

# Apéndice 3

## Módulos Estándar

---

En este apéndice se recoge la definición de los módulos estándar, que acompañan normalmente a cada compilador de Modula-2. Aunque de momento no existe una definición estándar de Modula-2 establecida por algún organismo internacional de normalización, se suele considerar como estándar la definición del lenguaje establecida en el libro de referencia siguiente:

**Niklaus Wirth: Programming in Modula-2. Springer-Verlag, 1982 (1ªEd.), 1983 (2ªEd.), 1985 (3ªEd.), 1988 (4ªEd.).**

En las sucesivas ediciones de esta obra se han ido introduciendo algunas modificaciones en la definición del lenguaje. La descripción de los módulos estándar que se incluye aquí es una adaptación de la que aparece en la 4ª edición, con ligeras correcciones para ajustarse en lo posible a los compiladores comerciales.

Los compiladores de Modula-2 disponibles comercialmente suelen seguir bastante bien la definición del lenguaje correspondiente al momento en que se han desarrollado. Sin embargo no hay demasiada uniformidad en cuanto a los módulos "estándar" que los acompañan. Los módulos descritos en el libro de referencia son:

- CursorMouse
- FileSystem (\*)
- GraphicWindows
- InOut (\*)
- MathLib0 (\*)
- Menu
- RealInOut (\*)
- Storage (\*\*)
- SYSTEM (\*\*)
- Terminal (\*)
- TextWindows
- Windows

Los módulos marcados con un asterisco (\*) suelen acompañar a casi todos los compiladores. Los módulos marcados con dos asteriscos (\*\*) son algo especiales. El módulo SYSTEM es un seudomódulo que forma parte del lenguaje en sí, por lo que siempre está disponible. También debe estar siempre disponible el módulo Storage, necesario para usar variables dinámicas, aunque éste suele ser un módulo normal.

Los compiladores comerciales suelen incluir un repertorio bastante amplio de módulos de utilidad, que incluye los que se han indicado y/o versiones ampliadas de los mismos, así como otros para poder invocar servicios del sistema operativo, manejar ristras de caracteres ("strings") en forma cómoda, realizar cálculos con una variedad mayor de funciones matemáticas, etc.

A continuación se presenta la definición de los módulos habitualmente disponibles.

## FileSystem

---

**DEFINITION MODULE** FileSystem;

(\* =====  
*Esta es una versión adaptada del módulo estándar para  
 manejo de ficheros propuesto en el libro de referencia*  
 ===== \*)

**TYPE**

Flag =  
   ( read, (\* modo de acceso en lectura \*)  
     write, (\* modo de acceso en escritura \*)  
     truncate, (\* truncar el fichero al cerrar \*)  
     again, (\* volver a leer el último carácter \*)  
     ....)

FlagSet = **SET OF** Flag; (\* Indicadores de estado \*)

FilePosition; (\* Posición en el fichero \*)  
 (\* Equivale a un valor CARDINAL con rango amplio \*)

FileHandler; (\* Código del manejador del fichero \*)  
 (\* Equivale a CARDINAL, o bien a ADDRESS \*)

Response = (\* Respuesta a una orden \*)  
 ( done, notdone, notsupported, callerror, ... );

File = (\* Tipo fichero \*)  
**RECORD**  
   fileno: FileHandler; (\* número del manejador \*)  
   flags: FlagSet; (\* indicadores de estado \*)  
   eof: **BOOLEAN**; (\* indicador de fin de fichero \*)  
   res: Response; (\* resultado de la última acción \*)  
   ....  
**END**;

**PROCEDURE** Create( VAR f: File; nombre: **ARRAY OF CHAR** );  
 (\* Crea un fichero nuevo con el nombre indicado. Si ya había un  
 fichero con ese nombre, da error \*)

**PROCEDURE** Lookup( VAR f: File; nombre: **ARRAY OF CHAR**; nuevo: **BOOLEAN** );  
 (\* Abre un fichero con ese nombre. El parámetro 'nuevo' indica que si  
 no existe un fichero con ese nombre, se crea automáticamente \*)

