;
76        UNTIL (F1.eof) OR (F2.res = eos) OR (Posicion >= Final);
77    END Uno;
78  (* ------------------------------------------------------------ *)
79    BEGIN
80      Cls;PxPy(10,5);Neg;
81      WriteString('Introduzca el nombre del fichero a imprimir: ');Y;
82      REPEAT
83        ReadString(fnombre);
84        Lookup(F1,fnombre,FALSE);
85        IF F1.res = notdone THEN
86           Bs(99);Cll;Neg;
87           WriteString('Ese fichero no se encuentra en el disco, Introduzca otro nombre: ');Y;
88        END;
89      UNTIL F1.res = done;
90      Lookup(F2,'prn',TRUE);
91      Pantalla;
92      CASE ch OF
93             '0':Abrir;Cero(0.25);Cerrar; (* 0 - 25% *)
94             '1':Abrir;Cero(0.50);Cerrar; (* 0 - 50% *)
95             '2':Abrir;Cero(0.75);Cerrar; (* 0 - 75% *)
96             '3':Abrir;Cero(1.);Cerrar ;  (* 0 - 100% *)
97             '4':Abrir;Uno(0.25,0.50);Cerrar; (* 25 - 50% *)
98             '5':Abrir;Uno(0.25,0.75);Cerrar; (* 25 - 75% *)
99             '6':Abrir;Uno(0.25,1.);Cerrar;   (* 25 - 100% *)
100            '7':Abrir;Uno(0.50,0.75);Cerrar; (* 50 - 75% *)
101            '8':Abrir;Uno(0.50,1.);Cerrar;   (* 50 - 100% *)
102            '9':Abrir;Uno(0.75,1.);Cerrar    (* 75 - 100% *)
103      ELSE
104      (* con cualquier otro caracter que no sea [0..9], se sale del programa *)
105      END
106   END Copia.

-------------------------------

   Modula-2/86 Compiler Version V 1.10

   =>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

9.3.2 RENAME, CREATE Y DELETE

Estos tres procedimientos, complementan el juego de rutinas para el manejo de ficheros de la libreria FileSystem.

PROCEDURE Rename(VAR f:File; nuevonombre:ARRAY OF CHAR);

Renombra el fichero f con 'nuevonombre', si este campo está vacío, el archivo pasa a ser temporal.

'nuevonombre' puede incluir el dispositivo, no obstante, el soporte en el que reside el fichero, no puede cambiarse.

PROCEDURE Create (VAR f:File; soporte:ARRAY OF CHAR);

Crea un fichero temporal f (sin nombre) en el dispositivo que se menciona.

Un fichero temporal no tiene nombre pero, se puede convertir en permanente (o definitivo) antes de cerrarlo renombrándolo mediante el procedimiento Rename.

A raiz de esto puede referenciarse al fichero, con el nombre f.

PROCEDURE Delete(nombre:ARRAY OF CHAR; VAR f:File);

Borra el fichero cuyo nombre se menciona, f es una variable del tipo File que usa el campo res para indicar el resultado de la operación. Si el fichero no se encuentra, o el disco está protegido contra escritura, el campo res toma el valor notdone.

A diferencia de otras rutinas de ficheros de disco, con ésta no es necesario ejecutar un LooKup previo.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

9.4 PROGRAMAS EJEMPLO

El módulo sustitu1.mod siguiente, muestra, como buscar una determinada palabra dentro de un fichero y reemplazarla por otra, utilizando las rutinas de alto nivel.

Modula-2/86 sustitu1.MOD

```
 1 MODULE Intercambio;
 2   FROM InOut IMPORT OpenInput,OpenOutput,CloseInput,CloseOutput,
 3   ReadString,WriteString,Done,Read,Write,WriteLn;
 4   FROM Strings IMPORT CompareStr;
 5   FROM Util IMPORT Cls;
 6   VAR
 7     ch,w:CHAR;
 8     x,y,i,z,c:CARDINAL;
 9     ver,entrada,salida:ARRAY [0..30] OF CHAR;
10 BEGIN
11   FOR i:=0 TO 30 DO ver[i]:=CHR(0);entrada[i]:=CHR(0);salida[i]:=CHR(0);END;
12   WriteString ('Introduzca la palabra a sustituir: ');
13   ReadString(entrada); WriteLn;
14   WriteString ('Introduzca la palabra a reemplazar: ');
15   ReadString(salida);WriteLn;
16
17   WriteString ('Introduzca el nombre del fichero de entrada: ');
18   REPEAT
19     OpenInput('');
20   UNTIL Done;
21
22   WriteString ('Introduzca el nombre del fichero de salida: ');
23   REPEAT
24     OpenOutput('');
25   UNTIL Done;
26
27   c:=0;
28   REPEAT
29     Read(ch);
30     IF (CAP(ch)>='A')AND(CAP(ch)<='Z') THEN
31       x:=0;
32       REPEAT
33         ver[x]:=ch;
34         x:=x+1;
35         Read(ch);
36       UNTIL (CAP(ch)<'A')OR(CAP(ch)>'Z');
37       ver[x]:=CHR(0);
38
39       IF CompareStr(entrada,ver)=0 THEN
40         y:=0;
41         REPEAT
42           Write(salida[y]);
43           INC(c);IF c=79 THEN WriteLn;c:=0 END;
44           y:=y+1;
45         UNTIL (CAP(salida[y])<'A') OR (CAP(salida[y])>'Z');
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
46
47        ELSE
48
49          z:=0;
50          REPEAT
51            Write(ver[z]);
52            INC(c);IF c=79 THEN WriteLn;c:=0 END;
53            z:=z+1;
54          UNTIL ver[z]=CHR(0);
55        END;
56      END;
57      IF ch # CHR(30) THEN
58        Write(ch);
59        INC(c);IF c=79 THEN WriteLn;c:=0 END;
60      ELSE
61        IF c#0 THEN  Write(ch);c:=0;
62        ELSE
63        END;
64      END;
65    UNTIL NOT Done;
66    CloseInput;
67    CloseOutput;
68  END Intercambio.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

El módulo que se presenta seguidamente, llamado Registr3.mod, nos sirve para guardar y manejar datos personales, tales como, nombre, calle, ciudad, teléfono y código postal, como si de una agenda se tratase.

Conviene que se examine detenidamente, pues engloba la mayoría de los conceptos estudiados hasta ahora.

Modula-2/86 registr3.MOD

```
1  MODULE listaP;
2        FROM Util IMPORT Vi,Vv,Sp,Bs,Cuu,Neg,Cls,PxPy,Cud;
3        FROM FileSystem IMPORT File,Response,Lookup,SetWrite,SetRead,
4        Close,ReadChar,WriteChar,Reset;
5        FROM InOut IMPORT Read, Write, WriteString, WriteLn, WriteCard,
6        ReadCard, EOL;
7        FROM Strings IMPORT CompareStr;
8        CONST
9              Tamlist = 100;
10       TYPE
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
11              direcciones = RECORD
12                      nombre: ARRAY [0..40] OF CHAR;
13                      calle: ARRAY [0..40] OF CHAR;
14                      ciudad: ARRAY [0..40] OF CHAR;
15                      tlf: ARRAY [0..14] OF CHAR;
16                      copos: ARRAY [0..5] OF CHAR;
17              END;
18      VAR
19 alista: ARRAY [0..Tamlist] OF direcciones; (*Matriz con las direcciones*)
20    eleccion: CHAR;
21    F3:File;
22                              (* PROCEDIMIENTO Gets *)
23      PROCEDURE Gets (VAR a:ARRAY OF CHAR);
24        CONST
25        BS = 8; (* Back Space *)
26        VAR
27        ch : CHAR;
28        i: CARDINAL;
29        BEGIN
30        i:=0;
31        REPEAT
32        LOOP
33        Read (ch);
34      IF (ORD (ch))>31) AND (ORD (ch)<127) OR (ORD (ch)=8) AND (i#0)
35      OR (ORD (ch)=30)
36  OR (ORD (ch))>159) AND (ORD (ch)<169) OR (ORD (ch)=130) OR (ORD(ch)=129)
37      THEN EXIT END;
38      END;
39        Write (ch);
40      IF ORD(ch)=BS THEN i:=i-1; Write '4(C'; Write (177C)
41        ELSIF (ch#CHR(30)) AND (i<HIGH(a)) THEN
42        a[i]:=ch;
43        i:=i+1;
44        END;
45        UNTIL (ch=CHR(30)) OR (i=HIGH(a));
46        IF i=HIGH(a) THEN Write(36C) END;
47        a[i]:=CHR(0); (* todas las cadenas deben terminar en 0 *)
48      END Gets;
49 (* ########################################################################## *)
50
51                (* PROCEDIMIENTO Puts *)
52      PROCEDURE Puts (s: ARRAY OF CHAR);
53      BEGIN
54      WriteString (s);
55      WriteLn;
56      END Puts;
57 (* ########################################################################## *)
58
59                (* PROCEDIMIENTO Mar... *)
60      PROCEDURE M...(VAR
61            VAR
62                  ch:CHAR;
63                  i:CARDINAL;
64      BEGIN
65            Cls;  (*Borrar pantalla*)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
66              PxPy(0,9);
67  Sp(22);Vi; Puts('1. Introducir una dirección    ');Vv;
68  Sp(22);Vi; Puts('2. Borrar una dirección        ');Vv;
69  Sp(22);Vi; Puts('3. Encontrar una dirección     ');Vv;
70  Sp(22);Vi; Puts('4. Listar todas las direcciones');Vv;
71  Sp(22);Vi; Puts('5. Salvar                      ');Vv;
72  Sp(22);Vi; Puts('6. Cargar                      ');Vv;
73  Sp(22);Vi; Puts('7. Imprimir                    ');Vv;
74  Sp(22);Vi; Puts('8. Abandonar el programa       ');Vv;
75  Sp(22);Vi;
76  WriteString ('########################         ');Vv; WriteLn;
77  Sp(22);Vi;
78  WriteString ('Introduzca su elección:           ');Vv; WriteLn;
79  Sp(22);Vi;
80  WriteString ('########################         ');
81  Bs(8);
82  Cuu(1);   (*cursor arriba*)
83  Vv;(*cancelar modo inverso*)
84
85          REPEAT
86            Read (ch);
87          UNTIL (ORD(ch)>=49) AND (ORD(ch)<=56);
88            Write (ch);
89            WriteLn;WriteLn;
90          RETURN ch;
91          END Menu;
92  (* ########################################################################## *)
93  (* PROCEDIMIENTO Busqueda:Indica a otros procedimientos los registros que
94  están vacios, en caso de que todos estén llenos, lo indicará con un -1 *)
95          PROCEDURE Busqueda():INTEGER;
96                  VAR
97                          i:CARDINAL;
98          BEGIN
99                  FOR i:=0 TO Taalist DO
100                 IF CompareStr(alista[i].nombre,"")=0 THEN RETURN i; END;
101                 END;
102                 RETURN -1;
103         END Busqueda;
104 (* ########################################################################## *)
105
106 (* Procedimiento Introduce: Introduce todos los datos en un registro.*)
107         PROCEDURE Introduce;
108                 VAR
109                         i: INTEGER;
110                 BEGIN
111                         Cls;Sp(25);Cud(9);
112                         i:= Busqueda();
113                         IF i#(-1) THEN
114                  Vi;WriteString('Introduzca un nombre:');Vv;WriteLn;
115             Sp(25); Gets(alista[i].nombre);
116             Sp(25);Vi;WriteString('Introduzca la calle:');Vv;WriteLn;
117             Sp(25); Gets(alista[i].calle);
118             Sp(25);Vi;WriteString('Introduzca la ciudad:');Vv;WriteLn;
119             Sp(25); Gets(alista[i].ciudad);
120             Sp(25);Vi;WriteString('Introduzca el teléfono:');Vv;WriteLn;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
121             Sp(25); Gets(mlista[i].tlf);
122         Sp(25);Vi;WriteString('Introduzca el código postal:');Vv;WriteLn;
123             Sp(25); Gets(mlista[i].copos);
124
125                 END;
126         END Introduce;
127  (* ################################################################## *)
128
129 (*Procedimiento Visualiza:Hace que se visualice por la pantalla,todos los
130  registros que le indique el procedimiento lista.*)
131         PROCEDURE Visualiza(ml:direcciones);
132         BEGIN
133                 Puts(ml.nombre);
134          Sp(25);Puts(ml.calle);
135          Sp(25);Puts(ml.ciudad);
136          Sp(25);Puts(ml.tlf);
137          Sp(25);Puts(ml.copos);
138                 WriteLn;Sp(25);
139         END Visualiza;
140  (* ################################################################## *)
141 (*Procedimiento Listar:Visualiza la lista de registros con información, al
142  completo *)
143         PROCEDURE Listar;
144                 VAR
145                         n:CHAR;
146                         y:CARDINAL;
147                         i:CARDINAL;
148         BEGIN
149                     Cls;Cud(2);Sp(25);y:=0;
150                 FOR i:=0 TO Tamlist DO
151                     IF CompareStr(mlista[i].nombre,"")#0 THEN
152                     Visualiza(mlista[i]);
153                     y:=y+1;
154                     WriteLn;Sp(25);
155  IF y MOD 3 =0 THEN Vi;WriteString('Pulse cualquier tecla para continuar:');Vv;
156  Read(n);Cls;Sp(25);Cud(2);END;
157                     END;
158                 END;
159
160  IF y MOD 3 # 0 THEN Vi;WriteString('Pulse cualquier tecla Para continuar:');Vv;
161  Read(n);END;
162
163         END Listar;
164      (* ################################################################## *)
165
166(*Procedimiento Encontrar:Encuentra el índice del registro de el nombre dado
167 *)
168         PROCEDURE Encontrar():INTEGER;
169                 VAR
170                         f:CARDINAL;
171                         s: ARRAY [0..40] OF CHAR;
172         BEGIN
173                 Sp(19);Cud(9);
174  Vi;WriteString('Introduzca el nombre que se quiere buscar:');Vv;WriteLn;
175                 Sp(19);Gets(s);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
176                 FOR f:=0 TO Tamlist DO
177                 IF CompareStr(s,alista[f].nombre)=0 THEN RETURN f END;
178                 END;
179                 RETURN -1;
180         END Encontrar;
181 (* ************************************************************************ *)
182
183 (*Procedimiento Localizar:Visualiza los datos de un registro al completo,
184 basándose en el un nombre dado. *)
185         PROCEDURE Localizar;
186                 VAR
187                         i:INTEGER;
188                         x:CHAR;
189                 BEGIN
190                         Cls;
191                         i:=Encontrar();
192         IF i#(-1) THEN  Cls;Sp(25);Cud(9); Visualiza(alista[i]);
193         Vi;WriteString('Pulse cuaquier tecla para continuar:');Vv;
194                         Read(x);
195                         ELSE
196  Sp(23);Vi;WriteString('Este nombre no esta en la agenda.');Vv;WriteLn;
197  Sp(21);Vi;WriteString('Pulse cualquier tecla para continuar:');Vv;
198                         Read(x);
199                         END;
200                 END Localizar;
201 (* ************************************************************************ *)
202
203 (* Procedimiento Borrar: borra el nombre del registro que se lleve,
204 considerándolo como un registro vacío. *)
205         PROCEDURE Borrar;
206                 VAR
207                         i:INTEGER;
208                 BEGIN
209                         Cls;
210                         i:=Encontrar();
211                         IF i#(-1) THEN alista[i].nombre:="";
212                         END;
213                 END Borrar;
214 (* ************************************************************************ *)
215
216 (* Procedimiento Inicializar: Inicializa todos los registros. *)
217         PROCEDURE Inicializar;
218                 VAR
219                         t:CARDINAL;
220         BEGIN
221                 FOR t:=0 TO Tamlist DO alista[t].nombre:=""; END;
222         END Inicializar;
223 (* ************************************************************************ *)
224
225     (*Procedimiento Salvar: Guarda, todos los datos en un archivo.
226         PROCEDURE Salvar;
227                 VAR
228                         t,i:CARDINAL;
229                         Fi:File;
230         BEGIN
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
            Lookup (F1,'DATOS',TRUE);
            IF F1.res = done THEN Reset(F1);SetWrite(F1) END;

        FOR i:=0 TO Tamlist DO
          IF CompareStr(mlista[i].nombre,"")#0 THEN

            FOR t:=0 TO HIGH(mlista[i].nombre) DO
            WriteChar(F1,mlista[i].nombre[t]) END;

            FOR t:=0 TO HIGH(mlista[i].calle) DO
            WriteChar(F1,mlista[i].calle[t]) END;

            FOR t:=0 TO HIGH(mlista[i].ciudad) DO
            WriteChar(F1,mlista[i].ciudad[t]) END;

            FOR t:=0 TO HIGH(mlista[i].tlf) DO
            WriteChar(F1,mlista[i].tlf[t]) END;

            FOR t:=0 TO HIGH(mlista[i].copos) DO
            WriteChar(F1,mlista[i].copos[t]) END;

          END; (*fin del IF*)
        END; (*fin del FOR*)

            Close(F1);

    END Salvar;
(* ******************************************************************** *)
(* Procedimiento Cargar:Carga datos desde un fichero.*)
    PROCEDURE Cargar;
    VAR
            F1:File;
            i,t:CARDINAL;
    BEGIN
            Lookup(F1,'DATOS',FALSE);
          IF F1.res = done THEN Reset(F1);SetRead(F1);ELSE RETURN END;

            i:=0;
            REPEAT

            FOR t:=0 TO HIGH(mlista[i].nombre) DO
            ReadChar(F1,mlista[i].nombre[t]) END;

            FOR t:=0 TO HIGH(mlista[i].calle) DO
            ReadChar(F1,mlista[i].calle[t]) END;

            FOR t:=0 TO HIGH(mlista[i].ciudad) DO
            ReadChar(F1,mlista[i].ciudad[t]) END;

            FOR t:=0 TO HIGH(mlista[i].tlf) DO
            ReadChar(F1,mlista[i].tlf[t]) END;

            FOR t:=0 TO HIGH(mlista[i].copos) DO
            ReadChar(F1,mlista[i].copos[t]) END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
286                 i:=i+1;
287                 UNTIL F1.eof AND (i<Taslist);
288                 Close(F1);
289         END Cargar;
290 (* ************************************************************************ *)
291 (* Procedimiento Imprimir:Lista por impresora todos los datos  *)
292         PROCEDURE Imprimir;
293                 VAR
294                  i:CARDINAL;
295                 BEGIN
296                         REPEAT
297                         Lookup(F3,'PRN',FALSE);
298                         UNTIL F3.res=done;
299                         SetWrite(F3);
300                           FOR i:=0 TO Taslist DO
301                            IF CompareStr(alista[i].nombre,"")#0 THEN
302                            PRINT (alista[i]);
303                            END;
304                           END;
305
306       WriteChar(F3,33C);WriteChar(F3,65C); (*Cancela Itálicas ESC 5*)
307       WriteChar(F3,33C);WriteChar(F3,122C);(*Cancela Españoles ESC R 0*)
308       WriteChar(F3,60C);
309
310                         Close(F3);
311         END Imprimir;
312 (* ************************************************************************ *)
313
314 (* ************************************************************************ *)
315 (* Procedimineto PRINT :Saca por impresora los registros. *)
316
317         PROCEDURE PRINT(al:direcciones);
318          VAR
319           x:CARDINAL;
320            BEGIN
321            WriteChar(F3,33C);WriteChar(F3,64C); (*Itálicas ESC 4*)
322            WriteChar(F3,33C);WriteChar(F3,122C);(*Españoles ESC R 7*)
323            WriteChar(F3,67C);
324
325            x:=0;
326            REPEAT
327            WriteChar(F3,al.nombre[x]);
328            x:=x+1;
329            UNTIL (al.nombre[x]=CHR(0)) OR (x=41);
330            WriteChar(F3,15C);WriteChar(F3,12C); (* CR;LF *)
331
332            y:=0;
333            REPEAT
334            WriteChar(F3,al.calle[x]);
335            x:=x+1;
336            UNTIL (al.calle[x]=CHR(0)) OR (x=41);
337            WriteChar(F3,15C);WriteChar(F3,12C); (* CR;LF *)
339
339            x:=0;
340            REPEAT
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
341           WriteChar(F3,al.ciudad[x]);
342           x:=x+1;
343           UNTIL (al.ciudad[x]=CHR(0)) OR (x=41);
344           WriteChar(F3,15C);WriteChar(F3,12C); (* CR;LF *)
345
346           x:=0;
347           REPEAT
348           WriteChar(F3,al.tlf[x]);
349           x:=x+1;
350           UNTIL (al.tlf[x]=CHR(0)) OR (x=15);
351           WriteChar(F3,15C);WriteChar(F3,12C); (* CR;LF *)
352
353           x:=0;
354           REPEAT
355           WriteChar(F3,al.copos[x]);
356           x:=x+1;
357           UNTIL (al.copos[x]=CHR(0)) OR (x=6);
358           WriteChar(F3,15C);WriteChar(F3,12C); (* CR;LF *)
359           WriteChar(F3,15C);WriteChar(F3,12C); (* CR;LF *)
360
361       END PRINT;
362 (* ################################################################### *)
363 (* ########################BLOQUE DEL PROGRAMA######################## *)
364
365       BEGIN
366             Inicializar;
367             REPEAT
368                   eleccion:=Menu();
369             CASE eleccion OF
370                   '1': Introduce ;
371                   '2': Borrar ;
372                   '3': Localizar ;
373                   '4': Listar ;
374                   '5': Salvar ;
375                   '6': Cargar ;
376                   '7': Imprimir ;
377                   '8': WriteLn;Sp(2);
378         Puts (' ESTAMOS EN EL SISTEMA OPERATIVO ');
379             END;
380             UNTIL eleccion='8';
381       END listaP
382 (* ################################################################### *)
383
```

```
Modula-2/86 Compiler Version V 1.10

==>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 10
RECURSOS DE BAJO NIVEL

10.1 INTRODUCCION

Como sabemos, M2 es un lenguaje de alto nivel, sin embargo, está dotado de una serie de herramientas necesarias para abordar posibles situaciones, que requieren de la presencia de procedimientos de bajo nivel.

Para permitir a los programas tener conocimiento y por lo tanto acceso al hardware, M2 nos proporciona:

1) La posibilidad de situar una variable en una posición especifica de memoria.

2) Funciones de transferencia de tipos.

3) Los tipos y procedimientos de la librería SYSTEM.

10.2 ALOJAMIENTO DE UNA VARIABLE EN UN LUGAR ESPECIFICO DE MEMORIA

Es sabido que, cuando se declara una variable, el compilador le asigna el lugar de memoria que cree conveniente, ahora bien, el programador puede elegir el lugar específico donde quiere que se situe la variable, simplemente, indicando la dirección entre corchetes inmediatamente después del nombre de la variable.

```
VAR
  <NombreVariable> [Dirección] : Tipo;
```

Por ejemplo:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
VAR
  X [10134] : CARDINAL;
```

En el caso de que el ordenador en que estemos trabajando, utilice un microprocesador del tipo 8086/8088, las direcciones deben especificarse con el formato base/desplazamiento (segment/offset), por ejemplo:

```
VAR
  Y [0B003H:0700H] : INTEGER;
```

Este tipo de direccionamiento segmentado, es típico en los µp en los que el bus de direcciones consta de 20 bits y la CPU, sólo dispone de registros de 16 bits.

Con 20 bits, se pueden direccionar 2^{20} = 1 Megabyte. Con la técnica de la segmentación, este espacio total, se divide en trozos de 64 Kbytes (2^{16}), que reciben el nombre de "segmentos".

Para calcular una dirección de memoria en el 8086/8088, se parte del contenido de uno de los segmentos, que actúa como base, después, se multiplica por 16 dicho contenido, lo que en binaro, significa añadirle 4 ceros a la derecha y convertirlo en una magnitud de 20 bits, finalmente se suma un desplazamiento al resultado de la multiplicación anterior. Abreviadamente, la fórmula para calcular una dirección de memoria es:

Dirección de memoria = (base X 16) + desplazamiento

10.3 FUNCIONES DE TRANSFERENCIA DE TIPOS

Estas funciones, nos permiten que el contenido de una variable de un tipo, se sitúe en otra variable de otro tipo,

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

usando el nombre del tipo al que se convierte, como nombre de la función y la variable a convertir, entre paréntesis como argumento de la función. Por ejemplo:

```
VAR
  C:CARDINAL;
  I:INTEGER;
BEGIN
  C:=20;
  I:=INTEGER(C);
```

A diferencia de VAL, ORD o CHR, las funciones de transferencia de tipos, no hacen ningún cálculo, simplemente transfieren la secuencia de bits desde una variable de un tipo a otra, por lo tanto, la conversión sólo es posible, cuando ambos tipos de datos ocupan el mismo número de bits.

Las funciones de transferencia de tipos, también pueden usarse con los tipos de datos definidos por el programador, siempre que tengan la misma longitud. Por ejemplo:

```
TYPE
  Registro1 = RECORD
                C:CARDINAL;
                nombre:ARRAY [0..100] OF CHAR;
              END;
  Registro2 = RECORD
                nombre:ARRAY [0..100] OF CHAR;
                C:CARDINAL;
              END;
VAR
  R1:Registro1;
  R2:REgistro2;
BEGIN
. . . .
R1:= Registro1(R2);
```

A continuación, se muestra el tamaño que se requiere en memoria, para albergar a determinados tipos de datos.

BYTE	8 bits	BITSET	16 bits
CHAR	8 bits	POINTER	4 bytes
INTEGER	16 bits	ADDRESS	4 bytes
CARDINAL	16 bits	Tipos enumerados	8 bits
REAL	8 bytes	ARRAY	variable
SET	16 bits	RECORD	variable

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

El siguiente módulo de programa, nos visualiza la representación en hexadecimal y binario de cualquier número entero, utilizando funciones de transferencia de tipos.

```
Modula-2/86                    transtyp.MOD

 1  MODULE HexBinario;
 2    FROM InOut IMPORT WriteInt,ReadInt,Done,Write,WriteLn,WriteHex,WriteString;
 3    VAR
 4      I:INTEGER;
 5
 6    PROCEDURE Palabra (a:INTEGER);
 7       VAR
 8         x:BITSET;
 9         n:CARDINAL;
10    BEGIN
11      x:=BITSET(a);
12      FOR n:=15 TO 0 BY -1 DO
13        IF (n) IN (x)
14        THEN Write('1')
15        ELSE Write('0');
16        END;
17        IF n MOD 4 = 0 THEN Write(' ') END;(*Espacio cada 4 bits para facilitar la lectura*)
18      END;
19    END Palabra;
20
21  BEGIN
22    Write(14C);
23    ReadInt(I);WriteLn;
24    WHILE Done DO (* Cuando no se lea un entero, se sale del lazo*)
25      WriteHex(CARDINAL(I),0);Write('H');WriteLn;
26      Palabra(I);WriteString(' palabra binaria. ');WriteLn;
27      ReadInt(I);WriteLn;
28    END;
29  END HexBinario.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

Para la mejor comprensión del procedimiento Palabra, hay que tener en cuenta que una variable declarada con el tipo BITSET, es un conjunto que puede tomar valores comprendidos entre 0 y n-1, donde n es el número de bits de la palabra del computador, generalmente 16, es decir;

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
TYPE
  BITSET = SET OF [0..15];(*tipo predefinido en el lenguaje*)
VAR
  x,y,z :BITSET;
```

y los conjuntos, son representados internamente por una secuencia de bits, donde el bit cero, indica que el elemento correspondiente, no está en el conjunto y el bit uno, que si está. Por ejemplo, las asignaciones;

```
x:=BITSET{}
y:=BITSET{0,1,13,15}
z:=BITSET{1..7,11}
```

tendrían las siguinetes representaciones internas:

Conjunto	Representación
{}	0000 0000 0000 0000
{0,1,13,15}	1010 0000 0000 0011
{1..7,11}	0000 1000 1111 1110

Esto se podría verificar con la ejecución del siguiente programa ejemplo.

Modula-2/86 verifica.MOD

```
 1 MODULE RepresentacionBinariaDeUnConjunto;
 2   FROM InOut IMPORT Write,WriteLn,WriteHex,WriteString;
 3   VAR
 4     x,y,z:BITSET;
 5
 6   PROCEDURE Palabra (x:BITSET);
 7     VAR
 8       n:CARDINAL;
 9   BEGIN
10     FOR n:=15 TO 0 BY -1 DO
11       IF (n) IN (x)
12       THEN Write('1')
13       ELSE Write('0');
14       END;
15       IF n MOD 4 = 0 THEN Write(' ') END;(*Espacio cada 4 bits para facilitar la lectura*)
16     END;
17   END Palabra;
18
19 BEGIN
20   Write(14C);
21
22   x:=BITSET{};
23   y:=BITSET{0,1,13,15};
24   z:=BITSET{1..7,11};
25
26   WriteString('{}         ');
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
27   WriteHex(CARDINAL(x),0); WriteString(':H = '); Palabra(x); WriteLn;
28   WriteString('(0,1,13,15)  ');
29   WriteHex(CARDINAL(y),0); WriteString(':H = '); Palabra(y); WriteLn;
30   WriteString('(1..7,11)    ');
31   WriteHex(CARDINAL(z),0); WriteString(':H = '); Palabra(z); WriteLn;
32
33 END RepresentacionBinariaDeUnConjunto.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

10.4 TIPOS DE LA LIBRERIA SYSTEM

Del módulo de librería SYSTEM, se pueden importar los tipos de datos BYTE, WORD y ADDRESS, los cuales nos permiten manipular variables. El tamaño de BYTE o WORD (8, 16, 32, 64 bits), dependerá de la máquina que se utilice. En la configuración IBM/PC, el tipo BYTE designa una palabra de 8 bits y el tipo WORD de 16 bits.

M2 no permite que se apliquen operaciones a valores de los tipos BYTE o WORD (excepto la asignación), porque considera que estos valores no son interpretables, sin embargo, la utilización de las funciones de transferencia de tipos, que se vio en el apartado anterior, hacen posible su aplicación.

