Francisco Javier Martínez de Pisón Ascacíbar

LENGUAJE <<TURBO C>> PARA ESTUDIANTES

Universidad de La Rioja

LENGUAJE «TURBO C»
PARA
ESTUDIANTES

MATERIAL DIDÁCTICO

Ingenierías

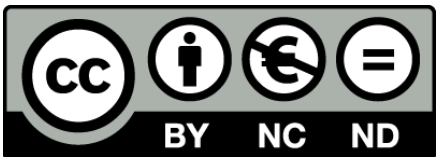
nº 2

Francisco Javier Martínez de Pisón Ascacíbar
Profesor del Departamento de Ingeniería Eléctrica
de la Universidad de La Rioja

LENGUAJE «TURBO C» PARA ESTUDIANTES

UNIVERSIDAD DE LA RIOJA

Servicio de Publicaciones



publicaciones.unirioja.es

E-mail: publicaciones@unirioja.es

ISBN: 978-84-694-0878-0

PROLOGO

El lenguaje C y su evolución Turbo C, Turbo C++, es un lenguaje hecho y escrito por programadores, y para programadores. Esta frase que se lee en numerosas obras, nos viene a indicar el problema que aparece cuando se trata de enseñar a personas que no han programado nunca. Es un lenguaje muy exigente con respecto a otros, pero las ventajas en velocidad de proceso y en el desarrollo de aplicaciones potentes, son lo suficientemente atractivas para que cualquier persona necesite aprenderlo. Cualquier ingeniero, matemático, etc.; necesitará crear programas que sean lo suficientemente rápidos y que aprovechen al máximo lasposibilidades de la máquina.

Este libro es un intento de crear un método de enseñanza del Lenguaje Turbo C, para todas aquellas personas que tengan que aprenderlo desde su inicio, sin tener conocimientos de antemano, de forma que se cree una buena base en el lector. Para ello he dispuesto los temas, como si de un curso se tratara, empezando desde una pequeña guía de MS-DOS, hasta asuntos olvidados en otros libros, como por ejemplo, el uso de ratón, cables de transmisión serie, etc.

Por otro lado, he intentado reunir mediante tablas el uso de las diferentes funciones de que consta el lenguaje Turbo C para que su comprensión resulte más sencilla.

La única y verdadera forma de aprender a programar es programando...

El autor.

INDICE

SISTEMA MS-DOS

Ficheros principales

io.sys	Fichero oculto.
msdos.sys	Fichero oculto.
command.com	Sistema operativo
config.sys	Fichero de configuración inicial del sistema
autoexec.bat	Fichero de configuración. Después del config.sys.

Comandos principales

dir: Muestra el directorio actual de ficheros.
'/w' muestra en columnas,
'/p' espera una tecla,
'/s' mira en todos los subdirectores.

tree : Muestra el arbol de directorios.
cd directorio: Entra en un subdirectorio
md directorio: Crea un directorio
rd directorio: Borra un directorio, si está vacio.
cd.. : Vuelve al directorio anterior
del fichero : Borrar fichero. **¡¡¡ Cuidado !!!**
undelete fichero :Recuperar fichero borrado.
cls : Limpiar la pantalla.
mem : Muestra la memoria libre que hay.
deltree directorio : Borra el directorio y todos sus subdirectorios.
¡¡¡ Mucho cuidado !!!
copy origen destino: copia el fichero de origen al destino.
format unidad: : Formatea un disco o un disco duro, **¡¡¡ Mucho cuidado !!!**
type archivo: Muestra un archivo de texto.

Ejemplos:
```
copy autoexec.bat a:
copy c:*.* a:
copy c:\dos\*.* c:\dosd\
copy *.exe a:
```

Unidades de almacenamiento

C:	Generalmente el disco duro de arranque del ordenador
D:	Generalmente segundo disco duro del sistema.
A:	Disquetera.
B:	Disquetera.

Otra Letra: Normalmente puede significar otro tipo de unidad de almacenamiento como Lector CD-ROM, unidad de almacenamiento de cinta, Unidad Floptical, etc.

Formas de especificar a los archivos.

Ejemplos:

autoexec.bat	Fichero específico.
*.dat	todos los ficheros con extensión dat
.	todos los ficheros.
pp??????.*	todos los archivos que empiezan por pp.
*.exe, *.com y *.bat	Archivos ejecutables.

Almacenamiento.

Bit=Mínima unidad de almacenamiento 1 o 0.

Byte=Compuesto por ocho bits. Formato binario 2^8 combinaciones. Representa un número decimal del 0 al 255.

Word=2 Bytes=16 Bits= Compuesto por dos bytes. Representa un número del 0 al 65535.

1 Kbyte=1024 bytes

1 Megabyte=1000 Kbytes=Corresponde a 1.024.000 de caracteres.

1 Gigabyte=1000 Megas=Un millón de Kbytes.

Partes en que se divide un ordenador

MEMORIA RAM:

Memoria Básica: Zona de la memoria donde se almacenan todos los programas que trabajan en el modo 8086. Como máximo 640 Kbytes.

Memoria Ampliada (EMS): El modo 8086 puede direccionar como máximo un Megabyte, de ellos 640 corresponden a la memoria Básica y los que quedan se dejan para direccionar la ROM. Pero como todas esas direcciones no se utilizan, se puede repaginar diversos bancos de memoria de 16 Kb de RAM mediante programas específicos que programan los registros de hardware.

Memoria Extendida (XMS): Los microprocesadores 80286, 80386 y 80486 pueden direccionar en modo protegido la memoria de forma lineal, sin tener que recurrir a bancos de paginación. La memoria que queda a partir del primer mega que tenemos en el ordenador se considera como memoria extendida. En el modo 8086 se puede redireccionar los primeros 64 Kbytes de memoria extendida como memoria básica, mediante el driver HIMEM.SYS. De esta forma la memoria básica la ampliamos a 640+64 K, estos 64 Ks de memoria extendida reciben el nombre de memoria alta (HMA).

MICROPROCESADOR (CPU) 80286, 80386, 80486: El cerebro del ordenador. Procesa las instrucciones en dos modos. En el modo real (Modo 8086), la memoria que puede direccionar es de 1 Megabyte, en bancos de 64 Kbytes, con este modo se consigue la compatibilidad con el MS-DOS. En el modo protegido, cada procesador tiene sus propias instrucciones y direcciona de forma lineal toda la memoria que tenga, además de poder trabajar en modo multitarea.

COPROCESADOR (NDP): Realiza las operaciones matemáticas en coma flotante, asi como diversas funciones matamáticas de forma mucho más rápida y liberando al micro de esa tarea.

PLACA BASE: Placa electrónica donde se encuentra el microprocesador y generalmente toda la circuiteria del ordenador (memoria, coprocesador).

SLOTS DE EXPANSION: Ranuras que sirven para conectar otras tarjetas en el ordenador.

TARJETA GRÁFICA: Se encarga de la comunicación de la pantalla con el ordenador. Se diferencian en los modos de resolución que puedan alcanzar, su velocidad y los colores que puedan representar. Tiene su propia memoria.

DISCO DURO: Unidad de almacenamiento masivo. Se almacenan los programas y la información que puede utilizar el ordenador. Su capacidad puede estar entre unos pocos Megabytes, 20 Megas por ejemplo hasta varios Gigabytes. Los mejores modelos Quantum, IBM y Conner (en la actualidad).

DISQUETERA: Unidad de almacenamiento en disco flexible. Actualmente existen en discos de 5 1/4 pulgadas (tienden a desaparecer) y en discos de 3 1/2 pul-

gadas. Con capacidades de 360K o 1.2M para los de 5 1/4 y con capacidades de 720K y 1.44M para los de 3 1/2.

CD ROM: Unidad lectora de discos ópticos. Cada disco óptico puede almacenar hasta 670 Megas (en la actualidad). Solo se pueden leer, no escribir en ellos.

UNIDADES MAGNETO ÓPTICAS: Tecnología que incorpora técnicas magnéticas unidas a ópticas, para almacenar en disquettes grandes capacidades de memoria. Pudiendose borrar y regrabar cuantas veces se quiera. Capacidades desde 128 Megas hasta varios Gigabytes.

PUERTOS SERIE (COM): Conectores que permiten la comunicación serie via cable, modem, dispositivos electrónicos, etc.

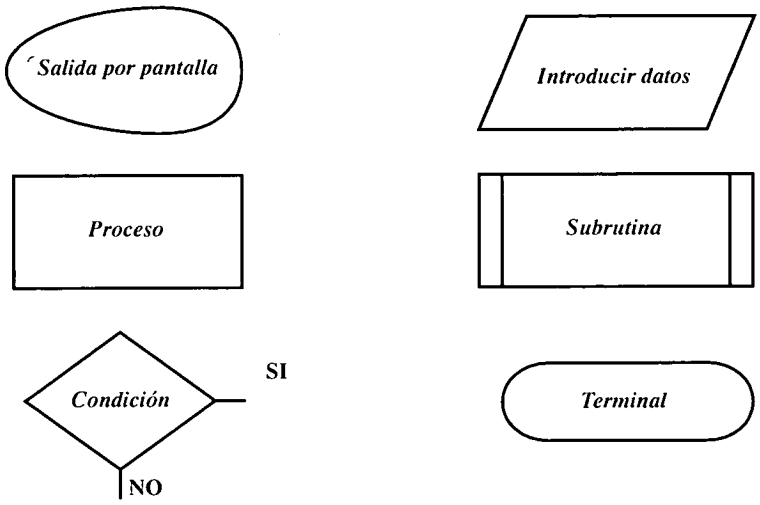
PUERTOS PARALELOS (LPT): Permiten la comunicación en paralelo, utilizado preferentemente por las impresoras.

TARJETA CONTROLADORA: Controla los dispositivos tales como el disco duro, disquetera, puertos de comunicaciones, ratón, joystick, etc. Puede ser una tarjeta externa (Tarjeta multifunción) o estar insertada en la placa base.

RATON
TECLADO
MONITOR

Técnicas de programación.
Organigramas o diagramas de flujo.

Los organigramas representan en forma simbólica las tareas concretas que realiza un programa, en forma de secuencia. **Los organigramas se utilizan, a la hora de realizar algoritmos complicados.**



NORMAS PARA UNA CORRECTA PROGRAMACION

1.- Organización: Organizar en qué partes se va a dividir el programa, cómo se van a representar los datos, que dispositivos se van a manejar, cómo va a ser el menu, etc. Preferiblemente primero se realiza un esquema en papel de las diferentes partes del programa.

2.- Orden: Comenzar con las partes más importantes del programa, **probando cada una** por separado, para ver su correcto funcionamiento.

3.- Prevención: Preveer todas las posibles ampliaciones antes de inicar la programación. Dejar puertas abiertas para poder ampliar o modificar el programa

con facilidad. Hay que darse cuenta que modificar o ampliar **un programa cuesta mucho más si no se han previsto futuros cambios.**

4.- Limpieza: Programar con limpieza las rutinas. Poniendo **textos explicativos al principio de cada rutina** y en los sitios importantes.

5.- Utilizar los tabuladores: Para diferenciar comandos que pertenecen a otros comandos. Bucles dentro de otros bucles, etc:

Ejemplo:

```
for (t=0;t<10;t++)
{
        for (j=1;j<100;j++)
        {
                if (j<t)
                        {
                        printf("J MENOR QUE T");
                        }
        }
}
```

6.- Optimizar las funciones: Intentar hacer lo más rápidas y con las menos instrucciones posibles las funciones. Luego serán más facilmente de entender por uno mismo y nos darán menos problemas. Cuesta más pero al final merece la pena.

7.- Intentar juntar las funciones parecidas en una: Si tenemos varias funciones que realizan cosas parecidas con pequeñas diferencias, es más util crear una función que las realice todas juntas. Ahorraremos memoria y complejidad en el programa.

8.- Acostumbrarse a hacer las cosas bien: No hacer cosas a medias ni chapuceras. Si nos acostumbramos a hacer las cosas bien, conseguiremos programar más y mejor en menos tiempo.

9.- Utilizar siempre las mismas variables para contadores, cadenas, cálculos matemáticos: Si usamos siempre las mismas variables, nos resultará más facil reconocerlas a la hora de leerlas.

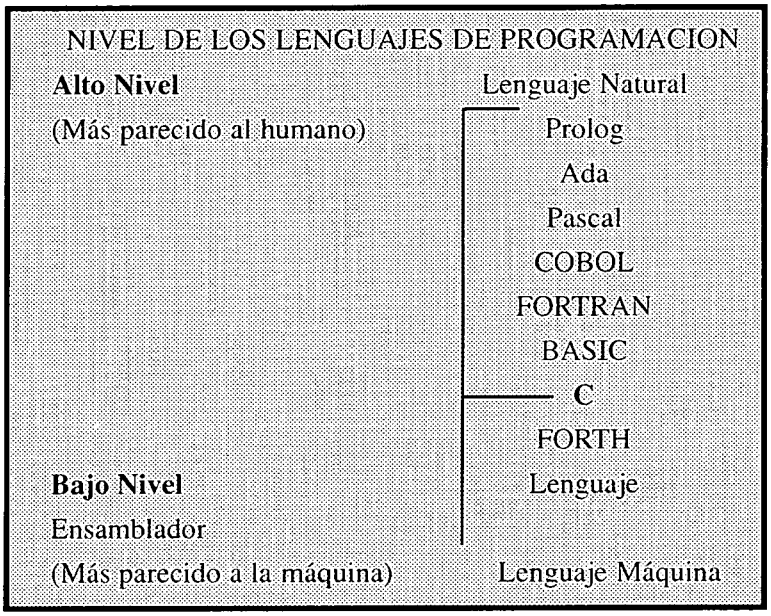
Problema

Realizar el organigrama de un programa que nos permita jugar a la ruleta apostando a un número par o impar. Al iniciar el programa, en una variable dinero=100000 tendremos el dinero de que disponemos y en otra variable banca=10000000 el dinero de la banca. El programa nos pedirá que apostemos a par o impar, y el dinero que apostamos (se almacenará en la variable apuesta), comprobando que el dinero apostado es menor que el dinero que tenemos. Una vez hecha la apuesta el programa calculará un número aleatorio dándonos el resultado y viendo si hemos ganado (dinero=dinero+apuesta, banca=banca-apuesta) o si hemos perdido (dinero=dinero-apuesta, banca=banca+apuesta). Para terminar, el programa tiene que mirar si estamos sin dinero o si la banca está a cero, si no es asi el programa nos preguntará si queremos seguir jugando.

1. PRIMEROS CONCEPTOS DEL LENGUAJE C

Introducción

El lenguaje C apareció durante los años setenta (creado por Dennis Ritchie en 1972 de los laboratorios Bell), con el objetivo de alcanzar la eficiencia de un lenguaje ensamblador unido a las ventajas que nos proporcionan los lenguajes de alto nivel (uso de librerias, estructuración de bloques, funciones complejas, etc).



Las palabras claves del C

LAS 32 PALABRAS CLAVES DEL LENGUAJE C

auto	break	case	char	const	continue	
default	do	double	else	enum	extern	
float	for	goto	if	int	long	
register	return	short	signed	sizeof	static	
struct	switch	typedef	union	unsigned		void
volatile						
while						

LAS 11 PALABRAS CLAVES AÑADIDAS POR TURBO C

asm	_cs	_ds	_es	_ss	cdecl
far	huge	interrupt	near	pascal	

Estructura de un programa en C.

```
Definición de librerías a utilizar
Variables globales;
funcion1()
{
  Variables locales;
  Instrucciones;
}
funcionX()
{
  Variables locales;
  Instrucciones;
}
main()
{
  Variables locales;
  Instrucciones;
}
```

Normas sintácticas del C

- Las palabras claves y funciones del Turbo C, se escriben siempre con minúsculas.

Ejemplo: clrscr(); limpiar la pantalla

- El final de cada instrucción o palabra clave termina con punto y coma, excepto si se compone de varias instrucciones, entonces termina con una llave "}".

```
Ejemplo: h=10;                          h igual a 10
         if (x==10)    {                si x igual a 10, h=10 y j=10.
                       h=10;
                       j=5;
                       }
```

- Las variables, nombres de funciones, etiquetas, etc., siempre tienen que empezar por una letra o símbolo de subrayado nunca por número. La ñ no se puede utilizar.

Correctos:	int menu;	Incorrectos:	int 12menu;
	int _contador;		int ¿21';
	int pe_12;		int tot...al

• El C distingue entre mayúsculas y minúsculas. Las variables Cuenta y CUENTA las toma como variables distintas.

Ejemplo de un programa en C y el código objeto que se produce al compilar.

A continuación se presenta un pequeño programa en C y el lenguaje ensamblador que se genera para ver como el compilador optimiza cada programa...

Programa en C

#include <stdio.h>	Librería "stdio.h" donde está la función printf, getch, ...
main()	Función principal.
{	
int h,j;	Dos variables enteras h y j (pueden almacenar entre 32767 y -32768)
h=10;j=5;	h igual a 10 y j igual a 5.
h=j+(h*10);	h=5+(10*10)=105
printf("EL VALOR DE h ES %d",h);	Muestra en pantalla el valor de h.
getch();	Espera a que se pulse una tecla
return;	Retorna al sitio desde donde hemos llamado al programa.
}	

Parte del código objeto generado (Lenguaje Ensamblador)

```
...
cabecera del fichero objeto
...
;
; main()
;
assume   cs:_TEXT
_main    proc    near
  push   bp
  mov    bp,sp
  push   si
  push   di

  ;
  ;      {
  ;      int h,j;
  ;      h=10;j=5;
  ;
  mov    si,10          h es el registro si
  mov    di,5           j es el registro di

  ;
  ;      h=j+(h*10);
  ;
  mov    ax,si          mueve al acumulador h
  mov    dx,10          mueve al dx 10
  imul   dx             multiplica el valor del acumulador con dx
  mov    dx,di          mueve a dx el valor de j
  add    dx,ax          súmale el resultado de la multiplicación anterior
  mov    si,dx          mueve el resultado a h

  ;
  ;      printf("EL VALOR DE h ES %d",h);
  ;
  push   si
  mov    ax,offset DGROUP:s@
  push   ax
  call   near ptr _printf
  pop    cx
  pop    cx
  ;
```

```
        ;       getch();
        ;
        call    near ptr _getch
        ;
        ;       return;
        ;
        jmp     short @1@50
@1@50:
        ;
        ;       }
        ;
        pop     di
        pop     si
        pop     bp
        ret
...
instrucciones de final del código objeto
...
```

Primeras instrucciones del lenguaje C (para el problema 1).

clrscr(); (Libreria **conio.h**)

Sirve para limpiar la pantalla en modo texto;

int var; (Palabra clave del lenguaje C)

Asigna una variable tipo entero con el nombre var. Las variables tipo int pueden almacenar números dentro del rango de 32767 a -32768.

Ejemplos:

```
int ab;
int contador=10;   Se asigna una variable con el nombre 'contador y con un valor inicial 10.
int a1, a2, a3;    Se definen 3 variables tipo int.
```

printf("cadena de caracteres", argumento); (Librería **stdio.h**)

Se utiliza para imprimir en la pantalla del ordenador la información deseada. En el problema lo utilizaremos para sacar en pantalla tex tos y el valor de las variables.