**PROCEDURE** Close( VAR f: File );  
 (\* Cierra el fichero. Si el flag 'truncate' vale TRUE, lo trunca \*)

**PROCEDURE** Delete( VAR f: File );  
 (\* Elimina el fichero (y por tanto, lo cierra) \*)

**PROCEDURE** Rename( VAR f: File; nombre: **ARRAY OF CHAR** );  
 (\* Cambia el nombre del fichero por el nuevo nombre \*)

**PROCEDURE** SetRead( VAR f: File );  
 (\* Fija el modo de acceso en lectura (flag 'read') \*)

**PROCEDURE** SetWrite( VAR f: File );  
 (\* Fija el modo de acceso en escritura (flag 'write') \*)

```

PROCEDURE SetModify( VAR f: File );
(* Fija el modo de acceso en modificación (flags 'read' y 'write') *)

PROCEDURE SetOpen( VAR f: File );
(* Fija el modo de acceso nulo (quita flags 'read' y 'write') *)

PROCEDURE Doio( VAR f: File );
(* Actualiza el contenido del fichero, haciendo que coincidan la información
   grabada en memoria externa y la información temporal en memoria interna *)

PROCEDURE SetPos( VAR f: File; posicion: FilePosition );
(* Pone el cursor de acceso al fichero en la posición indicada *)

PROCEDURE GetPos( VAR f: File; VAR posicion: FilePosition );
(* Obtiene la posición actual del cursor de acceso al fichero *)

PROCEDURE Length( VAR f: File; VAR posicion: FilePosition );
(* Devuelve la longitud del fichero (posición del último elemento) *)

PROCEDURE Reset(VAR f: File );
(* Pone el cursor de acceso señalando al comienzo del fichero *)

PROCEDURE Again(VAR f: File );
(* Hace que la siguiente operación de lectura vuelva a obtener, sólo
   por una vez, el último dato ya leído (flag 'again') *)

PROCEDURE ReadWord(VAR f: File; VAR w: WORD );
(* Lee una "palabra" de memoria del fichero, y avanza el cursor *)

PROCEDURE WriteWord( VAR f: File; w: WORD );
(* Graba una "palabra" de memoria en el fichero, y avanza el cursor *)

PROCEDURE ReadChar( VAR f: File; VAR e: CHAR );
(* Lee un carácter del fichero, y avanza el cursor *)

PROCEDURE WriteChar( VAR f: File; c: CHAR );
(* Graba un carácter en el fichero, y avanza el cursor *)

END FileSystem.

```



### Nota

Este módulo es el que presenta mayores diferencias entre unos compiladores y otros.

## InOut

```

DEFINITION MODULE InOut;
FROM FileSystem IMPORT File;

(* =====
   Operaciones de lectura y escritura de ficheros de texto
   ===== *)

CONST EOL = 36C; (* carácter de fin de línea *)

VAR
  Done: BOOLEAN; (* indicador de operación correcta *)
  termCH: CHAR; (* carácter que terminó la lectura *)
  in, out: File; (* ficheros en uso *)