Uno de los principales usos de BYTE y WORD, está en la declaración de los parámetros formales de un procedimiento, de forma que, en los parámetros actuales de llamada, podría colocarse cualquier tipo de dato que ocupe el tamaño de 8 o 16 bits respectivamente.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

El procedimiento Palabra, que se vio en el módulo transtyp.mod, se podría generalizar para cualquier tipo de dato de una longitud de 16 bits (CARDINAL, INTEGER, SET, BITSET),simplemente sustituyendo PROCEDURE Palabra(m:INTEGER), por PROCEDURE Palabra(m:WORD).

Otro ejemplo, podría ser el siguiente procedimiento función, que nos devuelve el cuadrado tanto de un número entero como de un cardinal.

```
PROCEDURE Cuadrado (x:WORD):CARDINAL;
   BEGIN
     RETURN (CARDINAL(x) * CARDINAL(x));
END Cuadrado;
```

El tipo ADDRESS denota las direcciones de las palabras de memoria, está definido como un puntero que apunta a datos de tipo WORD.

```
TYPE
  ADDRESS = POINTER TO WORD
```

También ADDRESS puede expresarse mediante la forma;

```
TYPE
  ADDRESS = RECORD
              SEGMENT:CARDINAL;
              OFFSET:CARDINAL;
            END;
```

según el modo de direccionamiento segmentado del 8086/8088. En general, una variable del tipo ADDRESS, direcciona un dato del tipo WORD.

Las únicas operaciones permitidas con datos de este tipo son, la comparación y la adición y sustracción cardinal, pero sólo procesan la parte de desplazamiento (OFFSET).

Una muestra de como puede ser usado ADDRESS, está en el siguiente programa ejemplo, que hace un vaciado de memoria, visualizando el contenido por pantalla en formato hexadecimal.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
MODULE DumpingDeMemoria;
  FROM SYSTEM IMPORT ADDRESS;
  FROM InOut IMPORT WriteString,WriteHex,ReadCard,WriteLn,WriteCard,Write,Done,Read;
  VAR
    desde, hasta:ADDRESS;
  PROCEDURE Vacia(principio,fin:ADDRESS);
    VAR
      t: CARDINAL; (*contador de palabras por linea*)
      l,n: CARDINAL; (*contador de lineas*)
      ch:CHAR;
      x:BITSET;
  BEGIN
    WriteCard(principio.SEGMENT,0); Write(':');WriteCard(principio.OFFSET,5); WriteString('=');
    WriteHex(principio.SEGMENT,4);WriteString(':H ');
    WriteHex(principio.OFFSET,4);WriteString(':H');
    t:=0;
    l:=0;
    WHILE ((principio.SEGMENT<fin.SEGMENT) OR (principio.SEGMENT =fin.SEGMENT)
          AND (principio.OFFSET <= fin.OFFSET)) AND (ch#33C) DO
      x:=BITSET(principio^); (*principio apunta a un dato del tipo WORD*)
      FOR n:=15 TO 0 BY -1 DO
        EXCL(x,n);
      END;
      WriteHex(CARDINAL(x),6);
      IF (principio.SEGMENT = 65535) AND (principio.OFFSET = 65534) THEN
        ch:=33C;
      ELSE
        IF principio.OFFSET = 65534 THEN
          principio.OFFSET:=0;
          INC (principio.SEGMENT);
        ELSE
          principio.OFFSET:=principio.OFFSET+1;
        END;
        INC(t);
        IF t=9 THEN
          t:=0;
          WriteLn;INC(l);
          IF l=24 THEN
            l:=0;
            WriteString('       *Pulse cualquier tecla para continuar o ESC para salir*');
            Read(ch);WriteLn;
          END;
          IF ch#33C THEN
            WriteCard(principio.SEGMENT,0); Write(':');WriteCard(principio.OFFSET,5); WriteString('=');
            WriteHex(principio.SEGMENT,4);WriteString(':H ');
            WriteHex(principio.OFFSET,4);WriteString(':H');
          END;
        END
      END
    END
  END Vacia;
BEGIN
  Write(14C);
  REPEAT
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
55    WriteString('Introduzca el segmento inicial: ');
56    ReadCard(desde.SEGMENT);WriteLn;
57  UNTIL Done;
58  REPEAT
59    WriteString('Introduzca el offset inicial: ');
60    ReadCard(desde.OFFSET);WriteLn;
61  UNTIL (Done) AND (desde.OFFSET <65535);
62  REPEAT
63    WriteString('Introduzca el segmento final: ');
64    ReadCard(hasta.SEGMENT);WriteLn;
65  UNTIL Done;
66  REPEAT
67    WriteString('Introduzca el offset final: ');
68    ReadCard(hasta.OFFSET);WriteLn;
69  UNTIL (Done) AND (hasta.OFFSET <65535);
70    IF (desde.SEGMENT > hasta.SEGMENT) OR (desde.SEGMENT = hasta.SEGMENT) AND
71       (desde.OFFSET > hasta.OFFSET)
72    THEN
73      WriteString('El formato inicial debe ser ≤ al final ');
74    ELSE
75      Write(14C);
76      Vacia(desde,hasta);
77    END;
78  END DumpingDeMemoria.
```

```
Modula-2/86 Compiler Version V 1.10
=>    0 Error(s) found
```

Este programa, sirve para direccionar cualquier posición de memoria, ahora bien, se debe saber que;

dirección de memoria = [SEGMENT] x 16 + [OFFSET]

y por lo tanto direccionar, por ejemplo, desde SEGMENT [1] OFFSET [0] a SEGMENT [1] OFFSET [100],es lo mismo que hacerlo desde SEGMENT [0] OFFSET [16] a SEGMENT [0] OFFSET [116].

10.5 PROCEDIMIENTOS DE LA LIBRERIA SYSTEM

El módulo de la librería SYSTEM, contiene tres importantes procedimientos función, que son:

```
PROCEDURE ADR (t:CualquierVariable):ADDRESS;
PROCEDURE SIZE (t:CualquierVariable):CARDINAL;
PROCEDURE TSIZE (t:CualquierTipo):CARDINAL;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

ADR devuelve la dirección en memoria de cualquier variable.

SIZE y TSIZE devuelven un número cardinal, que representa el tamaño en bytes que ocupa una variable o un tipo de dato respectivamente.

El siguiente programa, es un ejemplo de utilización de estos procedimientos.

Modula-2/86 prosys.MOD

```
 1 MODULE SizeTsizeAdr;
 2   FROM InOut IMPORT WriteCard,WriteHex,WriteString,Write,WriteLn;
 3   FROM SYSTEM IMPORT ADDRESS,SIZE,TSIZE,ADR;
 4   CONST
 5     Tamlist = 100;
 6   TYPE
 7     direcciones = RECORD
 8                     nombre:ARRAY [0..40] OF CHAR;
 9                     calle:ARRAY [0..40] OF CHAR;
10                     ciudad:ARRAY [0..40] OF CHAR;
11                     tlf:ARRAY [0..14] OF CHAR;
12                     copos:ARRAY [0..5] OF CHAR;
13                   END;
14   VAR
15     alista:ARRAY [0..Tamlist] OF direcciones;
16     x:ADDRESS;
17     i:CARDINAL;
18 BEGIN
19   WriteString('El tamaño del tipo direcciones es: ');
20   WriteCard(TSIZE(direcciones),0);WriteLn;
21
22   WriteString('El tamaño de la variable alista es: ');
23   WriteCard(SIZE(alista),0);WriteLn;
24
25   WriteString('El tamaño de la variable x es: ');
26   WriteCard(SIZE(x),0);WriteLn;
27
28   WriteString('El tamaño de la variable i es: ');
29   WriteCard(SIZE(i),0);WriteLn;
30
31   x:=ADR(alista);
32   WriteString('La localización de la variable alista es: ');
33   WriteHex(x.SEGMENT,0);WriteString(':H ');WriteHex(x.OFFSET,4);WriteString(':H ');
34   WriteString('y su contenido es: ') ; WriteHex(CARDINAL(x^),0); WriteLn;
35
36   WriteString('El tamaño de cada uno de los campos de alista es: ');
37   WriteCard(SIZE(alista[i].nombre),4);Write(';');
38   WriteCard(SIZE(alista[i].calle),4);Write(';');
39   WriteCard(SIZE(alista[i].ciudad),4);Write(';');
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
40    WriteCard(SIZE(alista[i].tlf),4);Write(';');
41    WriteCard(SIZE(alista[i].copos),4);Write(';');
42  END SizeTsizeAdr.
```

```
Modula-2/86 Compiler Version V 1.10

=)     0 Error(s) found
```

De la ejecución de este programa resultaría:

```
El tamaño del tipo direcciones es: 148
El tamaño de la variable alista es: 14948
El tamaño de la variable x es: 4
El tamaño de la variable i es: 2
La localización de la variable alista es: 2B20:H 0000:H y su contenido es: 3102
El tamaño de cada uno de los campos de alista es:   41;  41;  41;  15;   6;
```

10.6 ACCESO A FICHEROS BINARIOS DE FORMA ALEATORIA

En el capítulo anterior, en el apartado 9.3, se mostró como leer y escribir caracteres ASCII en ficheros.

Sin embargo, también se pueden leer y escribir datos binarios en ficheros, utilizando las rutinas descritas pero no comentadas, de ese apartado.

Estas rutinas no fueron examinados detenidamente, debido a que se utilizan en conjunción con los tipos y procedimientos de la librería SYSTEM.

Sus formas generales son;

```
ReadByte (<VarFile>,<VarByte>)
ReadWord (<VarFile>,<VarWord>)
WriteByte (<VarFile>,<Byte>)
WriteWord (<VarFile>,<Word>)
```

Donde VarFile es el descriptor de fichero, VarByte y VarWord, son variables para contener la información leída, y Byte y Word son los valores a escribir.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Al igual que con las demás rutinas de fichero, el resultado de la ejecución de la sentencia, se encuentra en el campo res de la variable VarFile.

Especial atención, requieren ReadNBytes y WriteNBytes, pues son las rutinas más utilizadas, debido a que pueden leer y escribir grandes bloques de datos, utilizando una sola sentencia.

ReadNBytes(<VarFile>,<DirVar>,<NumBytes>,<NumLeídos>)

DirVar, es la dirección de la variable, que contendrá la información a ser leída del fichero. NumBytes, es el número de bytes a ser leídos. NumLeídos, es una variable que contiene el número actual de bytes leídos.

La verificación de la ejecución correcta de la sentencia, se puede hacer, examinando el campo res del descriptor de fichero, o bien, viendo si coinciden el número de bytes a ser leídos (NumBytes) con los leídos (NumLeídos).

En cuanto al procedimiento WriteNBytes:

WriteNBytes(<VarFile>,<DirVar>,<NumBytes>,<NumEscritos>)

DirVar, es la dirección de la variable, que contendrá la información a ser escrita en el fichero. NumBytes, es el número de bytes a escribir. NumEscritos, es una variable de tipo cardinal que contiene el número actual de bytes escritos.

La verificación de la ejecución correcta de la sentencia, se puede hacer, examinando el campo res del descriptor de fichero, o bien, comprobando si coinciden el número de bytes a ser escritos (NumBytes) con los escritos (NumEscritos).

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

DirVar y NumBytes, se obtienen con los procedimientos de SYSTEM ADR y TSIZE respectivamente.

A continuación, se describen los procedimientos para los ficheros binarios, con sus listas de parámetros formales.

```
ReadByte (VAR f:File; VAR b:BYTE);

ReadWord (VAR f:File; VAR w:WORD);

WriteByte (VAR f:File; b:BYTE);

WriteWord (VAR f:File; w:WORD);

ReadNBytes (VAR f:File; apuntbuf:ADDRESS; numbytes:CARDINAL;
            VAR leidos:CARDINAL);

WriteNBytes (VAR f:File; apuntbuf:ADDRESS; numbytes:CARDINAL;
             VAR escritos:CARDINAL);
```

El siguiente módulo, es un ejemplo de la utilización de ReadNBytes y WriteNBytes, consiste, en escribir dos números reales en un fichero para posteriormente recuperarlos y visualizarlos.

Modula-2/86 nbytes.MOD

```
 1 MODULE ReadWriteNBytes;
 2   FROM InOut IMPORT ReadString,WriteString,WriteLn,Write;
 3   FROM FileSystem IMPORT File,Response,Lookup,SetModify,Close,Reset,Delete,
 4                          ReadNBytes,WriteNBytes;
 5   FROM RealInOut IMPORT WriteReal;
 6   FROM SYSTEM IMPORT ADR,TSIZE;
 7   VAR
 8     R1,Rb1,R2,Rb2:REAL;
 9     F1:File;
10     nombre:ARRAY [0..30] OF CHAR;
11     leido1,leido2,escrito1,escrito2:CARDINAL;
12
13 BEGIN
14   Write(14C);
15   R1:=73821.125E16;
16   R2:= -10114.E-7;
17   WriteString('Introduzca el nombre de un fichero de salida : ');ReadString(nombre);WriteLn;
18   WriteReal(R1,15);Write(' ');WriteReal(R2,12);WriteLn;(*Se visualiza R1 y R2 antes de manipularlos *)
19
20   REPEAT
21     Lookup (F1,nombre,TRUE);
22   UNTIL F1.res = done;
23   Reset(F1);
24   SetModify(F1);(* Abre el fichero a la lectura y escritura *)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
25
26    (* Escribimos R1 y R2 en el fichero *)
27    WriteNBytes(F1,ADR(R1),TSIZE(REAL), escrito1);
28    IF escrito1#TSIZE(REAL) THEN WriteString('Ha habido un error e1');WriteLn END;
29    WriteNBytes(F1,ADR(R2),TSIZE(REAL), escrito2);
30    IF escrito2#TSIZE(REAL) THEN WriteString('Ha habido un error e2');WriteLn END;
31
32    (* Leemos R1 y R2 del fichero *)
33    Reset(F1);
34    ReadNBytes(F1,ADR(Rb1),TSIZE(REAL), leido1);
35    IF leido1#TSIZE(REAL) THEN WriteString('Ha habido un error l1');WriteLn END;
36    ReadNBytes(F1,ADR(Rb2),TSIZE(REAL), leido2);
37    IF leido2#TSIZE(REAL) THEN WriteString('Ha habido un error l2');WriteLn END;
38    Close(F1);
39
40    Delete(nombre,F1); (* Borramos el fichero *)
41    IF F1.res = notdone THEN WriteString('disco protegido contra escritura');WriteLn END;
42
43    WriteReal(Rb1,15);Write(' ');WriteReal(Rb2,12); (* Se visualizan los datos obtenidos del fichero *).
44
45  END ReadWriteNBytes.
```

```
Modula-2/86 Compiler Version V 1.10

=>      0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 11
ADICIONES AL LENGUAJE

11.1 INTRODUCCION

Generalmente cuando se trabaja con un lenguaje, hay situaciones que no se pueden abordar directamente con él, tales como, hacer llamadas al sistema operativo, escribir rutinas con temporizaciones exactas o de gran velocidad de ejecución, etc.

Sin embargo, M2 está diseñado con flexibilidad para soportar esas adiciones al lenguaje, permitiéndonos escribir rutinas utilizando el ensamblador y hacer llamadas al sistema operativo.

Los ejemplos de este capitulo se basan en el sistema operativo MS-DOS y en el lenguaje ensamblador 8086/8088.

11.2 CODIGO ENSAMBLADOR INSERTADO

Cuando se escribe un procedimiento en lenguaje ensamblador, que debe interactuar con un código compilado en M2, se deben seguir unas normas de adaptación al compilador que estemos utilizando (en nuestro caso el Logitech), sólo de esta forma, se puede estar seguro de que el código insertado en lenguaje ensamblador, interactúa correctamente con el código de M2.

Hay que tener en cuenta, que M2 usa la pila para pasar parámetros a una subrutina y se debe acceder a ellos

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

utilizando el registro Puntero Base, BP, para el indexado dentro de ella.

Los parámetros valor, de un solo sentido, se pasan al stack, utilizando una palabra (16 bits), excepto los de tipo real, que necesitan 8 bytes y los punteros 2 palabras. En el caso de que sólo se necesite un byte, la mitad más significativa de la palabra, se pone a cero.

A diferencia de los anteriores, los parámetros variables, de dos sentidos, se pasan a la pila mediante sus direcciones, por lo tanto, utilizan 2 palabras.

Una vez se efectúa la llamada al procedimiento y se han alojado los parámetros en la pila, el compilador sitúa en ésta la dirección de retorno.

Para la inserción de un código ensamblador como parte de un programa M2, la librería SYSTEM proporciona el procedimiento CODE, cuya forma general es:

```
CODE (Valor,Valor,.....,Valor);
```

Donde Valor, es el código correspondiente a las instrucciones en ensamblador.

De la misma librería, también se exportan las siguientes constantes:

AX=0	DX=2	ES=8	SS=10
BX=3	SI=6	DS=11	SP=4
CX=1	DI=7	CS=9	BP=5

Estas constantes representan los registros del procesador y se suelen usar en conjunción con los procedimientos GETREG y SETREG, (también proporcionados por la librería SYSTEM) para leer y escribir respectivamente en

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

los registros de máquina.

GETREG (Reg:CARDINAL; VAR Val:BYTE o WORD);

Este procedimiento lee el registro del 8086 cuyo número se le indica en Reg y sitúa el resultado en la variable Val.

SETREG (Reg:CARDINAL; Val:BYTE o WORD);

Pone el valor que se le indica en Val, en el registro que se especifique en Reg.

Hay que tener en cuenta, que los registros CS, SS, SP y BP, no son modificables y por lo tanto no se pueden utilizar con SETREG.

El siguiente programa, utiliza el procedimiento 'multiplica', en forma de código insertado, para realizar el producto de dos números enteros.

```
Modula-2/86                    ensamb13.MOD

 1 MODULE InsCoEnsam;
 2   FROM SYSTEM IMPORT CODE,AX,GETREG;
 3   FROM InOut IMPORT WriteInt,ReadInt,WriteLn,Done,Write;
 4   VAR
 5     a,b,c:INTEGER;
 6   PROCEDURE multiplica(x,y:INTEGER):INTEGER;
 7     VAR resultado:INTEGER;
 8     BEGIN
 9       CODE (89H, 46H, 04H, (* MOV AX,[BP]+4 *)
10             0F7H, 66H, 06H); (* MUL [BP]+6 *)
11       GETREG (AX, resultado);
12       RETURN resultado
13   END multiplica;
14 BEGIN
15   Write(14C);
16     ReadInt(a);WriteLn;
17     ReadInt(b);WriteLn;
18   WHILE Done DO
19     c:= multiplica(a,b);
20     WriteInt(c,6); WriteLn;
21     ReadInt(a);WriteLn;
22     ReadInt(b);WriteLn;
23   END;
24 END InsCoEnsam.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Hay que observar, que una vez llamado el procedimiento, éste, accede a la pila para obtener los parámetros que se encuentran en el byte cuarto (a) y sexto (b) bajando por la pila, pués la dirección de retorno, que se salvó posteriormente a los parámetros variables a y b, utiliza cuatro bytes.

Se debe tener en cuenta que cuando se realizan cálculos o se ejecutan instrucciones, éstas, pueden destruir valores de registros escritos por SETREG o a ser leídos por GETREG, por lo que estos procedimientos deben usarse con variables declaradas como locales a los procedimientos, como es el caso de la variable 'resultado' del ejemplo anterior.

11.3 ACCESO A FUNCIONES DEL SISTEMA OPERATIVO

Generalmente hay funciones especiales que el sistema operativo es capaz de soportar, pero que no se encuentran en las rutinas de biblioteca proporcionadas por el compilador.

M2 soporta varios sistemas operativos, pero en el caso del MS-DOS que es el que nos ocupa, la forma de acceder a estas funciones, es a través de las interrupciones de software, que son llamadas a rutinas del sistema operativo, que en función del valor que tenga el registro AH u otros registros, determinará que función se va a realizar.

Existen 256 interrupciones posibles (0 - FFH), pero las que se refieren al sistema operativo, son de dos tipos:
BIOS (0 - 1F) (Sistema básico de entrada/salida). Son rutinas básicas de bajo nivel de E/S.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

DOS (20 - 3F) (Sistema operativo de disco). Son funciones de alto nivel del sistema operativo, suelen invocar a rutinas elementales del BIOS.

Una lista completa de las interrupciones con las diferentes funciones que realizan, se puede encontrar en un manual de referencia de IBM.

Veamos un ejemplo, utilizando la interrupción 10H del BIOS, función 6H, que hace un desplazamiento (scroll) hacia arriba de la página.

INT 10H función 6H: Desplaza la página hacia arriba.

AH=6H

AL= Número de lineas a desplazar, las lineas de la parte inferior de la ventana se borran. Si AL = 0, se borra toda la ventana.

CH= Fila esquina superior izquierda.

CL= Columna esquina superior izquierda.

DH= Fila esquina inferior derecha.

DL= Columna esquina inferior derecha.

BH= Atributo a ser usado para las lineas en blanco.

ATRIBUTO	CODIGO
Normal	0000 0111 = 07H
Video inverso	0111 0000 = 70H
Subrayado	0000 0001 = 01H
Invisible (negro)	0000 0000 = 00H
Invisible (blanco)	0111 0111 = 77H

Con este módulo de programa, lo que se consigue es visualizar dos ventanas en la pantalla.

Modula-2/86 ventanas1.MOD

```
1 MODULE Ventanas;
2   FROM InOut IMPORT WriteString,Write,Read;
3   FROM SYSTEM IMPORT CODE;
4   FROM Util IMPORT Vi,Vv,PxPy,Cls;
5   VAR
6     ni,CARDINAL;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 7     ch:CHAR;
 8  PROCEDURE  Ventana1;
 9  BEGIN
10    CODE(
11      0B5H,05,     (*MOV CH,5*)  (*FESI*)
12      0B1H,0EH,    (*MOV CL,14*) (*CESI*)
13      0B6H,09,     (*MOV DH,9*)  (*FEIB*)
14      0B2H,22H,    (*MOV DL,34*) (*CEIB*)
15      0B4H,06,     (*MOV AH,6*)  (*función 6*)
16      0B0H,00,     (*MOV AL,0*)  (*lineas a desplazar*)
17      0B7H,01,     (*MOV BH,1*)  (*atributo lineas en blanco*)
18      0CDH,10H);   (*INT 10 *)
19  END Ventana1;
20
21  PROCEDURE  Ventana2;
22  BEGIN
23    CODE(
24      0B5H,0FH,    (*MOV CH,15*) (*FESI*)
25      0B1H,2DH,    (*MOV CL,44*) (*CESI*)
26      0B6H,13H,    (*MOV DH,19*) (*FEIB*)
27      0B2H,40H,    (*MOV DL,64*) (*CEIB*)
28      0B4H,06,     (*MOV AH,6*)  (*función 6*)
29      0B0H,00,     (*MOV AL,0*)  (*lineas a desplazar*)
30      0B7H,01,     (*MOV BH,1*)  (*atributo lineas en blanco*)
31      0CDH,10H);   (*INT 10 *)
32  END Ventana2;
33
34  BEGIN
35    Cls;
36    Vi; FOR n:=1 TO 2500 DO Write(' ') END; Vv;
37    Ventana1;
38    Ventana2;
39    PxPy(21,6); WriteString('Ventana 1 ');
40    PxPy(51,16); WriteString('Ventana 2 ');
41    Read(ch);
42  END Ventanas.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

Teniendo en cuenta, que la librería SYSTEM proporciona el procedimiento SWI, cuya forma general es;

SWI (nº del vector de interrupción (≤225):CARDINAL);

y que genera una interrupción de software, el módulo anterior se podría escribir en conjunción con el procedimiento SETREG, de la siguiente forma.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 1 MODULE Ventanas;
 2   FROM InOut IMPORT WriteString,Write,Read;
 3   FROM SYSTEM IMPORT AX,BX,CX,DX,SETREG,SWI,CODE;
 4   FROM Util IMPORT Vi,Vv,PxPy,Cls;
 5   VAR
 6     n:CARDINAL;
 7     ch:CHAR;
 8 PROCEDURE  Ventana1;
 9 BEGIN
10   SETREG(CX,5000H);
11   SETREG(DX,9223H);
12   SETREG(AX,0600H);
13   SETREG(BX,1000H);
14   CODE(55H); (*PUSH BP*)
15   SWI(10H);
16   CODE(5DH); (*POP BP*)
17 END Ventana1;
18
19 PROCEDURE  Ventana2;
20 BEGIN
21   SETREG(CX,0F2CH);
22   SETREG(DX,1340H);
23   SETREG(AX,0600H);
24   SETREG(BX,1000H);
25   CODE(55H); (*PUSH BP*)
26   SWI(10H);
27   CODE(5DH); (*POP BP*)
28 END Ventana2;
29
30 BEGIN
31   Cls;
32   Vi; FOR n:=1 TO 2500 DO Write(' ') END; Vv;
33   Ventana1;
34   Ventana2;
35   PxPy(21,6); WriteString('Ventana 1 ');
36   PxPy(51,16); WriteString('Ventana 2 ');
37   Read(ch);
38 END Ventanas.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

Se puede observar, que antes de ejecutarse la interrupción, se salva el registro BP para posteriormente recuperar su valor. Esto es debido a que el fabricante del compilador, recomienda que cuando se utilce el procedimiento SWI para llamar al IBM-PC ROM BIOS, se debe salvar y

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

restaurar el Puntero Base pues éste, es esencial para el compilador, debido a que lo utiliza para acceder a variables locales y parámetros de procedimientos. Por lo tanto, debe insertarse CODE(55H) (PUSH BP) justo antes y CODE(5DH) (POP BP) justo después de la llamada a SWI.

Otro ejemplo con la INT 10H, pero en este caso utilizando la función 2H, lo tenemos en el siguiente módulo, que nos permite no visualizar el cursor en la pantalla.

INT 10H función 2H: Fija la posición del cursor.

AH=2H

DH:Fila.

DL:Columna.

BH:Número de la página de video.

```
Modula-2/86                    nocursor.MOD

 1 MODULE DesaparecaCursor;
 2 FROM InOut IMPORT WriteString, Read;
 3 FROM Util IMPORT Cls, PxPy;
 4 FROM SYSTEM IMPORT CODE, SWI, SETREG, AX, BX, DX;
 5 VAR
 6   ch:CHAR;
 7
 8 PROCEDURE Bo; (*Desaparece el cursor*)
 9 BEGIN
10   SETREG (AX,0200H);(*función 2H*)
11   SETREG (DX,1900H);(*fila 25=19H,columna 00*)
12   SETREG (BX,00H);   (*pag. de video*)
13   CODE(55H);
14   SWI(10H);
15   CODE(5DH);
16 END Bo;
17
18 BEGIN
19   Cls;PxPy(33,12);
20   WriteString('Cursor Borrado');
21   Bo;
22   Read(ch);
23 END DesaparecaCursor.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Bibliotecas Universitaria, 2006

Para la utilización de la interrupción 16H función OH en el próximo ejemplo, se debe saber que:

INT 16H función OH:Lee el código de inspección del teclado.

```
ENTRADA
  AH=OH
DEVUELVE
  AH:Código de posición.
  AL:Código ASCII del carácter.
```

El teclado del IBM PC consta de 83 teclas, divididas en tres grupos; teclas de función, teclas estándar de máquina de escribir y teclado numérico.

Cada vez que se pulsa una tecla, se genera una palabra o código de inspección del teclado, el byte más significativo o código de posición, tomará un valor de 1 a 83, dependiendo de la posición que ocupe ésta respecto al teclado, el byte menos significativo o código de carácter, tomará el valor ASCII correspondiente a la tecla, si éste existe.

Si el segundo código es 0, ésto indica que se trata de una tecla especial y que no existe código ASCII del carácter, tal es el caso de las teclas de función y las de control del cursor, en este caso, debe examinarse el código de posición, que es el que identificará la tecla.

Los procedimientos estándar de M2 del tipo Read, sólo permiten leer el código ASCII de un carácter, lo que no nos permite tener control sobre todo el teclado.

Las combinaciones de teclas como ALT, mayúsculas y CTRL acompañadas de otra tecla, también generan un código de este tipo.

El siguiente programa ejemplo, nos permite averiguar los códigos de inspección de cualquier tecla o combinación de

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

teclas, hasta que se pulse ALT - CTRL, < >,que nos devuelve al sistema operativo.

Modula-2/86 teclado.MOD

```
 1 MODULE CodigosTeclas;
 2   FROM InOut IMPORT WriteString, WriteHex, WriteCard, Write;
 3   FROM SYSTEM IMPORT CODE,GETREG,AX;
 4   FROM Util IMPORT PxPy, Cls, Neg, Vv;
 5 TYPE
 6   MSBLSB = ARRAY [0..1] OF CHAR;
 7 VAR
 8   p,q:CARDINAL;
 9   s:MSBLSB;
10 PROCEDURE Inspecciona():CARDINAL;
11   VAR
12     codigo:CARDINAL;
13 BEGIN
14   CODE (0B4H,0, (*MOV AH,0*)
15         0CDH,16H);(*INT 16*)
16   GETREG (AX,codigo);
17   RETURN codigo;
18 END Inspecciona;
19 BEGIN
20   REPEAT
21     Cls;
22     WriteString('pulse una tecla ');
23     p:=Inspecciona();
24     Cls;
25     s:=MSBLSB(p);
26     PxPy(27,10); WriteString('C. POSICION'); PxPy(45,10); WriteString('C. CARACTER');
27     PxPy(30,11); WriteCard(ORD(s[1]),0); PxPy(48,11); WriteCard(ORD(s[0]),0);
28     PxPy(30,12); WriteHex(ORD(s[1]),0); Write('H'); PxPy(48,12); WriteHex(ORD(s[0]),0); Write('H');
29     PxPy(10,18); Neg; WriteString('Pulse cualquier tecla para continuar o ALT-CTRL <> para salir'); Vv;
30     q:=Inspecciona();
31   UNTIL q = 2E0CH;
32   PxPy(0,25);
33 END CodigosTeclas.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

Una función BIOS interesante de estudiar es la 2H de la interrupción 17H, que tiene por misión, la de leer el estado de la impresora.

INT 17H función 2H:Lee el estado de la impresora.

ENTRADA

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

AH = 2H
DX = Impresora a usar (0-2)

DEVUELVE
AH = Byte de estado de la impresora.

El byte de estado tiene el siguiente significado.

bits	Contenido
7	Impresora no ocupada (0=ocupada, 1=ocupada)
6	Reconocimiento de la impresora (0=señal, 1=normal)
5	Fin del papel (0=con papel, 1=sin papel)
4	Seleccionada(0=impresora no en linea, 1=en linea)
3	Error de entrada/salida (0=error, 1=normal)
2	No se usa
1	No se usa
0	Tiempo excedido (0=no, 1=si)

Con la ejecución del siguiente programa, se podría verificar que:

Estado = 24576 = 6000H con impresora apagada.

Estado = 32768 = 8000H con impresora no en linea.

Estado = 36864 = 9000H con impresora en linea.