Ejemplos: printf("Mi coche es azul\n y bastante viejo");

El carácter '\n' realiza un salto de línea, es decir, en la pantalla aparecerá el siguiente texto:

Mi coche es azul

y bastante viejo

printf("El valor de la variable h=%d",h);

El carácter '%d' indica que en esa posición hay que imprimir una valor entero con signo. En este caso el valor de h. Si h es igual a 12320 en la pan talla aparecerá:

El valor de la variable h=12320

printf("HOY ES VIERNES");

Imprime en pantalla el texto:

HOY ES VIERNES

Ordenes de formato de printf

Código	Formato
%c	Imprime un carácter ASCII. (char)
%d	Valor número rico de un carácter. (char)
%d	Enteros decimales con signo (int)
%u	Enteros sin signo (unsigned int)
%ld	Números long con signo (long)
%lu	Números long sin signo (unsigned long)
%f	Coma flotante (float y double)
%i	Enteros decimales con signo
%e	Notación científica (con e minúscula)
%E	Notación científica (con E mayúscula)
%g	Usa %e o %f el más corto
%G	Usa %E o %f el más corto
%o	Octal sin signo
%s	Cadena de caracteres
%x	Hexadecimales sin signo (minúsculas)
%X	Hexadecimales sin signo (mayúsculas)
%p	Mostrar puntero
%n	Asigna un puntero con el número de caracteres escritos
%%	Imprime el signo %

Códigos de barra invertida

Código	Significado
\n	Salto de línea
\t	Tabulador horizontal
\b	Espacio atrás
\a	Alerta
\\	Barra invertida
\"	Comillas dobles
\'	Comilla simple
\r	Retorno de carro
\0	Nulo
\f	Salto de página
\v	Tabulación vertical
\o	Constante Octal
\x	Constante Hexadecimal

return; (Palabra clave del lenguaje C)

Retorna de la función en la que esté el programa.

Problema 1

Realiza un programa que almacene en las variables anio_nac y anio_act tu año de nacimiento y el año actual respectivamente. El programa tiene que imprimir en pantalla los datos de esta forma:

NOMBRE: FRANCISCO JAVIER

EDAD APROXIMADA: 24 AÑOS

La edad la calculará, de forma aproximada, el programa restando al año actual el año del nacimiento.

Problema 2

Ejecutar paso a paso el programa del problema 1 mediante el modo TRACE del compilador de TURBO C++, grabar el programa en el disquete, cargarlo, compilar por separado el programa.

Objetivo: Los dos problemas tienen como objetivo la familiarización con el compilador y el editor de TURBO C++, así como el uso de las instrucciones arriba descritas y la práctica en la sintaxis de programas escritos en C.

2. LAS VARIABLES EN EL C

Las variables y el rango de valores que pueden almacenar

Tipo	Tamaño (bits)	Rango
char *	8	127 a -128 (-2^7)
unsigned char *	8	255 a 0 (2^8)
signed char	8	127 a -128 (-2^7)
int *	16	32767 a -32768 (-2^{15})
unsigned int *	16	65535 a 0 (2^{16})
short int	16	32767 a -32768 (-2^{15})
signed short int	16	32767 a -32768 (-2^{15})
signed int	16	32767 a -32768 (-2^{15})
long *	32	2147483647 a -2147483648 (-2^{31})
unsigned long *	32	0 a 4294967295
long int	32	2147483647 a -2147483648 (-2^{31})
float *	32	3.4 E+38 a 3.4 E-38
double *	64	1.7E+308 a 1.7E-308
long double	64	1.7E+308 a 1.7E-308

(*) Más utilizadas

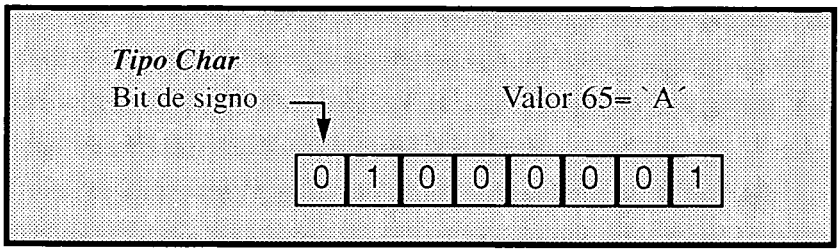
Variables tipo char (1 byte)

Cada variable esta formada por un byte, sirve para almacenar caracteres ASCII. Cada carácter ASCII viene representado por un número del 0 al 255. Para almacenar un carácter en una variable, se almacena el valor que le pertenece dentro de la tabla ASCII. Por ejemplo:

```
1.- char carac;          carac tipo char, en carac almacenamos la "A" (ver
    carac=65;            tabla ASCII). Valor de "A"=65. Imprime su valor.
    printf("CARACTER=%c",carac);
```

Lo mismo...

2.- char carac; carac tipo char, en carac almacenamos el valor de
carac='A'; en ASCII de A. El carácter entre comillas simples
printf("CARACTER=%c",carac); le indica al compilador que asigne
su valor ASCII.



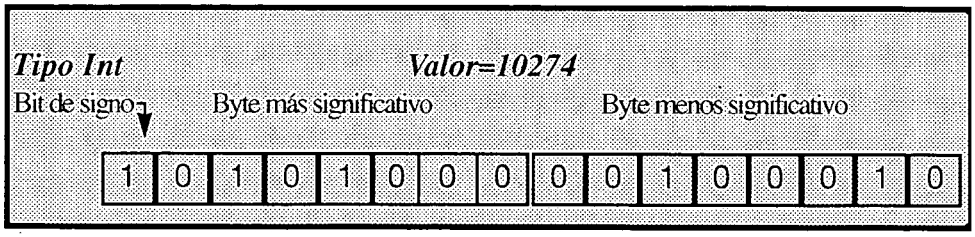
1.-signed char var1; var1 tipo char y la igualamos con el valor 120.
var1=120;

2.- char var2; var2 tipo char y la igualamos con el valor 123.
var2=-123;

3.-unsigned char var3; variable var3 sin signo tipo char igual a 254.
var3=254;

Variables tipo int (2 BYTES=1 WORD)

Generalmente son las variables más utilizadas. Sirven para almacenar valores enteros entre 32767 y -32768 con enteros con signo, y entre 65535 y 0 con enteros sin signo. Como el microprocesador utiliza registros de 16 bits, cada entero corresponde a un registro. De este modo, las operaciones aritméticas rápidas como multiplicar o dividir enteros pequeños se realizarán con este tipo de variable, tomando en cuenta que si dividimos dos enteros y el resultado lo almacenamos en otro, solo tendremos el cociente sin los decimales.

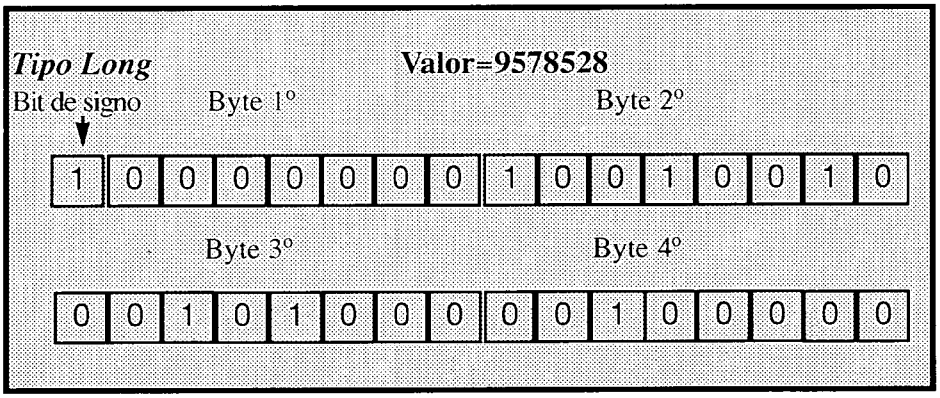


Ejemplos.

```
1. int var4, var5;                  var4 y var5 tipo int, igualados a 12312 y -
                                    12500
   var4=12312; var5=-12500;         respectivamente.
2.-unsigned int var6;               variable entera sin signo (0 a 65535).
   var6=50000;
```

Variables tipo long (4 BYTES=1 DWORD)

Se utilizan para manejar valores enteros muy elevados. El rango de estas variables es desde el 2147483647 hasta el -2147483648 para long con signo o de 0 a 4294967295 sin signo.



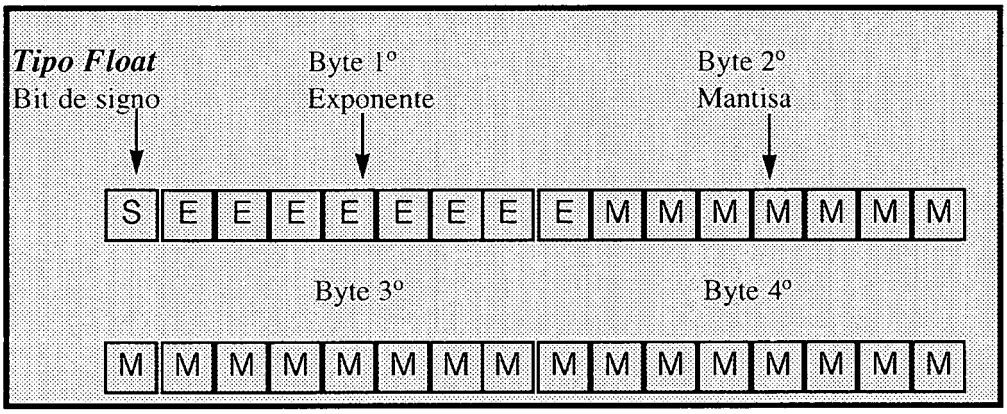
Ejemplos.

```
1.- long int var7, var8;      variables var7 y var8 tipo long, almacenan valores
   var7=1231231;              entre 2147483647 y -2147483648.

   var8=-31313321;
   printf("Valor de var7=%ld y de var8=%ld",var7,var8);
2.- unsigned long var9;
   var9=3148643232;
   printf("Valor de var9=%lu",var9);
```

Variables tipo float (4 BYTES)

Las variables tipo float almacenan cantidades numéricas con parte fraccional. La precisión que nos ofrecen este tipo de variables es de 6 decimales. Los números se almacenan utilizando 1 bit de signo, 8 bits para el exponente y 23 para la mantisa (la mantisa corresponde al número total eliminando el punto). Los números se almacenan en base dos, de forma que si la mantisa es demasiado grande el ordenador trunca los decimales. El rango de números oscila entre ±3.4 E+38 hasta ±3.4 E-38.



Ejemplos.

1.- float var10;
var10=1231.123;
printf("Valor de var10=%f",var10);

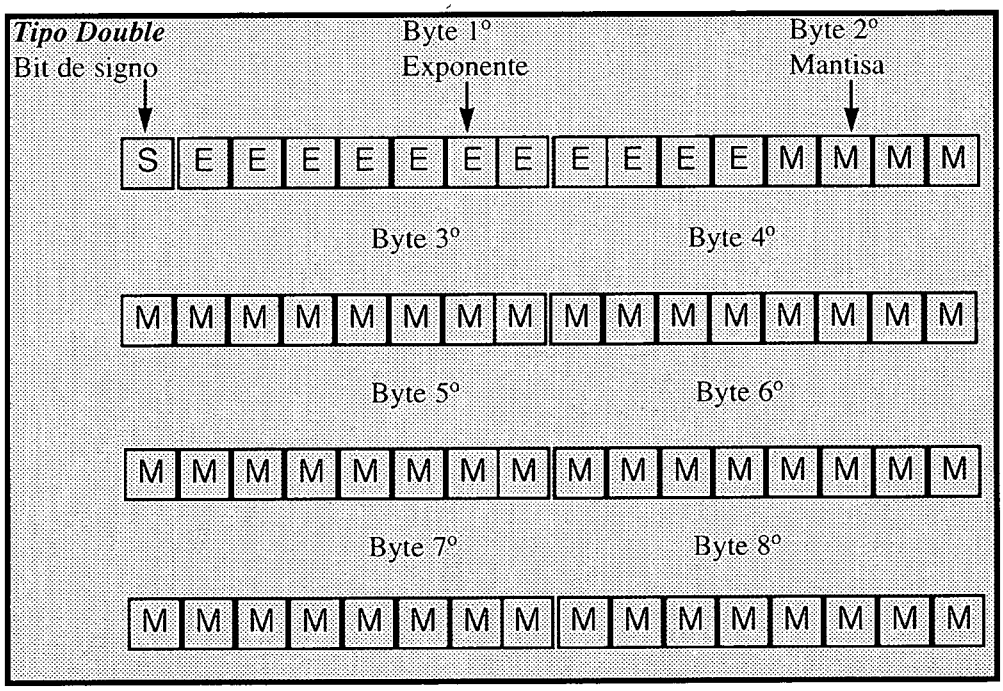
variable var10 tipo float, variables que almacenan números con decimales.

2.- float var11;
var11=1231.0;

Al dar valor a una variable tipo float, si el valor de los decimales es cero, es necesario inicializar la variable **con punto cero**.

Variables tipo double (8 BYTES)

Como las variables float, las variables double sirven para manejar números con parte fraccional, excepto que con este tipo de variables la precisión es mayor. El **número de decimales** que nos permite **es de 15** (según el tamaño de la mantisa). Para almacenar estos números, se utiliza 1 bit de signo, 11 bits para el exponente y 52 para la mantisa. El rango de este tipo está entre ±1.7 E+308 hasta ±1.7 E-307.



Ejemplos.

1.- double var12; variable var12 tipo double.
var12=1231321.233345787652
printf("El valor de la variable var12 es %f",var12);

2.- double var13;

var13=123121565212.0; Al dar valor a una variable tipo double, si el valor de los decimales es cero, es necesario inicializar la variable con punto cero(en algunos compiladores).

Variable tipo void (0 BYTES)

Las variables tipo void son variables que no almacenan ningún valor. Se utilizan fundamentalmente para designar funciones que no devuelven ningún valor. También para indicar que una función no va a requerir ningún parámetro y para designar punteros genéricos.

Ejemplos.

```
1.-void pon_nombre (void)
  {
        printf("Mi nombre es FRANCISCO JAVIER");
        return;
  }
```

Conversión de tipos de datos, problemática.

La conversión de tipos de datos ocurre cuando se mezclan variables de distintos tipos. Las reglas de conversión de tipos son:

1.- El valor del lado derecho de la asignación (el lado de la expresión), se convierte al tipo del lado izquierdo (variable que recibe el valor).

2.- Se realiza toda la operación matemática antes de convertiste al tipo del lado izquierdo.

3.- Si en la operación matemática existen variables de diferentes tipos, se convierten todas a la de más precisión. Siendo la de más precisión la tipo double, después float, long , int y la de menor precisión char.

Por ejemplo, en el programa siguiente:

```
void funcion(void)
{
        char    c;
        int     h;
        c=126;
        h = c + 1231;
        printf("VALOR DE h=%d",h);
        getch();
        return;
}
```

aparecen dos variables de diferente formato, y una sentencia donde se relacionan. En la operación de suma de la variable c más 1231, el valor de la variable c que es tipo char se convierte a variable tipo int (formato de la variable que recibe el valor, la variable tipo h) y se le suma el valor 1231.

En el ejemplo siguiente:

```
main()
{
        int k;
        float m;
        k=7;
        m=k / 2;
}
```

la variable m recibe el valor 3, resultante de dividir 7 entre 2 como números enteros. En cambio si en vez de poner esa linea así, la ponemos de esta forma:

```
m=k / 2.0;
```

el compilador convierte la variable k a una variable tipo float, dando el resultado 3.5.

Existen combinaciones que pueden provocar pérdida de información, hay que tener especial cuidado cuando se combinan variables de distinto tipo.

Tipo destino=Tipo expresión;

Tipo destino	Tipo de expresión	Pérdida de información
signed char	unsigned char	Si el valor es > 127, destino negativo
char	int	8 bits más significativos
char	long	24 bits más significativos
int	long	16 bits más significativos
int	char	Se almacena el valor en 2 bytes. No hay pérdida de información
int	unsigned char	Se almacena el valor en 2 bytes. No hay pérdida de información
int	float	Se pierde la parte fraccional
float	double	Se pierde precisión. El resultado se redondea.
float	int	Se almacena el entero sin perder información
float	long int	Se almacena el entero sin perder información
double	float	No se pierde información, se almacena el nº en más bytes.
double	int	Se almacena el entero sin perder información
double	long	Se almacena el entero sin perder información

Instrucciones del C (para el problema 3).

char var; (Palabra clave del C)

Asigna una variable tipo char con el nombre var.

int var; (Palabra clave del lenguaje C)

Asigna una variable tipo entero con el nombre var.

long int var; (Palabra clave del C)

long var;

Asigna una variable tipo long con el nombre var.

float var; (Palabra clave del C)

Asigna una variable tipo float con el nombre var.

double var; (Palabra clave del C)

Asigna una variable tipo double con el nombre var.

scanf ("formato",&var); (Libreria stdio.h)

Lee desde el teclado una información y la almacena en la variable var en el formato especificado.

Ejemplos:	int variab;	*Variable tipo int.*
scanf("%d",&variab);		*Espera a que se introduzca por teclado un número en formato int y lo almacena en la variable variab.*

Algunos formatos que admite scanf

Código	Significado
%c	Leer un carácter ASCII
%d	Leer un valor tipo char
%d	Leer un valor tipo int
%ld	Leer un valor tipo long int
%f	Leer un valor tipo float
%lf	Leer un valor tipo double
%e	Leer un valor tipo float
%i	Leer un valor tipo int
%o	Leer un valor en formato octal
%x	Leer un valor en hexadecimal
%s	Leer una cadena de caracteres
%p	Leer un puntero

!!Atención¡¡: El *scanf* cuando lee cadenas de caracteres considera el espacio o la coma, como si fuera el ENTER. No es aconsejable utilizarlo para pedir cadenas, para ello se utiliza el comando gets(*cadena*). Si se utiliza de todas formas, es aconsejable utilizar el comando *flushall*() para limpiar el buffer de teclado antes de hacer un scanf o un gets si se han hecho anteriores *scanf*, *gets* o *getch*;

Problema 3

Realiza un programa que pida que se introduzca por teclado los siguientes datos:

- *Año actual = Variable an_actual*
- *Año de tu nacimiento = Variable an_nacim*
- *DNI = Variable dni_var*
- *Altura (cm) = Variable con decimales altura*
- *Peso (Kg) = Variable con decimales peso*

determinar el tipo de variables donde se almacenarán estos datos e imprimir en pantalla el la edad aproximada, el DNI y un coeficiente que viene determinado por la siguiente formula:

$$Coefi = \frac{Edad\ (años)^* Altura\ (num)^* 0.1231}{Peso\ (gramos)}$$

Problema 4

Realiza un programa que pida el radio de una esfera y nos determine:

- *Diámetro de la esfera.*
- *Circunferencia máxima de la esfera.*
- *Sección máxima de la esfera.*
- *Volumen de la esfera.*

Problema 5

Tenemos una fábrica que realiza latas de conserva, queremos un programa que dándole el radio y la altura en mm nos muestre en pantalla:

- *Circunferencia de la base en mm.*
- *Superficie de la base en mm2.*
- *Superficie total de la lata en mm2.*
- *Volumen de la lata en cc.*

Además el programa nos determinará la superficie a cortar de dos planchas rectangulares. La primera corresponderá a la superficie lateral de la lata, teniendo las dimensiones Circunferencia de la base · Altura y la segunda correspondiente a un plancha donde se pueda realizar el corte de las dos bases. De esta segunda plancha el programa nos determinará el área desechada una vez realizado el corte de las dos bases.

Objetivo: Se persigue la familiarización con los distintos tipos de variables del C y la interrelación de éstos.

3. LOS OPERADORES

Operadores aritméticos

Sirven para realizar la operaciones aritméticas.

Operador	Acción
-	Resta dos elementos
+	Suma dos números
*	Multiplica
/	Divide
%	Devuelve el resto de la división

La división de dos enteros devuelve el cociente truncado (sin decimales). Por ejemplo, si efectuamos la división 5/2 siendo dos números enteros, el resultado será el 2.