```

```

PROCEDURE OpenInput( extension: ARRAY OF CHAR );
(* Lee un nombre de fichero por el terminal, le añade la extensión
   indicada, si no tiene, y lo abre como fichero de entrada *)

PROCEDURE OpenOutput( extension: ARRAY OF CHAR );
(* Lee un nombre de fichero por el terminal, le añade la extensión
   indicada, si no tiene, y lo abre como fichero de salida *)

PROCEDURE CloseInput;
(* Cierra el fichero abierto con 'OpenInput', y revierte la lectura al
   fichero de entrada por defecto *)

PROCEDURE CloseOutput;
(* Cierra el fichero abierto con 'OpenOutput', y revierte la escritura al
   fichero de salida por defecto *)

PROCEDURE Read( VAR c : CHAR );
(* Lee un carácter del fichero de entrada. Un salto de línea se lee como
   un único carácter igual a EOL. Si se llega al final del fichero se hace
   Done := FALSE, y se devuelve 0C *)

PROCEDURE ReadString(VAR s : ARRAY OF CHAR );
(* Lee una ristra, permitiendo funciones de edición, hasta encontrar un
   blanco o un carácter de control. El carácter terminador se asigna
   a 'termChar' *)

PROCEDURE ReadCard( VAR x : CARDINAL );
PROCEDURE ReadInt( VAR x: INTEGER );
(* Lee una ristra y la convierte al tipo indicado. La variable 'Done'
   indica el éxito de la operación *)

PROCEDURE Write( c: CHAR );
(* Escribe un carácter en el fichero de salida. Si es EOL graba un salto
   de línea *)

PROCEDURE WriteLn;
(* Graba un salto de línea en el fichero de salida ( = Write(EOL) ) *)

PROCEDURE WriteString(s: ARRAY OF CHAR );
(* Escribe los caracteres útiles de la ristra en el fichero de salida *)

PROCEDURE WriteCard(x: CARDINAL; ancho: CARDINAL );
PROCEDURE WriteInt( x: INTEGER; ancho: CARDINAL );
PROCEDURE WriteOct(x: CARDINAL; ancho: CARDINAL );
PROCEDURE WriteHex(x: CARDINAL; ancho: CARDINAL );
(* Escribe la representación del valor en forma de texto, con un
   número total de caracteres igual al ancho indicado, rellenando con
   espacios en blanco al comienzo, si es necesario. Si el ancho no es
   suficiente, se escriben tantos caracteres como haga falta *)

END InOut.

```



### Nota

Este módulo se debe apoyar en el módulo **Terminal** para la lectura o escritura por un terminal interactivo, y en el módulo **FileSystem** para la lectura o escritura de ficheros en memoria externa. El código exacto para **EOL** puede cambiar de un compilador a otro.

## MathLib0

---

```

DEFINITION MODULE Mathlib0;

(* =====
   Funciones matemáticas básicas
   ===== *)

PROCEDURE arctan( x: REAL ): REAL;
(* Arco tangente, en radianes, entre - /2 y + /2 *)

PROCEDURE cos( x: REAL ): REAL;
(* Coseno de un ángulo en radianes *)

PROCEDURE entier( x: REAL ): INTEGER;
(* 'x' truncado hacia negativo (mayor entero que sea <= 'x') *)

PROCEDURE exp( x: REAL ): REAL;
(* Exponencial: 'e' elevado a 'x' *)

PROCEDURE ln( x: REAL ): REAL;
(* Logaritmo neperiano *)

PROCEDURE real( x: INTEGER ): REAL; (* obsoleta *)
(* Valor de 'x' convertido a REAL *)

PROCEDURE sin( X: REAL ): REAL;
(* Seno de un ángulo en radianes *)

PROCEDURE sqrt( X: REAL ): REAL;
(* Raíz cuadrada *)

END MathLib0.

```



### Nota

En la 4a edición del libro de referencia la función **real** queda anticuada, ya que equivale a la nueva definición de **FLOAT**.

## RealInOut

---

```

DEFINITION MODULE RealInOut;

(* =====
   Lectura y escritura de valores reales
   ===== *)

VAR Done: BOOLEAN; (* indicador de operación correcta *)

PROCEDURE ReadReal( VAR x: REAL );
(* Lee una ristra y la convierte a valor REAL. La variable 'Done'
   indica el éxito de la operación *)

```

```

PROCEDURE WriteReal( x: REAL; ancho: CARDINAL );
(* Escribe la representación del valor en notación científica, con un
   número total de caracteres igual al ancho indicado, rellenando con
   espacios en blanco al comienzo, si es necesario. Si el ancho no es
   suficiente, se escriben tantos caracteres como haga falta *)

PROCEDURE WriteFixPt( x: REAL; ancho, decimales: CARDINAL );
(* Similar al anterior, pero usando la notación decimal habitual, sin
   factor de escala en forma de potencia de 10 *)

END RealInOut.

```



### Nota

Este módulo usa la misma entrada y salida que **InOut**, y constituye una extensión del mismo.