```
Modula-2/86                    impresor.MOD

 1 MODULE EstadoDeImpresora;
 2   FROM InOut IMPORT WriteHex,Write,WriteLn,WriteCard;
 3   FROM SYSTEM IMPORT CODE,GETREG,AX;
 4   VAR
 5     estado:CARDINAL;
 6 BEGIN
 7   Write(14C);
 8   CODE(
 9         0B4H,02,  (*MOV AH,2*)
10         0BAH,0,0, (*MOV DX,0*)
11         0CDH,17H);(*INT 17h*)
12   CODE(
13         0B0H,00); (*MOV AL,0*)
14   GETREG(AX,estado);
15   WriteHex(estado,2); Write('H'); WriteLn;
16   WriteCard(estado,2);
17 END EstadoDeImpresora.
```

```
Modula-2/86 Compiler Version V 1.10

==>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

La utilización de esta interrupción de software, toma sentido en programas donde se necesita verificar el estado de la impresora antes de imprimir algún documento.

Un ejemplo de ésto podría ser la llamada al procedimiento 'TestImp', para determinar si una impresora está preparada o no.

```
PROCEDURE TestImp():BOOLEAN;   (*determina el estado de la impresora*)
  VAR
   verdad:BOOLEAN;
   estado:CARDINAL;
BEGIN                          (*estado=24576=6000H con impresora off*)
  CODE(                        (*estado=32768=8000H con impresora off line*)
       0B4H,02,  (*MOV AH,2H*) (*estado=36864=9000H con impresora on line*)
       0B4H,0,0, (*MOV DX,0H*)
       0CDH,17H);(*INT 17h*)
  CODE(
       0B0H,00); (*MOV AL,0H*)
  GETREG(AX,estado);
  IF estado = 36864 THEN verdad:=TRUE ELSE verdad:=FALSE END;
  RETURN verdad;
END TestImp;
```

Como se puede observar, se accede a todas las funciones del sistema operativo de forma similar. Sin embargo, M2 proporciona el procedimiento DOSCALL, para llamar a las funciones de alto nivel del sistema operativo de la interrupción 21H de una forma sencilla.

Su forma general es:

DOSCALL (nº de función, registros);

nº de función, es el número de la función DOS a ser ejecutada y registros, es una lista de valores a ser colocados en los registros adecuados para que se realice la función, o bien pueden ser variables donde se albergan los datos devueltos por los registros tras la ejecución de la función.

Por ejemplo, la función 0BH de la interrupción 21H, que sirve para determinar si se ha pulsado una tecla, devuelve en

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

AL el resultado de la operación, que podrá ser OFFH = TRUE o OOH = FALSE.

El siguinte programa, imprime asteriscos en la pantalla hasta que se pulse una tecla.

```
Modula-2/86                    tecla1.MOD

  1 MODULE tecla;
  2   FROM SYSTEM IMPORT DOSCALL;
  3   FROM InOut IMPORT Write;
  4
  5 PROCEDURE Pulsatecla():BOOLEAN;
  6   VAR
  7     Estado:BOOLEAN;
  8 BEGIN
  9   DOSCALL(OBH,Estado);
 10   RETURN Estado;
 11 END Pulsatecla;
 12
 13 BEGIN
 14   LOOP
 15     Write('*');
 16     IF Pulsatecla() THEN EXIT END;
 17   END;
 18 END tecla.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

El siguiente módulo, supone otro ejemplo de acceso a funciones de alto nivel del sistema operativo.

Este utiliza la interrupción 21H función 2AH, que sirve para obtener la fecha del sistema.

```
Modula-2/86                    fecha.MOD

  1  MODULE DeterminaFecha;
  2    FROM SYSTEM IMPORT DOSCALL;
  3    FROM InOut IMPORT WriteCard,WriteLn,Write,WriteString;
  4    VAR
  5      dia,mes,ano:CARDINAL;
  6
  7  PROCEDURE Fecha(VAR dia,mes,ano:CARDINAL);
  8    VAR
  9      mesdia,n:CARDINAL;
 10      ax,di:BITSET;
 11  BEGIN
 12    DOSCALL(2AH,ano,mesdia);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
13    me:=BITSET{};
14    di:=BITSET{};
15    FOR n:=15 TO 8 BY -1 DO
16      IF n IN (BITSET(mesdia)) THEN INCL(me,n-8) END;
17    END;
18    FOR n:=7 TO 0 BY -1 DO
19      IF n IN (BITSET(mesdia)) THEN INCL(di,n) END;
20    END;
21    dia:=CARDINAL(di);
22    mes:=CARDINAL(me);
23  END Fecha;
24
25  BEGIN
26    Write(14C);
27    Fecha(dia,mes,ano);
28    WriteString('La fecha actual es:');
29    WriteCard(dia,1); Write('-');
30    WriteCard(mes,1); Write('-');
31    WriteCard(ano,1); WriteLn;
32  END DeterminaFecha.
```

```
Modula-2/86 Compiler Version V 1.10

==>    0 Error(s) found
```

Hay que tener en cuenta, que el año, se obtiene en la primera variable del argumento de la función DOSCALL y que el mes y el día los dan el octeto más significativo y menos significativo respectivamente de la segunda variable, por lo que ha habido que manipular la variable 'mesdia', para visualizar el mes y el día por separado.

A continuación presentamos un último ejemplo, que realiza llamadas a INT 10H funciones 2H y 6H, INT 16H función 0H, INT 17H funciones 1H y 2H e INT 21H funciones 2AH y 4CH.

Conviene saber que la función 1H de INT 17H del BIOS inicializa un puerto de impresora y devuelve un byte indicando el estado de dicho puerto.

INT 17H función 1H: Inicializa puerto de impresora.

ENTRADA

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

AH = 1H
DX = Número de impresora a usar (0-2).

DEVUELVE
AH = Byte de estado de la impresora.

La función 4CH de INT 21H, permite devolver el control al MS-DOS.

INT 21H función 4CH:Salir al MS-DOS.

AH = 4CH
AL = Código de retorno.

Este programa permite seleccionar los tipos de letras de una impresora. En caso de que la impresora donde se utilice, tenga unos códigos de impresión específicos, que no coincidan con los de éste módulo, éstos, se podrían corregir haciendo unas ligeras rectificaciones en el procedimiento 'Programar'.

Modula-2/86 iba5.MOD

```
MODULE ParaImpresora; (* Permite programar la impresora *)
  FROM InOut IMPORT ReadString, WriteString, WriteLn, Read, Write, WriteCard;
  FROM FileSystem IMPORT File, Response, Lookup, SetWrite, SetRead,
       Close, ReadChar, WriteChar, Length, SetPos, WriteWord;
  FROM Util IMPORT PxPy, Cls, Cll, Vi, Vv, Neg, Bs, Cuu, Cud;
  FROM DiskDirectory IMPORT CurrentDrive, CurrentDirectory;
  FROM Directories IMPORT DirQuery, DirQueryProc, DirResult;
  FROM ASCII IMPORT esc, si, EOL, dc2, bel, nul, del, cr,lf;
  FROM Strings IMPORT CompareStr;
  FROM Terminal IMPORT KeyPressed;
  FROM Lineas IMPORT Line,LineD;
  FROM SYSTEM IMPORT GETREG, SETREG, AX, BX, CX, DX, CODE, DOSCALL, SWI;
  FROM NumberConversion IMPORT CardToString;
  VAR
    h,l,n, dia,mes,ano, px, x,y:CARDINAL;
    ch, hh, ww, unidad:CHAR;
    fnombre: ARRAY [0..46] OF CHAR;
    directorio: ARRAY [0..80] OF CHAR;
    bis: ARRAY [0..3] OF CHAR;
    Ocho, NLQ, Enfatiza, DobleI, Comprimido, Elite, Italica, ponerfecha:BOOLEAN;
    F1, F2:File;
    resultado: DirResult;
(* ------------------------------------------------------------------------------------------*)
  PROCEDURE Pantalla1;
    CONST
      cero = '8 lineas por pulgada';
      uno  = 'Alta calidad        ';
      dos  = 'Enfatizado          ';
      tres = 'Doble Impresión     ';
      cuatro = 'Comprimido          ';
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
31      cinco = 'ELITE                ';
32      seis = 'Cursiva              ';
33    TYPE
34      DelTipoProcedimiento=PROCEDURE;
35    VAR
36      contador:[0..6];
37      Cteclado:CARDINAL;
38
39    PROCEDURE Inspecciona():CARDINAL;
40      VAR
41        codigo:CARDINAL;
42    BEGIN
43      CODE (0B4H,0,  (*MOV AH,0*)
44            0CDH,16H);(*INT 16H*)
45      GETREG(AX,codigo);
46      RETURN codigo;
47    END Inspecciona;
48
49    PROCEDURE Poner(argumento:DelTipoProcedimiento);
50      BEGIN
51      CASE contador OF
52        0: PxPy(31,10); argumento; WriteString(cero); Vv;Bo; |
53        1: PxPy(31,11); argumento; WriteString(uno); Vv;Bo; |
54        2: PxPy(31,12); argumento; WriteString(dos); Vv;Bo; |
55        3: PxPy(31,13); argumento; WriteString(tres); Vv;Bo; |
56        4: PxPy(31,14); argumento; WriteString(cuatro); Vv;Bo;|
57        5: PxPy(31,15); argumento; WriteString(cinco); Vv;Bo;|
58        6: PxPy(31,16); argumento; WriteString(seis); Vv;Bo;
59      ELSE
60      END;
61    END Poner;
62
63    PROCEDURE PonerNormal;
64      BEGIN
65      CASE contador OF
66        0:IF Ocho = FALSE THEN Poner(Vv); ELSE Poner(Neg) END;|
67        1:IF NLQ = FALSE THEN Poner(Vv); ELSE Poner(Neg) END;|
68        2:IF Enfatiza = FALSE THEN Poner(Vv); ELSE Poner(Neg) END;|
69        3:IF DobleI = FALSE THEN Poner(Vv); ELSE Poner(Neg) END;|
70        4:IF Comprimido = FALSE THEN Poner(Vv); ELSE Poner(Neg) END;|
71        5:IF Elite = FALSE THEN Poner(Vv); ELSE Poner(Neg) END;|
72        6:IF Italica = FALSE THEN Poner(Vv); ELSE Poner(Neg) END;
73      ELSE
74      END;
75    END PonerNormal;
76
77  BEGIN
78    Ocho:=FALSE; NLQ:=FALSE; Enfatiza:=FALSE; DobleI:=FALSE; Comprimido:=FALSE;
79    Elite:=FALSE; Italica:=FALSE;
80    Cls;
81    PxPy(29,9);Neg;Write('┌');LineD(0,22);Write('┐');Cud(1);Bs(1);LineD(6,7);Write('┘');
82    Bs(2);LineD(4,22);Write('└');Cuu(1);Bs(1);LineD(2,7);Vv;
83    PxPy(31,10);  Vi;WriteString(cero);Vv;
84    PxPy(31,11);  WriteString(uno);
85    PxPy(31,12);  WriteString(dos);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 86      PxPy(31,13);  WriteString(tres);
 87      PxPy(31,14);  WriteString(cuatro);
 88      PxPy(31,15);  WriteString(cinco);
 89      PxPy(31,16);  WriteString(seis);
 90
 91      PxPy(19,20);Neg;Write('┌');LineD(0,42);Write('┐');Cud(1);Bs(1);LineD(6,1);Write('┘');
 92      Bs(2);LineD(4,42);Write('└');Cuu(1);Bs(1);LineD(2,1);Vv;
 93      PxPy(20,21);Neg;Write(30C);Write(31C);Write(' ');Write(21C);Write('-');Write('┘');
 94      Write(':');Vv;WriteString('Seleccionar; ');
 95      Neg;WriteString('F1:');Vv;WriteString('Ejecutar; ');
 96      Neg;WriteString('Esc:');Vv;WriteString('Salir');Bo;
 97      contador:=0;
 98      REPEAT
 99        Cteclado:= Inspecciona();
100        CASE Cteclado OF
101    ($↑) 4800H: PonerNormal; IF contador = 0 THEN contador:=6 ELSE DEC(contador) END;
102                Poner(Vi);|
103    ($↓) 5000H: PonerNormal; IF contador = 6 THEN contador:=0 ELSE INC(contador) END;
104                Poner(Vi);|
105   ($CR) 1C0DH: CASE contador OF
106                  0: IF Ocho=FALSE THEN Ocho:=TRUE; Poner(Neg); INC(contador); Poner(Vi)
107                     ELSE Ocho := FALSE; Poner(Vv); INC(contador); Poner(Vi) END|
108
109                  1: IF NLQ = FALSE THEN NLQ := TRUE; Poner(Neg); INC(contador); Poner(Vi)
110                     ELSE NLQ := FALSE; Poner(Vv); INC(contador); Poner(Vi) END|
111
112                  2: IF Enfatiza = FALSE THEN Enfatiza := TRUE; Poner(Neg); INC(contador); Poner(Vi)
113                     ELSE Enfatiza := FALSE; Poner(Vv); INC(contador); Poner(Vi) END|
114
115                  3: IF DobleI = FALSE THEN DobleI := TRUE; Poner(Neg); INC(contador); Poner(Vi)
116                     ELSE DobleI := FALSE; Poner(Vv); INC(contador); Poner(Vi) END|
117
118                  4: IF Comprimido = FALSE THEN Comprimido := TRUE; Poner(Neg); INC(contador); Poner(Vi)
119                     ELSE Comprimido := FALSE; Poner(Vv); INC(contador); Poner(Vi) END|
120
121                  5: IF Elite = FALSE THEN Elite := TRUE; Poner(Neg); INC(contador); Poner(Vi)
122                     ELSE Elite := FALSE; Poner(Vv); INC(contador); Poner(Vi) END|
123
124                  6: IF Italica = FALSE THEN Italica := TRUE; Poner(Neg); contador:=0; Poner(Vi)
125                     ELSE Italica := FALSE; Poner(Vv); contador:=0; Poner(Vi) END
126                ELSE
127                END;|
128  ($ESC) 011BH: Quit;
129        ELSE
130        END;
131      UNTIL (Cteclado =3B00H)($F1);
132      Cls;
133    END Pantalla1;
134  (* ------------------------------------------------------------------------------------------------*)
135  PROCEDURE Programar;
136    BEGIN
137      WriteChar(F2,bel);(* Inicializar la impresora *)
138      WriteChar(F2,esc); WriteChar(F2,'@');
139      IF Ocho = TRUE THEN WriteChar(F2,esc); WriteChar(F2,'0') ELSE END;
140      IF NLQ = TRUE THEN WriteChar(F2,esc); WriteChar(F2,'4'); ELSE END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
141     IF Enfatiza = TRUE THEN WriteChar(F2,esc); WriteChar(F2,'E') ELSE END;
142     IF DobleI = TRUE THEN WriteChar(F2,esc); WriteChar(F2,'6') ELSE END;
143     IF Comprimido = TRUE THEN WriteChar(F2,esc); WriteChar(F2,si) ELSE END;
144     IF Elite = TRUE THEN WriteChar(F2,esc); WriteChar(F2,'M') ELSE END;
145     IF Italica = TRUE THEN WriteChar(F2,esc); WriteChar(F2,'I');WriteChar(F2,'1') ELSE END;
146  END Programar;
147  (* ------------------------------------------------------------------------------------*)
148    PROCEDURE Pantalla2;
149      BEGIN
150        Cls;
151        PxPy(18,5); Neg; WriteString('¿Qué porción de fichero quiere imprimir?: ');Vv;
152        PxPy(27,9);Neg;Write('┌');Line(0,22);Write('┐');Cud(1);Bs(1);Line(6,12);Write('┘');
153        Bs(2);Line(4,22);Write('└');Cuu(1);Bs(1);Line (2,12);Vv;
154        PxPy(27,20); Neg;Write('├');Line(0,22);Write('┤');Vv;
155
156        PxPy(28,10); Vi; WriteString(' 0: desde  0 al  25 % '); Vv;
157        PxPy(28,11); Vi; WriteString(' 1: desde  0 al  50 % '); Vv;
158        PxPy(28,12); Vi; WriteString(' 2: desde  0 al  75 % '); Vv;
159        PxPy(28,13); Vi; WriteString(' 3: desde  0 al 100 % '); Vv;
160        PxPy(28,14); Vi; WriteString(' 4: desde 25 al  50 % '); Vv;
161        PxPy(28,15); Vi; WriteString(' 5: desde 25 al  75 % '); Vv;
162        PxPy(28,16); Vi; WriteString(' 6: desde 25 al 100 % '); Vv;
163        PxPy(28,17); Vi; WriteString(' 7: desde 50 al  75 % '); Vv;
164        PxPy(28,18); Vi; WriteString(' 8: desde 50 al 100 % '); Vv;
165        PxPy(28,19); Vi; WriteString(' 9: desde 75 al 100 % '); Vv;
166        PxPy(28,21); Neg;WriteString('Introduzca su elección'); Vv;Bo;
167    END Pantalla2;
168  (* ------------------------------------------------------------------------------------*)
169    PROCEDURE Abrir;
170      BEGIN
171        SetRead(F1);
172        SetWrite(F2);
173    END Abrir;
174  (* ------------------------------------------------------------------*)
175    PROCEDURE Cerrar;
176      BEGIN
177        Close(F1);
178        Close(F2);
179    END Cerrar;
180  (* ------------------------------------------------------------------*)
181    PROCEDURE ResetPrn; (* Inicializa puerto de impresora *)
182      BEGIN
183        SETREG (AX,0100H);
184        SETREG (DX,0H);
185        CODE (55H);
186        SWI(17H);
187        CODE (5DH);
188    END ResetPrn;
189  (* ------------------------------------------------------------------*)
190    PROCEDURE Quit;      (* Devuelve al MS-DOS *)
191      BEGIN
192        Cls;
193        DOSCALL(4CH,0FFH);
194    END Quit;
195  (* ------------------------------------------------------------------*)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
196   PROCEDURE Bo; (* Borra el cursor *)
197     BEGIN
198       SETREG (DX,1900H);
199       SETREG (AX,0200H);
200       SETREG (BX,0H);
201       CODE (55H);
202       SWI(10H);
203       CODE (5DH);
204   END Bo;
205  (* ---------------------------------------------------------------*)
206  PROCEDURE Fecha(VAR dia,mes,ano:CARDINAL);
207   VAR
208     mesdia,n:CARDINAL;
209     me,di:BITSET;
210  BEGIN
211   DOSCALL(2AH,ano,mesdia);
212   me:=BITSET{};
213   di:=BITSET{};
214   FOR n:=15 TO 8 BY -1 DO
215     IF n IN (BITSET(mesdia)) THEN INCL(me,n-8) END;
216   END;
217   FOR n:=7 TO 0 BY -1 DO
218     IF n IN (BITSET(mesdia)) THEN INCL(di,n) END;
219   END;
220   dia:=CARDINAL(di);
221   mes:=CARDINAL(me);
222  END Fecha;
223  (* ---------------------------------------------------------------*)
224  PROCEDURE PreguntarFecha;
225   VAR
226     SN:CHAR;
227  BEGIN
228   Cls;
229   PxPy(12,11);Neg;Write('┌');LineD(0,56);Write('┐');Cud(1);Bs(1);LineD(6,1);Write('┘');
230   Bs(2);LineD(4,56);Write('└');Cuu(1);Bs(1);LineD(2,1);Vv;
231   PxPy(13,12);
232   WriteString('¿Quiere poner la fecha ');Write('0');Bs(1);
233   Fecha(dia,mes,ano);
234   IF dia>9 THEN
235     WriteCard(dia,2); Write('-');
236   ELSE
237     Write('0');WriteCard(dia,1); Write('-');
238   END;
239   IF mes>9 THEN
240     WriteCard(mes,2); Write('-');
241   ELSE
242     Write('0');WriteCard(mes,1); Write('-');
243   END;
244   WriteCard(ano,4);
245   WriteString(' en el documento? (S/N)');Bo;
246   SN:=0C;
247   REPEAT
248    Read(SN)
249   UNTIL (CAP(SN)='S') OR (CAP(SN)='N') OR (SN=esc) OR (SN=EOL);
250
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
251   IF CAP(SN)='S' THEN
252     ponerfecha:=TRUE;
253   ELSIF (CAP(SN)='N')  OR (CAP(SN)=EOL) THEN
254     ponerfecha:=FALSE;
255   ELSE
256    Quit;
257   END;
258 END PreguntarFecha;
259 (* ---------------------------------------------------------------*)
260 PROCEDURE MesyDia;
261 BEGIN
262   IF dia>9 THEN
263     CardToString(dia,bis,2);
264     FOR px:=0 TO 1 DO WriteChar(F2,bis[px]) END; WriteChar(F2,'-');
265   ELSE
266     CardToString(dia,bis,1);
267     WriteChar(F2,'0');WriteChar(F2,bis[0]);WriteChar(F2,'-');
268   END;
269   IF mes>9 THEN
270     CardToString(mes,bis,2);
271     FOR px:=0 TO 1 DO WriteChar(F2,bis[px]) END; WriteChar(F2,'-');
272   ELSE
273     CardToString(mes,bis,1);
274     WriteChar(F2,'0');WriteChar(F2,bis[0]);WriteChar(F2,'-');
275   END;
276 END MesyDia;
277 (* ---------------------------------------------------------------*)
278 PROCEDURE Cero(x:REAL);
279 VAR
280   salir:CHAR;
281   Puntero,Posicion:REAL;
282 BEGIN
283   PreguntarFecha;
284   Abrir;Programar;
285   IF ponerfecha=TRUE THEN
286     MesyDia;
287     CardToString(ano,bis,4);
288     FOR px:=0 TO 3 DO WriteChar(F2,bis[px]) END;
289     WriteChar(F2,cr); WriteChar(F2,lf); WriteChar(F2,lf);
290   ELSE END;
291   Length(F1,h,l);
292   Posicion:=(FLOAT(h)*2.E16 + FLOAT (l)) * x;
293   Puntero:=0.;
294   REPEAT
295     IF KeyPressed() THEN
296       Read(salir);
297       IF salir=esc THEN
298        Cerrar;ResetPrn;Quit;
299       END
300     END;
301     ReadChar(F1,ch);
302     IF ch#CHR(0) THEN WriteChar(F2,ch) END;
303     Puntero:=Puntero + 1.;
304   UNTIL (F1.eof) OR (F2.res = eom) OR (Puntero >= Posicion);
305   (*inicializar impresora y cerrar ficheros*)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
306   WriteChar(F2,esc); WriteChar(F2,'@');Cerrar
307 END Cero;
308 (* ------------------------------------------------------------*)
309 PROCEDURE Uno(x,y:REAL);
310 VAR
311   salir:CHAR;
312   Final,Posicion:REAL;
313 BEGIN
314   PreguntarFecha;
315   Abrir;Programar;
316   IF ponerfecha=TRUE THEN
317     MesyDia;
318     CardToString(ano,bis,4);
319     FOR px:=0 TO 3 DO WriteChar(F2,bis[px]) END;
320     WriteChar(F2,cr); WriteChar(F2,lf); WriteChar(F2,lf);
321   ELSE END;
322   Length(F1,h,l);
323   Posicion:=(FLOAT(h)*2.E16 + FLOAT (l)) * x;
324   Final:=(FLOAT(h)*2.E16 + FLOAT (l)) * y;
325   IF Posicion >65535.
326      THEN
327        h:=TRUNC((Posicion-65535.)/2.E16);
328        l:=65535
329      ELSE
330        h:=0;
331        l:=TRUNC(Posicion);
332   END;
333   SetPos(F1,h,l);
334   REPEAT
335     IF KeyPressed() THEN
336       Read(salir);
337       IF salir=esc THEN
338         Cerrar;ResetPrn;Quit;
339       END;
340     END;
341     ReadChar(F1,ch);
342     IF ch#CHR(0) THEN WriteChar(F2,ch) END;
343     Posicion:=Posicion + 1.;
344   UNTIL (F1.eof) OR (F2.res = eom) OR (Posicion >= Final);
345   (*inicializar impresora y cerrar ficheros*)
346   WriteChar(F2,esc); WriteChar(F2,'@');Cerrar
347 END Uno;
348 (*-------------------------------------------------------------------------*)
349   PROCEDURE gets (VAR a:ARRAY OF CHAR);
350     CONST
351     BS = 8;
352     VAR
353       ca : CHAR;
354       i: CARDINAL;
355     BEGIN
356      i:=0;
357      REPEAT
358        LOOP
359          Read (ca);
360          IF (ORD (ca)>31) AND (ORD (ca)<127) OR (ORD (ca)=8) AND (i#0)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
361           OR (ORD (ca)=30) OR (ORD (ca)=27)
362           OR (ORD (ca)>159) AND (ORD (ca)<169) OR (ORD (ca)=130) OR (ORD(ca)=129)
363           THEN EXIT END;
364         END;
365         IF ca # esc THEN
366           IF ca # EOL THEN Write (ca) END;
367           IF ORD(ca)=BS THEN
368             i:=i-1; Write (40C); Write (177C)
369           ELSIF (ca#EOL) AND (i<HIGH(a)) THEN
370             a[i]:=ca;
371             i:=i+1;
372           END;
373         ELSE Quit END;
374       UNTIL (ca=EOL) OR (i=HIGH(a));
375       a[i]:=CHR(0);
376     END gets;
377   (*-------------------------------------------------------------------------*)
378   PROCEDURE TestImp():BOOLEAN;(*determina si la impresora está preparada*)
379   VAR
380    verdad:BOOLEAN;
381    estado:CARDINAL;
382   BEGIN                              (*estado=24576=6000H con impresora off*)
383     CODE(                            (*estado=32768=8000H con impresora off line*)
384           0B4H,02,  (*MOV AH,2*)     (*estado=36864=9000H con impresora on line*)
385           0BAH,0,0, (*MOV DX,0*)
386           0CDH,17H);(*INT 17h*).
387     CODE(
388           0B0H,00); (*MOV AL,0*)
389
390     GETREG(AX,estado);
391     IF estado = 36864 THEN verdad:=TRUE ELSE verdad:=FALSE END;
392     RETURN verdad;
393   END TestImp;
394   (*-------------------------------------------------------------------------*)
395   PROCEDURE Dir (fichero: ARRAY OF CHAR; VAR VF:BOOLEAN);
396   VAR
397     pausa:CHAR;
398
399   PROCEDURE Ventana;
400   BEGIN
401     SETREG (CX,0A00H);
402     SETREG (DX,1854H);
403     SETREG (AX,600H);
404     SETREG (BX,700H);
405     CODE(55H);
406     SWI(10H);
407     CODE(5DH);
408   END Ventana;
409
410   BEGIN
411     VF:=TRUE;
412     IF (x=1) AND (y=11) THEN Ventana END;
413     PxPy(x,y); WriteString(fichero); x:=(x MOD 69) + 17;
414     IF x=17 THEN
415       x:=1;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
416       INC(y);
417       IF y=25 THEN
418         PxPy(18,8);
419         Read(pausa);
420         IF pausa = esc THEN Quit ELSE END;
421         x:=1; y:=11
422       END
423     END;
424   END Dir;
425   (*-------------------------------------------------------------*)
426   BEGIN
427     Cls;
428     Pantalla1;
429     Cls;PxPy(19,5);Neg;
430     WriteString('Introduzca el nombre del fichero a imprimir: ');Vv;
431     PxPy(17,7);Neg;Write('┌');LineD(0,46);Write('┐');Cud(1);Bs(1);LineD(6,1);Write('┘');
432     Bs(2);LineD(4,46);Write('└');Cuu(1);Bs(1);LineD(2,1);Vv;
433
434     REPEAT
435       fnombre:='';
436       PxPy(18,8);
437       gets(fnombre);
438       Lookup(F1,fnombre,FALSE);
439       IF (F1.res = notdone) AND (CompareStr(fnombre,'') # 0) THEN
440          PxPy(64,8);FOR n:=0 TO 45 DO Write(del) END;
441          PxPy(18,5);Neg;
442          WriteString('Fichero no encontrado, Introduzca otro nombre: ');Vv;
443          x:=1; y:=11;
444          CurrentDrive(unidad);CurrentDirectory(unidad,directorio);
445          PxPy(0,10); Vi; Write(unidad);WriteString(':\');WriteString(directorio);Vv;
446          DirQuery('*.*',Dir,resultado);
447       END
448     UNTIL (F1.res = done) OR  (CompareStr(fnombre,'') = 0);
449
450     Lookup(F2,'prn',TRUE);
451     Cls;
452     PxPy(19,12);Neg;Write('┌');LineD(0,38);Write('┐');Cud(1);Bs(1);LineD(6,1);Write('┘');
453     Bs(2);LineD(4,38);Write('└');Cuu(1);Bs(1);LineD(2,1);Vv;
454     REPEAT
455       PxPy(20,13); Write(bel);WriteString('Prepare la impresora y pulse ');
456       WriteString('Enter ');Write(21C);Write('-');Write('┘');Bo;
457       REPEAT
458         hh:=0C;
459         Read(hh);
460       UNTIL (hh = EOL) OR (hh =esc);
461     UNTIL (TestImp()) OR (hh=esc);
462     IF hh = esc THEN Quit END;
463     IF CompareStr(fnombre,'') = 0 THEN SetWrite(F2); Programar;Close(F2);
464     ELSE
465       Pantalla2;
466       REPEAT
467         Read(ww);
468         CASE ww OF
469           '0':Cero(0.25)| (* 0 - 25% *)
470            '1':Cero(0.50)| (* 0 - 50% *)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
471           '2':Cero(0.75)| (* 0 - 75% *)
472           '3':Cero(1.)|   (* 0 - 100% *)
473           '4':Uno(0.25,0.50)| (* 25 - 50% *)
474           '5':Uno(0.25,0.75)| (* 25 - 75% *)
475           '6':Uno(0.25,1.)|   (* 25 - 100% *)
476           '7':Uno(0.50,0.75)| (* 50 - 75% *)
477           '8':Uno(0.50,1.)|   (* 50 - 100% *)
478           '9':Uno(0.75,1.)|   (* 75 - 100% *)
479           EOL:Cero(1.)        (* 0 - 100% *)
480         ELSE
481         END;
482       UNTIL (ORD(ww)>=48) AND (ORD(ww)<=57) OR (ww=EOL) OR (ww=esc);
483     END;
484     Cls;
485   END ParaImpresora.
```

```
-------------------------------------------

Modula-2/86 Compiler Version V 1.10

==>     0 Error(s) found
```

11.4 MANIPULADORES DE INTERRUPCIONES

Se llama interrupción a una señal que hace que la unidad central del ordenador, suspenda la tarea que está realizando y transfiera el control a una subrutina especial llamada manipulador de interrupciones.

La familia 8086/8088 soporta 256 [0..FFH] niveles de interrupciones. Desde 0 a 1FH, corresponden a interrupciones de hardware y de los niveles 20H a 3FH a interrupciones software.

Las interrupciones de hardware, son aquellas cuyo comienzo está condicionado por hardware, su activación puede deberse a la pulsación de una tecla por parte del usuario, o bien cuando ocurre un condición excepcional durante el proceso de las instrucciones, intento de división por cero,

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

overflow, etc. .

Las interrupciones de software, son rutinas integradas en el sistema operativo, que utilizan el mecanismo de las interrupciones para acceder a ellas.

Para cada interrupción, existe una posición reservada en la memoria, llamada 'vector de interrupción', que especifica la posición donde se encuentra el programa manipulador de este tipo de interrupción.(La dirección del vector de interrupción correspondiente a la INT n, es: 4 x n, puesto que cada vector de interrupción ocupa 4 bytes).

El vector para la INT 5 está en (5 x 4=14H) 0000:0014H. Los cuatro bytes que comienzan en esta dirección, contienen la dirección de comienzo en ROM de la subrutina que ejecuta la INT 5.

El contenido de un vector de interrupción (al residir en RAM) puede modificarse para que, en vez de apuntar a una rutina predefinida en ROM o por el sistema operativo, lo haga a un manipulador de interrupciones creado por el propio programador.

El programa que prepara un manipulador de interrupciones debe:

1) Inicializar el vector para la interrupción de que se trate, haciéndole apuntar al nuevo manipulador de interrupciones.

Esto se puede realizar con la función 25H de la INT 21H donde:

AH = 25H.
AL = Nº de la interrupción.
DS:DX = Segment:Offset de la rutina de manejo de la INT.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

que se puede llamar con:

```
DOSCALL(25H; Vector:ADDRESS; NºVector:BYTEWORD);
        AH   DS:DX                AL
```

o bien con el procedimiento:

```
RestoreInterruptVector(NºVector:CARDINAL; Vector:ADDRESS);
```

importado del módulo Devices, que hace que el vector de interrupciones de NºVector, apunte a la dirección Vector.

2) Antes de que el programa finalice, se debe devolver el vector de interrupción a su estado original, en caso contrario, un programa subsiguiente puede llamar a la interrupción y saltar a un lugar de la memoria en el que la subrutina ya no está.

Esto también se puede realizar con la función 25H de la INT 21H o con RestoreInterruptVector, pero en este caso, Vector es la dirección a la que apuntaba originalmente el vector de interrupción y que previamente se había salvado antes de la instalación del nuevo manipulador de interrupciones, con la función 35H de la INT 21H donde:

```
AH =35H.
AL = Nº de la interrupción.