El operador % devuelve el resto. Si efectuamos la operación 5%2 el resultado será 1 (el resto).

Operadores aritméticos de incremento y decremento

Añade o resta un 1 a su operando. Corresponde a las instrucciones directas de ensamblador INC registro (incrementa en uno la variable) y DEC registro (decrementa en uno el registro).

Operador	Acción
++	*Incrementa en uno el operador*
—	*Decrementa en uno el operador*

La sintaxis correspondiente es:

++x;	*Incrementa en uno x*
x++;	*Incrementa en uno x*
—x;	*Decrementa en uno x*
x—;	*Decrementa en uno x*

La diferencia consiste en que la operación de incremento o decremento se realiza antes o después de utilizar el valor del operando. Por ejemplo en el siguiente trozo de programa:

```
int x,y,n;
x=10;
y=++x;
x=10;
n=x++;
```

y tiene el valor 11 ya que la variable x se incrementa antes de asignarse a la y. En cambio la variable n tiene el valor 10 por que aún no se ha incrementado la x cuando se asigna a n.

Dados estos dos ejemplos:
Ejemplos 1:

```
x=5;
if (++x>5) printf("X es mayor que cinco");
```

Ejemplo 2:

```
x=5;
if (x++>5) printf("X es mayor que cinco");
```

deducimos que en el primer ejemplo se realiza el printf, ya que primero se realiza el incremento y después la comparación. En el segundo ejemplo no se imprime el texto, debido a que primero se compara y después se incrementa x.

Operadores relacionales y lógicos

Operadores relacionales

Operador	Acción
= =	*Igual*
! =	*Distinto*
>	*Mayor que*
> =	*Mayor o igual que*
<	*Menor que*
< =	*Menor o igual que*

Operadores lógicos relacionales

Operador	Acción
&&	*AND (Y)*
||	*OR (O)*
!	*NOT (Negación)*

Las operaciones realizadas devuelven un 1 cuando la operación es cierta, y un 0 cuando la operación es falsa. Las operaciones se pueden combinar en formas más complejas:

Ejemplo: 5 < 12 && !(12 < 9) && 1 || (2>5) = 1 && 1 && 1 || 0 = 1

Orden de precedencia entre operadores		
Mayor	++,-	ARITMETICOS
	-(monario)	
	*,/,%	
	+,-	
	!	LOGICOS
	<,<=,>,>=	RELACIONALES
	==,!=	
	&&	LOGICOS
Menor	||	

Cuando en una expresión aparecen varios tipos de operadores, el compilador toma unos antes que otros según el orden de precedencia de la tabla de la izquierda. Para impedir errores, es usual utilizar paréntesis para diferenciar aquellas partes que queramos que se realicen sin tener que preocuparnos del orden de precedencia de los operadores. La velocidad del programa no varía, ya que los paréntesis los elimina el compilador.

Problema 6

Elaborar un programa que nos pida cuatro números y mediante operadores lógicos calcular cual de ellos es el mayor, sin utilizar la palabra clave if, sólo mediante operadores relacionales y lógicos.

Operadores a nivel de bits

Operadores a nivel de bits

Operador	Acción	Explicación
&	*AND (Y)*	*Realiza un and con cada bit*
|	*OR (O)*	*Realiza un OR con cada bit*
^ (ASCII 94)	*XOR (O Exclusiva)*	*Realiza un XOR con cada bit*
~ (ASCII 126)	*Complemento a uno*	*Invierte los 1 por 0 y los 0 por 1.*
<<	*Desplaza a la izquierda los bits*	*Se introducen 0 por la derecha*
>>	*Desplaza a la derecha los bits*	*Se introducen 0 por la izquierda*

Consideraciones:

• Los operadores a nivel de bits sólo son válidos para variables tipo char y tipo int.

• El desplazamiento se indica de esta forma: X=X>>3 Desplaza a la derecha tres veces

• El desplazamiento hacia la derecha de números negativos introduce bits por la izquierda de valor 1.

• El complemento a uno cambia los bits de ceros a unos y de unos a ceros: Ejemplo: 0010100 a 1101011.

• No se puede poner una variable que indique el número de bits que se desplazan.

Operadores especiales

Operador	Acción
sizeof	*Devuelve el número de bytes de la variable*
[,]	*Indica el índice de los arrays*
(,)	*Aumentan la precedencia de las operaciones*
,	*Encadena operaciones*
Exp1?Exp2:Exp3	*Si Exp1 Exp2, sino Exp3*

Ejemplo:

1.- x=10;
y=x>9 ? 100 : 200 ; Si x es mayor que 9 entonces y=100, si x no es mayor de 9, y=200.

2.- y=10;
x=(y=y+5, 30/y); La y es igual a 15 y la x igual a 30/15 es decir x=2.

3.- x[10]=100; El elemento 10 del array es igual a 100.

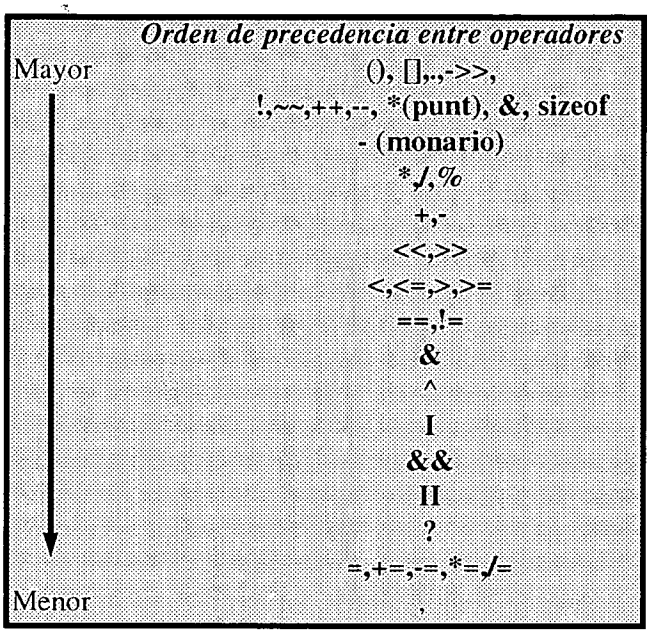
4.- int x;
b=sizeof(x); Devuelve el número de bytes de la varia ble x, es decir 2.

Abreviaturas

Operador	Acción
+=	*La variables es igual a si misma más ...*
-=	*La variable es igual a si misma menos ...*
*=	*La variable es igual a si misma por ...*
/=	*La variable es igual a si misma dividida por ...*
>>=	*Se desplaza N bits a la derecha*
<<=	*Se desplaza N bits a la izquierda*

Ejemplos

1.- x+=5; x es igual a x más 5.
2.- y-=x; y es igual a y menos x.
3.- y*=x; y igual a y por x.
4.- x/=6; x igual a x dividido por 6.



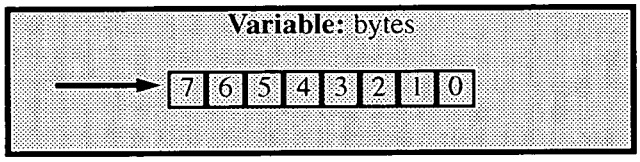
Problema 7

Se ha realizado un programa que simula la lectura de un puerto serie y la composición de cada byte mediante ocho lecturas del bit de datos del puerto. El programa realiza la composición de los bytes hasta que encuentra uno que sea igual a 13. Es el siguiente:

```
/* PROGRAMA RS.C */
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
unsigned char leer_puerto (void)
{
  char x;
  x=random(2)*16;
  x+=39;
  return(x);
}
main()
{
  unsigned char k,bytes,n;
```

```
int h;
do
{
        bytes=0;
        for (h=0;h<8;h++)
                {
                ...
                }
        gotoxy(10,wherey());
        printf(" El valor de bytes es=%03d\n",bytes);
}
while (bytes!=13);
getch();
return;
}
```

Suponiendo que el bit de datos es el bit número 4 del puerto serie, reconstruir el programa para que ocho bits de datos leídos consecutivamente formen el byte transmitido. Para ello se realiza un AND con el byte del puerto serie de forma que quede aislado el bit 4 de éste. Seguidamente se procede a convertirlo en bit 7 y mediante un OR asignarlo como bit 7 de la variable bytes. Por último se desplaza hacia la derecha el todos los bits.



4. SENTENCIAS DE CONTROL DE PROGRAMA.

La sentencia if

if (expresión) sentencia1; (Palabra clave del C)

else sentencia2;

Si es cierta la expresión, realiza la sentencia1, sino realiza la sentencia2. Las sentencias pueden estar formadas de varias instrucciones, para ello se introducen éstas entre llaves.

Ejemplos:

```
1.-   if (x==10)      printf("La x es igual a diez");
      else            printf("La x es distinta de diez");
2.-   if (x!=15)      printf("La x es distinta de quince");
3.-   if (x>10)       {
                      x=10;
                      printf("La x al ser mayor de 10 la iguala
                      mos con diez");
                      }
      else            {
                      y=x*10;
                      printf("y la igualamos con x por 10");
                      }
```

El operador ?

El operador ? puede reemplazar al if/else como operador ternario.

Ejemplo:

```
if (x>19) y=19;
else y=10;
```

Se puede cambiar por:

```
y= x>19 ? 19: 10;
```

El comando switch

```
switch (variable)          Comparar variable.
{
  case constante1:         ¿ Es igual a constante1 ?. Si es así realiza
  secuencia de instrucciones esta secuencia de instrucciones
  break;
  case constante2:         ¿ Es igual a constante2 ? Si es así realiza
  secuencia de instrucciones esta secuencia de instrucciones
  break;
```

```
  case constanten:          ¿ Es igual a constanten ? Si es así realiza
  secuencia de instrucciones  esta secuencia de instrucciones
  break;
  ...
  default:                  Si la variable no es igual a ninguna de
  secuencia de instrucciones  las constantes anteriores realiza esta
}                           secuencia de instrucciones.
```

Ejemplo:

```
c=getch();
switch(c)
{
        case 'a':
        case 'A':
                printf("Menu correspondiente a la tecla A");
                break;
        case 'b':
        case 'B':
                printf("Menu correspondiente a la tecla B");
                break;
        default:
                printf("No hay menu disponible para esa tecla");
}
```

Bucles: La instrucción for

El formato de la instrucción for es el siguiente:

for (inicialización; condición; incremento) sentencia;

Realiza la sentencia (o sentencias) hasta que se produzca la condición, efectuando el incremento cada vez y comenzando con la inicialización. El ejemplo tipo es:

```
int x;
for (x=0;x<100;x++) printf("\nValor de x=%d",x);
```

Inicializa la variable x con cero, imprime la x e incrementa x. Realiza ésto mientras la x sea menor que 100.

Ejemplos:

```
for (a=0, b=0; x<100; a+=5, b+=5) x=(a+b)/100;
```

Inicializa a y b con cero, realiza el cálculo de x incrementando a y b con 5

cada vez mientras x sea menor de 100. Lo mismo se puede escribir de esta forma:

Lo mismo que en el ejemplo se puede escribir de esta forma:

```
a=0;b=0;
for (;x<100;) {x=(a+b)/100; a+=5; b+=5;}
```

Bucles: La instrucción while.

El formato es:

while (condición) sentencia;

Mientras se produce la condición, es decir, mientras la condición es cierta realiza la sentencia. Primero mira si se cumple la condición y si es cierta, realiza la sentencia, si la condición es falsa salta a la linea siguiente.

Ejemplos:

```
1.-     x=0;
        while (x<100) x++;
2.-     char c;
        c=getch();
        while (c!=13)
        {
                printf("%c",c);
                c=getch();
        }
        printf("\n");
```

Bucles: La instrucción do/while.

Tiene el siguiente formato:

do sentencia;

while (condición);

Realiza la sentencia o serie de sentencias mientras se cumpla la condición. Se diferencia de la instrucción while en que ésta primero realiza la sentencia y después mira si se cumple la condición.

Ejemplos:

```
1.-     char c;
        do {
                c=getch();
```

```
            printf("%c",c);
        }
    while (c!=13);
```

2.-

```
    do
    {
            x++;
      }
    while (x<100);
```

3.-

```
    do
    {
            c=getch();
            switch(c)
            {
            case 'a':
            case 'A':
                    printf("Menu correspondiente a la tecla A");
                    break;
            case 'b':
            case 'B':
                    printf("Menu correspondiente a la tecla B");
                    break;
            default:
                    printf("No hay menu disponible para esa tecla");
            }
    } while (c!='a' && c!='b' && c!='A' && c!='B');
```

La instrucción break

La instrucción break tiene dos usos. Primero sirve para finalizar una instrucción case de un switch, ya visto anteriormente. Como segundo uso se utiliza para salir de un bucle anticipadamente.

Ejemplo:

1.-

```
    for (x=0;x<100;x++)
    {
            scanf("%d",&y);
            if (y==0) break;
            printf("%d",x/y);
```

```
        }
2.-     for (x=0;x<10000;x++)
        {
                printf("X=%d  ",x);
                if (kbhit()) break;          Si se pulsa una tecla sal
        }
```

La instrucción continue.

Fuerza una nueva iteración del bucle saltándose todo el código que exista entre medias.

Ejemplo:

```
x=-100;
do {
        if (x==0) continue;
        printf("DIVISION=%d",10.0/x);
} while (x++<200);
```

La instrucción exit().

(Libreria **stdlib.h**)

Provoca la finalización del programa y vuelta al sistema operativo. El argumento de exit() suele ser cero, pero se puede utilizar diferentes valores para indicar diferentes sitios de salida del programa.

Ejemplo:

```
c=getch();
if (c==27) exit(0);
```

La instrucción goto.

goto etiqueta;
...
...
etiqueta: ...

Desvía el flujo hacia la instrucción indicada por la etiqueta. La etiqueta y el goto tienen que pertenecer a la misma función. Nos es aconsejable utilizarlo, salvo en casos excepcionales (Ejemplo: salir de varios if anidados)

Ejemplo:

```
if (x==0)
        if (a==0)
                if (d==0) goto salida;
printf("Los numeros son distintos de cero");
exit();
salida:
printf("Los numeros son iguales a cero");
exit();
```

Problema 8

Realizar un programa que partiendo del juego de la ruleta, nos permita apostar una cierta cantidad de dinero contra una banca. El ordenador pedirá si se apuesta a par o impar generando posteriormente un número aleatorio entre 0 y 36 (mediante la instrucción random(37)).

Inicialmente partiremos con 1000 $ (variable efectivo) y la banca con 10000$ (variable banca). El ordenador nos pide si apostamos a impar o par y después la cantidad a jugar (variable apuesta) comprobando si tenemos efectivo suficiente. Una vez hecha la apuesta, el ordenador genera el número aleatorio y mira cual ha sido el resultado de la apuesta. Si ganamos se incrementa la apuesta a nuestro efectivo y se le resta la apuesta a la banca. Si perdemos restamos la apuesta de nuestro efectivo y se la sumamos a la banca. El ordenador en todo momento nos indicará el dinero que tenemos y el dinero de la banca.

El juego terminará cuando perdamos todo nuestro dinero o limpiemos a la banca. A la hora de apostar se puede implementar la opción de retirarse de la partida de esta forma.

printf("QUIERES APOSTAR A PAR/IMPAR o SALIR DE LA PARTIDA (p/i/s)");

Problema 9

Realizar el cálculo de las siguientes series:

a) ln 2=1 - 1/2 + 1/3 -1/4 +1/5 -1/6 + ···

b) p=4·(1 - 1/3 + 1/5 - 1/7 + 1/9 - ···)

c) ex= 1 + x + x2/2! + x3/3! + ···

5. FUNCIONES.

Expresión básica de una función

```
Especificador_de_tipo    nombre        (lista de parámetros)
{
  Lista de instrucciones que la componen
  return (argumento del tipo especificado)
}
```

• Especificador_de_tipo = Es el tipo de valor que nos va a devolver la función.

• Nombre= Nombre de la función.

• Lista de parámetros = Variables que reciben el valor cuando se llama a la función.

• Lista de instrucciones que la componen = Instrucciones que componen la función.

• Return = Instrucción que provoca el final de la función, devolviendo un valor del tipo al que está definida la función.Puede haber varios return, en distintas partes de la función

• Argumento del tipo especificado = Valor que devuelve la variable return. Tiene que ser del tipo de la función.

Ejemplos:

```
1.-   int calcula_cuadrado(int x)
      {
            int j;
            j*=x;
            return(j);
      }
2.-   calcula_cuadrado(int j)
      {
            return(j*j);
      }
3.-   float calcula_division(float k, float u)
      {
            float div;
            div=k/u;
            return(div);
      }
      main()
```

```
{
        printf("División=%f",calcula_division
        (10212.121,21222.0));
        return;
}
4.-     void pon_texto(char *car)
        {
                printf("TEXTO=%s",car);
        }
```

Uso de prototipos de funciones.

Un prototipo consiste en la declaración anticipada del valor que devuelve una función, de su nombre y de su lista de parámetros.

Ejemplos:

1.- Antigua.

```
int calcula_cuadrado();
main()
{
        printf("%d",calcula_cuadrado);
        return;
}
int calcula_cuadrado(int x)
{
        return(x*x);
}
```

2.- Actual.

```
int calcula_cuadrado(int x);
...
lo demás igual
```

3.- Mejor.

```
int calcula_cuadrado(int );
...
lo demás igual.
```

La definición permite al compilador la comprobación del uso correcto de las funciones y de si se llaman a las funciones con variables de diferente tipo de las que tiene asignadas.

Ejemplo:

```
int calcula_algo(int , float , double , char );
main()
{
        int o;
        o=calcula_algo(10.33, 1221, 21.1)
}
```

```
int calcula_algo(int a, float b, double c, char d)
{
        ...;
        ...;
}
...
```

dará error porque al llamar a la función sólo le damos valores a tres de los cuatro parámetros.

La función main()

La función main tiene definidas unas variables de esta forma:

```
main (int argc, char *argv[], char *env[])
{
...
}
```

Siendo argc el número de argumentos de la línea de órdenes. Cómo mínimo es uno, que corresponde al nombre del programa.

Argv[] es un puntero a un array de punteros de caracteres con los argumentos introducidos en la línea de órdenes. Por ejemplo, si llamamos desde el DOS a nuestro programa con los siguientes parámetros:

C> programa.exe uno dos tres

- argc=4
- argv[0]= Apunta a la cadena "programa.exe".
- argv[1]=Apunta a la cadena "uno".
- argv[2]=Apunta a "dos".
- argv[3]=Apunta a "tres"

De esta forma podemos ejecutar un programa escrito en C con varios parámetros. Ejemplo

C> fechahoy *.exe *.com

Puede ser un programa que cambie la fecha y la actualice de todos los ficheros con extensión .exe y extensión .com.

Env[] es un puntero a un array de punteros de caracteres que esta formado por todas las instrucciones "adicionales" del sistema. Pudiendo revisar variables propias que se han introducido en el sistema anteriormente.

El siguiente programa muestra el uso de estos argumentos:

```
#include <stdio.h>
int main(int argc, char *argv[], char *env[])
{
        int t;
        clrscr();
        printf("===== ARGUMENTOS INTRODUCIDOS==============\n");
        for (t=0;t<argc;t++)
                printf("%s\n",argv[t]);

        printf("\n=========ARGUMENTOS DEL SISTEMA ==========\n");
        for (t=0;env[t];t++)
                printf("%s\n",env[t]);
        printf("\n=====================================\n");
          getch();
          return 0;
}
```

Variables locales, globales, estáticas, externas, etc.

• Variable local: Son aquellas que se encuentran dentro de una función también reciben el nombre de variables automáticas llevan la palabra clave opcional auto. Se declaran dentro de la función y sólo se pueden utilizar dentro de la función. Realmente cuando el programa ejecuta el inicio de la función asigna espacio en la memoria para esa variable, cuando termina la función esa memoria se libera. Pueden existir variables locales con el mismo nombre pero en diferentes funciones.