## Storage

---

```

DEFINITION MODULE Storage;

(* =====
   Gestión dinámica de la memoria del computador
   ===== *)

FROM SYSTEM IMPORT ADDRESS;

PROCEDURE ALLOCATE( VAR ptr: ADDRESS; unidades: CARDINAL );
(* Crea una variable dinámica con tamaño 'unidades', y asigna su
   dirección al puntero 'ptr'. Si no hay espacio, devuelve NIL *)

PROCEDURE DEALLOCATE( VAR ptr: ADDRESS; unidades: CARDINAL );
(* Libera el espacio de tamaño 'unidades' que se supone ocupado por
   la variable dinámica apuntada por el puntero 'ptr' *)

PROCEDURE Available( unidades: CARDINAL ): BOOLEAN;
(* Indica si hay al menos una zona de memoria con tamaño suficiente
   disponible para crear una variable dinámica de tamaño 'unidades' *)

END Storage.

```



### Nota

En la 4ª edición del libro de referencia los nombres de estos procedimientos aparecen escritos con letras minúsculas, en la forma **Allocate** y **Deallocate**. La función **Available** no aparece mencionada. Además, los tamaños se expresan como **INTEGER** y no como **CARDINAL**.

# SYSTEM

```

DEFINITION MODULE SYSTEM;

(* =====
   Manejo básico de la memoria del computador
   ===== *)

TYPE
  ADDRESS                      (* Dirección de un elemento de memoria *)
    = POINTER TO WORD;
  WORD;                        (* Palabra de memoria *)
  BYTE;                        (* Byte de memoria, no estándar *)

PROCEDURE ADR( x: CualquieraTipo ): ADDRESS;
(* Dirección de memoria en que se encuentra 'x' *)

PROCEDURE TSIZE( CualquieraTipo ): CARDINAL;
(* Tamaño de la memoria ocupada por un valor de ese tipo *)

(* =====
   Manejo de corrutinas
   ===== *)

TYPE PROCESS = ADDRESS;      (* Corrutina *)

PROCEDURE NEWPROCESS
( p: PROC;                      (* Código de la corrutina *)
  a: ADDRESS;                   (* Dir. de una zona de memoria de trabajo *)
  n: CARDINAL;                 (* Tamaño de esa zona *)
  VAR pr: PROCESS );           (* Corrutina creada *)
(* Crea una nueva corrutina con los parámetros indicados *)

PROCEDURE TRANSFER( VAR desde, hacia: PROCESS );
(* Transfiere control desde la primera corrutina hacia la segunda *)

(* =====
   Manejo de interrupciones (no siempre disponible)
   ===== *)

PROCEDURE IOTRANSFER( VAR desde, hacia: PROCESS; vectorDeInterrupcion: CARDINAL );
(* Transfiere control desde la primera corrutina hacia la segunda,
   y prepara la interrupción para reanudar la primera *)

END SYSTEM.

```



## Nota

El tipo **ADDRESS** es compatible con cualquier tipo puntero y con el tipo **CARDINAL**. El tipo **PROCESS** aparecía como tipo predefinido opaco en las primeras versiones del lenguaje, y actualmente ha quedado obsoleto, habiendo sido reemplazado por **ADDRESS**. Se incluye aquí sólo para hacer más clara la interfaz de las operaciones con corrutinas.

## Terrninal

```

DEFINITION MODULE Terminal;

(* =====
   Manejo de un terminal interactivo, alfanumérico
   ===== *)

PROCEDURE Read( VAR c: CHAR );
(* Lee un carácter desde el teclado, sin eco, y lo almacena en 'c' *)

PROCEDURE BusyRead( VAR c: CHAR );
(* Lee un carácter del teclado sin esperar.
   Si no hay carácter pulsado, devuelve 0C *)

PROCEDURE ReadAgain;
(* Hace que la siguiente operación de lectura recupere de nuevo el
   último carácter pulsado *)

PROCEDURE Write( c: CHAR );
(* Escribe el carácter 'c' en el terminal *)

PROCEDURE WriteLn;
(* Envía un salto de línea al terminal *)

PROCEDURE WriteString( s: ARRAY OF CHAR );
(* Escribe la ristra 's' en el terminal *)

PROCEDURE ReadString( VAR s: ARRAY OF CHAR );
(* Lee la ristra 's' desde el teclado, con eco, permitiendo ciertas
   funciones de edición *)

END Terminal.

```



### Nota

El procedimiento **ReadString** no aparece mencionado en el libro de referencia, y su efecto puede cambiar de un compilador a otro.