DEVUELVE
  ES:BX =Segment:Offset del manipulador de la interrupción.
```

que se puede llamar con:

```
DOSCALL(35H; NºVector:BYTEWORD; VAR Vector:ADDRESS);
        AH       AL                     ES:BX
```

o bien con:

```
SaveInterruptVector(NºVector:CARDINAL; VAR Vector:ADDRESS);
```

también proporcionado por el módulo Devices, cuya misión es la misma, guardar el valor actual de un vector de interrupción.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

El propio manipulador de interrupción debe:

1) Guardar el contexto del sistema (registros, flags y cualquier otra cosa que pueda modificar el manipulador y que no sea almacenado automáticamente por la CPU).

2) Ejercer la acción apropiada para la interrupción.

3) Recuperar el contexto del sistema.

4) Recuperar la ejecución del proceso interrumpido.

A continuación se muestra un programa ejemplo que escribe asteriscos en pantalla y que reprograma la INT 5.

Hay que tener en cuenta que, cuando se pulsa la combinación de teclas Shift + PrtSc, se produce una interrupción de teclado que cede el control a la subrutina apuntada por el vector de la INT 5, cuya función, es volcar el contenido de la pantalla por impresora.

Sin embargo, se puede hacer apuntar este vector a un manipulador de interrupciones que nos realice una función diferente a la predefinida, en nuestro caso, será un simple retardo.

Modula-2/86 gestor.MOD

```
1  MODULE GestorDeInterrupciones;
2    FROM Util IMPORT Cls,PutPy;
3    FROM InOut IMPORT Read,Write;
4    FROM Terminal IMPORT KeyPressed;
5    FROM SYSTEM IMPORT CODE,ADDRESS,OUTBYTE,ENABLE,DOSCALL;
6    FROM Devices IMPORT SaveInterruptVector, RestoreInterruptVector;
7    VAR
8      z:CARDINAL;
9      ch:CHAR;
10     VectorViejoDeInterrupcion: ADDRESS;
11 (*------------------------------------------*)
12   PROCEDURE Aa; (*Retardo*)
13   VAR
14    x,y:CARDINAL;
15   BEGIN
16       FOR x:=0 TO 3 DO
17         FOR y:=0 TO 65535 DO END
18       END;
19   END Aa;
20 (*------------------------------------------*)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
21 PROCEDURE Mi; (*Manipulador de la interrupción*)
22 BEGIN
23   ENABLE;
24              (*PUSHF Generado por la INT*)
25              (*PUSH CS Generado por la INT*)
26              (*PUSH IP Generado por la INT*)
27              (*PUSH BP Generado por el compilador*)
28   CODE(50H); (*PUSH AX*)
29   CODE(51H); (*PUSH CX*)
30   CODE(52H); (*PUSH DX*)
31   CODE(53H); (*PUSH BX*)
32   CODE(56H); (*PUSH SI*)
33   CODE(57H); (*PUSH DI*)
34   CODE(1EH); (*PUSH DS*)
35   CODE(06H); (*PUSH ES*)
36
37   Aa;
38
39   CODE(07H); (*POP ES*)
40   CODE(1FH); (*POP DS*)
41   CODE(5FH); (*POP DI*)
42   CODE(5EH); (*POP SI*)
43   CODE(5BH); (*POP BX*)
44   CODE(59H); (*POP DX*)
45   CODE(58H); (*POP CX*)
46   CODE(58H); (*POP AX*)
47   CODE(5DH); (*POP BP*)
48
49   OUTBYTE(20H,20H); (*EOI --> 8259*)
50
51   CODE(0CFH);    (*IRET*)
52 END Mi;
53 (* ------------------------------------------------------------*)
54   BEGIN
55   (*DOSCALL(35H,5,VectorViejaDeInterrupcion);*)
56     SaveInterruptVector(5H,VectorViejaDeInterrupcion);
57
58   (*DOSCALL(25H,ADDRESS(Mi),5);*)
59     RestoreInterruptVector(5H,ADDRESS(Mi));
60
61     Cls; PxPy(0,10); ch:=0C; z:=0;
62
63     REPEAT
64       IF KeyPressed() THEN Read(ch) END;
65       Write('*');Write('-');
66       INC(z,2); IF z >= 1200 THEN z:=0; Cls; PxPy(0,10); END;
67     UNTIL ch=33C;
68
69   (*DOSCALL(25H,VectorViejaDeInterrupcion,5);*)
70     RestoreInterruptVector(5H,VectorViejaDeInterrupcion);
71
72 END GestorDeInterrupciones.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Hay ciertos aspectos respecto al manipulador de interrupciones de este módulo que conviene destacar.

Cuando la CPU detecta una interrupción, guarda el registro de los flags de estado, el registro de segmento de código (CS) y el puntero de instrucciones (IP) en la pila de la máquina, por lo que cualquier rutina de interrupción debe finalizar con IRET, instrucción que recupera de la pila el estado original de estos tres registros, devolviendo el control al lugar del programa donde se había producido la interrupción.

También con la interrupción se desactiva el sistema de interrupciones por lo tanto, será misión del manipulador activar inmediatamente el sistema de interrupciones por medio del procedimiento ENABLE, que equivale a la instrucción de ensamblador STI, para permitir que aparezcan otras de mayor prioridad dentro del procesado del manipulador.

Si además se trata de una interrupción de hardware, es decir, que se canaliza a través del PIC (controlador programable de interrupciones) 8259, se debe enviar un código especial (20H) llamado EOI (end of interrupt) a través del puerto (20H) que controla el registro de interrupciones en activo del propio PIC, con el fin de indicarle que se ha completado el proceso de la interrupción.

Al producirse una interrupción, el compilador genera un PUSH BP, por lo tanto, en el momento de recuperar el contexto del sistema, se debe incluir un POP BP.

El MS-DOS, en sus versiones actuales, es un sistema 'no reentrante', por lo que en ningún caso, se puede hacer

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

llamadas a funciones MS-DOS de E/S desde un manipulador de interrupciones hardware, durante el proceso real de la interrupción.

A continuación, se presenta el programa 'residir.MOD', que permite dejar residente en memoria (por medio de la función 31H de la INT 21H) un manipulador para la INT 5. Una vez instalado, puede ser llamado desde cualquier otro programa o desde el sistema operativo.

En el caso de que se ceda el control al manipulador (pulsando Shift + PrtSc), éste pregunta: '¿Continúa / Termina / Imprime?'. Si el usuario pulsa 'c' o 'C' para continuar, el manipulador devuelve todos los registros y realiza una operación IRET, cediendo nuevamente el control a la aplicación original. Por el contrario, si se pulsa 't' o 'T', el manipulador devuelve el control al MS-DOS mediante la función 4CH de la INT 21H. En el caso de 'i' o 'I', se imprimirá la información alfanumérica de la pantalla.

Debemos saber que:

```
INT 21H función 31H: Terminar y permanecer residente.
 AH = 31H.
 AL = código de retorno.
 DX = Párrafos de memoria (de 16 bits) a reservar.

INT 10H función 3H: Lee la posición del cursor.
 AH = 03H.
 BH = Nº de página.
DEVUELVE
 DH = Fila.
 DL = Columna.

INT 10H función 9H: Escribe carácter y atributo en la
                     posición del cursor.
 AH = 09H.
 AL = Código ASCII.
 BH = Nº de página.
 CX = Contador de caracteres a escribir.
 BL = Atributo.
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

ATRIBUTO	CODIGO
normal	07H
intenso	0FH
normal subrayado	01H
inteso subrayado	09H
inverso	70H
titilar normal	87H
titilar intenso	8FH
titilar normal subrayado	81H
titilar intenso subrayado	89H
titilar inverso	F0H

Modula-2/86 residir.MOD

```
 1 MODULE GestorDeInterrupcionesResidente;
 2   FROM InOut IMPORT WriteString,Write;
 3   FROM Util IMPORT Cls,PxPy,Neg,Tilt,Vv;
 4   FROM Devices IMPORT SaveInterruptVector, RestoreInterruptVector;
 5   FROM SYSTEM IMPORT CODE,ADDRESS,OUTBYTE,ENABLE,DOSCALL,BX,CX,DX,
 6                      AX,SETREG,GETREG,SWI;
 7 VAR
 8   VectorViejoDeInterrupcion:ADDRESS;
 9   posicion,codigo:CARDINAL;
10 (* ------------------------------------------------------------ *)
11 PROCEDURE Mi; (*Manipulador de la interrupción*)
12
13 (*---Posiciona el cursor---*)
14 PROCEDURE C (Posicion:CARDINAL);
15 BEGIN
16   SETREG(AX,0200H);
17   SETREG(BX,0000H);
18   SETREG(DX,Posicion);
19   CODE(55H);
20   SWI(10H);
21   CODE(5DH);
22 END C;
23
24 (*-Escribe carácter y atributo en la posición del cursor-*)
25 PROCEDURE W (Caracter,Atributo,mul:CARDINAL);
26 BEGIN
27   SETREG(AX,Caracter);
28   SETREG(BX,Atributo);
29   SETREG(CX,mul);
30   CODE(55H);
31   SWI(10H);
32   CODE(5DH);
33 END W;
34
35 BEGIN
36   ENABLE;
37   CODE(50H); (*PUSH AX*)
38   CODE(51H); (*PUSH CX*)
39   CODE(52H); (*PUSH DX*)
40   CODE(53H); (*PUSH BX*)
41   CODE(56H); (*PUSH SI*)
42   CODE(57H); (*PUSH DI*)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
43   CODE(1EH); {PUSH DS}
44   CODE(06H); {PUSH ES}
45
46   {-Guarda posición del cursor-}
47   SETREG(AX,0300H);
48   SETREG(BX,0);
49   CODE(55H);
50   SWI(10H);
51   CODE(5DH);
52   GETREG(DX,posicion);
53   {--------Escribe mensaje '¿Continúa/Termina/Imprime?'--------}
54   C(001AH); W(094EH,0007H,1); C(001BH); W(0943H,000FH,1);
55   C(001CH); W(096FH,0007H,1); C(001DH); W(096EH,0007H,1);
56   C(001EH); W(0974H,0007H,1); C(001FH); W(0969H,0007H,1);
57   C(0020H); W(096EH,0007H,1); C(0021H); W(0983H,0007H,1);
58   C(0022H); W(0961H,0007H,1); C(0023H); W(092FH,0007H,1);
59
60   C(0024H); W(0954H,000FH,1); C(0025H); W(0965H,0007H,1);
61   C(0026H); W(0972H,0007H,1); C(0027H); W(096DH,0007H,1);
62   C(0028H); W(0969H,0007H,1); C(0029H); W(096EH,0007H,1);
63   C(002AH); W(0961H,0007H,1); C(002BH); W(092FH,0007H,1);
64
65   C(002CH); W(0949H,000FH,1); C(002DH); W(0960H,0007H,1);
66   C(002EH); W(0970H,0007H,1); C(002FH); W(0972H,0007H,1);
67   C(0030H); W(0969H,0007H,1); C(0031H); W(096DH,0007H,1);
68   C(0032H); W(0965H,0007H,1); C(0033H); W(093FH,0007H,1); C(1900H);
69
70   REPEAT
71     CODE(0B4H,0);{MOV AH,0}{lee carácter del teclado}
72     CODE(55H);
73     SWI(16H);
74     CODE(5DH);
75     GETREG(AX,codigo);
76   UNTIL (c,C,t,T,i,I) (codigo =2E63H) OR (codigo=2E43H)
77                       OR (codigo=1474H) OR (codigo=1454H)
78                       OR (codigo=1766H) OR (codigo=1746H);
79
80   C(001AH); W(0920H,0007H,1AH);{Se borra el mensaje}
81
82   IF (codigo=1474H) OR (codigo =1454H) THEN
83     C(0000H); W(0920H,0007H,2000); DOSCALL(4CH,0FFH)
84   ELSIF (codigo=1766H) OR (codigo=1746H) THEN
85     RestoreInterruptVector(5H,VectorViejoDeInterrupcion);
86     CODE(55H);
87     SWI(5H);
88     CODE(5DH);
89     RestoreInterruptVector(5H,ADDRESS(Mi));
90   ELSE
91   END;
92
93   {-Repone posicion del cursor-}
94   SETREG(AX,0200H);
95   SETREG(BX,0);
96   SETREG(DX,posicion);
97   CODE(55H);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 98    SWI(10H);
 99    CODE(58H);
100
101    CODE(07H); (*POP ES*)
102    CODE(1FH); (*POP DS*)
103    CODE(5FH); (*POP DI*)
104    CODE(5EH); (*POP SI*)
105    CODE(5BH); (*POP BX*)
106    CODE(5AH); (*POP DX*)
107    CODE(59H); (*POP CX*)
108    CODE(58H); (*POP AX*)
109    CODE(5DH); (*POP BP*)
110    OUTBYTE(20H,20H);(*EOI -->8259*)
111    CODE(0CFH);(*IRET*)
112  END Mi;
113  (* ------------------------------------------------------------ *)
114    BEGIN
115      SaveInterruptVector(5H,VectorViejoDeInterrupcion);
116      RestoreInterruptVector(5H,ADDRESS(Mi));
117      Cls;PxPy(17,12); Neg; Tilt;
118      WriteString('Manejador de interrupciones de PrtSc instalado'); Vv; PxPy(0,24);
119      DOSCALL(31H,0,3000); (*Termina y permanece residente*)
120  END GestorDeInterrupcionesResidente.
```

```
Modula-2/86 Compiler Version V 2.00

=>     0 Error(s) found
```

11.5 LISTA DE INTERRUPCIONES

A continuación se presenta un conjunto parcial de rutinas tanto del BIOS, como del DOS, a las que se puede acceder a través de interrupciones, junto con una tabla, siempre útil, de códigos ASCII.

Una lista completa del sistema, ampliamente documentada, está disponible en los manuales de referencia técnica de IBM.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Interrupción 10H: Realiza funciones del BIOS de E/S de video.

Función 02H: Posiciona el cursor.

Se llama con: AH = 02H.
DH = Fila.
DL = Columna.
BH = Nº de página de video.

Función 03H: Lee posición del cursor.

Se llama con: AH = 03H.
BH = Nº de página de video.

Devuelve: DH = Fila.
DL = Columna.

Función 06H: Desplaza el contenido de una ventana en sentido ascendente un número determinado de lineas.

Se llama con: AH = 06H.
AL = Nº de lineas a desplazar (0 para todas).
CH = Fila de la esquina superior izquierda de la ventana.
CL = Columna de la esquina superior izquierda de la ventana.
DH = Fila de la esquina inferior derecha de la ventana.
DL = Columna de la esquina inferior derecha de la ventana.
BH = Atributo de la zona que se borra.
Normal = 07H
Video inverso = 70H
Subrayado = 01H
Invisible (negro) = 00H
Invisible (blanco) = 77H

Función 07H: Desplaza el contenido de una ventana en sentido descendente un número determinado de lineas.

Se llama con: Igual que en la función 06H y AH = 07H.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Función 08H: Lee carácter y atributo en la posición del cursor.

Se llama con: AH = 08H.
BH = Nº de página de video.

Devuelve: AL = Código ASCII del carácter.
AH = Byte de atributo.

Normal	07H
Intenso	0FH
Normal subrayado	01H
Inteso subrayado	09H
Inverso	70H
Titilar normal	87H
Titilar intenso	8FH
Titilar normal subrayado	81H
Titilar intenso subrayado	89H
Titilar inverso	F0H

Función 09H: Escribe carácter y atributo en la posición del cursor.

Se llama con: AH = 09H.
AL = Código ASCII del carácter.
CX = Nº de caracteres a escribir.
BH = Nº de página de video.
BL = Byte de atributo.

Función 0AH: Escribe carácter en la posición del cursor.

Se llama con: AH = 0AH.
BH = Nº de página de video.
CX = Nº de caracteres a escribir.
AL = Código ASCII del carácter.

Función 0EH: Escribe carácter en la pantalla y avanza el cursor.

Se llama con: AH = 0EH.
AL = Código ASCII del carácter.
BH = Nº de página de video.

Interrupción 16H: Realiza funciones del BIOS de E/S por teclado.

Función 00H: Lee el código de inspección del teclado.

Se llama con AH = 00H.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Devuelve: AH = Código de posición.
AL = Código ASCII del carácter.

Función 02H: Obtiene status del teclado.

Se llama con: AH = 02H.
Devuelve: AL = Flags del teclado.
Bit 7 = Ins.
Bit 6 = Caps Lock.
Bit 5 = Num Lock.
Bit 4 = Scroll Lock.
Bit 3 = Alt.
Bit 2 = Ctrl.
Bit 1 = Shift izquierdo.
Bit 0 = Shift derecho.

Interrupción 17H: Realiza funciones BIOS de E/S por impresora.

Función 00H: Escribe carácter en la impresora.

Se llama con: AH = 00H.
AL = Carácter a escribir.
DX = Nº de impresora (0 - 2).

Devuelve: AH = Status de impresora.
Bit 7 = Impresora no ocupada.
Bit 6 = Reconocimiento.
Bit 5 = Sin papel.
Bit 4 = Impresora On Line.
Bit 3 = Error de E/S.
Bit 2 = No se usa.
Bit 1 = No se usa.
Bit 0 = Tiempo excedido.

Función 01H: Inicializa puerto de impresora.

Se llama con: AH = 01H.
DX = Nº de impresora (0 - 2).

Devuelve: AH = Status de impresora.

Función 02H: Lee status de impresora.

Se llama con: AH = 02H.
DX = Nº de impresora (0 - 2).

Devuelve: AH = Status de impresora.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Interrupción 21H: Realiza funciones del DOS de alto nivel.

Función 01H: Lee carácter del teclado.

Se llama con: AH = 01H.

Devuelve: AL = Carácter.

Función 02H: Visualiza un carácter en la pantalla.

Se llama con: AH = 02H.
DL = Carácter.

Función 05H: Imprimir un carácter en un dispositivo de impresión.

Se llama con: AH = 05H.
DL = Carácter.

Función 07H: Lee carácter del teclado, pero no lo visualiza.

Se llama con: AH = 07H.

Devuelve: AL = Carácter.

Función 0BH: Detecta si existe un carácter disponible en el teclado.

Se llama con: AH = 0BH.

Devuelve: AL = 0FFH, si se ha pulsado una tecla.

Función 25H: Asigna vector de interrupción.

Se llama con: AH = 25H.
AL = Nº de la interrupción.
DS:DX = Segment:Offset de la rutina de manejo de la Interrupción.

Función 2AH: Obtiene fecha del sistema.

Se llama con: AH = 2AH.

Devuelve: CX = Año (1980 a 2099).
DH = Mes (1 a 12).
DL = Día (1 a 31).

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Función 2BH: Asigna fecha al sistema.

Se llama con: AH = 2BH.
CX = Año (1980 a 2099).
DH = Mes (1 a 12).
DL = Día (1 a 31).

Función 2CH: Obtiene hora del sistema.

Se llama con: AH = 2CH.

Devuelve: CH = Hora (0 a 23).
CL = Minutos (0 a 59).
DH = Segundos (0 a 59).
DL = Centésimas de segundo (0 a 99).

Función 2DH: Asigna hora al sistema.

Se llama con: AH = 2DH.
CH = Hora (0 a 23).
CL = Minutos (0 a 59).
DH = Segundos (0 a 59).
DL = Centésimas de segundo (0 a 99).

Función 31H: Terminar y permanecer residente.

Se llama con: AH = 31H.
AL = Código de retorno.
DX = Tamaño en párrafos de memoria a reservar.

Función 35H: Obtiene vector de interrupción.

Se llama con: AH = 35H.
AL = Nº de la interupción.

Devuelve: ES:BX = Segment:Offset del manipulador de la interrupción.

Función 4CH: Salida final al MS/DOS.

Se llama con: AH = 4CH.
AL = Código de retorno.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Decimal	Octal	Hexadecimal	Character
0	000	00	NUL
1	001	01	SOH
2	002	02	STX
3	003	03	ETX
4	004	04	EOT
5	005	05	ENQ
6	006	06	ACK
7	007	07	BEL
8	010	08	BS
9	011	09	HT
10	012	0A	LF
11	013	0B	VT
12	014	0C	FF
13	015	0D	CR
14	016	0E	SO
15	017	0F	SI
16	020	10	DLE
17	021	11	DC1
18	022	12	DC2
19	023	13	DC3
20	024	14	DC4
21	025	15	NAK
22	026	16	SYN
23	027	17	ETB
24	030	18	CAN
25	031	19	EM
26	032	1A	SUB
27	033	1B	ESC
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
32	040	20	SP
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?

Decimal	Octal	Hexadecimal	Character
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	^
95	137	5F	_
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~
127	177	7F	DEL

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

CAPITULO 12
PROCESOS CONCURRENTES Y CORRUTINAS

12.1 INTRODUCCION

Uno de los aspectos más interesantes de M2, es la posibilidad de diseñar corrutinas y procesos concurrentes, recursos básicos para la multiprogramación, no obstante, se pueden distinguir varios tipos de sistemas de multiprogramación, dependiendo de la arquitectura del computador utilizado, y que son:

1) Concurrencia real

   En este caso, el sistema se compone de varios procesadores idénticos, de forma que cada proceso, puede ser ejecutado en un procesador diferente.

2) Cuasi concurrencia

   El sistema dispone de un único procesador, que deberá ser compartido por todos los procesos conmutando entre ellos.

3) Concurrencia sobre periféricos

   El sistema sólo dispone de un procesador central, pero varios en las unidades periféricas, en este caso, algunos procesos debido a su naturaleza, sólo se podrán ejecutar con procesadores de estos periféricos, de forma que, determinados procesos se pueden ejecutar simultáneamente a este nivel.

En este capítulo se tratará fundamentalmente el segundo

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

caso, por ser el más común, debido a que la mayoría de los ordenadores personales, disponen de un único procesador.

12.2 PROCESOS CONCURRENTES

Un proceso, es una tarea ejecutándose en un computador, que se puede concebir, como un programa o un procedimiento. Si dos procesos se están ejecutando simultáneamente en el mismo computador, se les llama procesos concurrentes.

Para que exista una concurrencia real, es necesario que el ordenador en que estemos trabajando disponga de varios procesadores, de forma que cada proceso sea ejecutado por un procesador diferente. Sin embargo, la mayoría de los computadores personales disponen de un único procesador, por lo que sólo la pseudo - concurrencia es posible, es decir, el procesador que existe, debe ser compartido por todos los procesos, simplemente conmutando rápidamente entre ellos, lo cual da la apariencia de una concurrencia real.

Para M2, no hay diferencia de tratamiento entre la concurrencia real y la cuasi concurrencia, pues las afronta de la misma manera, con el módulo de librería Processes, que contiene las siguientes rutinas.

SEND --> Envía una señal y reactiva el proceso que la espera.

WAIT --> Suspende un proceso hasta que reciba una señal.

Init --> Inicializa una señal.

StartProcess --> Empieza un proceso.

Awaited --> Verifica si existe algún proceso a la espera de recibir una señal.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

StartProcess tiene la forma general;

StartProcess(Proc,TamañoAreaTrabajo)

donde Proc, es el nombre del proceso, (hay que tener en cuenta que un proceso se define como un procedimiento sin parámetros, de tipo estándar) y TamañoAreaTrabajo, es el número de bytes que se requiere en memoria para salvar las variables locales del proceso, cuando éste se encuentra inactivo.

Los precesos se comunican entre ellos, usando variables del tipo (exportado por Processes) SIGNAL.

Para enviar una señal, se requiere el uso de SEND, cuya forma general es;

SEND(Señal)

donde señal es una variable del tipo SIGNAL.

Para que un proceso reciba una señal, éste debe estar esperándola a través del WAIT, que tiene la forma:

WAIT(Señal)

Cuando en un proceso se ejecuta WAIT , éste se suspende (en espera de que se le envíe una señal) y se reanuda otro. Si no hay otro proceso que pueda ser ejecutado, el programa termina.

SEND se utiliza para reactivar un proceso que está esperando por una señal, ahora bien, si no hay ningún proceso esperando, no hace nada.

Awaited (Señal), verifica si algún proceso se encuentra en espera de recibir una señal. La salida será TRUE, si y sólo si al menos unos de los procesos, se encuentra en espera de una señal.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Antes del uso de las señales, éstas se deben inicializar a través del Init(Señal).

Después de la inicialización de la señal s, Awaited(s) resultaría FALSE.

A continuación se pueden ver los procedimientos de Processes, con sus listas de parámetros formales.

```
PROCEDURE StartProcess(P:Proceso;TamañoAreaTrabajo:CARDINAL);
PROCEDURE SEND (VAR Señal:SIGNAL);
PROCEDURE WAIT (VAR Señal:SIGNAL);
PROCEDURE Awaited (Señal:SIGNAL):BOOLEAN;
PROCEDURE Init (VAR Señal:SIGNAL);
```

El siguiente módulo, es un ejemplo típico de ejecución concurrente, donde dos procesos sincronizados por una señal, se pasan información a través de una variable compartida.

El proceso Productor lee caracteres del teclado, y los almacena en al variable 'buf', para que consumidor visualice su contenido, estableciéndose una transferencia de control continua entre procesos hasta que se decida abandornar la ejecución del programa, pulsando la tecla ESC.

Modula-2/86 concur.mod

```
 1 MODULE Pconcur;
 2   FROM InOut IMPORT WriteLn,WriteString,Read,EOL,Write;
 3   FROM Processes IMPORT WAIT,SEND,StartProcess,SIGNAL,Init;
 4   VAR
 5     S:SIGNAL;
 6     buf: ARRAY [0..100] OF CHAR; (*Variable compartida*)
 7 (*------------------------------------------------------*)
 8 PROCEDURE Consumidor;
 9 BEGIN
10   LOOP
11     WriteString('ESPERANDO ');WriteLn;
12     WAIT(S);
13     WriteString('SACAMOS EL CONTENIDO DEL BUFFER ');WriteString(buf);
14     WriteLn; WriteLn;
15   END
16 END Consumidor;
17 (*------------------------------------------------------*)
18   PROCEDURE Productor;
19     VAR
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
20        contador:CARDINAL;
21        ch:CHAR;
22      BEGIN
23        contador:=0;
24        LOOP
25          Read(ch);
26          IF ch = 33C THEN EXIT END; (*ESC para salir*)
27          IF ch # EOL THEN
28            buf[contador]:=ch;
29            Write(ch);
30            INC(contador);
31          ELSE
32            buf[contador]:=CHR(0);
33            WriteLn;
34            SEND(S);
35            contador:=0;
36          END;
37        END;
38      END Productor;
39  (*------------------------------------------------------------*)
40
41  BEGIN
42    Write(14C);
43    Init(S);
44    StartProcess (Consumidor,1000);
45    Productor;
46  END Pconcur.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

Hay que hacer notar, que en primer lugar, el control se transfiere a consumidor debido al StartProcess, una vez ahi, se suspende consumidor y se salta a la linea 45 de programa, el SEND devuelve el control a consumidor en la linea 13 y al volverse a ejecutar WAIT, se transfiere el control a la linea 35 y asi sucesivamente hasta que se pulse ESC.