Los parámetros de una función, son también variables locales.

• Variables globales: Son aquellas que se definen fuera de las funciones, se conocen en todo el programa manteniendo su valor durante todo el desarrollo del programa. Es muy aconsejable declarar todas las variables globales al principio del

programa. Cuando existen dos variables con el mismo nombre y una sea local y la otra global, la función que tenga la variable local se referirá a ella.

• Variables externas: El lenguaje Turbo C permite enlazar varios archivos a la vez. Si se quiere utilizar la variable global de un archivo en otro archivo se recurre a la asignación extern. De esta forma en una archivo se define la variable y en otro se dice que esa variable corresponde a otro archivo.

Archivo 1: int a; Archivo 2: extern int a;

• Variables estáticas: Se definen con el modificador static a aquellas variables que las definimos como variables permanentes, manteniendo su valor entre llamadas. Si la variable estática es local el valor cuando sale de su función se mantiene, no se destruye al salir de la función:

Ejemplo:

```
int cuenta (int y)
{
static int e=0;
if (y) return e;
else    e++;
return 0;
}
```

Si la variable estática es global solo puede ser llamada por las funciones de su propio archivo. De esta forma si introducimos esas funciones con esa variable global estática en una biblioteca, sólo es usada por las funciones de su archivo, pudiéndose utilizar en otras partes del programa otra variable con el mismo nombre sin que exista ningún error. Las variables static permiten al programador crear partes de un programa extenso que no puedan ser modificados por error desde otras partes del mismo.

• Variables registro: Solo se puede aplicar a variables locales tipo int o char. Le especifica al compilador para que esa variable permanezca en un registro de la CPU en vez de ocupar memoria. De esta forma el acceso a la variable es mucho más rápido. El compilador utiliza a la vez dos registros de la CPU para variables register como máximo. Normalmente se especifica en contadores de bucle, o variables que van a utilizarse mucho en la función. De esta forma se acelera considerablemente los programas.

El compilador asigna normalmente por defecto las dos primeras variables como register para optimizar los programas. El especificador es register.

Ejemplo: register int h;

Problema 10

Una estación meteorológica, nos pide que realicemos un dispositivo capaz de determinar la distancia de un Rayo a partir de su relámpago y el sonido que produce, considerando que la velocidad de la luz es infinita con respecto a la velocidad del sonido que la estimamos en 330 metros por segundo. El dispositivo electrónico está compuesto por una célula fotosensible y un micrófono debidamente ajustados. El circuito utiliza el puerto paralelo (puerto de impresora) dándonos la información mediante el byte de estado de impresora. Cuando el bit 7 está a uno se ha activado la célula, es decir, ha ocurrido el relámpago. A partir de este momento el programa calculará el tiempo hasta que se activa el bit 6 del estado de impresora que será cuando ha sonado el trueno mediante la instrucción clock().

Como simulación aleatoria del relámpago se realizará una función que calcule un número aleatorio entre 0 y 20000. Repitiendo este proceso hasta que ese número sea 1000 (el número se almacenará en una variable local tipo register unsigned int j). Cuando el programa salga de este bucle se producirá el relámpago, almacenando el primer valor aproximado del tiempo en la variable tipo long tiempl y saliendo de la función. La duración entre el relámpago y el trueno la simularemos con otra función, mediante dos bucles anidados uno dentro de otro mediante dos variables locales tipo register unsigned int h y j. La variable h formará el bucle interno con un valor 5000 y la variable j será la variable del bucle externo, con un valor aleatorio entre 0 y 30000. Una vez han finalizado estos bucles se almacena en la variable global tiemp2. La función devolverá la diferencia long2-longl.

Sabiendo que esta diferencia dividida por 18.2 nos da los segundos de tiempo, calcular la distancia estimando 330 metros por segundo, como la velocidad del sonido en el aire y almacenarla en la variable global dist. Sacar en pantalla las tres variables globales.

6. LOS ARRAYS.

Un *array* es una colección de variables que se denominan por un mismo nombre y se determinan por un índice. Un array es una matriz de n dimensiones.

Arrays unidimensionales

Forma general:

tipo *nombre_de_variable[tamaño];*

Ejemplos:

```
char   cadena1 [10];   Crea una matriz de 10 caracteres numerados desde el 0 al 9.
int    datos [100];    Matriz de 100 enteros con índices del 0 al 99.
```

Tamaño en memoria (bytes)= sizeof(tipo)*número de elementos.

Ejemplo:**int a[6];**

Indice (Variable int)	Posición en memoria	Valor
a[0]	Dir_inicial+0	1000
a[1]	Dir_inicial+2	25000
a[2]	Dir_inicial+4	-2311
a[3]	Dir_inicial+6	-1312
a[4]	Dir_inicial+8	-11222
a[5]	Dir_inicial+10	+12222

Cuando asignamos un array, el compilador deja espacio para ese número de elementos. Como el lenguaje C es un lenguaje desarrollado para programadores que quieran realizar programas a alta velocidad, éste no realiza las comprobaciones para ver si estamos utilizando índices que se salen del array. Por ejemplo: si nos fijamos en la tabla anterior, observamos que está formado por 6 elementos, si nosotros introducimos un valor a la variable con índice a[10] de esta forma:

```
a[10]= 21234;
```

se asignará a la dirección 10000+(10*2)=10020, como sólo está asignada memoria para 6 variables, el programa estará introduciendo datos en otra parte, que puede ser código del programa, pudiendo realizar auténticos desastres. **Hay que tener especial cuidado con el límite de los arrays.**

Arrays bidimensionales

Tienen el siguiente formato:

tipo *nombre_de_variable [número de filas][número de columnas];*

Tamaño en bytes= sizeof(tipo)*número de filas*número de columnas;

Ejemplo:

```
#include <stdio.h>
main(void)
{
int h,j;
int var[5][5];
for (h=0;h<5;h++)
        {
        for (j=0;j<5;j++)
                {
                var[h][j]=h*j;
                printf("Variable var[%d][%d]=%d",h,j,var[h][j]);
                }
        }
getch();
return 0;
}
```

Posición en memoria de la variable var[ ][ ];

1ª pos. var[0][0];	2ª pos. var[0][1];	3ª pos. var[0][2];	4ª pos. var[0][3];	5ª pos. var[0][4];
6ª pos. var[1][0];	7ª pos. var[1][1];	8ª pos. var[1][2];	9ª pos. var[1][3];	10ª pos. var[1][4];
11ª pos. var[2][0];	12ª pos. var[2][1];	13ª pos. var[2][2];	14ª pos. var[2][3];	15ª pos. var[2][4];
16ª pos. var[3][0];	17ª pos. var[3][1];	18ª pos. var[3][2];	19ª pos. var[3][3];	20ª pos. var[3][4];
21ª pos. var[4][0];	22ª pos. var[4][1];	23ª pos. var[4][2];	24ª pos. var[4][3];	25ª pos. var[4][4];

Arrays multidimensionales

Su definición es:

tipo *nombre_variable[tamaño1][tamaño2]...[tamañoN];*

y el tamaño que ocupa en bytes es:

Tamaño en bytes=sizeof(tipo)*tamaño1*tamaño2*...*tamañoN.

Inicialización de arrays

Los arrays se pueden inicializar al definirlos:

Ejemplos:

```
int num[10]={1,2,3,4,5,6,7,8,9,10};
int multi[4][4]={1,2,3,4,2,4,6,8,3,6,9,12,4,8,12,16};
char cadena[7]={'J','A','V','I','E','R','\0'};
char cadena[7]="JAVIER";
```

Un array multidimensional se puede especificar de forma indeterminada, y dejar que el compilador asigne el espacio correspondiente:

Ejemplo:

```
int multi[][4]={1,2,3,4, 2,4,6,8, 3,6,9,12, 4,8,12,16};
```

De esta forma le decimos al compilador que cada fila está formada por cuatro elementos.

Ejemplos:

```
int i[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17};
char cad_p[]="Hola esto es una cadena";
int is[][3][2]={1,2,3,4,5,6,...,3,5,5,3,3};
```

Cadenas

Una cadena consisten en un sucesión de caracteres que termina en un carácter nulo "\0", de esta forma hay que considerar la longitud de la cadena como el número de caracteres más uno:

Ejemplo:

```
char cadena[19]="Esto es una cadena";
```

Como cada carácter ocupa un byte (tipo char) una cadena se define como un **array unidimensional** de N caracteres:

Las cadenas se designan de la siguiente forma:

```
char cadena1[122];
```

Cómo puede ser muy tedioso el sumar todos los caracteres de una cadena, se puede dejar al compilador que determine su longitud cuando inicializamos una cadena, de esta forma:

Ejemplo:

```
char cadena2[]="ESTO ES UN ARRAY INDETERMINADO";
```

Un array de cadenas consiste en un número determinado de cadenas como un array bidimensional, de la siguiente forma:

```
char nombre_cadenas[num_cadenas][longitud de las cadenas];
```

Ejemplo:

```
char errores[][25]={"ERROR DE SINTAXIS\n", "ESE ARCHIVO NO EXISTE\n", "VALOR DEMASIADO GRANDE\n", "VALOR SOLO POSITIVO\n"};
```

Forma de pasar los arrays a una función

Como no se pueden pasar todos los elementos de un array a una función, los que se indica es la dirección inicial del array. Por ejemplo si tenemos el array unidimensional c[] se pasará a la función sólo el nombre del array, de esta forma:

```
funcion(char *x)          Puntero tipo char
{
  ...
  ...
}
main(void)
{
  char c[123];
  funcion(c);             Manda la dirección inicial del char.
  ...
  ...
}
```

Funcion() puede tener definida de tres formas la variable de entrada:

- Puntero: funcion(char *x)
- Array delimitado: funcion(char x[123])
- Array sin delimitar: funcion(char x[])

La primera forma, consiste en declarar un puntero. Un puntero consiste en una variable que apunta a una dirección y que sabemos que es de un formato determinado. Por ejemplo char *x indica que x es una dirección donde están valores tipo char que se acceden mediante el operador de puntero *x.

En la segunda definición, no asigna memoria a 123 elementos tipo char, sino que le decimos a la función que va a recibir un array tipo char de 123 elementos, aunque la longitud no tiene importancia por que el Turbo C no tiene en cuenta los límites de los arrays.

La elección más aceptable es la primera, la del puntero.

Tomando en cuenta char c[21] podemos advertir que es lo mismo:

c	Dirección inicial del array c[]
&c[0]	Dirección del elemento primero del array c[]

Arrays bidimensionales

La dirección inicial de un arrays bidimensional se determina de la siguiente forma:

char cadena[12][30];

Dif. formas de pasar elementos de un array.

funcion(cadena[2])	Pasa la dirección de la tercera cadena
funcion(cadena)	Pasa la dirección de la primera cadena
funcion(cadena[2][3])	Pasa el carácter cuarto de la cadena tercera

Formas que puede recibir una función

funcion(char *x)	Recibe la dirección de un puntero tipo char
funcion(char c)	Recibe un carácter
funcion(char c[])	Recibe un array unidimensional, char (una cadena)
funcion(char c[][3])	Recibe un array bidimensional

Para array tipo char cadena[10][10];

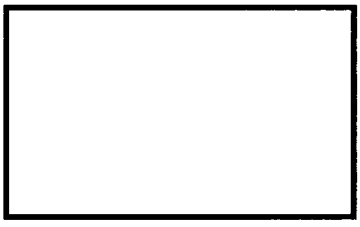
cadena	dirección inicial de la cadena
cadena[0][0]	primer elemento de la cadena
cadena[2]	Dirección inicial de la fila 2
cadena[][5]	Array de cadenas con un núm indet. de cad.
cadena[3][3]	Elemento 3,3 del array de cadenas
&cadena[3][3]	Dirección del elemento 3,3

Instrucciones para el problema 11

gotoxy(int x, int y); (Libreria **conio.h**)

Posiciona el cursor en la columna x, y fila y de la pantalla. Normalmente en modo texto las colum nas son 80 (desde 1 hasta 80) y las filas 25 (desde 1 al 25).

(1,1)



(25,80)

Problema 11

El programa a realizar consistirá en una serie de funciones que nos haga la multiplicación de dos matrices y su correspondiente visualización. La función que realice la multiplicación necesitará 5 variables: fila, el número de filas de la primera matriz; columna, el número de columnas de la primera matriz; los datos de la primera matriz en un array int m1[10][10]; los datos de la segunda matriz en un array int m2[10][10]; una matriz long r[10][10] donde se almacenará el resultado de la multiplicación. La fila y la columna de la segunda matriz corresponde al contrario de la matriz primera, ya que para que dos matrices se puedan multiplicar es necesario que el número de filas de una corresponda al número de columnas de la otra, y viceversa.

Otra función nos pedirá el número de filas y de columnas y los valores de cada matriz. Llamándose dos veces una para cada matriz que interviene en la multiplicación.

*Otra función imprimirá los datos de la matriz resultado en pantalla, situando cada dato en la posición (1+colu_dato*8,1+fila_dato*2).*

Cada dato de una matriz no puede ser mayor de 999 ni número negativo. El número de filas o de columnas no puede exceder las 10 ni ser menor de 1.

Todos los errores que ocurren al introducir números erróneos, los imprimirá en pantalla una función que incluya un array de cadenas. Una cadena para cada error.

Problema 12

Disponemos de la siguiente tabla:

	M=100	300	500	800	1200	1800	2500	3100	4200
L=50000	0.12	1.56	1.56	2.12	2.78	1.32	1.45	1.45	1.65
70000	0.12	0.23	0.54	0.98	2.12	1.23	0.65	0.41	0.52
95000	0.41	0.12	0.21	1.21	1.23	1.65	2.54	2.71	1.76
102000	0.92	0.12	0.56	0.74	0.12	0.45	1.21	1.20	1.03
121000	3.33	2.22	2.01	2.12	2.12	2.02	3.00	1.01	1.02
128000	1.23	1.21	3.01	2.11	0.14	0.55	0.67	0.69	098
139000	3.22	2.12	1.32	1.24	1.22	1.54	0.14	0.62	2.51
150000	0.33	1.02	1.54	0.25	0.12	0.32	0.21	1.21	1.32
173000	2.12	1.45	1.21	1.56	1.89	1.85	1.95	1.68	1.94
189000	2.14	2.65	2.98	2.91	2.02	2.06	2.51	2.24	2.02
200500	2.1	3.2	3.5	3.6	3.8	3.10	3.12	4.01	1.02

Inicializar tres arrays, una array con los valores de la primera fila, otro con los valores de la primera columna y el último array bidimensional con los restantes valores. El programa nos pedirá un valor a buscar en la fila M y uno en la fila L, de forma que localizará el valor más próximo en cada uno. El programa devolverá el valor de la matríz unidimensional que corresponda. Por ejemplo para M=305 y L=138000 el valor que nos encontrará el programa será el 2.12.

Problema 13

El programa a realizar servirá para codificar una serie de datos mediante una clave de forma que no puedan ser leidos sin no se conoce ésta. Dado un array tipo char de N elementos y un array con la clave de M elementos de longitud; el método de criptografía se basa en realizar un XOR de cada byte del array de datos con cada byte del array donde está almacenado la clave. De forma que si se vuelve a hacer, recuperamos la información.

Ejemplo:

char clave[]={"PEPITO"};

char datos[]={"ESTA INFORMACION ES CONFIDENCIAL"};

método: **datos[0]=datos[0] XOR clave[0]**

datos[1]=datos[1] XOR clave[1]

datos[2]=datos[2] XOR clave[2]

...

7. LOS PUNTEROS.

La programación en Turbo C tiene como arma más potente el uso de punteros. Los punteros nos permiten además de pasar argumentos complejos a las funciones, aumentar la eficacia de ciertas rutinas. Por otra parte su uso inadecuado resulta **muy peligroso.**

Definición: **Un puntero es una variable que contiene una dirección de memoria.** Esa dirección puede apuntar a otra variable, una función, una zona de memoria especial, etc.

Su forma general: **tipo** *nombre_de_la_dirección;

El nombre de la dirección, corresponde al nombre del puntero que apunta a una dirección determinada. El tipo corresponde al formato de la variable a la que apunta. Por ejemplo, si el puntero es tipo int cada vez que incrementemos éste lo hará de dos en dos bytes. Es muy importante tomar en cuenta ésto a la hora de realizar aritmética con punteros.

Ejemplo: int *punt_x;

Ejemplo

Dirección en memoria	Valor	Nombre de la Variable
1000	1003	Puntero **puntx**
1001		
1002		
1003	5	Var. **calculo**

En el ejemplo anterior tenemos el puntero puntx que apunta a la variable calculo.

Los operadores de punteros

&: **Devuelve la dirección de...**

Ejemplo:

puntx=&calculo; puntx recibe la dirección de calculo, es decir 1003.

* **Lo que hay en la dirección que apunta ...**

Ejemplo:

```
char c;
c=*puntx;          c recibe lo que hay en la dirección que
                   apunta puntx, es decir el valor 5.
```

Programa de ejemplo:

```
#include <stdio.h>
main(void)
{
        char c,calculo;
        char *puntx;
        calculo=5;
        puntx=&calculo;
        c=*puntx;
        printf("VALOR DE C=%d",c);
        getch();
        return;
}
```

!!! Cuidado !!! Si utilizamos un tipo distinto para el puntero, el valor que devolverá corresponderá a este tipo produciendo posiblemente un error en el programa sin ser detectado por el compilador. Por ejemplo, si el puntero fuera de tipo int, el valor que devuelve es entero. Los punteros también deben apuntar a datos del tipo correcto.

Aritmética de punteros

Con los punteros sólo se puede utilizar dos operaciones matemáticas: la suma y la resta.

A la hora de utilizar la aritmética de punteros, hay que tomar en cuenta que: **Cada vez que se incrementa o decrementa un puntero, éste lo hace en relación a su tipo base.** Por ejemplo, si tenemos un puntero de tipo base int, que apunta a una dirección 2000, si incrementamos éste apuntará a la dirección 2002. Si es tipo char irá de uno en uno, si es long de cuatro en cuatro, etc.

Ejemplo:	int *p;	Dirección	2000
	p++;	Dirección	2002
	p=p+5;	Dirección	2002+(5*2)=2012
	p—;	Dirección	2010
Ejemplo:	char *p;	Dirección	2000
	p++;	Dirección	2001
	p=p+5;	Dirección	2001+5=2006
	p—;	Dirección	2005

Ejemplo:

```
float *p;          Dirección   2000
p++;               Dirección   2004
p=p+5;             Dirección   2004+(5*4)=2024
p—;                Dirección   2020
```

Ejemplo:

```
char *p,c;
char cade[]="ABCDEFGHIJK";
p=cade;
c=*p+1;            c es igual a 1 más el valor del elemento de
                   la dirección p
c=*(p+1);          c es igual al valor de la dirección (p+1)
```

!!! Cuidado !!!. No es lo mismo *p+1 que *(p+1)

Otra operación que se puede realizar es la resta de dos punteros que apuntan a un mismo array, generalmente un cadena, para ver cuantos elementos hay entre uno y otro. No se pueden sumar, ni multiplicar, ni dividir, etc.

Ejemplos:

```
#include<stdio.h>
#include<conio.h>
main()
{
        char *p,*p1;
        int car;
        char cadena[]="BUENOS DIAS MISTER";
        p=p1=cadena;
        while (*p1!=' ')p1++;
        car=p1-p;
        printf("Longitud 1ª Palabra=%d",car);
        getch();
        return;
}
```

Comparar punteros

Se utilizan en punteros que apuntan a un objeto común:

```
Ejemplo:    if (p<q)      printf("p apunta a una dirección de memo
                          ria menor que q");
```

Punteros y arrays

Los punteros y los arrays están muy relacionados entre sí. El nombre de una array sin índice es la dirección del primer elemento.

Ejemplo:

```
char cade[]="HOLA";
char *p;
p=cade;                 Es lo mismo
p=&cade[0];             Es lo mismo
```

Ejemplo:

```
int array1[10][8];
int *p,h;
p=array;
h=array[5][4];          Elemento quinto de la fila sexta.
                        Empiezan por cero.
h=*(p+(5*8)+4);         Dirección inicial más
                        (fila*longitud_fila)+elemento_fila
```

Hay que recordar que los arrays comienzan en 0. Por tanto el elemento quinto de la fila sexta es [5][4].

Para saber que método utilizar se recurre al siguiente axioma:

• Si vamos a utilizar los elementos del array en forma secuencial, se recurre a los punteros

• Si el acceso va a ser de forma aleatoria se recurre a los índices.

Problema 14

Realizar una función cuyo argumento sea un puntero tipo char para poder pasarle a ésta una cadena de caracteres. Toda cadena termina con el carácter nulo \0, es decir con un cero. Sabiendo ésto, la función contará el número de caracteres de la cadena sacando este número en pantalla. Posteriormente el programa empezando desde el final de la cadena, imprimirá en cada línea de la pantalla cada palabra al revés. Para ello detectará el espacio o la coma , como separación de palabras.

Problema 15

Realizar un programa que pida una cadena y nos diga cuantas vocales hay en ella.

Problema 16

Realizar una función que tenga como argumento un puntero char de forma que reciba la dirección de una cadena, a esta cadena el programa le eliminará todos los espacios, comas y puntos.

Técnicas Avanzadas con Punteros

Devolución de punteros

Para crear una función que devuelva punteros, se define ésta como un puntero con el tipo base de la variable de la cual se va a devolver su dirección.

Ejemplo:

```
char *busca_espacio(char *cade)        Función tipo puntero que pide la
                                       dirección inicial de una cadena
                                       (puntero cade).
        {
        while (*cade!=' ') cade++;     Busca carácter 'espacio'.
        return (cade);
        }
```

Arrays de punteros

Una array de punteros es un conjunto de punteros que se pueden utilizar mediante un índice, al igual que un array.

Ejemplo:

```
int *punt[10];
int h,i,j,k,l,m;
int a,b,c,d,e,f
punt[0]=&h;
punt[1]=&i;
punt[2]=&j;
punt[3]=&k;
punt[4]=&l;
punt[5]=&m;
a=*punt[0];          Valor de h
b=*punt[1];          Valor de i
c=*punt[2];          Valor de j
d=*punt[3];          Valor de k
e=*punt[4];          Valor de l
f=*punt[5];          Valor de m
```

Ejemplo:

```
char *errores[]={"Número demasiado grande\n", "Número demasiado
                 pequeño\n"
                 , "Dispositivo no responde\n", "Error de lectura\n", "Error
                 de escritura"};
```

En el ejemplo anterior, se ha creado un array de punteros que apunta cada uno a la dirección inicial de cada cadena. Para imprimir la cadena "Número demasiado pequeño\n":

```
printf("%s", errores[1]);
```

Punteros a punteros

Un array de punteros es lo mismo que un puntero a punteros. Si consideramos que en el ejemplo anterior teniamos un array de punteros, podemos asignarle la dirección inicial de ese array a otro puntero, es decir un puntero que apunte al primer puntero del array. Se define de la siguiente forma:

```
int     **punt;
```

Indica que punt es un puntero a otro puntero que apunta a una variable int.

Ejemplo:

```
#include <stdio.h>
main()
{
        int a, *p, **punt;
        a=888;
        p=&a;
        punt=&p;
        printf("Valor de a=%d",**punt);
        return;
}
```

Dirección en memoria	Valor	Nombre de la Variable
996	1004	Puntero **punt
998		
1000		
1002		
1004	1010	Puntero *p
1006		
1008		
1010	888	Var. a

Un puntero a punteros normalmente se utiliza para apuntar a un array de punteros, siendo este array a si mismo variable.

Punteros a funciones

Todas las funciones aunque no son variables, tienen una posición física en la memoria, es decir, una dirección en memoria. Se puede asignar un puntero para que apunte a esa dirección y poder ser llamada la función mediante este puntero.

Toda función funciona como un array, es decir, si sólo utilizamos el nombre sin sus paréntesis nos estamos refiriendo a su dirección en memoria.

Un puntero a función se designa de la forma siguiente:

tipo (*nombre) ();

Normalmente si en un programa tenemos un menú de selección grande con muchas opciones utilizaremos la instrucción switch() para designar cada llamada a una función. Una forma más eficiente consiste en designar un array de punteros a funciones y llamar a la función correspondiente dependiendo de la opción elegida.

Ejemplo:

```
void    (*opcion[])()={imprimir, introducir_datos, grabar, recuperar,
        borrar, fin};     Funciones
main()
{
  int i;
```

```
  i=menu();                    Pide opción
  (*opcion[i])();              Llama a la opción número i.
  return;
}
```

Si la función va a tener argumentos:
Ejemplos de definición:
. int (*p) (char *, char *);
. char (*p[10]) (int , int);
. void (*p[10]) (int *, char , int);

Problemas con los punteros

Los punteros aunque dan una gran potencia a la hora de programar, pueden producir grandes errores en el programa. Estos son los errores más comunes:

Declarar un puntero sin asignarle un valor. De esta forma, el puntero tiene una dirección desconocida y si se utiliza para introducir datos, podemos introducir datos en cualquier parte del programa.

Ejemplo:

```
void funcion1()
{
        int *p, h=124;
        *p=h;
        ...
}
```

El puntero no está inicializado con ninguna dirección, de forma que al introducir el valor de h en la dirección p que puede estar dentro del código del programa o en los datos.

Inicializar erroneamente un puntero.

Ejemplo:

```
void funcion2()
{
        int *p,x=100;
        p=x;
        ...
}
```

En vez de inicializar el puntero con la dirección de x, lo hacemos con el valor del x, 100.

Problema 17

El programa a realizar consistirá en un menú con las siguientes opciones:

1.- Pasar número decimal a binario
2.- Pasar número binario a decimal
3.- Pasar número decimal a hexadecimal
4.- Pasar número hexadecimal a decimal
5.- Pasar número binario a hexadecimal
6.- Pasar número hexadecimal a binario
7.- Salir del programa

El texto del menú estará formado por un array de cadenas, y la llamada a cada función por una array de punteros a función. Realizar las funciones de cada opción.

8. ESTRUCTURAS, UNIONES Y ENUMERACIONES.

Estructuras

Una estructura es una colección de variables que aparecen bajo un mismo nombre. Su forma tipo es:

```
struct  nombre_tipo {
                variables;
                ...
                } nombres_de_estructuras;
```

Siendo nombre_tipo el nombre del tipo de estructura y nombres_de_estructuras las diferentes estructuras definidas.

Ejemplo:

```
 struct   dd1 {
          char nombre[40];
          long dni;
          char anio_nac;
          char mes_nac;
          char dia_nac;
          float peso;
          float altura;
          }        persona1;
```

Para acceder a un elemento de una estructura se designa el nombre de la estructura seguido de un punto y el nombre del elemento.

Ejemplo:

```
          persona1.dni=16561436;
          printf("%ld",persona1.dni);
          printf("\INTRODUCE NOMBRE:");
          gets(persona1.nombre);
```

Se puede definir una estructura y luego designar las estructuras:

Ejemplo:

```
 struct   dd{
          char nombre[40];
          char mes_nac;
          char anio_nac;
          char dia_nac;
          };
 struct dd pers1;
```

Array de estructuras

El uso más común y utilizado es el del array de estructuras, consiste en definir N estructuras.

Ejemplo:

```
struct ff {
        char nombre[40];
        long dni;
        char direccion[50];
        } fichas[1000];
```

Para acceder al dni de la ficha 500, se escribe:

```
fichas[500].dni=121313;
```

o para acceder al carácter 10 del elemento char nombre de la ficha 100:

```
fichas[100].nombre[10]='a';
```

Paso de elementos a estructuras

El paso del elemento de una estructura a una función se realiza del mismo modo que una variable normal:

Ejemplos:

```
struct ff {
        char nombre[40];
        long dni;
        char direccion[50];
        } fichas[1000];
funcion(fichas[500].nombre[4]);
funcion(fichas[100].dni);
funcion(fichas[200].direccion);
funcion(&fichas[150].dni);
funcion(&fichas[200].nombre[4]);
```

Pasa el carácter 4 de nombre.
Pasa el valor de dni.
Pasa la dirección inicial de la cadena direccion.
Pasa la dirección de la variable dni de la ficha 150.
Pasa la dirección del carácter 4 de nombre.

Paso de estructuras completas a funciones

Cuando se define una estructura como argumento de una función, los valores de la estructura se trasladan a la que forma parte como argumento de la función. Es importante por lo tanto que sea del mismo tipo las dos estructuras. Para ello lo más acertado es definir la estructura globalmente.

Ejemplo:

```
...
struct tipo_a {
        int a,b,c;
        char d;
        };
void func(struct tipo_a mm)
{
        int h;
        char c;
        h=mm.a*mm.b*mm.c;
        c=mm.d;
        ...
        return;
}
main(void)
{
        struct tipo_a jj;
        jj.a=10;
        jj.b=20;
        jj.c=10;
        jj.d='e';
        func(jj);
        ...
        return;
}
```

Punteros a estructuras

Los punteros a estructuras sirven para pasar la dirección de una estructura y poderla modificar desde otra función.

Se definen de la siguiente forma:

Ejemplo:

```
...
struct direc {
        int a,b,c;
```

```
            char f;
            } ficha;
  struct direc *p1;
  p1=&ficha;
  ...
```

De esta forma para pasar un puntero a una función, se **declara el puntero como parámetro y se pasa su dirección.**

```
struct direc {
            int a,b,c;
            char f;
            };

funcion (struct direc *p2)
{
  ...
}

main()
{
  ...
  struct direc ficha;
  funcion (&ficha);
  ...
}
```

Para poder acceder a un elemento de la estructura, hay que utilizar el operador flecha (signo menos seguido del signo mayor). Por ejemplo, para ver el elemento f del puntero p2 del programa anterior, lo designamos de la siguiente forma:

printf("Valor de f=%d",p2->f); *Se usa el operador flecha, en vez del punto.*

Se accede a los elementos de la estructura usando el puntero *p2 de la siguiente forma:

- p2->a
- p2->b
- p2->c
- p2->f

Estructuras anidadas

Cuando una estructura está dentro de otra estructura se le denomina estructura anidada.

Ejemplo:

```
struct persona {
                char nombre[40];
                long dni;
                long telefono;
                char direccion [40];
                };
struct direc {
        int a[10][20];
        struct persona juan;
        float f;
        } ficha;
```

Los elementos se designarían de la siguiente forma:

- ficha.a[5][10]=100;
- ficha.f=22.22;
- ficha.juan.nombre="PERICO DE LOS PALOTES";
- ficha.juan.dni=16562622;
- ficha.juan.telefono=221928;
- ficha.juan.direccion="Calle la huerta x";

Problema 18

Desarrollar un programa que sirva para almacenar los datos de los clientes de un videoclub y que esté formado por las siguientes funciones:

- ***Función para introducir las fichas.***
- ***Función para buscar una ficha introduciendo el DNI.***
- ***Función para buscar con una cadena dentro de los campos: nombre, dirección y película; mediante la función strstr() de la librería string.h.***

Cada ficha estará formada por una estructura con cuatro variables: nombre, dirección, DNI y vídeo.

Campos de bits

En una estructura además de los diferentes tipos de variables que le podemos asignar, existe la posibilidad de acceder a bits individualmente. Se definen de la siguiente forma:

Ejemplo:

```
struct   nombre {
         float f;
         long li;
         unsigned hay_cliente : 1;
         unsigned tiene_ss : 1;
                  nivel : 4;
                };
```

Los campos de un solo bit se designan unsigned. Si definimos un campo con varios bits y no lo declaramos unsigned, el bit más significativo será bit de signo.

La máxima longitud de un campo de bits es 16 bits.

No se puede tomar la dirección de un campo de bits, no se puede saber si van de derecha a izquierda o al revés.

Uniones

Una unión es una posición de memoria que es utilizada por diferentes variables. Se declaran como una estructura:

```
...
union  gg {
           int i;
           unsigned char c;
           };
union gg nnn;
nnn.i=65535;
printf("%d",nnn.c);
...
```

Enumeraciones

Se denomina enumeración a un conjunto de variables enteras que van recibiendo un valor. Empiezan con un valor cero y va incrementándose.

Ejemplo:

enum meses {enero=1, febrero, marzo, abril, mayo, junio, julio, agosto, septiembre, octubre, noviembre, diciembre};

enum meses mes; Define a la variable mes del tipo de las enum.

Para imprimir el valor de una variable enum hay que poner un molde para evitar mensajes de 'warnings'.

Ejemplo:

printf("%d",(int)febrero);

Typedef

Asigna otro nombre a un tipo de formato:

Ejemplo: typedef long largo;

largo dni; Es lo mismo que poner long dni

Con el typedef no se sustituye el nombre de un tipo con otro nombre, sino que creamos otra forma de llamar a ese tipo. Es decir, en el ejemplo anterior podemos definir una variable tipo long de dos formas con la palabra long o la palabra largo.

Problema 19

Decidimos crear una estructura que está formada por los siguientes campos:

- ***4 bits indicando el tipo de elemento a poner en pantalla: 0 icono, 1 rectán gulo, 2 rectángulo-relleno, 3 marco, 4 texto, ...***
- ***1 bit si está a cero no imprime el elemento, si está a uno si.***
- ***1 bit para el rectángulo y el marco. Si es uno borra lo de dentro, si es cero no.***
- ***int x,y;***
- ***int x2,y2;***
- ***int anchura,altura;***
- ***int color;***
- ***int numero_icono;***
- ***char texto[80];***

De esta forma mediante un array de estructuras podemos definir los elementos que van a aparecer en la pantalla. Mediante un enum determinaremos el nombre de cada elemento.

9. FUNCIONES DE ENTRADA Y SALIDA, ARCHIVOS DE DISCO.

Entrada y salida por consola (teclado-pantalla).

int getche(void);	Lee un carácter del teclado y lo imprime en panta lla, devuelve el carácter en el byte bajo.
int putchar(int c);	Imprime en pantalla el carácter c.
int getchar(void);	Va guardando en un buffer de entrada los carácte res hasta que se pulsa el ENTER.
int getch(void);	Lee un carácter del teclado pero no lo imprime.
char *gets(char *cad);	Lee una cadena del teclado hasta que se pulsa el ENTER. Se introduce en el array cad.
	Ejemplo: gets(cadena);
char *puts(char *cad);	Imprime una cadena, más rápido que printf.
int prinf(const char *cad_fmt, variables);	Imprime con formato.
int scanf(const char *cad_fmt, &variables);	Lee datos desde el tecla do.

Entrada y salida a archivos

Existen tres sistemas de acceso de E/S:

• ANSI, Sistema de archivos con buffer. (Librería stdio.h)

• UNIX, sistema de archivos sin buffer. Desarrollado para los compiladores de C de UNIX. (Librería conio.h)

• Funciones de bajo nivel que acceden directamente al hardware de la computadora.

El sistema ANSI C

Las funciones del sistema ANSI C se encuentran en la librería stdio.h y son las siguientes:

Tabla de las funciones ANSI C

Función	Lo que hace
fopen();	Abre un flujo
fclose();	Cierra un flujo
putc();	Escribe un carácter en el flujo.
getc();	Lee un carácter en el flujo.
fseek();	Se sitúa en una parte del flujo.
fprintf();	Imprime en un flujo
fscanf();	Lee valores del flujo
feof();	Devuelve cierto si es el final del archivo
ferror();	Devuelve cierto si se ha producido un error
rewind();	Se coloca al principio del archivo
remove();	Elimina un archivo

La función fopen()

La función fopen abre un flujo y lo asocia a un archivo cuyo nombre se especifica. La función fopen devuelve un puntero tipo FILE, que es un puntero a una información que define varias cosas sobre él, nombre, estado y posición actual. Tiene como prototipo:

FILE *fopen(const char *nombre_archivo, const char *modo);

Siendo nombre_archivo el nombre del archivo con el que se van a almacenar los datos. Siendo modo el formato de los datos y el estado de apertura de los datos.

Valores de modo

Modo	Significado
"r"	Abre un archivo para lectura
"w"	Crea un archivo para escritura
"a"	Abre un archivo de texto para añadir
"rb"	Abre un archivo binario para lectura
"wb"	Crea un archivo binario para escritura
"ab"	Abre un archivo binario para añadir
"r+"	Abre un archivo de texto para Lectura/Escritura
"w+"	Crea un archivo de tecto para Lectura/Escritura
"a+"	Abre o Crea un archivo de texto para L/E
"w+t"	Crea un archivo de texto para L/E
"a+t"	Abre o Crea un archivo de texto para L/E
"a+b"	Abre o Crea un archivo binario para L/E
"rt"	Abre un archivo de texto para lectura
"wt"	Crea un archivo de texto para escritura
"at"	Abre un archivo de texto para añadir
"r+t"	Abre un archivo de texto para L/E
"rb+"	(*) Abre un archivo binario para L/E
"wb+"	(*) Crea un archivo binario para L/E

(*).- Al aparecer problemas con los otros formatos, el autor recomienda sólo la utilización de estos dos.

La función fopen devuelve un NULL si no se ha podido abrir el archivo. La forma general de utilización de la función fopen se indica con el siguiente ejemplo:

```
FILE *p_file;
if (((p_file=fopen("nombre", "wb+"))= =NULL)
        {
        puts("Error al abrir el archivo\n");
        exit(1);
        }
```

Hay que tomar en cuenta estos puntos:

• Si abrimos un archivo para escritura, se crea un archivo con ese nombre. Si

existía un archivo con ese nombre se borra y se crea uno nuevo

• Si abrimos un archivo para lectura, sólo se puede abrir cuando existe ese archivo.

• Si abrimos un archivo para Lectura y Escritura, si no existe se crea uno nuevo. Si existe se abre el archivo sin borrarlo.

• Los archivos en modo texto, convierten las secuencias de retorno y salto de línea. En los archivos binarios no se produce ningún tipo de conversión.

Putc()

Escribe caracteres en un flujo previamente abierto, su prototipo es:

int putc(int c, FILE *p);

Ejemplo: putc('a', p_file);

Una vez escrito, incrementa la posición en el fichero.

Getc()

Lee caracteres de un flujo previamente abierto, su prototipo es:

int getc(FILE *p);

Ejemplo: char c;

```
c=getc(p_file);
```

Una vez leído incrementa la posición en el fichero.

Getc devuelve EOF cuando se ha alcanzado el final de un archivo de texto:

Ejemplo:
```
char c;
FILE *p_file;
if ((if (((p_file=fopen("nombre", "w"))= =NULL)
        {
        puts("Error al abrir el archivo\n");
        exit(1);
        }
do  c=getc(p_file));
while (c!=EOF);
```

Función feof()

Para determinar el final de un archivo binario se utiliza la función feof(). Su prototipo es:

int feof(FILE *p);

Ejemplo: while (!(feof(p_file)) getc(p_file);

Función fclose()

La función fclose() cierra un flujo que previamente fue abierto. Escribe al archivo toda la información que se encuentre en el buffer del disco. Su prototipo es:

int fclose(FILE *p);

Ejemplo: fclose(p_file);

Si devuelve un EOF es que ha ocurrido un error al intentar cerrar un archivo.