A continuación se puede observar un nuevo ejemplo de concurrencia, con el resultado de su ejecución, donde cada vez que un proceso toma el control, incrementa la variable compartida 'contador'.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
 1  MODULE Variablecompartida;
 2    FROM InOut IMPORT WriteLn,WriteString,WriteCard;
 3    FROM Processes IMPORT SIGNAL,SEND,WAIT,Init,StartProcess;
 4    FROM Util IMPORT Cls,Sp;
 5    VAR
 6      contador:CARDINAL; (* Variable compartida *)
 7      ok:SIGNAL;
 8
 9    PROCEDURE proceso2;
10    BEGIN
11       Sp(27);WriteString ('El proceso2 ha comenzado ');WriteLn;
12       LOOP
13         WAIT (ok);
14         Sp(24); WriteString ('En el proceso2 contador =');
15         INC (contador);WriteCard(contador,2);WriteLn;
16       END
17    END proceso2;
18
19    BEGIN (* El programa principal, es el proceso 1*)
20      Cls;
21      WriteString (' El Proceso1 ha comenzado');
22      Init(ok);contador:=0;
23      StartProcess(proceso2,1000);
24
25      LOOP
26        WriteString('En el proceso1 contador =');
27        INC (contador);WriteCard(contador,2);
28        IF contador >30 THEN EXIT END;
29        SEND(ok);
30      END;
31    WriteLn;WriteLn;
32    WriteString('contador=');WriteCard(contador,3);WriteLn;
33    WriteString('Fin del programa')
34
35  END Variablecompartida.
```

```
Modula-2/86 Compiler Version V 1.10

==>     0 Error(s) found
```

```
 El Proceso1 ha comenzado
En el proceso1 contador = 1
En el proceso1 contador = 3
En el proceso1 contador = 5
En el proceso1 contador = 7
En el proceso1 contador = 9
En el proceso1 contador =11
En el proceso1 contador =13
En el proceso1 contador =15
En el proceso1 contador =17
En el proceso1 contador =19
En el proceso1 contador =21
En el proceso1 contador =23
En el proceso1 contador =25
En el proceso1 contador =27
En el proceso1 contador =29
En el proceso1 contador =31

contador= 31
Fin del programa
```

```
 El proceso2 ha comenzado
En el proceso2 contador = 2
En el proceso2 contador = 4
En el proceso2 contador = 6
En el proceso2 contador = 8
En el proceso2 contador =10
En el proceso2 contador =12
En el proceso2 contador =14
En el proceso2 contador =16
En el proceso2 contador =18
En el proceso2 contador =20
En el proceso2 contador =22
En el proceso2 contador =24
En el proceso2 contador =26
En el proceso2 contador =28
En el proceso2 contador =30
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

El módulo sinvarc, se incluye también como ejemplo de concurrencia. Consiste en una conmutación entre dos procesos, uno de los cuales cuenta números y el otro letras, en este caso no hay variables compartidas.

Modula-2/86 sinvarc.MOD

```
 1 MODULE SinVariableCompartida;
 2   FROM InOut IMPORT Read,Write,WriteCard;
 3   FROM Processes IMPORT WAIT,SEND,SIGNAL,StartProcess,Init;
 4   FROM Terminal IMPORT KeyPressed;
 5   FROM Util IMPORT Cls,PxPy,Vi,Neg,Vv;
 6   VAR
 7     ch:CHAR;
 8     n,l:SIGNAL;
 9 (*-------------------------------------*)
10   PROCEDURE Numero; (*Proceso1 cuenta números*)
11     VAR
12       num,retardo:CARDINAL;
13   BEGIN
14     num:=0;
15     LOOP
16       WAIT(n);
17       FOR retardo:=0 TO 15000 DO END;
18       IF num=26 THEN num:=0 END;
19       PxPy(0,0);
20       Vi;WriteCard(num,2);Vv;
21       INC(num);
22     END;
23   END Numero;
24 (*-------------------------------------*)
25   PROCEDURE Letra; (*Proceso2 cuenta letras*)
26     VAR
27       let,retardo:CARDINAL;
28   BEGIN
29     let:=65;
30     LOOP
31       WAIT(l);
32       FOR retardo:=0 TO 15000 DO END;
33       IF let=91 THEN let:=65 END;
34       PxPy(78,0);
35       Neg;Write(CHR(let));Vv;
36       INC(let);
37     END;
38   END Letra;
39 (*-------------------------------------*)
40   PROCEDURE Sincroniza;
41   BEGIN
42     LOOP
43       IF NOT KeyPressed() THEN
44         SEND(n);
45         SEND(l);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
46        ELSE
47          Read(ch);
48          IF ch=33C THEN RETURN ELSE END;
49        END;
50      END;
51    END Sincroniza;
52  (*------------------------------------------*)
53  BEGIN
54    Cls;
55    Init(n);Init(1);(*Se inicializan las señales*)
56    StartProcess(Numero,1000);
57    StartProcess(Letra,1000);
58    Sincroniza;
59  END SinVariableCompartida.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```

Al principio del capítulo, se comentó que M2 afrontaba la concurrencia real y la cuasi concurrencia de la misma forma, es decir con las herramientas de la librería Processes.

Hemos visto como se han utilizado variables compartidas, para transferir datos entre procesos cuasi concurrentes.

Sin embargo, en el caso de un computador con varios procesadores, donde los procesos se ejecutarían en concurrencia real, tendríamos problemas con la utilización de variables compartidas, tal y como las hemos estado utilizando hasta ahora, pues un proceso puede estar realizando una operación crítica con una variable y otro proceso interferir alterando los valores.

Una solución a esto, sería encapsular las variables comunes dentro de un módulo llamado Monitor, de forma que se

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

garantice la exclusión mutua entre procesos, es decir, que sólamente un proceso en un instante dado, puede tener acceso al Monitor.

Un Monitor, es un módulo local, que se utiliza para evitar interferencias entre procesos que se ejecutan en concurrencia real.

Para ilustrar ésto, presentamos un ejemplo clásico de múltiprogramación, donde dos procesos se intercambian datos a través de un buffer compartido.

Un proceso tiene por misión, almacenar datos en un buffer y otro extraerlos del mismo. El buffer, junto con los procedimientos Deposite y Retire que manejan la cooperación entre procesos, se encuentran encapsulados en un Monitor.

Modula-2/86 multipro.MOD

```
 1  MODULE ProductorConsumidor;
 2    FROM InOut IMPORT ReadString, WriteString, WriteLn;
 3    FROM Processes IMPORT StartProcess, Awaited, Init,
 4                          SIGNAL, WAIT, SEND;
 5    TYPE
 6      tipocadena = ARRAY [0 .. 79] OF CHAR;
 7    VAR
 8      esperafinal: SIGNAL;
 9
10  (*--------------------MONITOR--------------------*)
11  MODULE BufferCompartido;
12     IMPORT SIGNAL, SEND, WAIT, Awaited, Init;
13     EXPORT Depositar, Retirar;
14     VAR
15       n, entrada, salida : CARDINAL;
16       NoLleno, NoVacio : SIGNAL;
17       buffer : ARRAY [0 .. 15] OF INTEGER;(* buffer compartido*)
18
19     PROCEDURE Depositar (entero : INTEGER);
20     BEGIN
21       IF n = 16 THEN WAIT (NoLleno); END;
22       INC (n);
23       buffer [entrada] := entero;
24       entrada := (entrada + 1) MOD 16;
25       IF Awaited (NoVacio) THEN SEND (NoVacio); END;
26     END Depositar;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
27
28    PROCEDURE Retirar (VAR entero : INTEGER);
29    BEGIN
30      IF n = 0 THEN WAIT (NoVacio); END;
31      DEC (n);
32      entero := buffer [salida];
33      salida := (salida + 1) MOD 16;
34      IF Awaited (NoLleno) THEN SEND (NoLleno); END;
35    END Retirar;
36
37  BEGIN
38    n := 0; entrada := 0; salida := 0;
39    Init (NoLleno); Init (NoVacio);
40  END BufferCompartido;
41  (*-------------------------------------------------------*)
42
43    PROCEDURE Almacena;(*proceso1*)
44      VAR
45        i : INTEGER;
46        cadena : tipocadena;
47    BEGIN
48      WriteString ('Introduzca una cadena: ');
49      i := -1;
50      ReadString (cadena);
51
52      REPEAT
53         INC (i);
54         Depositar (ORD (cadena [i]));
55      UNTIL cadena [i] = 0C;
56
57    END Almacena;
58  (*-------------------------------------------------------*)
59
60    PROCEDURE Extrae;(*proceso2*)
61      VAR
62        i, orden : INTEGER;
63        cadena : tipocadena;
64    BEGIN
65      i := -1;
66
67      REPEAT
68        INC (i);
69        Retirar (orden);
70        cadena [i] := CHR (orden);
71      UNTIL cadena [i] = 0C;
72
73      WriteLn;
74      WriteString ('Despues de la transferencia: ');
75      WriteString (cadena);
76      SEND(esperafinal);
77    END Extrae;
78  (*-------------------------------------------------------*)
79
80  BEGIN
81    Init (esperafinal);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
82    StartProcess (Extrae, 1000);
83    StartProcess (Almacena, 1000);
84    WAIT (esperafinal);
85 END ProductorConsumidor.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

En un computador de un solo procesador, la interferencia entre procesos es imposible por definición, por lo tanto la utilización de un Monitor, resulta redundante.

12.3 LAS CORRUTINAS

Una corrutina, es un programa secuencial idéntico en esencia a un proceso. Las diferencias básicas entre ellos son:

1) Las corrutinas se ejecutan en cuasi concurrencia con otras corrutinas.
2) El programador, establece la trasferencia de control entre corrutinas, a través de transferencias explícitas.

En los procesos, las señales sirven para sincronizar, y hay una transferencia de control no explícita, por ejemplo, el proceso que ejecuta un SEND, desconoce a quien va a alertar.

La llamada a una corrutina, transfiere el control, o bien al principio de ésta, en el caso de la primera llamada, o bien al sitio donde la corrutina había sido abandonada.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Para el tratamiento de las corrutinas, M2 nos proporciona en el módulo SYSTEM, el tipo de variable PROCESS y los procedimientos NEWPROCESS y TRANSFER.

Una variable que se declara con el tipo PROCESS, puede considerarse como un puntero, que apunta a una corrutina.

NEWPROCESS nos va a crear una corrutina con un entorno de trabajo, sin lanzarla, es decir, sin dar comienzo a su ejecución. Su forma general es;

NEWPROCESS (NProc, DirAreaTrab, TamAreaTrab, P)

donde NProc, es el nombre del procedimiento (sin parámetros) que queremos convertir en corrutina.

DirAreaTrab y TamAreaTrab son respectivamente, la dirección de la región de memoria que contendrá la corrutina y el número de bytes de longitud de esta región o área de trabajo.

Para la determinación del área de trabajo, se suele declarar una variable del tipo matriz, y posteriormente, con los procedimientos ADR y SIZE aplicados a ésta, se obtienen DirAreaTrab y TamAreaTrab.

Hay que tener en cuenta que el array que se declara, debe ser lo suficientemente grande, para almacenar los datos locales y la pila de la corrutina.

P, es una variable de tipo PROCESS, que apunta a la corrutina y nos sirve para designarla.

Para otorgar el control a una corrutina, hay que ejecutar el procedimiento TRANSFER, cuya forma general es;

TRANSFER (DesdeC1,HaciaC2)

donde DesdeC1 y HaciaC2, son variables del tipo PROCESS.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Su invocación, provoca la suspensión de la corrutina DesdeC1 (que se podrá reanudar posteriormente en la sentencia que sigue a TRANSFER) y la reactivación de la corrutina HaciaC2, en el último punto de suspensión. En el caso de que se llame a una corrutina por primera vez, esta se ejecuta desde el princpio.

Para lanzar una corrutina, ésta debe haber sido creada previamente, de lo contrario se cometería un herror.

Los procedimientos NEWPROCESS y TRANSFER, expresados con sus listas de parámetros formales, tienen la forma:

```
PROCEDURE NEWPROCESS (P:PROC; D:ADRESS; T:CARDINAL;
                      VAR C:PROCESS);

PROCEDURE TRANSFER (VAR C1,C2:PROCESS);
```

El siguiente programa, nos muestra a través de una serie de mensajes, el funcionamiento de dos corrutinas.

Inicialmente, se transfiere el control a Corruti1, sin embargo, si se pulsa una tecla, se pasa el control momentáneamente a Corruti2.

Modula-2/86 corutina.MOD

```
 1  MODULE Corrutinas;
 2    FROM InOut IMPORT WriteString,Read;
 3    FROM Keyboard IMPORT KeyPressed;
 4    FROM SYSTEM IMPORT PROCESS,ADR,SIZE,NEWPROCESS,TRANSFER;
 5    FROM Util IMPORT Cls,PxPy,Vi,Vv;
 6    VAR
 7      salida, C1, C2: PROCESS;  (*variables de tipo PROCESS*)
 8      AreaTrab1, AreaTrab2: ARRAY[1..1000] OF CHAR; (*areas de trabajo*)
 9
10    PROCEDURE Retardo(a,b:CARDINAL);
11      VAR
12        x,y:CARDINAL;
13    BEGIN
14      FOR y:=0 TO a DO
15        FOR x:=0 TO b DO END;
16      END;
17    END Retardo;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
18 (*------------------------------------------------------------------*)
19   PROCEDURE Corruti1;
20     VAR
21       ch:CHAR;
22   BEGIN
23     LOOP
24       Cls;
25       WHILE NOT KeyPressed() DO
26         PxPy(8,1);Vi; WriteString('Estamos en Corruti1 ');
27         PxPy(0,3);Vi; WriteString('Cada vez que se pulse una tecla, se');
28         PxPy(0,4);Vi; WriteString('hace una transferencia momentanea a');
29         PxPy(0,5);Vi; WriteString('Corruti2                            ');
30         PxPy(8,7);Vi; WriteString('Pulse ESC para salir');Vv;
31       END;
32       Read(ch);
33       IF ch=33C THEN  (*ESC para salir*)
34         TRANSFER (C1,salida);
35       ELSE
36         TRANSFER (C1,C2);
37       END;
38       Cls;
39       PxPy(0,14);Vi; WriteString('Hemos vuelto a Corruti1, por el mismo');Vv;
40       PxPy(0,15);Vi; WriteString('sitio en que la habiamos abandonado. ');Vv;
41       Retardo(10,22000);
42     END;
43   END Corruti1;
44 (*------------------------------------------------------------------*)
45   PROCEDURE Corruti2;
46     VAR
47       x,y:CARDINAL;
48   BEGIN
49     LOOP
50       Cls;
51       PxPy(32,0);Vi;WriteString('Estamos en Corruti2');Vv;
52       Retardo(10,25000);
53       TRANSFER(C2,C1); (*Devuleve el control a C1*)
54       Cls;
55       PxPy(43,14);Vi; WriteString('Hemos vuelto a Corruti2, por el mismo');Vv;
56       PxPy(43,15);Vi; WriteString('sitio en que la habiamos abandonado. ');Vv;
57       Retardo(10,22000);
58     END;
59   END Corruti2;
60 (*------------------------------------------------------------------*)
61
62 BEGIN
63  NEWPROCESS(Corruti1, ADR(AreaTrab1), SIZE(AreaTrab1), C1);
64  NEWPROCESS(Corruti2, ADR(AreaTrab2), SIZE(AreaTrab2), C2);
65  TRANSFER(salida, C1);  (* empieza el programa *)
66 END Corrutinas.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

En este programa ejemplo, se puede ver claramente la diferencia entre un procedimiento y una corrutina. Cuando se llama a un procedimiento, éste, se ejecuta siempre desde el principio. Sin embargo, cuando una corrutina llama a otra, la ejecución se reinicia desde el último punto que se ejecutó antes de la transferencia de control previa.

Hay que hacer notar, la declaración de la variable 'salida' y su utilización, ésta, nos permite la transferencia al final del programa, para que haya una terminación adecuada.

El siguiente programa es capaz de contar el tiempo, para lo cual, dispone de una corrutina que contabiliza las horas, otra los minutos y una tercera los segundos.

En previsión de que el tiempo de retardo varíe en función del computador que estemos utilizando, existe una cuarta corrutina, que puede ser llamada en cualquier momento pulsando la tecla 'r', de forma que podríamos asignar a la variable 'ret', cualquier otro valor que se considere oportuno, para que la temporización sea la adecuada.

```
Modula-2/86                    reloj.MOD

 1  MODULE Temporizador;
 2    FROM Util IMPORT Bs,Sp,Cls,Cll,Vi,Vv,PxPy;
 3    FROM SYSTEM IMPORT PROCESS,NEWPROCESS,TRANSFER,ADR,SIZE;
 4    FROM InOut IMPORT WriteCard,Write,Read,WriteString,ReadCard,Done;
 5    FROM Keyboard IMPORT KeyPressed;
 6    VAR
 7      ret:CARDINAL;
 8      salida,hr,min,seg,retardo:PROCESS;
 9      AreaTrab1,AreaTrab2,AreaTrab3,AreaTrab4:ARRAY[0..1000] OF CHAR;
10    PROCEDURE Retardo;
11    BEGIN
12      LOOP
13        REPEAT
14          Vv;PxPy(10,12);
15          WriteString('Introduzca un número entero positivo como retardo: ');
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
16          ReadCard(ret);Bs(70);C11;
17        UNTIL Done;
18        Vi;
19        TRANSFER (retardo,seg);
20      END;
21    END Retardo;
22  (*-------------------------------------------------*)
23    PROCEDURE Horas;
24      VAR
25        hh:CARDINAL;
26    BEGIN
27      Bs(5);
28      hh:=1;
29      LOOP
30        IF hh=24 THEN
31          hh:=0;Write('0');WriteCard(0,1);
32        ELSE
33          IF hh<=9 THEN
34            Write('0');WriteCard(hh,1);
35          ELSE
36            WriteCard(hh,2);
37          END;
38        END;
39        hh:=hh+1;
40        Sp(3);
41        TRANSFER(hr,min);
42        Bs(5);
43      END;
44    END Horas;
45  (*-------------------------------------------------*)
46    PROCEDURE Minutos;
47      VAR
48        mm:CARDINAL;
49    BEGIN
50      Bs(3);
51      mm:=1;
52      LOOP
53        IF mm=60 THEN
54          Write('0');Write('0');
55        ELSE
56          IF mm<=9 THEN
57            Write('0');WriteCard(mm,1);
58          ELSE
59            WriteCard(mm,2);
60          END;
61        END;
62        mm:=mm+1;
63        IF mm>=61 THEN mm:=1;TRANSFER(min,hr);END;
64        TRANSFER(min,seg);
65        Bs(3);
66      END;
67    END Minutos;
68  (*-------------------------------------------------*)
69    PROCEDURE Segundos;
70      VAR
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
71      ch:CHAR;
72      ss,x,y:CARDINAL;
73    BEGIN
74      ss:=0;
75      ret:=47200;
76      LOOP
77        WHILE NOT KeyPressed() DO
78          IF ss<=9 THEN
79            Write('0');WriteCard(ss,1);Bs(2);
80          ELSE
81            WriteCard(ss,2);Bs(2);
82          END;
83          FOR y:=0 TO 1 DO (*Retardo*)
84            FOR x:=0 TO ret DO END;
85          END;
86          INC (ss);
87          IF ss>59 THEN ss:=0; TRANSFER(seg,min);Bp(1) END;
88        END;
89        Read(ch);
90        CASE ch OF
91         (*ESC*)  33C: TRANSFER(seg,salida);
92         (* r *) 162C: TRANSFER(seg,retardo);PxPy(7,0);
93        ELSE
94        END;
95      END;
96    END Segundos;
97  (*-------------------------------------------*)
98  BEGIN
99    Cls;Vi;
100   Write('0');Write('0');Write(':');Write('0');Write('0');Write(':');
101   Write('0');Write('0');Bs(2);
102   NEWPROCESS(Horas,ADR(AreaTrab1),SIZE(AreaTrab1),hr);
103   NEWPROCESS(Minutos,ADR(AreaTrab2),SIZE(AreaTrab2),min);
104   NEWPROCESS(Segundos,ADR(AreaTrab3),SIZE(AreaTrab3),seg);
105   NEWPROCESS(Retardo,ADR(AreaTrab4),SIZE(AreaTrab4),retardo);
106   TRANSFER (salida,seg);Vv;
107 END Temporizador.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

12.4 IOTRANSFER

En el capitulo anterior se vio como se podían implementar manipuladores de interrupciones, sin embargo, M2 permite asociar un Mi a una determinada interrupción de una forma más sencilla, a través del procedimiento IOTRANSFER

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

proporcionado por la librería SYSTEM, cuya forma general es:

IOTRANSFER(VAR Mi,ProcesoInterrumpido:PROCESS;NºINT:CARDINAL)

Su ejecución hace que se devuelva el control desde el Mi, (el mismo que ejecuta dicho procedimiento) al proceso interrumpido y además hace que se produzca la transferencia inversa, es decir, un TRANSFER automático desde el proceso en curso, a la instrucción siguiente a IOTRANSFER, cuando se ocasiona la interrupción vigente.

A continuación se muestra un programa ejemplo, que realiza la misma función que el módulo gestor.MOD del capitulo anterior, pero en este caso el manipulador de interrupciones, se implementa con la ayuda del procedimiento IOTRANSFER.

```
Modula-2/86                    iotrans.MOD

 1  MODULE GestorDeInterrupciones;
 2    FROM Util IMPORT Cls, PxPy;
 3    FROM InOut IMPORT Read, Write;
 4    FROM Terminal IMPORT KeyPressed;
 5    FROM Devices IMPORT SaveInterruptVector, RestoreInterruptVector;
 6    FROM SYSTEM IMPORT PROCESS,NEWPROCESS,TRANSFER,IOTRANSFER,ADR,SIZE,DISABLE,ADDRESS;
 7    VAR
 8      ch:CHAR;
 9      z:CARDINAL;
10      Programa,Mi PROCESS;
11      AreaTrab: ARRAY[1..1000] OF CHAR;
12      VectorViejoDeInterrupcion:ADDRESS;
13  (*-------------------------------------------*)
14    PROCEDURE As  (VAR ta:ARRAY OF CHAR);
15    VAR
16     x,y:CARDINAL;
17    BEGIN
18      FOR x:=0 TO 3 DO
19        FOR y:=0 TO 65535 DO END
20      END;
21    END As;
22  (*-------------------------------------------*)
23  PROCEDURE ManipuladorInterrupcion;
24  BEGIN
25    LOOP
26      IOTRANSFER(Mi,Programa,5H);
27      DISABLE;
28      As;
29    END;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
30  END ManipuladorInterrupcion;
31  (* ------------------------------------------------------ *)
32    BEGIN
33      SaveInterruptVector(5H,VectorViejoDeInterrupcion);
34      NEWPROCESS(ManipuladorInterrupcion,ADR(AreaTrab),SIZE(AreaTrab),Mi);
35      TRANSFER(Programa,Mi);
36
37      Cls; PxPy(0,10); ch:=0C; z:=0;
38
39      REPEAT
40        IF KeyPressed() THEN Read(ch) END;
41        Write('*');Write('-');
42        INC(z,2); IF z >= 1200 THEN z:=0; Cls; PxPy(0,10); END;
43      UNTIL ch=33C;
44
45      RestoreInterruptVector(5H,VectorViejoDeInterrupcion);
46
47  END GestorDeInterrupciones.
```

```
Modula-2/86 Compiler Version V 1.10

=>    0 Error(s) found
```

El siguiente módulo permite realizar divisiones enteras entre números, de tal forma que, cuando se produzca una división por cero, se transfiera el control al manipulador de interrupciones preparado para tal efecto. El Mi de la INT 0 permite que se tomen dos decisiones; o continuar con el programa, o bien abandonarlo.