Función ferror()

La función ferror() nos muestra si se ha producido un error en la última operación sobre un archivo. La función tiene el siguiente prototipo:

int ferror(FILE *p);

La función devuelve cierto si se ha producido un error; sino devuelve falso.

Función rewind()

Inicializa el indicador de posición al principio del archivo:

void rewind(FILE *p);

Getw() y putw()

Funcionan igual que getc y putc, excepto que escriben y leen dos bytes en vez de uno:

Ejemplo:

```
int h;FILE *p;
...
putw(65000, p);
h=getw(p);
```

Fread() y fwrite()

Sirven para leer y escribir bloques de datos, su formato es:

fread(void *buffer, num_bytes, cuenta, FILE *p);

fwrite(void *buffer, num_bytes, cuenta, FILE *p);

siendo:

• buffer= puntero de memoria donde escribir (al leer) o leer (al escribir) los datos.

• num_bytes= longitud del elemento a leer;

• cuenta= número de elementos a leer de longitud num_bytes.

• p= puntero a archivo.

Ejemplo:

```
float buff[20];
FILE *p;
...
...
...
fread(buff, sizeof buff, 1, p);
..
fwrite(buff, sizeof buff, 1, p)
...
```

Fputs() y fgets()

Graban y leen cadenas en/del archivo. Tienen el siguiente formato:

int fputs(const char *cad, FILE *p);

char *fgets(char *cad,int longitud, FILE *p);

Ejemplo:

```
char cadena[200];
FILE *p;
...
fputs("GUARDA ESTA CADENA",p);
...
fgets(cadena, 100, p);          Lee 100 caracteres del archivo
```

Fprintf() y fscanf()

Estas funciones se comportan igual que el printf y el scanf, pero operando

con archivo de disco. Sus prototipos son:

int fprintf(FILE *p, const char *cadena, var...);

int fscanf(FILE *p, const char *cadena, &var...);

Ejemplos:

```
FILE *p;
...
fprintf(p, "%s%d%d", cadena, h, j);
```

Guarda cadena, h y j en el fichero con puntero p.

```
...
fscanf(p,"%s%d%d", cadena, &h, &j);
```

Lee cadena, h y j del fichero con puntero p.

Se guardan y leen los números en formato ASCII (tal como se imprimirían en pantalla o se introducirían por teclado), no en binario.

E/S de acceso directo con fseek()

Se sitúa el indicador de posición de archivo en la posición deseada. Su formato es el siguiente:

int fseek(FILE *p, long num_bytes, int origen);

Siendo:

p el puntero a un archivo

num_bytes=longitud en bytes a partir de origen

origen=Puede tener estos tres valores:

Valores de origen

Nombre	Origen	Valor
SEEK_SET	Principio del archivo	0
SEEK_CUR	Posición actual	1
SEEK_END	Final del archivo	2

- Se utiliza fseek sólo con archivo de tipo binario.
- La función fseek devuelve un cero cuando ha tenido éxito y un valor dis tinto de cero cuando hay un error.
- Si situamos fseek fuera del archivo (Siempre valores positivos, ejemplo un archivo que tiene 100 bytes y hacemos un fseek(p, 150, SEEK_SET)), y escribimos en él, aumentará el archivo hasta esa nueva posición.
- Num_bytes es una variable tipo long para poder acceder a ficheros más grandes de 64k.

Remove()

Sirve para eliminar archivos:

int remove(const char *nombre_archivos);

Ejemplo:

```
remove ("c:\\tc\\programs\\Programa.exe");
```

Si tiene éxito devuelve un cero, en caso contrario devuelve un número distinto de cero.

Problema 20

Partiendo del problema 18 (programa para almacenar los datos de los ficheros de los clientes), ampliar el programa para que permita almacenar/leer los datos en disco.

Problema 21

Desarrollar un programa que grabe en disco los 10000 primeros números primos empezando por el 1. Una vez grabado en un archivo 'primos.bin', nos pedirá la posición del número primo que queramos que nos muestre. En otro archivo 'primos.txt' introducirá en ASCII el siguiente texto:

"NUMERO PRIMO EN LA POSICION %d=%ld\n"

para 100 números primos a partir de una posición que nosotros determinaremos.

10. FUNCIONES GRAFICAS EN TURBO C

Modos de vídeo del PC

Las tarjetas gráficas de los ordenadores compatibles PC, soportan diferentes modos de vídeo. Un modo de vídeo constituye una configuración básica de la pantalla. Inicialmente los modos se dividen en dos tipos principales:

• **Modo texto**: Sólo se imprimen caracteres ASCII. La pantalla está compuesta por posiciones de carácter.

• **Modo gráfico:** Se accede en la pantalla punto a punto, se puede dibujar cualquier tipo de gráfico.

Los distintos tipos de adaptadores de vídeo tienen los siguientes nombres: monocroma, CGA (Color Graphics Adapter), EGA (Enhanced Graphics Adapter), VGA (Vídeo Graphics Array) y Super-VGA. cada uno de ellos admite diferentes tipos de modos texto y gráfico.

Modo texto

En el modo texto se imprimen caracteres únicamente. La particularidad del lenguaje Turbo C, consiste en el uso de ventanas especiales, de forma que todas las operaciones de lectura y escritura se realicen dentro de una ventana determinada. Por ejemplo, el compilador de Turbo C está compuesto por diversas ventanas.

Las instrucciones básicas para el manejo de la pantalla en modo texto son las siguientes:

Funciones del modo texto

Función	Propósito
window()	Define y activa una ventana
textmode()	Establece el modo de texto de la pantalla
gotoxy()	Coloca el cursor en una posición dentro de la ventana
cprintf()	Igual que prinf pero dentro de la ventana activa
cputs()	Escribe una cadena en la ventana activa
putch()	Imprime un carácter en la ventana activa
getche()	Lee un carácter de la ventana activa
cgets()	Lee una cadena de la ventana activa
clrscr()	Limpia la ventana activa
clreol()	Borra desde el cursor hasta el final de la línea actual
delline()	Borra la línea sobre la que está el cursor
insline()	Inserta una línea vacía debajo de la posición del cursor
gettext()	Copia parte de la pantalla en un buffer de caracteres
puttext()	Copia texto de un buffer a la pantalla
movetext()	Copia texto de una parte de la pantalla a otra
highvideo()	Muestra el texto brillante
lowvideo()	Muestra el texto en baja intensidad
normvideo()	Muestra el texto en la intensidad original
textattr()	Poner el color del texto y del fondo
textbackground()	Poner el color del fondo
textcolor ()	Poner el color del texto
gettextinfo ()	Devuelve información de la ventana de texto actual
wherex ()	Devuelve la coordenada x del cursor
wherey ()	Devuelve la coordenada y del cursor

Función window()

Esta función activa una ventana de texto con unas dimensiones específicas. Su prototipo es:

void window(int izquierda, int arriba, int derecha, int abajo);

Todas las referencias de coordenadas son relativas a la nueva ventana creada:

Ejemplo:

```
window(1, 1, 40,25);
gotoxy(5, 5);
cprintf("HOLA");
```

Función textmode()

Se utiliza para cambiar el modo de texto. Su prototipo:

void textmode(int modo);

siendo modo:

Valores de modo

Nombre	Valor	Descripción
BW40	0	Blanco y negro, 40 columnas
C40	1	Color, 40 columnas
BW80	2	Blanco y negro, 80 columnas
C80	3	Color, 80 columnas
MONO	7	Monocromo de 80 columnas
LASTMODE	-1	Modo anterior
C4350	64	En EGA, 80x43; en VGA, 80x25

Función gotoxy()

Sitúa el cursor en la posición determinada. Siendo la coordenada (1,1) la de la esquina superior izquierda. Su prototipo es:

void gotoxy(int x, int y);

Función cprintf()

Funciona igual que la función printf pero dentro de una ventana activa. Se diferencia de printf en que para pasar de línea se utiliza el cambio de línea de esta forma "\n\r". Su prototipo es:

int cprintf(const char *fmt, ...);

Función cputs()

Imprime una cadena en la ventana activa:

int cputs(const char *cad);

Ejemplo:

```
cputs("ESTA CADENA SE IMPRIME EN LA VENTANA ACTIVA");
```

Función cgets()

Lee una cadena de la ventana activa. Su prototipo es:

char *cgets(char *cad);

Es necesario poner en cad[0] la longitud máxima de la cadena, en cad[1] devuelve el número de caracteres leídos. La cadena empieza en cad[2]. La función devuelve un puntero al primer carácter de la cadena leida:

Ejemplo:

```
char cadena[50], *p;
printf("Introduce la cadena:" );
cadena[0]=48;
p=cgets(cadena);
cprintf("\n\rLA CADENA ES=%s y SU LONGITUD=%d", p, cadena[1]);
```

Funciones clrscr(), clreol(), delline() y insline()

Son funciones que se encargan de manipular la pantalla de texto.

La función **clrscr**() limpia la ventana activa. Su prototipo es:

void clrscr(void);

La función **creol()** limpia desde la posición actual del cursor hasta el límite derecho de la ventana. Tiene como prototipo:

void crleol(void);

La función **delline()**, borra la línea donde está el cursor y desplaza las de abajo hacia arriba. Su prototipo es:

void delline(void);

La función **insline()**, inserta una línea donde está el cursor bajando todas la líneas siguientes. Prototipo:

void insline(void);

Funciones gettext(), puttext() y movetext()

Son funciones para mover, guardar y poner texto de la pantalla activa. Las funciones gettext() y puttext() se usan para coger y poner texto en la pantalla activa. Tienen los siguientes prototipos:

int gettext(int c_izquierda, int c_arriba, int c_derecha, int c_abajo, void *buffer);

int puttext(int c_izquierda, int c_arriba, int c_derecha, int c_abajo, void *buffer);

El tamaño de buffer corresponde a Tamaño= filas x columnas x 2=bytes, cada carácter está formado por un byte con el atributo y otro con el valor ASCII de éste. Con las coordenadas seleccionamos un trozo del texto y lo asignamos a un buffer.

La función movetext() copia texto de una parte de la pantalla a otra, se designa un trozo de la ventana y después su nueva coordenada.

int movetext(int c_izquierda, int c_arriba, int c_derecha, int c_abajo, int nueva_izquierda, int nueva_arriba);

Funciones highvideo(), lowvideo(), normvideo()

Estas funciones ponen la pantalla en alta intensidad, baja intensidad y intensidad normal. No son aplicables en el modo monocromo. Sus prototipos son:

void highvideo(void);	Pone los caracteres en alta intensidad.
void lowvideo(void);	Pone los caracteres en baja intensidad.
void normvideo(void);	Pone los caracteres en la intensidad que estaba al ejecutarse el programa.

Funciones textattr(), textbackground() y textcolor()

Cambian los colores del texto y del fondo. Las funciones hacen lo siguiente:

void textcolor(int color);	Pone el color del texto.
void textbackground(int color_fondo);	Pone el color del fondo. Sólo entre 0 y 7.

Los valores de color por defecto son:

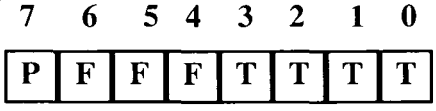
Colores por defecto

Nombre	Color	Valor
BLACK	NEGRO	0
BLUE	AZUL	1
GREEN	VERDE	2
CYAN	AZUL CELESTE	3
RED	ROJO	4
MAGENTA	MAGENTA	5
BROWN	MARRON	6
LIGHTGRAY	GRIS CLARO	7
DARKGRAY	GRIS OSCURO	8
LIGHTBLUE	AZUL CLARO	9
LIGHTGREEN	VERDE CLARO	10
LIGHTCYAN	AZUL CELESTE CLARO	11
LIGHTRED	ROJO CLARO	12
LIGHTMAGENTA	MAGENTA CLARO	13
YELLOW	AMARILLO	14
WHITE	BLANCO	15
BLINK	PARPADEANTE	128

Para cambiar el color del texto y del fondo a la vez se utiliza la función:

void textattr(int atributo);

Siendo *atributo* un byte codificado de la siguiente manera:



- **P= bit de parpadeo.**
- **F= color del fondo.**
- **T= color del texto.**

Hay que darse cuenta **que el cambio de color sólo afecta a las escrituras siguientes, no a las anteriores.**

Funciones gettextinfo(), wherex() y wherey()

La función **gettextinfo()** devuelve el estado de la ventana actual en una estructura definida en la biblioteca conio.h. El prototipo es el siguiente:

void gettextinfo(struct text_info *info);

Para llamarla se define una estructura tipo **text_info** y se pasa la dirección de ésta:

Ejemplo:

```
struct  text_info p;
gettextinfo(&p);
```

La estructura definida en conio.h tiene la siguiente forma:

```
struct text_info
  {
  unsigned char winleft;          Coordenada x izquierda
  unsigned char wintop;           Coordenada y arriba
  unsigned char winright;         Coordenada x derecha
  unsigned char winbottom;        Coordenada y abajo
  unsigned char attribute;        Atributos del texto
  unsigned char normattr;         Atributos normales
  unsigned char currmode;         Modo de vídeo
  unsigned char screenheight;     Líneas de la pantalla (altura)
  unsigned char screenwidth;      Columnas de la pantalla (anchura)
  unsigned char curx;             Coordenada x del cursor.
  unsigned char cury;             Coordenada y del cursor.
  };
```

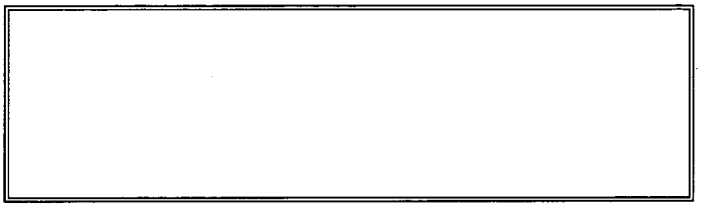
Las funciones wherex() y wherey() tienen el siguiente prototipo:

int wherex(void); Devuelve la coordenada X del cursor.

int wherey(void); Devuelve la coordenada Y del cursor.

Problema 22

Realizar una función que cree una ventana, con un borde alrededor de ésta. Los argumentos que le pasaremos a la función serán: coordenada x izquierda, coordenada y arriba, coordenada X derecha, coordenada y abajo, color del fondo y color del texto. El dibujo del borde de la ventana será el siguiente:



Funciones en modo gráfico

Modo gráfico

El modo gráfico nos permite realizar dibujos a nivel de puntos, circunferencias, gráficas, texto con diferentes tamaños, etc. Igual que con el modo texto, en el modo gráfico todas las operaciones se realizan en la ventana gráfica activa, por defecto la ventana gráfica es toda la pantalla.

Toda función gráfica requiere de la librería graphics.h.

Inicialmente para tratar un modo gráfico, hay que inicializar la tarjeta a uno de sus modos gráficos. Para ello se utiliza la función initgraph()

Función initgraph()

La función initgraph carga en memoria el controlador de gráficos correspondiente. Su prototipo es:

void far initgraph(int far **controlador*, int far **modo*, char far **camino*);

Siendo controlador el número del controlador, modo apunta a un entero que es el modo de vídeo que se va a usar y camino el directorio donde está el controlador (si no ponemos camino buscará en el directorio actual).

Valores para controlador

Macro	Equivalente
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8154	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

Si igualamos controlador a DETECT, initgraph() detecta automáticamente el tipo de tarjeta gráfica que tiene y selecciona el modo de vídeo de resolución más alto.

Valores de modo

Controlador	Modo	Equivalente	Resolución
CGA	CGAC0	0	320 x 200
	CGAC1	1	320 x 200
	CGAC2	2	.320 x 200
	CGAC3	3	320 x 200
	CGAHI	4	640 x 200
MCGA	MCGAC0	0	320 x 200
	MCGAC1	1	320 x 200
	MCGAC2	2	320 x 200
	MCGAC3	3	320 x 200
	MCGAMED	4	640 x 200
	MCGAHI	5	.640 x 480
EGA	EGALO	0	640 x 200
	EGAHI	1	640 x 350
EGA64	EGA64LO	0	640 x 200
	EGA64HI	1	640 x 350
EGAMONO	EGAMONOH	I3	640 x 350
HERC	HERCMONOHI	0	720 x 348
ATT400	ATT400C0	0	320 x 200
	ATT400C1	1	320 x 200
	ATT400C2	2	320 x 200
	ATT400C3	3	320 x 200
	ATT400CMED	4	640 x 200
	ATT400CHI	5	640 x 400
VGA	VGALO	0	640 x 200
	VGAMED	1	640 x 350
	VGAHI (*)	2	640 x 480
PC3270	PC3270HI	0	720 x 350
IBM8514I	BM8514HI	0	1024 x 768
	IBM8514LO	0	640 x 480

(*) más usada

Un ejemplo típico de selección es el siguiente:

Ejemplo:

```
#include<graphics.h>
void pon_modo_grafico(void)
{
```

```
        int controlador, modo;
        controlador=VGA;
        modo=VGAHI;
        initgraph(&controlador, &modo, "C:\\tc\\bgi");
        return;
}
```

Función closegraph()

La función **closegraph**() cierra el modo gráfico y restablece el modo texto anterior. Las dos funciones que se pueden utilizar son:

void far closegraph(void);	Restablece el modo gráfico anterior
void far restorecrtmode(void);	Restablece el modo antes de la primera llamada a initgraph().

Paletas de colores en los diferentes modos

Cada modo de vídeo puede representar una cantidad de colores determinada, para cada modo se presentan sus paletas y colores por defecto.

MODO CGA

En el modo CGA podemos representar cuatro colores a la vez. Se puede elegir entre cuatro paletas siendo el color 0 de cada paleta el color del *fondo*.

Paletas y colores del modo CGA

COLOR	PALETA 0	PALETA 1	PALETA 2	PALETA 3
0	fondo	fondo	fondo	fondo
1	verde	cyan	verde claro	cyan claro
2	rojo	magenta	rojo claro	magenta claro
3	amarillo	blanco	amarillo	blanco

El número de paleta se elige cuando seleccionamos el modo, por ejemplo en modo CGAC3 selecciona el modo CGA paleta 3ª.

El único color que se puede cambiar es el del fondo.

MODO EGA y VGA

En estos dos modos se pueden representar 16 colores entre 64 posibles.

Función setpalette()

Nos permite cambiar el color numero n de la paleta de colores. Su prototipo es:

void far setpalette(int indice, int color);

siendo índice el número del color a cambiar dentro de la paleta.

En el modo CGA sólo se puede cambiar el color del fondo.

Los colores son los siguientes:

Colores para el modo CGA (fondo)

MACRO	VALOR
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14.
WHITE	15

Colores para el modo EGA y VGA

MACRO	VALOR
EGA_BLACK	0
EGA_BLUE	1
EGA_GREEN	2
EGA_CYAN	3
EGA_RED	4
EGA_MAGENTA	5
EGA_BROWN	6
EGA_LIGHTGRAY	7
EGA_DARKGRAY	8
EGA_LIGHTBLUE	9
EGA_LIGHTGREEN	10
EGA_LIGHTCYAN	11
EGA_LIGHTR	12
EGA_LIGHTMAG	13
EGA_YELLOW	14
EGA_WHITE	15

para el modo CGA se cambiaría el fondo del siguiente modo:

Ejemplo:

```
sepalette(0, RED);
```

Para el modo EGA y VGA se pueden mostrar 16 colores a la vez, para cambiar el color número 3 por un rojo claro:

Ejemplo:

```
setpalette(3, EGA_LIGHTRED);
setpalette(3, 12);              Es lo mismo.
```

Función setbkcolor()

Sirve para cambiar sólo el color del fondo, se utiliza la tabla de colores del CGA. Su prototipo es:

void setbkcolor(int color);

Ejemplo: setbkcolor(GREEN);

Función setallpalette()

Sirve para establecer todos los colores a la vez. Su prototipo es:

void far setallpalette(struct palettetype far *pal);

Siendo la estructura palettetype la siguiente:

```
struct palettetype
        {
        unsigned char size;
        signed char colors[MAXCOLORS+1];
        };
```

MAXCOLORS especifica el número máximo de colores de la paleta.