Modula-2/86 divid0.MOD

```
 1  MODULE GestorDeInterrupciones;
 2    FROM Util IMPORT Cls,PxPy,Cll,Vi,Vv;
 3    FROM Devices IMPORT SaveInterruptVector, RestoreInterruptVector;
 4    FROM InOut IMPORT Read,Write,WriteLn,ReadCard,WriteCard,WriteString;
 5    FROM SYSTEM IMPORT ADDRESS,DOSCALL,TRANSFER,IOTRANSFER,NEWPROCESS,PROCESS,
 6                       ADR,SIZE;
 7    VAR
 8      a,b,c:CARDINAL;
 9      ch,caracter:CHAR;
10      Programa,Mi:PROCESS;
11      AreaTrab: ARRAY[1..1000] OF CHAR;
12      VectorViejoDeInterrupcion:ADDRESS;
13  (*------------------------------------------------------*)
14  PROCEDURE ManipuladorInterrupcion;
15  BEGIN
16    LOOP
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
17      IOTRANSFER(Mi,Programa,OH);
18
19      Write(33C);Write('[');Write('s'); (*Secuencia ESC,Guarda posición del cursor*)
20
21      PxPy(25,0);Vi;WriteString('Continúa o Termina');Vv;
22
23      REPEAT
24        Read(caracter);
25        Write(7C); (*beep*)
26      UNTIL (CAP(caracter)='C') OR (CAP(caracter)='T'); (*c,C,t,T*)
27
28      PxPy(0,0); Cl.; (*Borra mensaje*)
29
30      IF CAP(caracter)='T' THEN DOSCALL(4CH,0FFH) ELSE END
31
32      Write(33C);Write('[');Write('u'); (*Secuencia ESC,Restaura pcs.ción del cursor*)
33    END;
34  END Manipulador(Interrupcion);
35  (* ------------------------------------------------------------ *)
36    BEGIN
37      SaveInterruptVector(OH,VectorViejoDeInterrupcion);
38      NEWPROCESS(Manipulador Interrupcion,ADR(AreaTrab),SIZE(AreaTrab),Mi);
39      TRANSFER(Programa,Mi);
40
41      Cls; PxPy(0,2); ch:=0C;
42
43      REPEAT
44        Write('a'); Write(':'); ReadCard(a); WriteLn;
45        Write('b'); Write(':'); ReadCard(b); WriteLn; c:=(a)DIV(b);
46        Write('c'); Write(':'); WriteCard(c,0);WriteLn;
47        WriteString('Division realizada, pulse ESC para salir');
48        Read(ch);Cls; PxPy(0,2);
49      UNTIL ch=33C;
50
51      RestoreInterruptVector(OH,VectorViejoDeInterrupcion);
52
53  END GestorDeInterrupciones.
```

```
Modula-2/86 Compiler Version V 1.10

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

APENDICE

SISTEMA DE CONTROL DE ESTABILIDAD DE UNA CENTRAL ELECTRICA

A.1 INTRODUCCION

En cualquier sistema de suministro de potencia por parte de una central eléctrica, es tarea esencial evitar la inestabilidad de la misma. Esta se produce fundamentalmente, cuando las cargas que tiene conectadas demandan mayor potencia que la que la propia central puede suministrar. Una situación de este tipo hace que las tensiones de los generadores de la central decaigan, debido a que sus velocidades disminuyen y en consecuencia también sus frecuencias.

Hay varias formas de concebir un sistema de control de este tipo.

En primer lugar, existe un método tradicional, el más extendido, que consiste en ubicar equipos de protección en lugares estratégicos de la red, que constan de relés de baja frecuencia en estado sólido e interruptores.

Este método está siendo desechado por presentar una serie de deficiencias solventables con otros tipos de procedimientos.

Por su propia naturaleza son un foco de posibles valores inestables, además la actuación de los relés se produce cuando la inestabidídad se ha consumado, sometiéndose a los grupos a esfuerzos indebidos. Tampoco es selectivo, porque desconecta las líneas de una forma preseleccionada, sin

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

atender a las cargas que ocasionan la inestabilidad.

Un segundo método consistiría en un control inteligente basado en un microprocesador, donde se está continuamente adquiriendo datos de la central y tomando decisiones de control óptimas.

Este sistema posee la ventaja de que al ser continua la adquisición de datos a través del tiempo, no permite que se produzca la inestabilidad, actuando de una forma selectiva en el momento de la detección de una anomalía es decir, desconectando cargas en función de la inestabilidad que ocasionan y no de forma predeterminada.

En nuestro caso, se pretende desarrollar un sistema de control inspirado en el firmware del método anterior, evitando no sólo la complejidad de su implementación hardware, si no también los problemas de estandarización de este tipo de productos, ayudándonos de un entorno ampliamente extendido como es el IBM/PC o compatibles, lo que además facilita el mantenimiento del sistema.

A.2 FUNCIONES DEL SISTEMA DE CONTROL

El objetivo fundamental es crear un sistema a través del cual, cuando la central tiende a la inestabilidad, no caiga nunca en ella, pasando de un estado estable a otro transitorio y de uno transitorio de nuevo al estable.

Nuestro sistema de control debe informar en todo momento de los datos que está recepcionando, alertándonos en las situaciones indeseadas, así como de las acciones de control

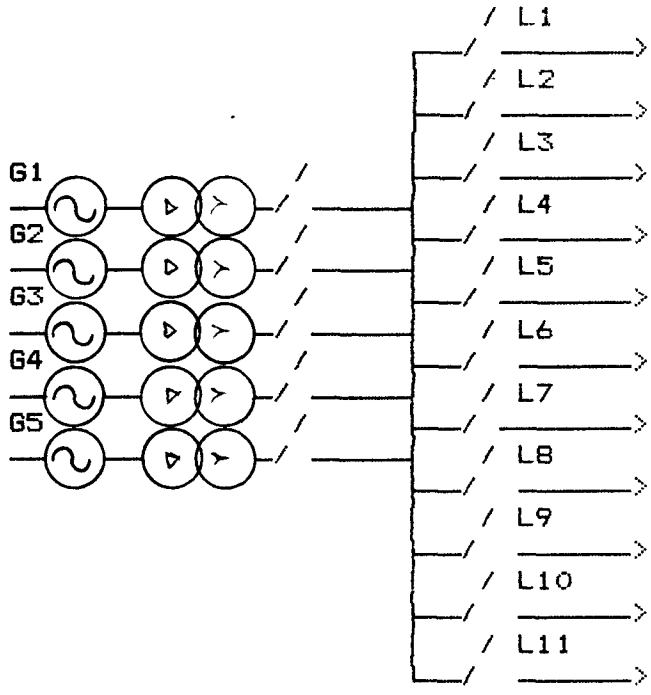
© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

que se están llevando a cabo en los casos necesarios.

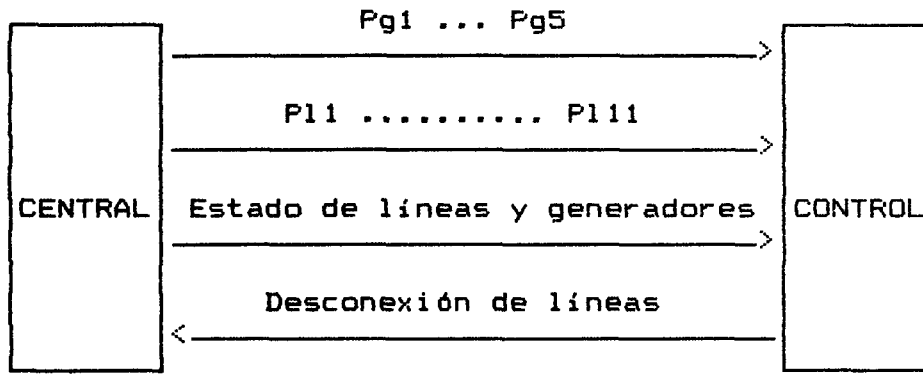
Los parámetros que recibe el sistema de control son; Los valores de potencia de las líneas y de los generadores, así como de su estado, es decir, si están conectados a la red o no.

Las acciones de control se limitarán a la desconexión de líneas de la central en los casos necesarios, según unos criterios (que se comentarán posteriormente) que comparan los valores recibidos con otros de potencia nominal de líneas y generadores, predefinidos en el propio sistema, si bien los de los generadores pueden alterarse a través del terminal.

En las siguientes figuras se presentan la disposición de las líneas y de los generadores en la red, así como los flujos de entrada y salida entre el proceso de control y la central.



© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006



A.3 CRITERIOS DE ACTUACION

El sistema de control debe en todo momento visualizar los valores de potencia de líneas y de generadores, así como sus estados ON - OFF (es decir, si están conectados a la red o no), alertándonos emitiendo sonidos o haciendo titilar los datos en las situaciones preocupantes.

Las razones que harán flashear un dato de este tipo son:

- El estado OFF de una línea o generador.
- Una potencia presente en un generador o en una línea cuando pasa el 100% de su valor nominal.

Se emitirá un pitido de advertencia sólo en el caso de que se detecte una potencia excesiva en algún generador, es decir, cuando se sobrepase el 100% del valor nominal.

Otros datos interesantes a contemplar son, la sumatoria de potencia consumida en las líneas y la sumatoria de la potencia total suministrada por los generadores. Es más debe cumplirse que:

$$\Sigma Pgi = \Sigma Pli$$

Esto nos permite detectar posibles anomalías como la de un cortocircuito en una de las líneas, en el caso que la sumatoria de potencias de las líneas sea mayor que la

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

sumatoria de potencias de los generadores.

No obstante, se podrá permitir un margen de error del 2% en dicha igualdad, pues habrá que tener en cuenta el consumo por parte de los servicios auxiliares más las pérdidas producidas en cualquier otro lado.

Por lo tanto, se harán titilar estos datos en el caso de que, la sumatoria de potencias de los generadores sea menor que la de la suma de las potencias en las líneas en más de un 2%, pues se considerará una situación de error.

Es interesante por otro lado, ofrecer en pantalla las potencias nominales de los generadores, valores que pueden ser alterados desde el terminal.

Es nuestro propósito, disponer la visualización de los datos en el terminal del sistema de control, de la siguiente forma.

CONTROL DE ESTABILIDAD ΣPg=42465 ΣPl=49259

	POTENCIA		P. NOMINAL		ESTADO
GENERADOR 1:	8724	KVA	9400	KVA	ON
GENERADOR 2:	8105	KVA	9400	KVA	ON
GENERADOR 3:	8611	KVA	9400	KVA	ON
GENERADOR 4:	8425	KVA	9400	KVA	ON
GENERADOR 5:	8600	KVA	9400	KVA	ON

	POT.		EST.		POT.		EST.		POT.		EST.
LIN.1	4500	KVA	ON	LIN.5	6123	KVA	ON	LIN.9	3463	KVA	ON
LIN.2	6300	KVA	ON	LIN.6	2375	KVA	ON	LIN10	4243	KVA	ON
LIN.3	2320	KVA	ON	LIN.7	6784	KVA	ON	LIN11	6528	KVA	ON
LIN.4	4280	KVA	ON	LIN.8	2343	KVA	ON				

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Para tomar decisiones de control, el sistema debe estar continuamente adquiriendo datos y comparándolos con los valores nominales, diferenciando entre situaciones estables y aquellas que tienden a la inestabilidad.

Un generador tenderá a la inestabilidad si funciona por encima del 100% de su valor nominal, pudiendo mantenerse trabajando entre el 100% y el 110% una hora como máximo y con una sobrecarga superior al 10% de la potencia nominal, 5 segundos. Transcurridos estos tiempos en uno y otro caso, las tensiones de los generadores decaerán, situación que se debe evitar con la desconexión de líneas.

Existe la posibilidad de ruidos inducidos por campos electromagnéticos en el momento de la adquisición de los datos, pudiendo considerar el sistema que son datos alarmantes, (como valores de potencia $\geq$ 110%) por lo que desconectará líneas innecesariamente.

Estas falsas alarmas deben ser tenidas en cuenta, por lo que habrá que implementar un software que contemple las siguientes consideraciones.

Siempre que se detecte que un generador supera el 90% de su potencia nominal de trabajo, habrá que vigilarlo y observar si su valor de potencia se incrementa de forma brusca (90% --> $\geq$110%) lo que indicará que probablemente se trate de un ruido, o lo hace progresivamente (90% --> $<$110%), lo cual nos indicará que se trata de una verdadera situación de alarma.

En definitiva lo que se pretende diferenciando entre

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

estos dos tipos de incrementos, es realizar una espera prudencial que dependerá del tipo de incremento, en el caso de un incremento brusco, el tiempo de retardo dependerá de la brusquedad del incremento, con un límite máximo de 2'5 segundos, tras lo cual se verificará si se ha reestablecido el generador o no.

En el primer caso, la situación de peligro habrá pasado y no será preciso poner líneas fuera de servicio, en el segundo caso se procederá a la descarga de una línea.

Una vez tomada la decisión de dejar una línea fuera de servicio, debe seleccionarse una entre las que se encuentran conectadas a la red.

Como lo que se intenta, es liberar al sistema del exceso K existente en él, donde:

$$Pgs - Png = K;$$

Pgs = Potencia del generador sobrecargado;

Png = Potencia nominal de dicho generador;

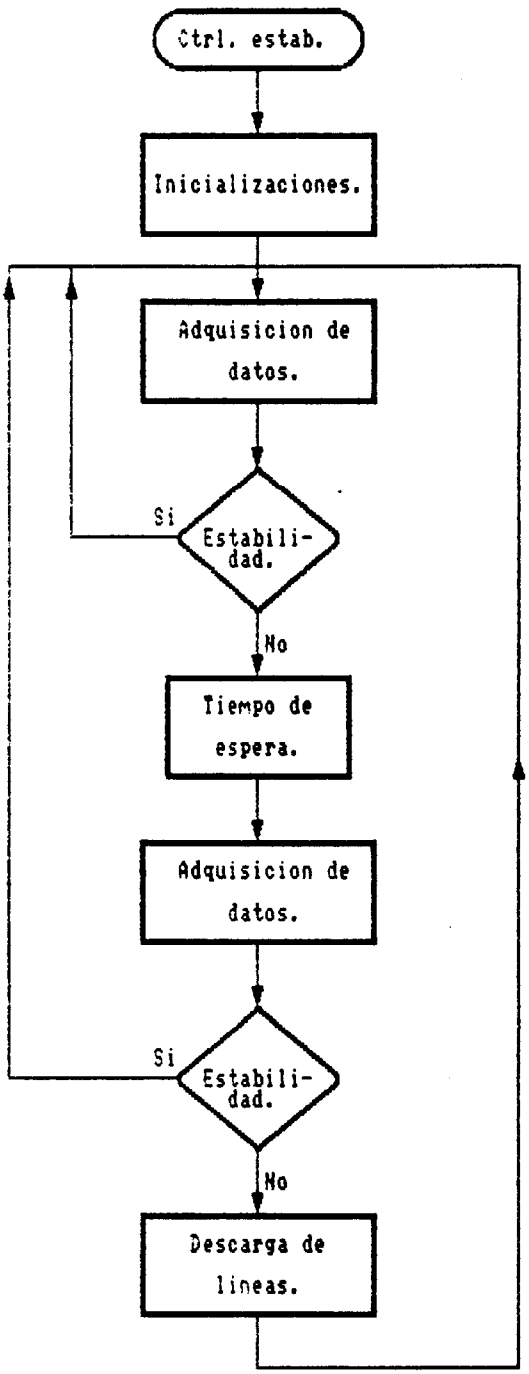
La línea óptima a desconectar, sería aquella cuya potencia demandada sea igual al valor K. En el caso de no existir ninguna con dicha potencia de consumo, se optaría por la descarga de la más próxima a este valor, pero por encima de él.

Después de descargar una línea para aligerar la carga de los grupos, debe realizarse un retardo de espera de aproximadamente medio segundo, que es el tiempo que el sistema tarda en reestablecerse.

En el siguiente organigrama, se reflejan las

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

consideraciones vistas hasta ahora.



© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Una cuestión a considerar, es la posibilidad de que un generador (G2) entre en una situación de alarma en el momento en que otro, que ya lo estaba (G1), decrementaba su tiempo (T1) de espera prudencial antes de la posible desconexión.

Si ignoramos esta situación, se podría dar el caso de que transcurrido T1, no se desconectase ninguna línea por haberse tratado de una falsa alarma y posteriormente, se empezase a decrementar el T2 de G2, si en este segundo caso se tratase de una verdadera situación de alarma y tuviese que ser desconectada una línea, esta acción se produciría transcurrido el tiempo T, donde:

$$T = T1 + T2;$$

Probablemente, sería demasiado tarde y como resultado se perdería la estabilidad de la central.

Por lo tanto, se optará por una comprobación constante de los demás generadores, mientras uno de ellos está decrementando su tiempo de espera, de forma que cuando se produzca una situación anómala en un segundo generador, se le asignará un tiempo de espera, decrementándose ambos simultáneamente.

Una vez acabado uno de los dos tiempos, bien T1 o T2, su correspondiente generador es comprobado, pudiéndose dar dos casos.

En el caso de que continúe mal, se descargará una línea que probablemente aliviará el sistema, reestableciéndose los generadores de éste, por lo que no es necesario actuar tras acabar una segunda cuenta, pudiéndose resetear los

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

contadores.

En el caso de que se haya recuperado, se esperará a que el generador que aun está en fase de decremento de su contador finalice la cuenta. después de este tiempo se volverá a plantear los dos mismos casos y dependiendo del valor del generador correspondiente, habrá descarga o no.

A.4 ADQUISICION DE DATOS

Para la toma de datos, así como para la transmisión de las acciones de control, se ha optado por la utilización de la tarjeta PCL - 712 de U.S. Technologies, que nos proporciona 16 entradas analógicas, además de 16 entradas y 16 salidas digitales, lo cual convierte al PC compatible en una poderosa herramienta de adquisición de datos.

Es capaz de realizar conversiones A/D de ±5 V. en un tiempo inferior a los 30 microsegundos, con una resolución de 12 bits, siendo aproximadamente de 400 microsegundos por término medio, el tiempo que se tarda en muestrear y convertir las 16 A/I, con nuestro software de control.

Si bien en nuestro software se ha considerado la posibilidad de la adquisición de determinados datos erroneos, debido a las inducciones electromagnéticas creadas por la central, además, el sistema de control posee una etapa previa al convertidor A/D que filtra las señales.

Para la adquisición de los datos de estado de líneas y generadores, que se hacen a través de la entrada digital, se

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

utiliza una etapa previa que consiste en una tarjeta del tipo PCLD - 782 de U.S. Technologies, que nos proporciona 16 canales de entrada optoacoplados, lo cual también nos aislará de posibles inducciones electromagnéticas, cuya salida atacará al D/I de la PCL - 712.

Como interface entre el D/O de nuestro sistema de control y la central, se utilizará la tarjeta PCLD - 785 de la misma marca, que proporciona 16 relés de salida capaces de conmutar en un tiempo inferior a los 10 milisegundos.

A continuación se muestran las hojas de especificaciones de las tarjetas PCL - 712, PCLD - 782 y PCLD - 785.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

Model PCL-712

Multi-Lab Card (12 bit)A/D+D/A+DIO

Part No. 00712-90001
Printed in Taiwan

1. INTRODUCTION TO THE PCL-712

1.1. General Description

The PCL-712 Multi-Lab card is a multifunction analog/digital I/O card that offers 5 most desirable measurement and control functions on a single board.

Its versatile functions include 16 single ended analog inputs, 2 analog outputs, 16 digital inputs, 16 digital outputs and a programmable interval timer.

Designed with affordability and versatility in mind, this full size card offers 12 bit resolution for both D/A and A/D conversion and turns the IBM PC* into a powerful data acquisition instrument. In addition to its highly condensed features, the functions of the card can be further enhanced with the use of optional daughter cards such as PCLD-780, PCLD-782 and PCLD-785.

The PCLD-785 is a 16-channel relay output card which can be driven by the digital output of the PCL-712 card. The PCLD-782 is a 16-channel opto-isolated digital input card which provides an easy way to input digital data to the PCL-712. The PCLD-780 wiring terminal board allows analog and/or digital I/O's be easily connected to the PCL-712. All three daughter boards can be connected to the PCL-712 via 20 pin flat ribbon cables.

As a further convenience in your application, the PCL-712 card is also supported by the PC-LabDAS software, a handy and friendly menu driven data acquisition software package.

With all the features and supports together, the PCL-712 truly provides a total and low cost solution for your application need.

1.2. Features, Applications and Specifications

Features

- 16 Single ended analog input with 12 bit resolution
- 2 analog output with 12 bit resolution
- Hardware successive approximation for faster conversion time and higher throughput
- A/D conversion time less than 30 microseconds
- Jumper selectable voltage range
- 16 digital input channels
- 16 digital output channels
- All digital input and output channels are TTL compatible.
- Programmable interval timer

Applications

- Industrial measurements/automation
- Laboratory measurements/automation
- Signal analysis
- Process control/monitoring
- Contact closure monitoring

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

- Switch panel status sense
- Industrial on/off control
- BCD interface
- Digital I/O control
- Period and pulse width measurement
- Event and frequency counting

Specifications

Analog to digital
- Input Range : Bipolar +/-5V
- Input Channels : 16 single ended
- Accuracy : +/-0.2% at +/-5V range
  +/-0.3% at +/-1V range
- Input Impedance: > 10 mega Ohms
- Conversion time: < 30 microseconds
- Resolution : 12 bits

Digital to analog
- Output Range : 0V to 5V
- Output Channels: 2
- Accuracy : 0.1%
- Settling time : < 30 microseconds for 5V step
- Resolution : 12 bits

Digital Input
- Input Low Level : Min. -0.5V, max. 0.8V
- Input High Level: Min. 2.0V, max. 5.0V
- Input Loading : 0.2 mA at 0.4V
- Input Hysteresis: Typical 0.4V, min. 0.2V

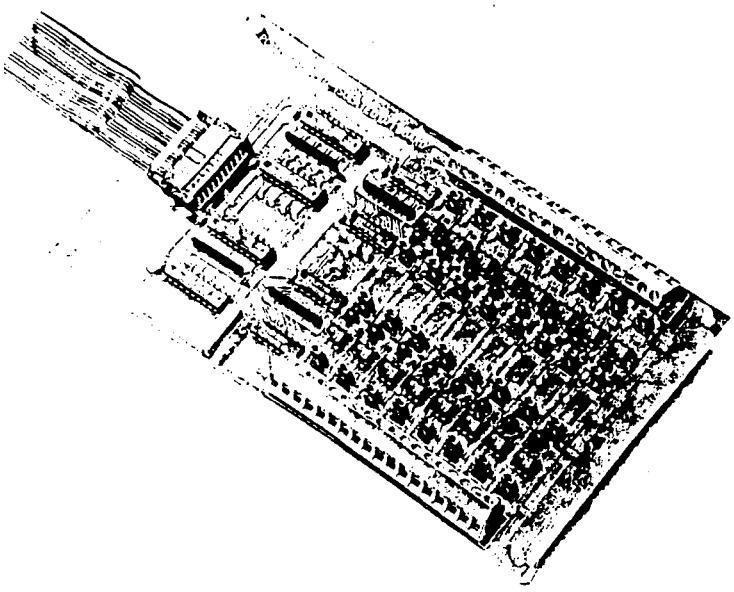
Digital Output
- Output Low Level : Max. 0.5V at 8 mA
  (sink) Max. 0.4V at 4 mA
- Output High Level: Min. 2.7V at 0.4 mA
  (source)

Power Consumption : < 800 mA at 5V
< 50 mA at +12V
< 50 mA at -12V

General
- Dimensions : 13 3/8" x 3 3/4"
  34 cm x 9.5 cm
- Bus : IBM PC bus
- Slot : One 62-pin slot
- I/O Port Base Address :
  Hex 200 - hex 3F0
- I/O Port Space : 12

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

PCLD-782 16 Channel Opto-isolated D/I Board



INTRODUCTION

The PCLD-782 16 Channel Opto-Isolated Digital Input Board is a daughter board to be used with several PC-LabCards which contain the digital input ports (e.g. the PCL-712 Digital I/O card). The PCLD-782 provides screw terminals and opto-isolators for singal connection and conditioning. The opto-isolators allow the input signals be completely floated and prevent the ground loops.

Each input channel is equipped with one red LED to indicate the Hi/Lo status of input signal, and is buffered with voltage comparator to allow flexible signal conditioning. User may select the isolated or non-isolated modes for each input by changing the on-board jumper setting.

APPLICATIONS

- Digital signal sensing
- Switch contact status monitoring
- Limit switch monitoring

FEATURES

- **Compatible with digital input port of PCL-712, PCL-714 and PCL-720**
- **16 Opto-isolated digital input channels**
- **On-board signal conditioning circuits**
- **Built-in screw terminals for easy wiring**
- **LEDs indicate input logic status**
- **Inputs buffered with voltage comparators**

SPECIFICATIONS

- **Type of opto-isolator:** 4N25
- **Input channels:** 16
- **Input current:** 80mA max. for isolated input. Inputs buffered by voltage comparators.
- **Threshold voltage:** 1.5 VDC, adjustable by changing voltage dividing resistors.
- **Input mode:** Isolated or non-isolated digital input (jumper selectable)
- **Withstanding Voltage:** 1500 VDC
- **Screw terminal:** Accept #22 te #12 AWG wire
- **Connector for digital bus:** 20 pin flat cable connector
- **Size:** 20.5 cm (L) x 11.43 cm (W) or 8.07" (L) x 4.5" (W)

ORDERING INFORMATION

☐ **PCLD-782** 16 Channel Opto-Isolated Digital Input Board.

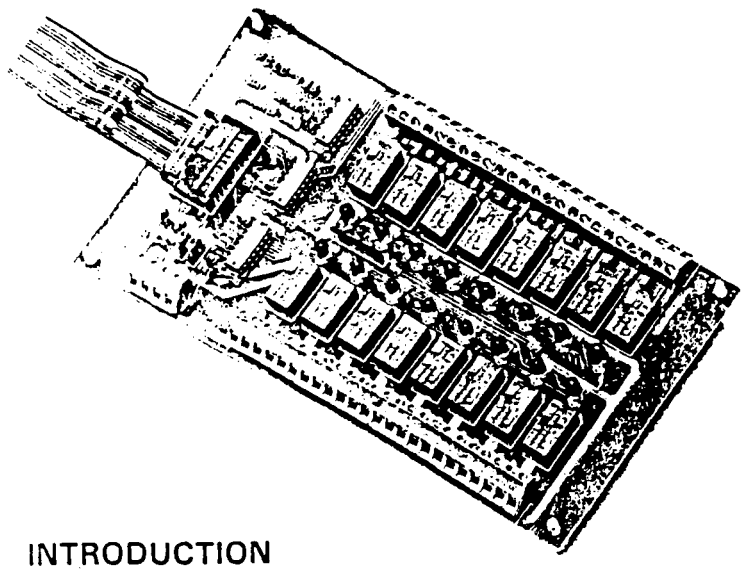
Accessories Furnished: One 1 meter 20-pin flat cable assembly (P/N PCL-10420-1), nylon standoffs for table-top mounting, screws and washers for panel mounting.

Accessories Available:

☐ PCL-10420-1: 20 pin flat cable assembly, 1 m
☐ PCL-10420-2: 20 pin flat cable assembly, 2 m
☐ PCL-10420-0.7: 20 pin flat cable assembly, 0.7 m

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

PCLD-785 16 Channel Relay Output Board



INTRODUCTION

The PCLD-785 Relay Output Board provides 16 electromechanical SPDT relays driven by 16 bit digital output port of several PC-Lab Cards. The relays can be used for general switching purposes, extending the number of analog inputs, setting up test configurations, power switching etc. The normal open, normal close and common contacts of each relay are brought out to the screw connector strip. A red LED, adjacent to each relay, lights up when the relay is energized.

Each relay on the board consumes about 33mA when energized, so with all relays active the board takes about 0.53 A from the computer's +12V power bus. In this case you may have an overload problem to the computer's +12V supply. The PCLD-785 has a jumper switch allowing the user switch to external +12V source connected to the on-board screw connector.

APPLICATIONS

- Signal switching
- ON/OFF control of external devices
- Valve/solenoid control
- Drive external high power relays
- Activate alarms
- Annunciator control

FEATURES

- **Compatible with digital output port of PCL-712, PCL-714 and PCL-720**
- **16 single-pole-double-throw relays**
- **120V/1 Amp contact rating**
- **On-board relay driver circuits**
- **Built-in screw terminals for easy wiring**
- **LED indicators to show activated relays**
- **Complete accessories for table-top, panel or wall mounting**

SPECIFICATIONS

- **Relay type:** SPDT (Form C)
- **Contact rating:** 120V AC/DC, 1A
- **Breakdown voltage:** 500V AC/DC minimum
- **Relay on time:** 3 msec typical
- **Relay o f time:** 2 msec typical
- **Total switching time:** 10 msec typical
- **Insulation Resistance:** 100 Mohm minimum
- **Life expectancy:** >5 million operations at full load
- **Connector for digital bus:** 20 pin flat cable connector
- **Screw terminal:** Accept #22 — #12 AWG wire
- **Power consumption:** +12V: 33mA for each relay, total 0.53A if all relays are energized. +5V: <0.2A
- **Power supply:** Use +12V power from computer bus or external power supply, selectable by on-board jumpers.
- **Size:** 20.5 cm (L) x 11.43 cm (W) or 8.07" (L) x 4.5" (W)

ORDERING INFORMATION

☐ **PCLD-785: 16 Channel Relay Output Board**

Accessories Furnished: One 1 meter 20-pin flat cable assembly (P/N PCL-10420-1), nylon standoffs for table-top mounting, screws and washers for panel mounting.

Accessories Available:

☐ PCL-10420-1: 20 pin flat cable assembly, 1 m
☐ PCL-10420-2: 20 pin flat cable assembly, 2 m
☐ PCL-10420-0.7: 20 pin flat cable assembly, 0.7 m

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A.5 VERIFICACIONES DEL DISPOSITIVO DE CONTROL

Los puertos de E/S de la PCL-712, poseen una dirección base, seleccionable por un conjunto de cinco interruptores, que puede ir desde 200H hasta 3F0H. Nosotros optaremos por utilizar la dirección base 220H, que además, es la que viene preseleccionada de fábrica.

A.5.1 CONVERSION A/D

Tal y como se puede ver en las especificaciones anteriores, la PCL-712 nos proporciona 16 canales de entrada analógica convertibles a un valor digital con una resolución de 12 bits.

Para acceder a una conversión A/D, en primer lugar, debemos seleccionar el canal del que pretendemos conocer su tensión, lo cual lo hacemos a través de los cuatro bits menos significativos del puerto BASE+10.

Una vez seleccionado el canal, se debe disparar la conversión, poniendo a 1 el bit 0 del puerto BASE+11.

Realizada esta secuencia, se podrá leer el byte bajo de la conversión en BASE+4 y los 4 bits restantes en los bits (0-3) del BASE+5, siempre que el bit 4 del BASE+5, sea igual a 1, en caso contrario, que sea 0, nos indicará que la conversión no está disponible, por lo que deberemos hacer una nueva lectura.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A continuación, se presenta un módulo ejemplo, que no sólo ilustra la adquisición de datos analógicos con la PCL-712, si no que además sirve para hacer los ajustes de offset, así como la calibración a escala completa de los conversores A/D.

Modula-2/86 offsetp.MOD

```
(* El siguiente programa muestrea los 16 canales A/D con una A/I  de ±1 V. *)
(* y visualiza los resultados. También permite ajustar tanto el offset como*)
(* la calibración a escala completa del ADC *)
MODULE TriggerMode;
  FROM Util IMPORT Cls,PxPy;
  FROM RealInOut IMPORT WriteReal;
  FROM Terminal IMPORT KeyPressed;
  FROM SYSTEM IMPORT INBYTE, OUTBYTE;
  FROM RealConversions IMPORT RealToString;
  FROM InOut IMPORT Write, WriteCard, WriteString, Read;
  CONST
    BASE = 220H;
  VAR
    V:REAL;
    ch:CHAR;
    resul:BOOLEAN;
    HI, LO, x:CARDINAL;
    str:ARRAY [0..15] OF CHAR;
  (*-------------------------------*)
   PROCEDURE Ret; (*Retardo*)
   VAR
     Stuff,Count:CARDINAL;
   BEGIN
     FOR Count:=1 TO 6500 DO
        Stuff:=32200 DIV Count;
     END;
   END Ret;
  (*-------------------------------*)
BEGIN
  Cls;
  REPEAT
     FOR x:=0  TO 15 DO
        OUTBYTE (BASE+10,x);  (*Selecciona el canal*)
        OUTBYTE (BASE+11,1);  (*Dispara la conversión*)

        REPEAT                (* Loop hasta que la conversión esté disponible*)
          INBYTE (BASE+5,HI); (*Byte alto*)
        UNTIL HI < 16;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
39
40          INBYTE (BASE+4,LO);    (*Byte bajo*)
41
42          V:=(FLOAT(HI)*256.0+FLOAT(LO)-2048.0)*2.0/4096.0;
43
44          PxPy(0, 1+x);WriteString('A/D'); WriteCard(x,0); Write('=');
45
46          RealToString(V,3,7,str,resul);
47          IF resul THEN WriteString(str) END;
48          Ret;
49       END;
50    UNTIL KeyPressed();
51    Read(ch);
52
53     (*Esta segunda porción de código, nos permite hacer tanto el ajuste
54       del OFFSET, así como la calibración a escala completa,de una forma
55       más rigurosa *)
56
57     (* OFFSET--> Con entrada 0 debe visualizarse un valor entre  2047 y
58      2049 ajustando VR4 *)
59
60     (* ESCALA COMPLETA--> Aplicando +1V al canal A/D15, debe obtenerse
61        una lectura entre 4094 y 4095 ajustando VR5.*)
62
63    OUTBYTE (BASE+10,15);  (*Selecciona el canal 15*)
64    REPEAT
65       OUTBYTE (BASE+11,1);  (*Dispara la conversión del canal 15*)
66
67       REPEAT                (* Loop hasta que la conversión esté disponible*)
68         INBYTE (BASE+5,HI); (* Byte alto*)
69       UNTIL HI < 16;
70
71       INBYTE (BASE+4,LO);    (*Byte bajo*)
72
73       PxPy(30,20); WriteString('A/D15='); WriteCard(HI*256+LO,0);(*Visualiza el dato*)
74
75       Ret;
76    UNTIL KeyPressed();
77    Read(ch);
78 END TriggerMode.
```

```
Modula-2/86 Compiler Version V 2.00

==>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A.5.2 E/S DIGITAL

La palabra de 16 bits presenta a la entrada digital, puede ser leída en los puertos BASE+7 y BASE+6, fragmentada en su byte más significativo y menos significativo respectivamente.

La transmisión de un valor a cualquiera de los 16 canales de la palabra de salida digital, se hace a través de la escritura del MSB y LSB del valor deseado en los puertos BASE+14 y BASE+13.

El siguiente módulo ejemplo, nos permite visualizar los datos que enviamos a los canales de salida digital, así como los que recepcionamos por los canales de entrada digital.

Modula-2/86 relesp.MOD

```
(* Programa que permite visualizar los datos que enviamos
   a D/O, asi como los que recibimos po D/I.          *)
MODULE EntradaSalidaDigital;
  FROM Terminal IMPORT KeyPressed;
  FROM Util IMPORT Cls, PxPy, Neg, Vv;
  FROM InOut IMPORT Write, WriteHex, WriteCard, WriteString;
  FROM SYSTEM IMPORT INBYTE, OUTBYTE, SETREG, AX ,DX, BX, SWI;
  CONST
    BASE = 220H;
  VAR
    x,ALTO,BAJO,y,HI,LO:CARDINAL;
(* ------------------------------------------------------------ *)
PROCEDURE Bo;          (* Oculta el cursor *)
BEGIN
  SETREG (DX,1900H);
  SETREG (AX,0200H);
  SETREG (BX,0H);
  SWI(10H);
END Bo;
(*------------------------------------------------------------------*)
  PROCEDURE Palabra (m:CARDINAL);   (* Visualiza palabra *)
  VAR                               (* en forma binaria  *)
    z:BITSET;
    n:CARDINAL;
  BEGIN
    z:=BITSET(m);
    FOR n:=15 TO 0 BY -1 DO
      IF (n) IN (z) THEN
        Write('1')
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
30        ELSE
31          Write('0');
32        END;
33        IF n MOD 4 = 0 THEN (*Espacio cada 4 bits para facilitar la lectura*)
34          Write(' ')
35        END;
36      END;
37    END Palabra;
38  (*-------------------------------------------------------------------*)
39    PROCEDURE Ret(zp5:CARDINAL);   (*Retardo*)
40    VAR
41      Stuff,Count:CARDINAL;
42    BEGIN
43      FOR Count:=1 TO zp5 DO
44        Stuff:= 32000 DIV Count;
45      END;
46    END Ret;
47  (*-------------------------*)
48  BEGIN
49    Cls;
50    FOR x:=0 TO 65535 DO
51      IF KeyPressed() THEN x:=65535 END; (* Para salir y ver si cae la tensión *)
52
53      (* Visualizamos en Pantalla el dato que enviamos al D/O *)
54      PxPy(28,12); Neg; WriteString('D/O= '); Vv; Palabra(x); Bo;
55      ALTO := x DIV 256; BAJO :=  x MOD 256;
56      OUTBYTE (BASE+14,ALTO); OUTBYTE (BASE+13,BAJO); (*Lo mandamos*)
57
58      Ret(1000); (* Retardo para que de tiempo a la adquisición del D/I *)
59
60      (* Leemos y visualizamos el valor del D/I *)
61      INBYTE (BASE+7,HI); INBYTE (BASE+6,LO); (*Leemos del D/I*)
62      y:=HI*256+LO;
63      PxPy(28,13); Neg; WriteString('D/I= '); Vv; Palabra(y); Bo;
64
65      Ret(26600); (* dos segundos de retardo *)
66    END;
67    Ret(53200);
68
69     (* Leemos estados de la E/S/D antes de abandonar el programa *)
70    x:=0;
71    PxPy(28,12); Neg; WriteString('D/O= '); Vv; Palabra(x);
72    OUTBYTE (BASE+14,x); OUTBYTE (BASE+13,x); (*Ponemos a 0 D/O*)
73
74    Ret(1000); (* Retardo para que de tiempo a la adquisición del D/I *)
75
76    INBYTE (BASE+7,HI); INBYTE (BASE+6,LO); (*Leemos el D/I*)
77    y:=HI*256+LO;
78    PxPy(28,13); Neg; WriteString('D/I= '); Vv; Palabra(y); (* Leemos el D/I *)
79  END EntradaSalidaDigital.
```

```
Modula-2/86 Compiler Version V 2.00

=>     0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A continuación se presenta un nuevo módulo de programa que nos permite, inspeccionando la D/O:0 con la ayuda de un osciloscopio, determinar el tiempo que tarda la conversión de los datos junto con la asignación de éstos a una variable.

El tiempo que esté a nivel bajo la D/O:0, nos indicará el tiempo que tarda en realizarse la conversión, con lo que se podrá observar que ésta, es de aproximadamente de 40 microsegundos.

Modula-2/86 tconverp.MOD

```
(* El siguiente programa, y la inspección con la ayuda de
   un osciloscopio, del D/O:0, nos permite determinar el
   tiempo de conversión de los datos.                    *)
MODULE TiempoDeConversion;
  FROM Util IMPORT Cls;
  FROM Terminal IMPORT KeyPressed;
  FROM SYSTEM IMPORT INBYTE, OUTBYTE;
  CONST
    BASE = 220H;
  VAR
    V:ARRAY [0..15] OF CARDINAL;
    HI, LO, x:CARDINAL;
BEGIN
  Cls;
REPEAT
  FOR x:=0  TO 15 DO
    OUTBYTE (BASE+13,0);  (* Sacamos un 0 por D/O:0  *)

    OUTBYTE (BASE+10,x);  (*Selecciona el canal*)
    OUTBYTE (BASE+11,1);  (*Dispara la conversión*)

    REPEAT                (*Loop hasta que la conversión esté disponible*)
      INBYTE (BASE+5,HI); (*Byte alto*)
    UNTIL HI < 16;

    INBYTE (BASE+4,LO);   (*Byte bajo*)

    V[x]:=HI*256+LO;

    OUTBYTE (BASE+13,1); (* Sacamos un 1 por D/O:0  *)
  END;
UNTIL KeyPressed();
END TiempoDeConversion.
```

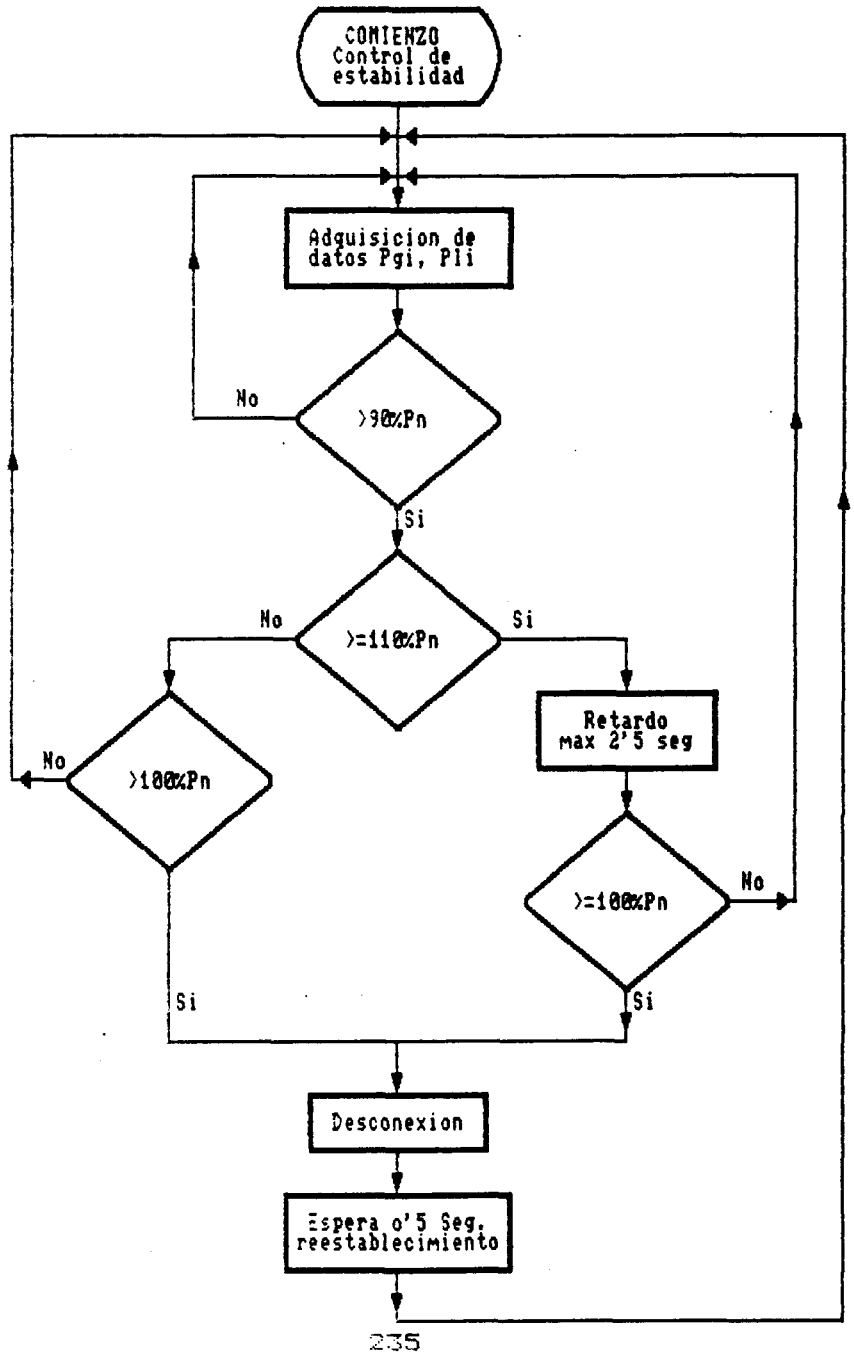
```
Modula-2/86 Compiler Version V 2.00

=>    0 Error(s) found
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A.6 IMPLEMENTACION SOFTWARE

Este apartado describe con flujogramas, los criterios de actuación descritos con anterioridad y que han sido implementados en nuestro programa de control.

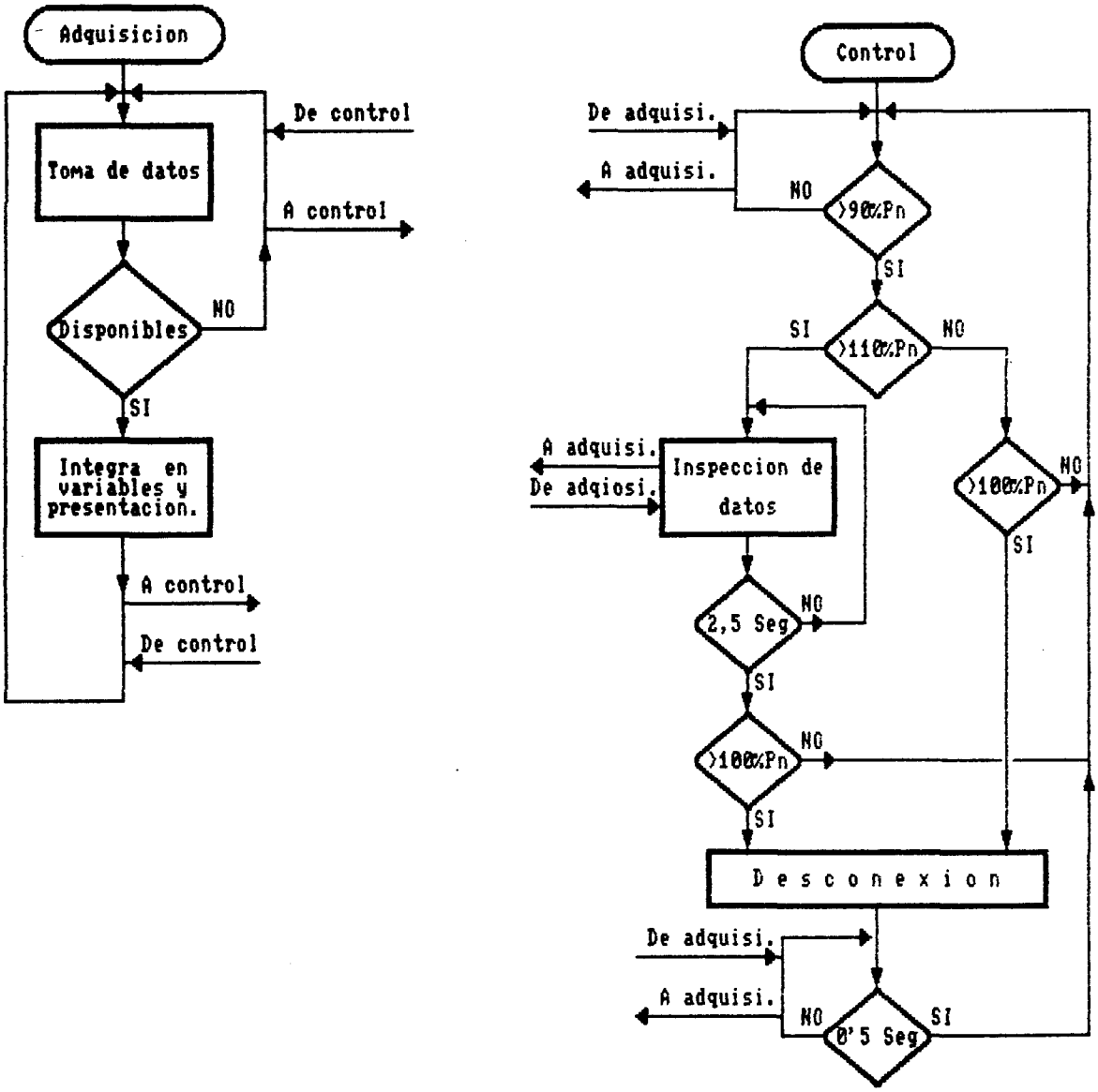
Descendiendo de nivel respecto al diagrama de bloques de la página 221, y siguiendo una metodología "TopDown",



© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

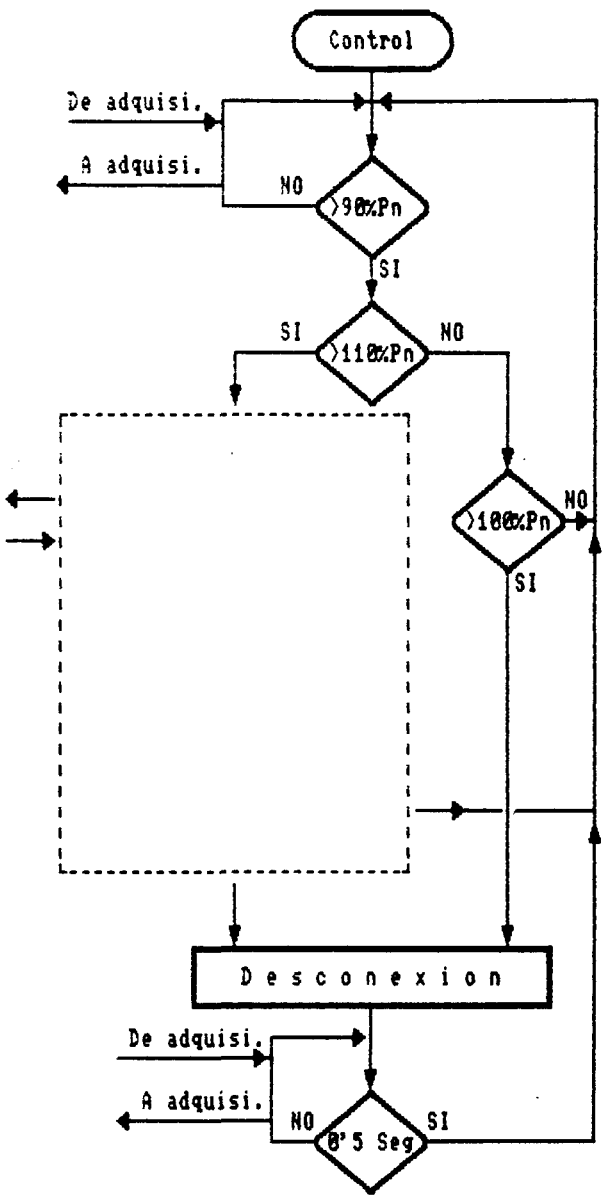
reflejamos en este primer flujograma, los criterios de actuación de una forma menos genérica.

Teniendo en cuenta que para la implementación del software de control, se ha utilizado el recurso de la concurrencia entre procesos, que MODULA - 2 proporciona. El siguiente flujograma pretende reflejar la conmutación entre los dos procesos más relevantes de nuestro programa. El proceso de adquisición de datos y el de control.

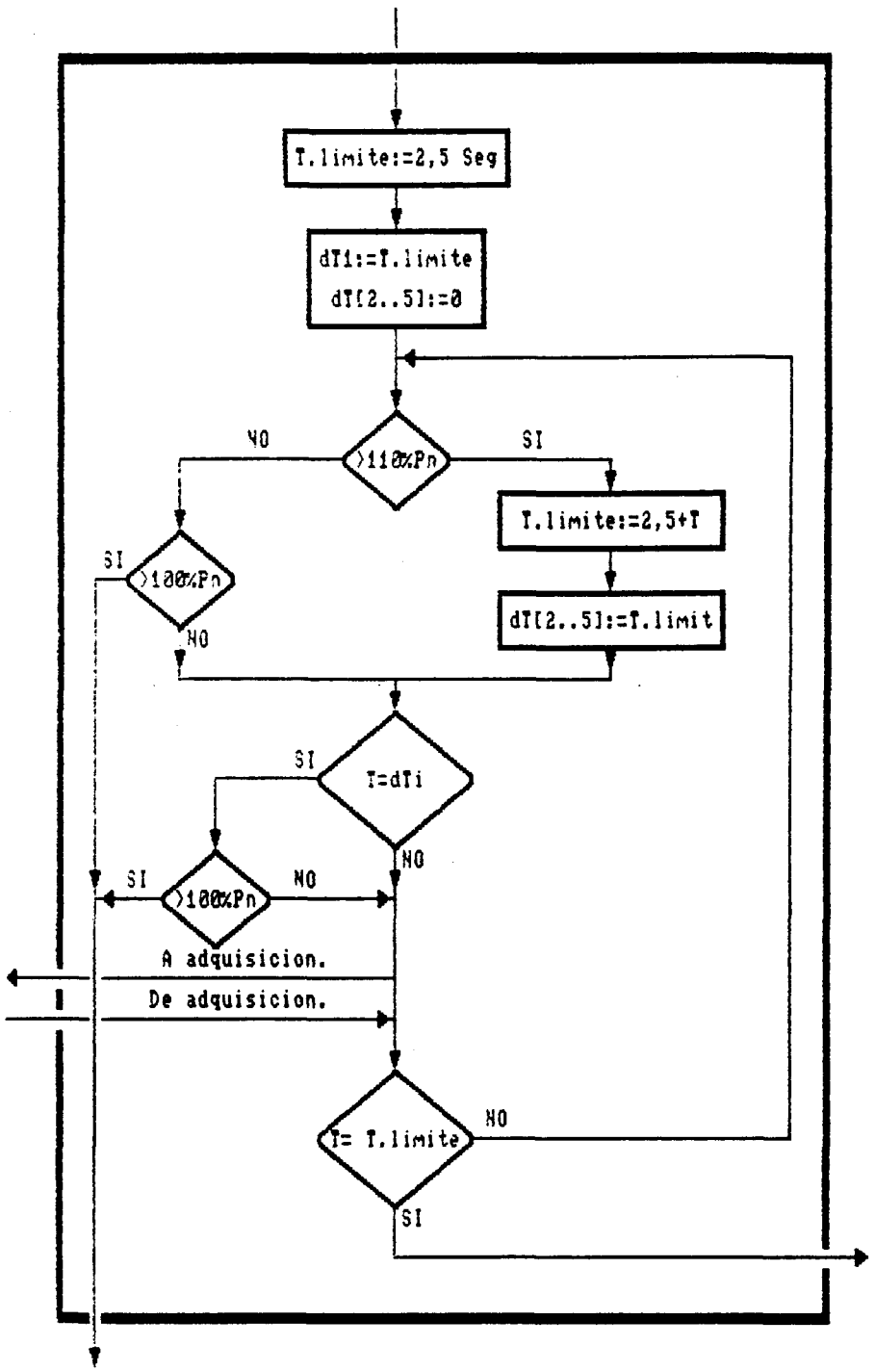


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

A continuación de desarrolla de una forma más concreta la porción omitida en el siguiente flujograma del proceso de control.



© Del documento, los autores. Digitalización realizada por ULPGC. Bibliotecas Universitaria, 2006



© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
MODULE UNELCO;
FROM Lineas IMPORT LineD, Line;
FROM RealInOut IMPORT WriteReal;
FROM Terminal IMPORT KeyPressed;
FROM RealConversions IMPORT RealToString;
FROM InOut IMPORT Write, Read, WriteString, WriteCard;
FROM Processes IMPORT WAIT, SEND, SIGNAL, StartProcess, Init;
FROM Util IMPORT Cls, Cll, PxPy, Neg, Tilt, Vi, Vv, Bs, Sp, Cud, Cuu;
FROM SYSTEM IMPORT DOSCALL, GETREG, SETREG, INBYTE, OUTBYTE, OUTWORD, AX ,DX, BX, CODE, SWI;

CONST
   BASE = 220H;
   Ls1 = 'LINEA 1'; Ls2 = 'LINEA 2'; Ls3 = 'LINEA 3'; Ls4 = 'LINEA 4';
   Ls5 = 'LINEA 5'; Ls6 = 'LINEA 6'; Ls7 = 'LINEA 7'; Ls8 = 'LINEA 8';
   Ls9 = 'LINEA 9'; Ls10 = 'LINEA10'; Ls11 = 'LINEA11';
   Gs1 = 'GENERADOR 1:'; Gs2 = 'GENERADOR 2:'; Gs3 = 'GENERADOR 3:';
   Gs4 = 'GENERADOR 4:'; Gs5 = 'GENERADOR 5:';

VAR
   ch:CHAR;
   t, co:SIGNAL;

   PG:ARRAY [1..5] OF CARDINAL;  PL:ARRAY [1..11] OF CARDINAL; (* Potencias G y L*)

   PNG:ARRAY [1..5] OF CARDINAL; PNL: CARDINAL;  (* Potencias nominales *)

   VG: ARRAY [1..5] OF BOOLEAN;  VL: ARRAY [1..11] OF BOOLEAN; (* Vigilan G y L*)

   Dif: BOOLEAN; (* Vigila IPg y IPl *)
   SPg,SPl: REAL;(* IPg y IPl *)

   SalidaReal: ARRAY [0..15] OF CHAR; (* Para el procedimiento RealToString *)
   OkReal: BOOLEAN;

   ESTADO:BITSET; (*Contendrá el estado de las lineas y generadores ON OFF*)
   EG: ARRAY [1..5] OF BOOLEAN;  EL: ARRAY [1..11] OF BOOLEAN; (* Vigilan Estado G y L*)

   ALTO,BAJO:CARDINAL; (*Bytes del D/O*)

   PantallaParada:BOOLEAN;(*Para poder retener los datos en pantalla con control*)

   HoraMinuto,SegundoMiliseg:CARDINAL;(*Para leer y asignar hora al sistema*)