Funciones básicas para crear gráficos

Estas son las funciones más utilizadas:

Funciones gráficas básicas

Función	Acción	Prototipo
initgrafph ()	Inicializa el modo grafico	**void far initgraph** (int far *contro lador, int far *modo, char far *cami no);
closegraph ()	Cierra el modo gráfico.	**void far closegraph** (void);
restorecrtmode()	Cierra el modo gráfico	**void far restorecrtmode** (void);
setpalette ()	Cambia un color de la paleta	**void far setpalette** (int indice, int color);
setallpalette ()	Cambia toda la paleta	**void far setalpalette** (struct palet tetype far *pal);
setbcolor ()	Cambia el color del fondo	**void far setbkcolor** (int color);
putpixel ()	Dibuja un punto	**void far putpixel** (int x, int y, in color);
line ()	Dibuja una linea	**void far line** (int xini, int yini, intx fin, int yfin);
circle ()	Dibuja un círculo	**void far cicle** (int x, int y, intradio);
bar ()	Dibuja un rectangulo relleno	**void far bar** (int xarr, int yarr, int xaba, int yaba);
setcolor	Color actual	**void far setcolor** (int color);
floodfill ()	Rellena una figura cerrada	**void far floodfill** (int x, int y, int colorborde);
setfillstyle ()	Estilo del relleno	**void far setfillstyle** (int patron, int color);
setfillpattern ()	Definición propia del patrón de relleno	**void setfillpattern** (char far *patron int color);
outtext ()	Imprime texto	**void far outtext** (char far *cad)
settextsyle ()	Cambia el tipo del texto, tamaño etc.	**void settextsyle** (in fuente, intdirec- int tamañocar);
clearviewport ()	Limpia la ventana gráfica	**void far clearviewport** (void);
getimage ()	Copia una parte de una ventana gráfica	**void far getimage** (int xizq, inyizq, int xder, int aba, void far * bud);
putimage ()	Copia el contenido de un buffer en la ventana gráfica	**void far putimage** (int xizq, intyizq void far *buf, int op);
setactivepage ()	Determina que página se verá afectada por las rutinas gráficas	**void far setactivepage** (int numpa gina);
setviewport ();	Crea una ventana gráfica	**void far setviewport** (int izq,intarr, int der, int aba, int indclip);
setvisualpage	Determina la página que se muestra	**void far setvisualpage** (int num pa- gina);

Función putpixel()

Pone un punto en la coordenada (x,y) de la pantalla con el color de la paleta especificado. Su prototipo es:

void far putpixel(int x, int y, int color);

Ejemplo:

putpixel(100, 100, 10); Pone el punto (100,100) con el color 10.

Función line()

Dibuja una linea desde la coordenada (xini, yini) hasta la (xfin, yfin). Su prototipo es:

void far line(int xini, int yini, int xfin, int yfin);

Ejemplo:

```
setcolor (10);
line(100, 100, 200, 200); Dibuja una línea desde el punto (100,100) hasta
                          el (200,200).
```

Función circle()

Dibuja un circulo con centro (x, y) y radio igual a radio. Su prototipo es:

void far circle(int x, int y, int radio);

Ejemplo:

```
setcolor (10);
circle(100, 100, 50);   Dibuja un circulo con centro en (100,100) y radio
                        50 pixeles.
```

Función bar()

Dibuja un rectangulo relleno con el color por defecto. Su prototipo es:

void far bar(int xarr, int yarr, int xaba, int yaba);

Ejemplo:

```
setfillstyle (1,color);
bar(100, 100, 250, 250);Dibuja un rectángulo relleno desde (100,100)
                        hasta (250,250).
```

Función setcolor()

Selecciona un color de la paleta para dibujar con él. Su prototipo es:

void far setcolor(int color);

Ejemplo:

setcolor (11); Pone el color 11 de la paleta por defecto.

Función floodfill()

Rellena una figura cerrada delimitada por el color colorborde con el color del relleno, desde la posición (x,y). Su prototipo es:

void far floodfill(int x, int y, int colorborde);

Ejemplo:

```
setcolor(15);
circle(100,100,30);
floodfill(100,100,15);
```

Función setfillstyle()

Seleciona el tipo de relleno y su color.Su prototipo es:

void far setfillstyle(int patron, int color);

Los valores para patron son:

Valores para patron

Nombre	Valor	Significado
EMPTY_FILL	0	Rellena con el color del fondo
SOLID_FILL	1	Rellena con un color liso
LINE_FILL	2	Rellena con líneas
LTSLASH_FILL	3	Rellena con barras finas
SLASH_FILL	4	Rellena con barras
BKSLASH_FILL	5	Rellena con barras invertidas
LTBKSLASH_FILL	6	Rellena con barras invertidas finas
HATCH_FILL	7	Rellena con trama fina
XHATCH_FILL	8	Rellena con trama
INTERLEAVE_FILL	9	Rellena dejando espacios
WIDE_DOT_FILL	10	Rellena con puntos espaciados
CLOSE_DOT_FILL	11	Rellena con puntos poco espaciados
USER_FILL	12	Rellena con el patrón del usuario

Ejemplo:

```
setcolor(15);
circle(100,100,30);
setfillstyle(SOLID_FILL, 2);
floodfill(100,100,15);
```

Función setfillpattern()

Sirve para definir un tipo de relleno propio. El array debe tener un longitud mínima de 8 bytes formado por una rejilla de bits de modo que por cada bit puesto a uno se pone el pixel con el color actual, si es cero se dibuja el pixel correspondiente con el color del fondo. Su prototipo es:

void setfillpattern(char far *patron, int color);

Ejemplo:

```
char p[]={0,255,0,255,0,255,0,255};
...
setcolor(15);
circle(100,100,30);
setfillpattern(p,10);
floodfill(100,100,15);
```

Funciones outtext(),outtextxy()

Con la función outtext() podemos visualizar en pantalla una cadena de texto con outtextxy() visualizamos una cadena de texto en las coordenadas (x,y). Sus prototipos son:

void far outtext(char far *cad);

void far outtextxy(int x, int y, char far *cad);

Ejemplo:

```
outtext("CADENA EN MODO TEXTO");
outtextxy(100,100, "CADENA EN LA COORDENADA 100,100");
```

Función settextstyle()

Esta función nos permite seleccionar diferentes tipos de texto. El prototipo de esta función es:

void settextstyle(int fuente, int direccion, int tamañocar);

Siendo tamañocar un valor que puede estar entre 0 y 10, definimos el tamaño de los caracteres. Con direccion podemos indicar si el texto va a aparecer de forma horizontal HORIZ_DIR (Valor 0) o vertical VERT_DIR (Valor 1). Por último con fuente seleccionamos el formato de los caracteres, siendo los valores siguientes:

Valores de fuente

Nombre	Valor	Tipo de fuente
DEFAULT_FONT	0	Mapa de bits de 8 x 8
TRIPLEX_FONT	1	Fuente Triplex
SMALL_FONT	2	Fuente pequeña
SANS_SERIF_FONT	3	Fuente Sans Serif
GOTHIC_FONT	4	Fuente gótica

Ejemplo:

```
...
settextstyle(GOTHIC_FONT, HORIZ_DIR, 2);
outtextxy(100, 100, "ESTO ES UNA CADENA DE CARACTERES GOTICOS");
```

Funciones getimage() y putimage()

Con la función getimage() copiamos un trozo de pantalla a un buffer y con putimage() pegamos en la pantalla un trozo de gráfico procedente de un buffer, imagesize() devuelve el tamaño en bytes del trozo de pantalla. Sus prototipos son:

void far getimage(int xizq, int yizq, int xder, int aba, void far *buf);

void far putimage(int xizq, int yizq, void far *buf, int op);

unsigned imagesize(int xizq, int yizq, int xder, int yder);

Siendo xizq, yizq, xder y yder las coordenadas del trozo de pantalla y *buf un puntero que apunta al buffer.

El valor op corresponde a la forma de imprimir en pantalla el gráfico.

Valores de op

Nombre	Valor	Significado
COPY_PUT	0	Copia igual
XOR_PUT	1	Copia con XOR
OR_PUT	2	Copia haciendo un OR
AND_PUT	3	Copia haciendo un AND
NOT_PUT	4	Copia invirtiendo la imagen

Ejemplo:

```
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
main()
{
        int cont=DETECT, mode=0,h;
        void *buf;
        unsigned tam;
        initgraph(&cont,&mode, "c:\\tc\\bgi");
        randomize();
        for (h=0;h<40;h++)
        {
                setcolor(h);
                line(h,0,40-h,50);
        }
        tam=imagesize(0,0,40,50);
        buf=malloc(tam);
        getimage(0,0,40,50,buf);
        for (h=0;h<40;h++)
        {
                putimage(random(400),random(300),buf,OR_PUT);
        }
        getch();
        closegraph();
        return;
}
```

Funciones setactivepage() y setvisualpage()

En ciertos modos gráficos se pueden seleccionar distintas páginas (si la memoria de vídeo y el modo lo permiten), de forma que podemos dibujar en una página oculta y después visualizarla instantáneamente. Para ello se utiliza las funciones setactivepage() que nos determina en que página se van a volcar todos los dibujos, y setvisualpage() que indica qué página visualizar. Sus prototipos son:

void far setvisualpage(int numpagina);

void far setactivepage(int numpagina);

Ejemplo:

```
setvisualpage(1);
setactivepage(0);
for (h=0;h<300;h++)
        {
        setcolor(h);
        line(h,0,400-h,300);
        }
setvisualpage(1);
```

Función setviewport()

Crea una ventana gráfica. Su prototipo es:

void far setviewport(int izq, int arr, int der, int aba, int indclip);

Siendo (izq,arr,der,aba) las coordenadas de la ventana gráfica y indclip un valor entre 0 y 1. Si es 1 la salida en la pantalla se recorta según los bordes de la ventana, si es cero no se recorta.

Problema 23

Realizar un programa que represente en coordenadas cartesianas una función matemática cualquiera a elección. Para ello definiremos una ventana rectangular gráfica de dimensiones 500x300 pixeles en un modo gráfico de 640x480 pixeles y 16 colores. Una vez inicializado el modo gráfico, el programa dibujará dentro de la ventana los ejes cartesianos divididos por rayitas. el fondo de la ventana será de color azul oscuro, los ejes de color gris claro. Se situará en cada eje el tipo de variables que representa (X e Y). Por último una vez dibujado todo se procederá a representar la forma de la función en la pantalla.

11. FUNCIONES PARA MANEJAR EL RATON.

Manejo del ratón

Las funciones para manejar el ratón se definen a continuación:

```
#include<stdio.h>
#include<dos.h>
#include<graphics.h>

int estado_raton;
int xmouse,ymouse,botones;

typedef struct
  {
  int      x;
  int      y;
  int      mascara_cursor[2][16];
  } tipo_cursor;

/* DIBUJO DE LOS CURSORES

  MASCARA              CURSOR

  0011111111111111     1000000000000000
  0001111111111111     1100000000000000
  0000111111111111     1110000000000000
  0000011111111111     1111000000000000
  0000001111111111     1111100000000000
  0000011111111111     1111000000000000
  0000011111111111     0011000000000000
  1100001111111111     0001100000000000
  1100001111111111     0001100000000000
  1110000111111111     0000110000000000
  1110000111111111     0000110000000000
  1111000011111111     0000011000000000
  1111000011111111     0000011000000000
  1111100111111111     0000000000000000
  1111111111111111     0000000000000000
  1111111111111111     0000000000000000
*/
tipo_cursor cursor_flecha_izda=
  {1,1,
  {{0x3FFF,0x1FFF,0x0FFF,0x07FFF,0x03FFF,0x7FFF,0x7FFF,0xC3FF
//máscara
```

```
      ,0xC3FF,0xE1FF,0xE1FF,0xF0FF,0xF0FF,0xF9FF,0xFFFF,0xFFFF}
      ,{0x8000,0xC000,0xE000,0xF000,0xF800,0xF000,0x3000,0x1800   //grá-
fico
      ,0x1800,0x0C00,0x0C00,0x0600,0x0600,0x0000,0x0000,0x0000}
      }};

    /* (1) */
    /*=====================================================
                        PONE EL RATON
      ====================================================*/

    void       far pon_raton    (void)
    {
      union REGS registros;
      registros.x.ax=1;
      int86(51,&registros,&registros);
    }

    /* (2) */
    /*=====================================================
                        QUITA EL RATON
    ====================================================*/

    void       far quitar_raton (void)
    {
      union REGS registros;
      registros.x.ax=2;
      int86(51,&registros,&registros);
    }

    /* (3) */
    /*=====================================================
          DEFINE LAS COORDENADAS LIMITE DEL RATON
          MOUSEBORDER
    ====================================================*/

    void define_coordenadas  (int far xiz, int far yiz, int far xder, int far yder)
    {
      union REGS registros;
      registros.x.ax=7;
      registros.x.cx=xiz;
      registros.x.dx=xder;
```

```
  int86(51,&registros,&registros);
  registros.x.ax=8;
  registros.x.cx=yiz;
  registros.x.dx=yder;
  int86(51,&registros,&registros);
}

/* (4) */
/*=======================================================
        DEFINE EL TIPO DE CURSOR DEL RATON
=======================================================*/
void define_cursor          (void)
{
  tipo_cursor *tc=&cursor_flecha_izda;
  union REGS registros;
  struct SREGS reg_seg;
  registros.x.ax=9;
  registros.x.bx=tc->x;
  registros.x.cx=tc->y;
  registros.x.dx=FP_OFF(tc->mascara_cursor);
  reg_seg.es=FP_SEG(tc->mascara_cursor);
  int86x(51,&registros,&registros,&reg_seg);
}

/* (5) */
/*=======================================================
              PONE EL RATON EN X e Y
=======================================================*/

void pon_coord    (int x,int y)
{
  union REGS registros;
  registros.x.ax=4;
  registros.x.cx=x;
  registros.x.dx=y;
  int86(51,&registros,&registros);
}

/* (6) */

/*=======================================================
                INICIALIZA EL RATON
```

```
            SALIDA -1 = BIEN :  SALIDA 0 = ERROR
=================================================*/
int inicializar_raton        (void)
{
  union REGS registros;

  registros.x.ax=0;
  int86(51,&registros,&registros);
  if (registros.x.ax==0xFFFF)
            {
            estado_raton=0xffff;
            define_coordenadas(0,0,639,479);
            define_cursor();
            pon_coord(320,175);
            return -1;
            }
  else
            {
            estado_raton=0x0000;
            return 0;
            }
}

/* (7) */
/*=================================================
        COORDENADAS DEL RATON Y BOTON PULSADO
=================================================*/
void       far posicion_botones     (int far *xb,int far *yb,int far *botonesb)
{
  union REGS registros;
  disable();
  registros.x.ax=3;
  int86(51,&registros,&registros);
  *botonesb=registros.x.bx;
  *xb=registros.x.cx;
  *yb=registros.x.dx;
  enable();
}
/*      FIN DE RUTINAS DE CONTROL DE RATON */
```

Para manejar las diferentes funciones del ratón, casi todos los drivers utilizan

la interrupción 51 (33 Hex). Para ello se llama a la interrupción con los registros del microprocesador que necesite, el registro AX corresponde al número de función siendo las más importantes y compatibles con casi todos los tipos de ratón las siguientes:

Funciones de ratón.

Número	Función
0	Inicializa el ratón
1	Habilita el ratón
2	Oculta el ratón
3	Lee las coordenadas del ratón y los botones
4	Pone el ratón en unas coordenadas específicas
7	Define el rango X del ratón
8	Define el rango Y del ratón
9	Define el gráfico del cursor

Función 0. Inicializar el ratón

Con AX puesto a cero se inicializa el ratón, si está el driver devuelve en el registro AX=0xFFFF y en BX= el número de botones del ratón. Si no devuelve un valor en AX distinto de 0xFFFF.

Ejemplo:

```
union REGS registros;
registros.x.ax=0;
int86(51, &registros, &registros);
if (registros.x.ax==0xFFFF) {printf("RATON ENCONTRADO, NUMERO DE BOTONES= %d", registros.x.bx);}
else     printf("RATON NO ENCONTRADO");
```

Función 1. Pone el cursor en pantalla

Ejemplo:

```
union REGS registros;
registros.x.ax=1;
int86(51, &registros, &registros);
```

Función 2. Esconde el cursor de la pantalla

Ejemplo:

```
union REGS registros;
registros.x.ax=2;
int86(51, &registros, &registros);
```

Función 3. Devuelve las coordenadas del ratón y el botón pulsado

Con el registro AX puesto a 3 nos devuelve en BX un entero correspondiente al botón pulsado:

- Si es 0 ningún botón pulsado
- Si es 1 (Bit 0 a uno), botón de la izquierda pulsado.
- Si es 2 (Bit 1 a uno), botón de la derecha pulsado.
- Si es 4 (Bit 2 a uno), botón del centro pulsado.

En CX devuelve la x del ratón y en DX la y del ratón

Ejemplo:

```
union REGS registros;
registros.x.ax=3;
int86(51, &registros, &registros);
gotoxy(1,1);
printf("X=%3d Y=%3d BOTON=%3d",registros.x.cx,registros.x.dx,regis
tros.x.bx);
```

Función 4. Pone el cursor en unas coordenadas específicas

Si ponemos el registro AX con el valor 4, podemos situar el cursor en cualquier posición, para ello ponemos en el registro CX la coordenada x y en DX la coordenada y del ratón.

Ejemplo:

```
union REGS registros;
registros.x.ax=4;
registros.x.cx=200;
registros.x.dx=100;
int86(51, &registros, &registros);
```

Función 7 y 8. Poner rango horizontal y vertical respectivamente

Con AX a 7 ponemos el rango horizontal en CX x mínima y en DX x máxima. Con AX a 8 ponemos el rango vertical en CX y mínima y en DX y máxima.

Ejemplo:

```
union REGS registros;
registros.x.ax=7;
registros.x.cx=10;
registros.x.dx=400;
int86(51, &registros, &registros);
registros.x.ax=8;
registros.x.cx=50;
registros.x.dx=200;
int86(51, &registros, &registros);
```

Función 9. Definir el gráfico del cursor.

En modo texto el cursor del ratón aparece como un cuadrado que se mueve por la pantalla, en cambio para el modo gráfico podemos definir el gráfico del cursor que queramos. Para ello tenemos que definir su máscara y el gráfico del cursor. La máscara sirve para dejar coger el fondo de la pantalla y situar un espacio donde dibujar el cursor. Se hace un AND de la máscara con el fondo y después se sitúa el gráfico del cursor con un OR. El cursor está definido por una estructura formada por:

```
struct flecha {int xs;int ys;int mascara[2][16]};
```

Siendo xs e ys los valores a sumar de las coordenadas partiendo de la esquina superior izquierda del gráfico. Es decir, es lo que hay que sumarle a las coordenadas para saber cual es el punto real del cursor como activo. Si tenemos una flecha cuya punta está en centro del gráfico, los valores corresponderán al los pixeles que hay que sumarle en la x y en la y.

En BX ponemos xs, en CX los valores de ys y en DX el OFFSET del puntero al gráfico que está en la estructura (ver ejemplo). Por último se pone el registro índice ES con el SEGMENTO del puntero al gráfico.