   Pitando,Pitidos:BOOLEAN; (*Para los beep*)
(* ------------------------------------------------------------ *)
PROCEDURE INCREMENTAR (VAR suma:CARDINAL);
BEGIN
   IF suma # 65534 THEN
      suma := suma + 1;
   ELSE
      suma :=0;
   END;
END INCREMENTAR;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
54 (* ---------------------------------------------------------- *)
55 PROCEDURE DECREMENTAR (VAR resta:CARDINAL);
56 BEGIN
57    IF resta # 0 THEN
58       resta := resta - 1;
59    ELSE
60       resta := 65534;
61    END;
62 END DECREMENTAR;
63 (* ---------------------------------------------------------- *)
64 PROCEDURE CambiarPNG;
65 TYPE
66   DelTipoProcedimiento = PROCEDURE (VAR CARDINAL);
67 VAR
68    SN:CHAR;
69    Cteclado:CARDINAL;
70    (* ---------------------------------------------------------- *)
71    PROCEDURE AlterarPNG;
72    VAR
73       Kontador:[0..4];
74       PROCEDURE QuitaPone(PuntaDeFlecha:CHAR);     (* Quita ó Pone > *)
75       BEGIN
76          CASE Kontador OF
77             0: PxPy(32,8); Write (PuntaDeFlecha); |
78             1: PxPy(32,10); Write (PuntaDeFlecha );|
79             2: PxPy(32,12); Write (PuntaDeFlecha );|
80             3: PxPy(32,14); Write (PuntaDeFlecha );|
81             4: PxPy(32,16); Write (PuntaDeFlecha );
82          END; Bo;
83       END QuitaPone;
84       (* ---------------------------------------------------------- *)
85       PROCEDURE IncDec (INCDEC: DelTipoProcedimiento );  (* Incrementa ó  Decrementa *)
86          (* ---------------------------------------------------------- *)
87          PROCEDURE IncrementaDecrementa(VAR Elemento:CARDINAL; EnX, EnY :CARDINAL);
88          BEGIN
89             REPEAT
90                REPEAT
91                   INCDEC (Elemento);
92
93                   PxPy(EnX,EnY); WriteCard(Elemento,5); Bo;
94
95                   IF Cteclado = 5200H THEN RETURN END; (*Si Ins, no entra en el lazo*)
96                UNTIL KeyPressed();
97                Cteclado := Inspecciona();
98                IF Cteclado = 011BH THEN DOSCALL(4CH,0FFH) END; (* Esc => Salir al DOS *)
99                IF Cteclado = 5200H THEN Cteclado := 0 END; (* Para que Ins, no controle el lazo *)
100            UNTIL (Cteclado = 4A2DH) OR (Cteclado = 4E2BH); (* - ó + *)
101         END IncrementaDecrementa;
102         (* ---------------------------------------------------------- *)
103      BEGIN
104         CASE Kontador OF
105            0: IncrementaDecrementa (PNG[1],40,8); |
106            1: IncrementaDecrementa (PNG[2],40,10); |
107            2: IncrementaDecrementa (PNG[3],40,12); |
108            3: IncrementaDecrementa (PNG[4],40,14); |
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
109             4: IncrementaDecrementa (PNG[5],40,16);
110           END;
111       END IncDec;
112       (* ------------------------------------------------------------ *)
113   BEGIN
114     Kontador:=0;
115     QuitaPone('>');
116     REPEAT
117       Cteclado:=Inspecciona();
118       CASE Cteclado OF
119         (* | *)  4800H: QuitaPone(' ');
120                         IF Kontador = 0 THEN
121                           Kontador:= 4
122                         ELSE
123                           DEC(Kontador)
124                         END;
125                         QuitaPone('>'); |
126
127         (* | *) 5000H: QuitaPone(' ');
128                         IF Kontador = 4 THEN
129                           Kontador:= 0
130                         ELSE
131                           INC(Kontador)
132                         END;
133                         QuitaPone('>'); |
134
135         (* + *) 4E2BH: IncDec (INCREMENTAR); |
136
137         (* - *) 4A2DH: IncDec (DECREMENTAR); |
138
139       (* Ins *) 5200H: IncDec (INCREMENTAR); |
140
141       (*Esc *)  011BH: Cls; DOSCALL(4CH,0FFH); (* Salir al DOS *)
142       ELSE
143       END;
144     UNTIL  (Cteclado = 1C0DH); (* EOL *)
145   END AlterarPNG;
146   (* ------------------------------------------------------------ *)
147 BEGIN
148   PxPy(20,4); Neg; WriteString('Las potencias nominales de los generadores');
149   PxPy(26,5);      WriteString('predefinidas en el sistema son:');
150
151   PxPy(38,7); Write(332C); Line(0,8); Write(277C); Cud(1); Bs(1); Line(6,9);
152               Write(331C); Bs(2); Line(4,8); Write(300C); Cuu(1); Bs(1); Line(2,9);
153
154   PxPy(28,19); WriteString('¿Quiere Cambiarlas? (S/N)'); Vv;
155
156   PxPy(33,8); Neg; WriteString('PNG1:'); Vv; Sp(2); WriteCard(PNG[1],5);
157   PxPy(38,9); Neg; Write(303C); Line(0,8); Write(264C); Vv;
158
159   PxPy(33,10); Neg; WriteString('PNG2:'); Vv; Sp(2); WriteCard(PNG[2],5);
160   PxPy(38,11); Neg; Write(303C); Line(0,8); Write(264C); Vv;
161
162   PxPy(33,12); Neg; WriteString('PNG3:'); Vv; Sp(2); WriteCard(PNG[3],5);
163   PxPy(38,13); Neg; Write(303C); Line(0,8); Write(264C); Vv;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
164
165     PxPy(33,14); Neg; WriteString('PNG4:'); Vv; Sp(2); WriteCard(PNG[4],5);
166     PxPy(38,15); Neg; Write(303C); Line(0,8); Write(264C); Vv;
167
168     PxPy(33,16); Neg; WriteString('PNG5:'); Vv; Sp(2); WriteCard(PNG[5],5); Bo;
169
170     REPEAT
171        Read(SN);
172     UNTIL (CAP(SN) = 'B') OR (CAP(SN) = 'N') OR (SN = 33C) OR (SN = 36C); (*Esc,EOL*)
173
174     IF CAP(SN) = 'B' THEN
175        PxPy(0,19); Cll;
176        PxPy(17,20);Neg;Write('┌');LineD(0,46);Write('┐');Cud(1);Bs(1);LineD(6,1);Write('J');
177        Bs(2);LineD(4,46);Write('L');Cuu(1);Bs(1);LineD(2,1);Vv;
178
179        PxPy(18,21);Neg;Write(30C);Write(31C);Write(' ');Write(21C);Write('-');Write('J');
180        Write(':');Vv;WriteString('Seleccionar; ');
181
182        Neg;WriteString('+ - Ins:');Vv;WriteString('Alterar; ');
183        Neg;WriteString('Esc:');Vv;WriteString('Salir');Bo;
184
185        AlterarPNG;
186     ELSIF  SN = 33C  THEN
187        Cls; DOSCALL(4CH,0FFH);
188     ELSE
189     END;
190     Cls;
191  END CambiarPNG;
192  (* ------------------------------------------------------------------ *)
193  PROCEDURE Inicializaciones;  (* Inicializaciones del programa *)
194  VAR
195    x:CARDINAL;
196  BEGIN
197    FOR x:=1 TO 5 DO  (* Inicialización de las potencias nominales de los G y L*)
198      PNG[x]:= 9400;
199    END;
200    PNL:=9400;
201
202    FOR x:=1 TO 5 DO  (* Inicialización de las potencias de los G y L*)
203      PG[x]:= 9000;
204    END;
205    FOR x:=1 TO 11 DO
206      PL[x]:= 4000;
207    END;
208
209    FOR x:=1 TO 5 DO (* Inicialización para que el flashear de los G y L*)
210      VG[x]:= FALSE; (* se corrija sólo en el caso necesario *)
211    END;
212    FOR x:=1 TO 11 DO
213      VL[x]:= FALSE;
214    END;
215
216    Dif:=FALSE;      (* "   "   "   "   "   "   "   "     IPg y  IPl *)
217
218    FOR x:=1 TO 5 DO (* "   "   "   "   "   "   "   "      ON y  OFF *)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
219      EG[x]:= TRUE;
220    END;
221    FOR x:=1 TO 11 DO
222      EL[x]:= TRUE;
223    END;
224
225    PantallaParada:=FALSE;
226
227    ESTADO:=BITSET(65535); (* Inicializamos la palabra de estado (Todo en ON) *)
228     ALTO:=CARDINAL(ESTADO) DIV 256; BAJO:=CARDINAL(ESTADO) MOD 256;
229     OUTBYTE(BASE+14,ALTO); OUTBYTE(BASE+13,BAJO);
230
231    Pitidos:=TRUE; Pitando:=FALSE;
232
233  END Inicializaciones;
234  (* ------------------------------------------------------------------------ *)
235  PROCEDURE Inspecciona():CARDINAL;      (* Lee Código del teclado *)
236  VAR
237    codigo:CARDINAL;
238  BEGIN
239     CODE(0B4H,0,0CDH,16H); (*MOV (AH,0); INT(16H)*)
240     GETREG(AX,codigo);
241     RETURN codigo;
242  END Inspecciona;
243  (* ------------------------------------------------------------------------ *)
244  PROCEDURE Bo;          (* Oculta el cursor *)
245  BEGIN
246     SETREG (DX,1900H);
247     SETREG (AX,0200H);
248     SETREG (BX,0H);
249     SWI(10H);
250  END Bo;
251  (* ------------------------------------------------------------------------ *)
252  PROCEDURE BEEP;
253  VAR
254    PPI:CARDINAL;
255    resultado:BITSET;
256  BEGIN
257    INBYTE(61H,PPI);                    (*ACTIVA*)
258    resultado:= BITSET(PPI) + BITSET(03H);
259    OUTBYTE(61H,CARDINAL(resultado));
260
261    OUTBYTE(43H,0B6H);                  (*PROGRAMA*)
262
263    OUTWORD(42H,10);                    (*CARGA PERIODO*)
264    OUTWORD(42H,10);
265  END BEEP;
266  (* ------------------------------------------------------------------------ *)
267  PROCEDURE NOBEEP;
268  VAR
269    PPI:CARDINAL;
270    resultado:BITSET;
271  BEGIN
272    INBYTE(61H,PPI);                    (*DESACTIVA*)
273    resultado:= BITSET(PPI) * BITSET(0FCH);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
274    OUTBYTE(61H,CARDINAL(resultado));
275  END NOBEEP;
276  (* ------------------------------------------------------------------------ *)
277  PROCEDURE Pantalla;   (* ******Pantalla****** *)
278
279     PROCEDURE KVA (SP:CARDINAL);  (* Posiciona KVA *)
280     BEGIN
281     Sp(SP); WriteString('KVA');
282     END KVA;
283
284     PROCEDURE Lin(l:CARDINAL);  (* Posiciona Linea *)
285     BEGIN;
286     Write(303C); Line(0,1); Write(264C);
287     END Lin;
288
289     PROCEDURE Ventana (Px, Py, Base, Altura:CARDINAL);          (* Ventanas *)
290     BEGIN
291        PxPy(Px,Py);
292        Write(332C); Line(0,Base-2); Write(277C);  Cud(1); Bs(1); Line(6,Altura-2);
293        Write(331C); Bs(2); Line(4,Base-2); Write(300C); Cuu(1); Bs(1); Line(2,Altura-2);
294        Bo;
295     END Ventana;
296
297  BEGIN
298     PxPy(0,0);
299     Neg; Write(311C); LineD(0,76); Write(273C);  Cud(1); Bs(1); LineD(6,23);
300     Write(274C); Bs(2); LineD(4,76); Write(310C); Cuu(1); Bs(1); LineD(2,23); Bo;
301
302     PxPy(28,1);  Write(313C);  PxPy(51,1); Write(313C);
303     PxPy(28,2); Write(272C); WriteString('CONTROL DE ESTABILIDAD'); Write(272C);
304     Sp(2); WriteString('IPg= '); Sp(7); WriteString('IP1= ');
305     PxPy(28,3);  Write(310C); LineD(0,22);  Write(274C); Bo;
306
307     Ventana(16,5,10,11); Ventana(37,5,10,11); Ventana(60,5,10,11);
308
309     Ventana(9,17,8,9); PxPy(9,25); Write(317C); LineD(0,6); Write(317C);
310     Ventana(20,17,6,9); PxPy(20,25); Write(317C); LineD(0,4); Write(317C);
311
312     Ventana(35,17,8,9); PxPy(35,25); Write(317C); LineD(0,6); Write(317C);
313     Ventana(46,17,6,9); PxPy(46,25); Write(317C); LineD(0,4); Write(317C);
314
315     Ventana(61,17,8,7); Ventana(72,17,6,7);
316
317     PxPy(3,6); WriteString(Gs1); KVA(11); KVA(18);
318     PxPy(3,8); WriteString(Gs2); KVA(11); KVA(18);
319     PxPy(3,10); WriteString(Gs3); KVA(11); KVA(18);
320     PxPy(3,12); WriteString(Gs4); KVA(11); KVA(18);
321     PxPy(3,14); WriteString(Gs5); KVA(11); KVA(18);
322
323
324     PxPy(2,18); WriteString(Ls1); KVA(8); Sp(8); WriteString(Ls5); KVA(8);
325     Sp(8); WriteString(Ls9); KVA(8);
326     PxPy(2,20); WriteString(Ls2); KVA(8); Sp(8); WriteString(Ls6); KVA(8);
327     Sp(8); WriteString(Ls10); KVA(8);
328     PxPy(2,22); WriteString(Ls3); KVA(8); Sp(8); WriteString(Ls7); KVA(8);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
329     Sp(8); WriteString(Ls11); KVA(8);
330     PxPy(2,24); WriteString(Ls3); KVA(8); Sp(8); WriteString(Ls8); KVA(8);
331
332
333     PxPy(17,4); WriteString('POTENCIA'); Sp(12); WriteString('P. NOMINAL');
334     Sp(15); WriteString('ESTADO');
335     PxPy(9,16); WriteString('POTENCIA'); Sp(3); WriteString('ESTADO');
336        Sp(9); WriteString('POTENCIA'); Sp(3); WriteString('ESTADO');
337        Sp(9); WriteString('POTENCIA'); Sp(3); WriteString('ESTADO');
338
339     PxPy(16,7); Lin(8); Sp(11); Lin(8); Sp(13); Lin(8);
340     PxPy(16,9); Lin(8); Sp(11); Lin(8); Sp(13); Lin(8);
341     PxPy(16,11); Lin(8); Sp(11); Lin(8); Sp(13); Lin(8);
342     PxPy(16,13); Lin(8); Sp(11); Lin(8); Sp(13); Lin(8);
343
344     PxPy(9,19); Lin(6); Sp(3); Lin(4); Sp(9); Lin(6); Sp(3); Lin(4); Sp(9); Lin(6); Sp(3); Lin(4);
345     PxPy(9,21); Lin(6); Sp(3); Lin(4); Sp(9); Lin(6); Sp(3); Lin(4); Sp(9); Lin(6); Sp(3); Lin(4);
346     PxPy(9,23); Lin(6); Sp(3); Lin(4); Sp(9); Lin(6); Sp(3); Lin(4); Vv; Bo;
347
348     PxPy(39,6); WriteCard(PNG[1],5); PxPy(39,8); WriteCard(PNG[2],5);
349     PxPy(39,10); WriteCard(PNG[3],5); PxPy(39,12); WriteCard(PNG[4],5);
350     PxPy(39,14); WriteCard(PNG[5],5); Bo;
351
352     PxPy(63,6);  WriteString(' ON'); PxPy(63,8);  WriteString(' ON');
353     PxPy(63,10); WriteString(' ON'); PxPy(63,12); WriteString(' ON');
354     PxPy(63,14); WriteString(' ON');
355                                  PxPy(21,18); WriteString(' ON');
356     PxPy(21,20); WriteString(' ON'); PxPy(21,22); WriteString(' ON');
357     PxPy(21,24); WriteString(' ON'); PxPy(47,18); WriteString(' ON');
358     PxPy(47,20); WriteString(' ON'); PxPy(47,22); WriteString(' ON');
359     PxPy(47,24); WriteString(' ON'); PxPy(73,18); WriteString(' ON');
360     PxPy(73,20); WriteString(' ON'); PxPy(73,22); WriteString(' ON');
361
362 END Pantalla;
363 (* ------------------------------------------------------------------------ *)
364   PROCEDURE Sincroniza;  (* Sincroniza los dos procesos *)
365   BEGIN
366     LOOP
367        WHILE NOT KeyPressed() DO
368          SEND(t);
369          SEND(co);
370        END;
371        CASE Inspecciona() OF
372           011BH: NOBEEP; Cls; DOSCALL(4CH,0FFH);! (* Esc => Quit *)
373           3B00H: IF PantallaParada = FALSE THEN  (*F1 => Control Pantalla*)
374                     PantallaParada:= TRUE;PxPy(2,2);WriteString('Pantalla OFF');Bo;
375                     DOSCALL(2DH,0,0,SegundoMiliseg);
376                  ELSE
377                     PantallaParada:= FALSE;PxPy(2,2); WriteString('            ');Bo;
378                  END;!
379           3C00H: IF Pitidos = TRUE THEN  (*F2 => Control Pitidos*)
380                     Pitidos:= FALSE; Pitando:=FALSE; NOBEEP;
381                     PxPy(2,3);WriteString(' Sonido OFF');Bo;
382                  ELSE
383                     Pitidos:=TRUE; Pitando:=FALSE;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
384                    PxPy(2,3); WriteString('          ');Bo;
385                  END;
386        ELSE
387        END;
388     END;
389   END Sincroniza;
390 (* ************************************************************************************************ *)
391 PROCEDURE TomaDeDatos;
392     (* ------------------------------------------------------------------------ *)
393     PROCEDURE VigilaD (A, B, Px, Py:CARDINAL; cadena:ARRAY OF CHAR; VAR Vig:BOOLEAN);
394     BEGIN                                  (*Vigila datos de lineas y generadores*)
395       IF (A > B) AND (Vig = FALSE) THEN
396         PxPy(Px,Py); Tilt; Vi; WriteString(cadena); Vv; Bo; Vig:=TRUE;
397       ELSIF  (A <= B) AND (Vig = TRUE) THEN
398         PxPy(Px,Py); Neg; WriteString(cadena); Vv; Bo; Vig:=FALSE;
399       END;
400     END VigilaD;
401     (* ------------------------------------------------------------------------ *)
402     PROCEDURE DeterminaError (SumaA, SumaB:REAL);(* Er max:A-B=A -->e=100% *)
403     VAR                                    (* Er ZX :  ?   -->e= ZX *)
404       Diferencia:REAL;                     (*     ==>  ?:=A*2/100   *)
405     BEGIN
406       Diferencia := SumaA - SumaB ;
407       IF (Diferencia >= SumaA * 4.0 / 100.0) AND (Dif=FALSE) THEN
408         PxPy(54,2); Tilt;Neg; WriteString('IPg='); Vv;
409         PxPy(66,2); Tilt;Neg; WriteString('IPl='); Vv; Bo; Dif:=TRUE;
410       ELSIF (Diferencia < SumaA * 4.0 / 100.0) AND (Dif=TRUE) THEN
411         PxPy(54,2); Neg; WriteString('IPg=');
412         PxPy(66,2);      WriteString('IPl='); Vv; Bo; Dif:=FALSE;
413       END;
414     END DeterminaError;
415     (* ------------------------------------------------------------------------ *)
416     PROCEDURE VigilaE (VarEstadoLineas:CARDINAL; VAR Vie:BOOLEAN; x,y:CARDINAL);
417     BEGIN
418       IF (VarEstadoLineas IN ESTADO) AND (Vie = FALSE) THEN
419         PxPy(x,y); WriteString(' ON'); Bo; Vie:= TRUE;
420       ELSIF
421         ( NOT(VarEstadoLineas IN ESTADO) ) AND (Vie = TRUE) THEN
422          PxPy(x,y); Tilt; WriteString('OFF'); Vv; Bo; Vie:=FALSE;
423       END;
424     END VigilaE;
425     (* ------------------------------------------------------------------------ *)
426 VAR
427   HI, LO, x:CARDINAL;
428 BEGIN
429   LOOP
430     FOR x:=0  TO 15 DO
431       INBYTE (BASE+7,HI); INBYTE (BASE+6,LO); (*Lee estado de los G y L*)
432       ESTADO:=BITSET(HI*256+LO);
433
434       OUTBYTE (BASE+10,x);  (*Selecciona el canal*)
435       OUTBYTE (BASE+11,1);  (*Dispara la conversión*)
436       INBYTE (BASE+5,HI);   (*Lee el byte alto*)
437
438       WHILE HI >=16 DO      (* Si la conversión no está disponible *)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
439         WAIT(t);            (* pasa a Control *)
440         INBYTE (BASE+5,HI);
441       (*HI:=15; (********Para Borrar********)*)
442       END;
443
444       INBYTE (BASE+4,LO);   (*Lee el byte bajo*)
445
446       (* Introducimos valores en sus variables y los visualizamos *)
447       IF x<=4 THEN (* Son valores de generadores *)
448         PG[x+1]:=TRUNC((FLOAT(HI)*256.0+FLOAT(LO))*1.1*FLOAT(PNG[x+1])/4095.0);
449         IF PantallaParada=FALSE THEN
450           CASE x OF
451             0: PxPy(18,6);  WriteCard(PG[1],5); Bo; VigilaD (PG[1], PNG[1], 3,  6, Gs1, VG[1]);
452                VigilaE (0,EG[1],63,6);|
453             1: PxPy(18,8);  WriteCard(PG[2],5); Bo; VigilaD (PG[2], PNG[2], 3,  8, Gs2, VG[2]);
454                VigilaE (1,EG[2],63,8);|
455             2: PxPy(18,10); WriteCard(PG[3],5); Bo; VigilaD (PG[3], PNG[3], 3, 10, Gs3, VG[3]);
456                VigilaE (2,EG[3],63,10);|
457             3: PxPy(18,12); WriteCard(PG[4],5); Bo; VigilaD (PG[4], PNG[4], 3, 12, Gs4, VG[4]);
458                VigilaE (3,EG[4],63,12);|
459             4: PxPy(18,14); WriteCard(PG[5],5); Bo; VigilaD (PG[5], PNG[5], 3, 14, Gs5, VG[5]);
460                VigilaE (4,EG[5],63,14);
461           ELSE
462           END;
463         END;
464       ELSE (* Son valores de lineas *)
465         PL[x-4]:=TRUNC((FLOAT(HI)*256.0+FLOAT(LO))*11280.0/4095.0);
466         IF PantallaParada=FALSE THEN
467           CASE x OF
468             5: PxPy(10,18);  WriteCard(PL[1],5); Bo; VigilaD (PL[1], PNL, 2, 18, Ls1, VL[1]);
469                VigilaE (5,EL[1],21,18);|
470             6: PxPy(10,20);  WriteCard(PL[2],5); Bo; VigilaD (PL[2], PNL, 2, 20, Ls2, VL[2]);
471                VigilaE (6,EL[2],21,20);|
472             7: PxPy(10,22);  WriteCard(PL[3],5); Bo; VigilaD (PL[3], PNL, 2, 22, Ls3, VL[3]);
473                VigilaE (7,EL[3],21,22);|
474             8: PxPy(10,24);  WriteCard(PL[4],5); Bo; VigilaD (PL[4], PNL, 2, 24, Ls4, VL[4]);
475                VigilaE (8,EL[4],21,24);|
476             9: PxPy(36,18);  WriteCard(PL[5],5); Bo; VigilaD (PL[5], PNL,28, 18, Ls5, VL[5]);
477                VigilaE (9,EL[5],47,18);|
478            10: PxPy(36,20);  WriteCard(PL[6],5); Bo; VigilaD (PL[6], PNL,28, 20, Ls6, VL[6]);
479                VigilaE (10,EL[6],47,20);|
480            11: PxPy(36,22);  WriteCard(PL[7],5); Bo; VigilaD (PL[7], PNL,28, 22, Ls7, VL[7]);
481                VigilaE (11,EL[7],47,22);|
482            12: PxPy(36,24);  WriteCard(PL[8],5); Bo; VigilaD (PL[8], PNL,28, 24, Ls8, VL[8]);
483                VigilaE (12,EL[8],47,24);|
484            13: PxPy(62,18);  WriteCard(PL[9],5); Bo; VigilaD (PL[9], PNL,54, 18, Ls9, VL[9]);
485                VigilaE (13,EL[9],73,18);|
486            14: PxPy(62,20);  WriteCard(PL[10],5);Bo; VigilaD (PL[10],PNL,54, 20, Ls10,VL[10]);
487                VigilaE (14,EL[10],73,20);|
488            15: PxPy(62,22);  WriteCard(PL[11],5);Bo; VigilaD (PL[11],PNL,54, 22, Ls11,VL[11]);
489                VigilaE (15,EL[11],73,22);
490           ELSE
491           END;
492         END;
493       END;(*if*)
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
494
495         IF PantallaParada=TRUE THEN
496           DOSCALL(2CH,HoraMinuto,SegundoMiliseg);
497           IF HoraMinuto MOD 256 = 1 THEN
498             PantallaParada:=FALSE; PxPy(2,2); WriteString('           ');Bo;
499           END;
500         END;
501
502         IF (Pitidos = TRUE) AND (Pitando = FALSE) THEN
503              IF (PG[1]>=PNG[1]) OR (PG[2]>=PNG[2]) OR (PG[3]>=PNG[3]) OR
504                 (PG[4]>=PNG[4]) OR (PG[5]>=PNG[5]) THEN
505                 BEEP; Pitando:=TRUE;
506              END;
507         ELSIF (Pitidos = TRUE) AND (Pitando = TRUE) THEN
508              IF (PG[1]<PNG[1]) AND (PG[2]<PNG[2]) AND (PG[3]<PNG[3]) AND
509                 (PG[4]<PNG[4]) AND (PG[5]<PNG[5]) THEN
510                 NOBEEP; Pitando:=FALSE;
511              END;
512         ELSE
513         END;
514
515         WAIT(t); (* pasa a Control *)
516       END;(*for*)
517
518       IF PantallaParada=FALSE THEN
519
520         SPg := FLOAT(PG[1])+FLOAT(PG[2])+FLOAT(PG[3])+FLOAT(PG[4])+FLOAT(PG[5]);
521
522         SP1 := FLOAT(PL[1])+FLOAT(PL[2])+FLOAT(PL[3])+FLOAT(PL[4])+ FLOAT(PL[5]) +
523                FLOAT(PL[6])+FLOAT(PL[7])+FLOAT(PL[8])+FLOAT(PL[9])+FLOAT(PL[10]) +
524                FLOAT(PL[11]);
525
526         RealToString(SPg,0,7,SalidaReal,OkReal);  (* Visualizamos las sumatorias *)
527         IF OkReal THEN PxPy(58,2); WriteString(SalidaReal); Bo; END;
528
529         RealToString(SP1,0,7,SalidaReal,OkReal);
530         IF OkReal THEN PxPy(70,2); WriteString(SalidaReal); Bo; END;
531
532         IF SPg >= SP1 THEN
533           DeterminaError(SPg,SP1);
534         ELSE
535           DeterminaError(SP1,SPg);
536         END;
537
538       END;(*de pantalla*)
539     END; (*loop*)
540   END TomaDeDatos;
541   (* ########################################################################################################## *)
542   PROCEDURE Control;
543     (* ------------------------------------------------------------------------------------------------------ *)
544     PROCEDURE Desconexion (k:CARDINAL);
545     TYPE
546       rango = [0 .. 11];
547     VAR
548      indice, IndiceBuscado : rango;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
549   (* ---------------------------------------------------------- *)
550   PROCEDURE Retardo;      (* Retardo de medio segundo *)
551   VAR
552   Stuff,Count,LimSup:CARDINAL;
553   BEGIN
554
555   IF PantallaParada=TRUE THEN (*Porque va más rapido*)
556     LimSup:=30;
557   ELSE
558     LimSup:=16;
559   END;
560
561   FOR Count:=1 TO LimSup DO
562     WAIT(co);
563   END;
564
565   END Retardo;
566   (* ---------------------------------------------------------- *)
567   BEGIN
568     indice:=0;
569     REPEAT
570       INC(indice);
571     UNTIL (PL[indice] = k ) AND (indice+4 IN ESTADO) OR (indice = 11);
572
573     IF (k = PL[indice])AND(indice+4 IN ESTADO) THEN
574        EXCL (ESTADO,indice+4);(*Descargamos aquella linea conectada cuyo valor sea = k*)
575     ELSE
576       indice:=0;                    (*En caso de que niguna línea  sea=k *)
577       REPEAT                        (*habrá que elegir aquella que esté  *)
578         INC(indice);                (* más próxima a k pero por encima.  *)
579       UNTIL (PL[indice] > k ) AND (indice+4 IN ESTADO) OR (indice = 11);
580
581       IndiceBuscado:=indice;
582
583       IF IndiceBuscado = 11 THEN
584          IF (PL[11] > k) AND (15 IN ESTADO) THEN
585            EXCL (ESTADO,15);
586          ELSE       (*Si ningún valor >= k desconectamos una cualquiera*)
587            indice:=0;
588            REPEAT
589              INC(indice);
590            UNTIL (indice+4 IN ESTADO) OR (indice = 11);
591            EXCL (ESTADO,indice+4);
592          END;
593       ELSE
594         REPEAT
595           INC(indice);
596           IF (PL[indice] <= PL[IndiceBuscado]) AND (PL[indice] >k) AND (indice+4 IN ESTADO) THEN
597             IndiceBuscado:=indice;
598           END;
599         UNTIL  indice = 11;
600         EXCL (ESTADO,IndiceBuscado+4);
601       END;
602     END;
603     ALTO:=CARDINAL(ESTADO) DIV 256; BAJO:=CARDINAL(ESTADO) MOD 256;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
604       OUTBYTE(BASE+14,ALTO); OUTBYTE(BASE+13,BAJO);
605       Retardo;
606    END Desconexion;
607    (* ------------------------------------------------------------------------------ *)
608    PROCEDURE ABC (VAR A,a, B,b, C,c, D,d, E,e:CARDINAL) ;
609    VAR
610    T1,T2,T3,T4:CARDINAL;
611    cont,limite:CARDINAL;
612    generador2,generador3,generador4,generador5:BOOLEAN;
613       (*-------------------------------------------------*)
614       PROCEDURE MasRapid;
615       BEGIN
616          IF PantallaParada=TRUE THEN (*Porque va más rapido*)
617             WAIT(co);
618          END;
619       END MasRapid;
620       (*-------------------------------------------------*)
621    BEGIN
622
623      IF FLOAT (A) >= FLOAT(a)*110.0/100.0 THEN (* Retardo *)
624          generador2:=FALSE; generador3:=FALSE;
625          generador4:=FALSE;generador5:=FALSE;
626          limite:=28; T1:=0; T2:=0; T3:=0; T4:=0;
627
628          FOR cont:=1 TO limite DO (* RETARDO de 2.5 Seg.*)
629             WAIT(co);
630
631             IF generador2 = FALSE THEN
632                IF FLOAT (B) >= FLOAT (b) * 110.0/100.0 THEN (* Retardo para G2 *)
633                   T1:=limite + cont; limite:=T1; generador2 := TRUE;
634                ELSIF (B > b) AND (FLOAT (B) < FLOAT(b) * 110.0/100.0)  THEN
635                   Desconexion(B-b); RETURN;
636                END;
637             END;
638
639
640             IF generador3 = FALSE THEN
641                IF FLOAT (C) >= FLOAT (c) * 110.0/100.0 THEN (* Retardo para G3 *)
642                   T2:=limite + cont; limite:=T2; generador3 := TRUE;
643                ELSIF (C > c) AND (FLOAT (C) < FLOAT(c) * 110.0/100.0) THEN
644                   Desconexion(C-c); RETURN;
645                END;
646             END;
647
648             MasRapid;
649
650             IF generador4 = FALSE THEN
651                IF FLOAT (D) >= FLOAT (d) * 110.0/100.0 THEN (* Retardo para G4 *)
652                   T3:=limite + cont; limite:=T3; generador4 := TRUE;
653                ELSIF (D > d) AND (FLOAT (D) < FLOAT(d) * 110.0/100.0) THEN
654                   Desconexion(D-d); RETURN;
655                END;
656             END;
657
658
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
659             IF generador5 = FALSE THEN
660                IF FLOAT (E) >= FLOAT (e) * 110.0/100.0 THEN (* Retardo para 65 *)
661                   T4:=limite + cont; limite:=T4; generador5 := TRUE;
662                ELSIF (E > e) AND (FLOAT (E) < FLOAT(e) * 110.0/100.0) THEN
663                   Desconexion(E-e); RETURN;
664                END;
665             END;
666
667             IF cont=28 THEN
668                IF A > a THEN Desconexion(A-a); RETURN;END;
669             ELSIF cont=T1 THEN
670                IF B > b THEN Desconexion(B-b); RETURN;END;
671             ELSIF cont=T2 THEN
672                IF C > c THEN Desconexion(C-c); RETURN;END;
673             ELSIF cont=T3 THEN
674                IF D > d THEN Desconexion(D-d); RETURN;END;
675             ELSIF cont=T4 THEN
676                IF E > e THEN Desconexion(E-e); RETURN;END;
677             END;
678
679          END;(* END DEL RETARDO *)
680
681       ELSIF  (A > a) AND (FLOAT(A) < FLOAT(a) * 110.0/100.0) THEN
682          Desconexion(A-a); RETURN;
683       ELSE
684       (***No hace nada***)
685       END;
686       END ABC;
687     (* ------------------------------------------------------------ *)
688  BEGIN
689    LOOP
690      IF FLOAT (PG[1]) > FLOAT (PNG[1]) * 90.0/100.0  THEN
691         ABC (PG[1],PNG[1], PG[2],PNG[2], PG[3],PNG[3], PG[4],PNG[4], PG[5],PNG[5]);
692      END;
693      IF FLOAT (PG[2]) > FLOAT (PNG[2]) * 90.0/100.0  THEN
694         ABC (PG[2],PNG[2], PG[1],PNG[1], PG[3],PNG[3], PG[4],PNG[4], PG[5],PNG[5]);
695      END;
696      IF FLOAT (PG[3]) > FLOAT (PNG[3]) * 90.0/100.0  THEN
697         ABC (PG[3],PNG[3], PG[1],PNG[1], PG[2],PNG[2], PG[4],PNG[4], PG[5],PNG[5]);
698      END;
699      IF FLOAT (PG[4]) > FLOAT (PNG[4]) * 90.0/100.0 THEN
700         ABC (PG[4],PNG[4], PG[1],PNG[1], PG[2],PNG[2], PG[3],PNG[3], PG[5],PNG[5]);
701      END;
702      IF FLOAT (PG[5]) > FLOAT (PNG[5]) * 90.0/100.0 THEN
703         ABC (PG[5],PNG[5], PG[1],PNG[1], PG[2],PNG[2], PG[3],PNG[3], PG[4],PNG[4]);
704      END;
705      WAIT(co);
706    END;
707  END Control;
708  (* ********************************************************************************************************** *)
709  BEGIN (* Bloque del programa *)
710     Cls;
711     Inicializaciones;
712     CambiarPNG;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

```
714     Pantalla;
715     Init(t); Init(co); (*Se incializan las señales*)
716     StartProcess (TomaDeDatos,1000);
717     StartProcess (Control,1000);
718     Sincroniza;
719 END UNELCO.
```

```
Modula-2/86 Compiler Version V 2.00

=>      0 Error(s) found
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

BIBLIOGRAFIA

MODULA-2 A SOFTWARE DEVELOPMENT APPROACH.
G.Ford/R.Wiener/ Ed. John Willey & Sons, 1985.

PROGRAMACION EN MODULA-2.
Niklaus Wirth/ Ed. El Ateneo, 1987.

MODULA-2 MADE EASY.
Herbert Schildt/ Ed. Osborne McGraw-Hill, 1986.

MODULA-2 LENGUAJE Y COMPILADOR PARA EL IBM/PC.
A.B. FONTAINE/ Ed. Masson, 1987.

ADVANCED MODULA-2.
Herbert Schildt/ Ed. Osborne McGraw-Hill, 1987.

MODULA-2 DESARROLLO DE SOFTWARE.
Carlos Galán/ Ed. Paraninfo, 1987.

MODULA-2 TUTORIAL.
Gordon Dodrill/ Ed. Coronado Enterprises, 1987.

LOGITECH MODULA-2/86 SOFTWARE DEVELOPMENT SYSTEM.
LOGITECH, 1986.

8088/8086/8087 PROGRAMACION ENSAMBLADOR EN ENTORNO MS/DOS.
M. A. Rodríguez Roselló/ Ed. Anaya Multimedia, 1987.

MS/DOS AVANZADO.
Ray Duncan/ Ed. Anaya Multimedia, 1988.

MICROPROCESADORES DE 16 BITS.
J. Mª. Angulo Usategui/ Ed. Paraninfo, 1985.

SOLUCIONARIO DEL PROGRAMADOR PARA IBM PC,XT,AT Y COMPATIBLES.
Robert Jourdain/ Ed. Anaya Multimedia, 1988.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

INDICE DE MODULOS EJEMPLO

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2006