Ejemplo:

```
tipo_cursor *tc=&cursor_flecha_izda;
union REGS registros;
struct SREGS reg_seg;
```

```
registros.x.ax=9;
registros.x.bx=tc->x;
registros.x.cx=tc->y;
registros.x.dx=FP_OFF(tc->mascara_cursor);
reg_seg.es=FP_SEG(tc->mascara_cursor);
int86x(51,&registros,&registros,&reg_seg);
```

Problema 24

Realizar un programa que permita manejar el ratón y ver en pantalla las coordenadas del cursor (x,y) y el valor de los botones. Primero aparecerá en modo texto y pulsando el botón derecho del ratón pasará al modo gráfico.

En el modo gráfico dividiremos la pantalla en una matriz de ocho por ocho cuadros. Cada cuadro tendrá un número diferente.

Cuando pulsemos el botón derecho el programa nos indicará el número del cuadro donde está el cursor.

Pantalla

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

12. MANEJO DEL PUERTO SERIE Y PARALELO.

Puerto serie

La transmisión de datos por el puerto serie se realiza de bit en bit, la velocidad de transmisión viene determinada por BAUDIOS. Los BAUDIOS son el número de bits que se transmiten por segundo, es decir, si un módem transmite a 24000 baudios, transmite 24000 bits por segundo.

Para poder controlar el puerto de comunicación asíncrona RS232 se utiliza la función bioscom() que se encuentra en la librería bios.h. El BIOS tiene una serie de rutinas para controlar el puerto serie, a estas rutinas se les llama con la instrucción bioscom().

Su prototipo es:

int bioscom(int orden, char byte, int puerto);

El valor de orden nos indica la operación a realizar. Sus valores son:

Valores de orden

Valor	Acción
0	Inicializar puerto
1	Enviar carácter
2	Recibir carácter
3	Devolver el estado del puerto

El valor de puerto indica el puerto a utilizar: 0 para el COM1, 1 para el COM2, 2 para el COM3...

Inicializar puerto

Antes de poder usar el puerto serie hay que inicializarlo, para ello se utiliza la opción 0 de orden.

La variable byte es un valor char cuyos bits significan lo siguiente:

7	6	5	4	3	2	1	0
B	B	B	P	P	Bp	Bd	Bd

BBB (BITS)	BAUDIOS
111	9600
110	4800
101	2400
100	1200
011	600
010	300
001	150
000	110

PP (BITS)	PARIDAD
00	Sin paridad
01	Impar
11	Par
Bd (BITS)	**Bits datos**
00	
01	7
11	8

Bp (BIT)	Bit parada
0	1 Bit
1	2 Bits

Imaginemos que queremos configurar el puerto para transmitir a una velocidad de 4800 BAUDIOS, con paridad impar, un bit de parada y transmisión de ocho bits.

byte=110(4800BAUDIOS)+01(IMPAR)+0(1BIT)+11(8 Bits)=11001011=128+64+8+2+1=203

Entonces para configurar el COM0 sería:

```
bioscom(0,203,0);
```

La función **bioscom()** devuelve un entero. El byte de mayor orden tiene el siguiente significado:

Byte de mayor orden

Bit	Significado puesto a uno
0	Datos preparados
1	Error de sobreescritura
2	Error de paridad
3	Error de estructura
4	Error de interrupción
5	Registro de mantenimiento de transferencia vacío
6	Registro de desplazamiento de transferencia vacío
7	Error de temporización

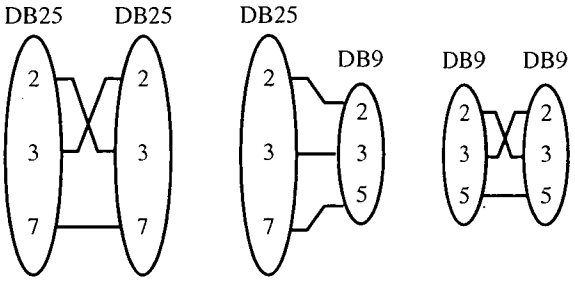
Byte de menor orden (orden=0, 1 o 3)

Bit	Significado puesto a uno
0	Cambio listo para enviar
1	Cambio en datos listo
2	Detector de llamada
3	Cambio en la señal de línea
4	Listo para enviar
5	Datos listos
6	Indicador de llamada
7	Detección de señal de línea

Con orden igual a 1 se manda el valor de byte por el puerto serie.

Con orden igual a 2 se recibe el valor de byte en el byte de menor orden que nos devuelve la función bioscom().

Los cables para comunicación serie dependiendo del tipo de conectores, tienen esta configuración:



- DB25 a DB25 = 2 con 3, 3 con 2, 7 con 7.
- DB25 a DB9 = 2 con 2, 3 con 3, 7 con 5.
- DB9 a DB9 = 2 con 3, 3 con 2, 5 con 5.

Ejemplo:

```
#include <stdio.h>
#include <bios.h>
#include <string.h>
#include <time.h>

int manda_puerto_serie(char *cadena)
{
clock_t ini,fin;
unsigned int salida,h;
bioscom(0,251,0);
for (h=0;h<strlen(cadena);h++)
  {
  ini=clock();
  do  {
          salida = bioscom(1, cadena[h], 0);
          fin=clock();
          if (((fin-ini)/CLK_TCK)>2) return 1;     Si tarda más de 2 segun
                                                   dos en pasar el carácter
                                                   vuelve devolviendo un 1
          }
  while (!(salida & 0x8000));
  }
bioscom(1, 13, puerto);
return 0;
}
```

```
main()
{
char cadena[]={"ESTA CADENA SE VA A PASAR POR EL PUERTO SERIE"};
manda_puerto_serie(cadena);
return;
}
```

Problema 25

Realizar un programa que nos permita recibir o mandar cualquier array de datos a una velocidad de 9600 BAUDIOS, con paridad par y 2 bits de parada por el puerto serie.

Puerto paralelo

En la transmisión en paralelo se transmiten los bytes enteros de uno en uno. El control de los puertos paralelos lo podemos realizar mediante la función biosprint() que se encuentra en la librería bios.h. La mayoría de las impresoras se conectan al puerto paralelo. El prototipo de biosprint() es:

int biosprint(int orden, int byte, int puerto);

La orden puerto especifica el puerto a controlar (0 para LPT1, 1 para LPT2, 2 para LPT3,...).

La acción de la función se define mediante orden. Sus valores son:

Valores de orden

Valor	Acción
0	Imprime el carácter byte
1	Inicializa el puerto de impresora
2	Devuelve el estado del puerto

Para imprimir un carácter cualquiera se utiliza la función 0, en el ejemplo siguiente:

Ejemplo:

```
char *cade={"Se imprime esta cadena"};
while(*cade) biosprint(0, *cade++, 0); se imprime toda la cadena
en la impresora.
```

Existen otros caracteres que nos permiten pasar de línea, indicar el fin de texto, el fin de página, etc. Estos caracteres reciben el nombre de caracteres de control. Son los siguientes:

Caracteres de control

Código decimal	Código hex	Tecla control	Nombre	Descripción
0	00	^(Control)@	NUL	Carácter nulo
1	01	.... ^A	SOH	Inicio de cabecera
2	02	^B	STX	Inicio de texto
3	03	^C	ETX	Fin de texto
4	04	^D	EOT	Fin de transmisión
5	05	^E	ENQ	Requerimiento
6	06	^F	ACK	Reconocimiento
7	07	^G	BEL	Campana
8	08	^H	BS	Espacio atrás
9	09	^I	HT	Tabulación horizon .tal
10	0A	^J	LF	Avance de línea
11	0B	^K	VT	Tabulación vertical
12	0C	^L	FF	Página nueva
13	0D	^M	CR	Retorno de carro
14	0E	^N	....SO	shift desactivado
15	0F	^O	....SI	shift activado
16	10	^P	....DEL	Borrar
17	11	^Q	....DC1	Control dispositivo 1
18	12	^R	....DC2	Control dispositivo 2
19	13	^S	....DC3	Control dispositivo 3
20	14	^T	....DC4	Control dispositivo 4
21	15	^U	....NAK	Reconocimiento neg tivo
22	16	^V	....SYN	Sincronización
23	17	^W	....ETB	Fin de bloque de texto
24	18	^X	....CAN	Cancelar
25	19	^Y	....EM	Fin de medio
26	1A	^Z	....SUB	Sustituir
27	1B	^[	....ESC	Escape
28	1C	^/	....FS	Separador de fichero
29	1D	^]	....GS	Separador de grupo
30	1E	^	....RS	Separador de reistro
31	1F	^_	....US	Separador de unidad

El estado de la impresora se devuelve en el byte de menor orden del valor devuelto con la opción 2 de orden, de esta forma:

Bits del byte menos significativo devuelto

Bit	Significado
0	Error de temporización (1)
1	No usado
2	No usado
3	Error de E/S (0)
4	Impresora On Line (1)
5	No hay papel (1)
6	Imp. preparada (0)
7	Impresora ocupada (0)

Un programa que imprime un texto determinado sería el siguiente:

Ejemplo:

```
int num_linea=0;
void iprint(char *cad)
{
        while(*cad) biosprint(0, *cad++, 0);    Imprime el carác
                                                ter *cad hasta el
                                                último
        biosprint(0,10,0);                      Siguiente línea
        num_linea++;
        if (num_linea>32)                       Si imprime más
                                                de 32 líneas...
                {
                num_linea=0;            Número de líneas a cero
                biosprint(0,12,0);      Página nueva
                }
}
main()
{
        clrscr();
        biosprint(1,0,0);               Inicializa el puerto LPT1
        iprint("CADENA 1 A IMPRIMIR");
        iprint("SEGUNDA CADENA QUE SE IMPRIME");
```

```
        iprint("TERCERA Y ULTIMA CADENA A IMPRI
        MIR");
        biosprint(0,12,0);          Página nueva
        biosprint(0,4,0);           Fin de transmisión.
        return;
}
```

Problema 26

Partiendo del programa que realiza el fichero de clientes de un videoclub y su grabación en disco, insertar dos opciones nuevas al menú. La primera nos permitirá sacar por impresora la ficha del cliente seleccionado mediante su nombre. La segunda opción nos imprimirá en la impresora la lista de todos los clientes (sólo los nombres y direcciones).

13. MODELOS DE MEMORIA. USO DE MALLOC ()

Los microprocesadores 80286, 80386, 80486 y PENTIUM tienen varios modos de funcionamiento. Cuando el microprocesador funciona en modo real, emula a un microprocesador 8086 pero más rápido. El sistema MS-DOS y el modo de programación para este S.O. utiliza los registros y modo de manejo de memoria del 8086.El microprocesador está compuesto por los siguentes registros:

El direccionamiento de la CPU 8086

Los microprocesadores en modo 8086 sólo pueden direccionar 1 megabyte (1000Ks), el problema viene a la hora de direccionar este megabyte, ya que con un registro de 16 bits sólo podemos direccionar 64 Kbytes, para realizar el direccionamiento se utilizan dos registros de 16 bits, de esta forma:

Registro de segmento: Registro de desplazamiento.

El registro de segmento corresponde a la posición de un bloque de 16 bytes (párrafo) y el registro de desplazamiento la dirección de 64k a partir de la dirección a la que apunta el r. de segmento. Por ejemplo, la dirección siguiente:

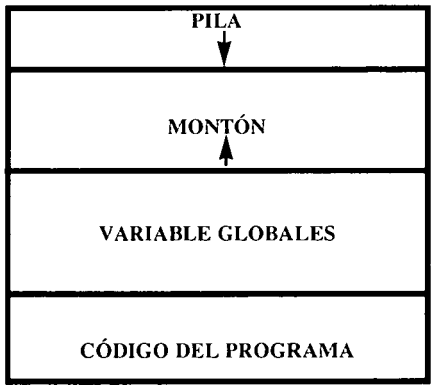
00FF:0E00 Equivalente decimal=255:3570

La dirección real es 255*16+3570=4080+3570=7650

El mapa de memoria de Turbo C

Para cada programa realizado en Turbo C, éste se divide en cuatro bloques:

Mapa de memoria de un programa en Turbo C



• El ***código del programa*** es la zona de memoria donde están las instrucciones del programa.

• La siguiente región es donde se guardan las ***variables globales***.

• El ***montón*** es la zona de memoria donde se van a guardar todas las asignaciones dinámicas que se hagan (con malloc).

• Por último aparece el espacio para la ***pila***. Las flechas indican hacia donde crecen esas zonas de memoria.

Punteros lejanos y punteros cercanos

Si tenemos que acceder a direcciones dentro de un espacio de 64K, con sólo utilizar el registro de desplazamiento podemos hacerlo. A los punteros que acceden sólo a un área de 64K cuya posición inicial está indicada por el registro de segmento propio, se les denominan punteros *near* (punteros cercanos).

En cambio si queremos poder acceder a cualquier dirección de memoria dentro del Megabyte que puede direccionar el 8086 tenemos que utilizar puntero tipo far. Estos punteros llamados lejanos, utilizan dos registros, el de segmento y el de desplazamiento.

Los punteros cercanos son más rápidos que los punteros lejanos, por lo tanto, siempre que se pueda hay que intentar utilizar punteros cercanos.

Modelos de memoria

Modelo diminuto: tiny.

En este modelo de memoria, el compilador hace que todos los registros de segmento tengan el mismo valor, de forma que datos, código y pila comparten 64 K. Con este modo de compilación se realizan programas más rápidos.

Podemos convertir el programa en un archivo .COM con la opción /t del enlazador de Turbo C++.

Modelo pequeño: small.

Este es el modo de compilación por defecto. El código del programa está separado del de pila, datos y montón. de esta forma ocupan espacios de 128 K. De ellos 64K son para código y los otros 64K para datos, pila y montón. Podemos tener más instrucciones que en el modo diminuto sin realentizar el programa.

Modelo mediano: medium.

Este modelo es adecuado para programas extensos que manejan pocos datos. El código utiliza punteros lejanos para el código del programa pero para los datos utiliza punteros cercanos.

El programa será más lento a la hora de llamar a las funciones debido a que se llaman por punteros lejanos, pero los datos se manejarán rápidamente.

Modelo compacto: compact.

Con este modelo, al revés que el modo medium, es indicado para programas pequeños (menos de 64K), pero que manejan gran cantidad de datos. El código de programa se direcciona mediante punteros cercanos y los datos se direccionan mediante punteros lejanos.

En este caso los datos precisan direccionamiento de 20 bits (con punteros lejanos) y el código de programa sólo en 64 Kbytes.

Modelo grande: large.

Con este modelo de memoria podemos tener varios segmentos para código de programa y para datos, pero los elementos de datos no pueden sobrepasar los 64K. Por ejemplo el array:

```
int dat[40000];
```

No cabría ya que ocupa 40000*2=80000 bytes.

Modelo enorme: huge.

Este modelo es igual que el grande, pero nos permite tener elementos de datos más grandes de 64K.

Es evidente que la velocidad de acceso a los datos será más lento.

Consideraciones principales

Para definir un array mayor de 64K se utiliza el comando huge en el modo de memoria enorme, de esta forma:

Ejemplo:

```
int huge var[200000];
main()
{
long h;
for (h=0;h<200000;h++) var[h]=h;
return;
}
```

Los punteros lejanos se declaran con el comando far y los cercanos con near. Para poder acceder a una serie de datos que sobrepasan los 64K se utiliza los punteros huge, un puntero far permite acceder a una dirección compuesta por segmen-

to y desplazamiento, pero el incremento de éste solo incrementa el desplazamiento (offset) no el segmento. Para utilizar datos en bloque mayores de 64K, se utiliza los punteros huge de la siguiente forma:

Ejemplo:

```
int huge datos[200000];
int huge *punt;
void prueba()
{
        long h;
        punt=datos;
        for (h=0;h<200000;h++)
                {
                printf("(%ld)=%d,",h,*punt++);
                }
        return;
}
```

La función malloc(). Asignación dinámica de memoria.

De las diferentes regiones de memoria que utiliza el Turbo C, hemos visto que una de ellas correspondía al montón. El montón es el área de memoria libre gestionada por la funciones malloc() y free(). Estas funciones nos permiten asignar un trozo de memoria libre y después cuando ya no se utilice más liberarla.

Pertenecen a la librería **stdlib.h** siendo los prototipos de estas funciones son:

void *malloc(size_t num_bytes);

void free(void *p);

La función **malloc**() nos devuelve un puntero al principio de la zona de memoria de tamaño num_bytes con **free**() liberamos esa memoria. Si la función malloc() no puede asignar esa memoria, devuelve un puntero nulo (NULL).

Ejemplo:

```
int *p;
p=malloc(15000*sizeof(int));
if (p==NULL) {puts("No hay memoria");exit(0);}
...
...     Instrucciones que utilizan esa zona de memoria
...
free(p);
```

Problema 27

Crear un array de 200000 elementos tipo long y almacenar en él los 200000 primeros números primos. Acostumbrarse al manejo de arrays tipo huge.

Problema 28

Asignar un espacio de memoria de 5000 elementos tipo double, almacenar en cada elemento la raiz cuadrada de un número aleatorio entre 0 y 10000, sacar en pantalla los valores de 100 elementos y por último liberar esa memoria.

14. CREACION DE PROYECTOS Y LIBRERIAS CON TURBO C.

Creación de proyectos

En la creación de programas complejos, es conveniente dividir éste en diferentes ficheros que realicen funciones diferenciadas. El entorno Turbo C nos permite tener y manejar todos estos ficheros y compilarlos. A un programa compuesto por archivos múltiples se le denomina proyecto. Una de ventajas de utilizar un proyecto consiste en que si sólo modificamos uno de los ficheros, sólo se compila éste siendo mucho más rápido la depuración del programa.

Supongamos que tenemos tres ficheros:

- RATON.C *Funciones para manejar el ratón*
- GRAFICOS.C *Funciones para crear gráficos*
- PRINCIPAL.C *Programa que va a utilizar las funciones de RATON.C y de GRAFICOS.C*

Para poder unirlos mediante un proyecto, se selecciona en el menú **project** la opción open y definimos el nombre del proyecto (con extensión PRJ), por ejemplo lo llamamos PROYECTO.PRJ. Una vez creado aparecerá una nueva ventana que pertenece al proyecto recién creado.

Utilizando la opción añadir, unimos los archivos RATON.C, GRAFICOS.C y PRINCIPAL.C. Con la opción delete podemos eliminar cualquier archivo del proyecto.

Si tenemos definidas una variables en la función RATON.C y queremos utilizarlas desde otra función, hay que declararlas como externas. Se puede crear un archivo de cabecera de la siguiente forma:

Ejemplo:
RATON.C

```
...
int xmouse, ymouse, botones;
...
```

PRINCIPAL

```
#include "variab.h"
#include <stdio.h>
#include <conio.h>
...
```

Desde aqui se usan las variables externas.
VARIAB.H *(en el directorio include)*

```
extern int xmouse,ymouse, botones;
```

Creación de bibliotecas de Turbo C

Mediante el programa **TLIB.EXE** que aparece a partir de la versión 1.5 de Turbo C, podemos crear nuestras propias bibliotecas de funciones. La ventaja de crear y utilizar una biblioteca consiste en que sólo se compilan aquellas funciones que se van a utilizar.

Podemos por ejemplo crear una librería de creación de gráficos, uso de ratón, etc. La forma de usar el programa **TLIB.EXE** es la siguiente:

TLIB nombre_librería [opción] nombre_módulo [opción] nombre_módulo ...

Siendo opción:

Valores de opción

Op	Significado
+	Añade un .OBJ a la biblioteca
-	Elimina un .OBJ de la biblioteca
*	Extrae un archivo .OBJ de la biblioteca
-+ ó +-	Reemplaza el .OBJ por uno nuevo
-* o´*-	Extrae un .OBJ de la biblioteca y después lo elimina de ella.

Todos los módulos a añadir a la biblioteca tienen que estar en formato OBJ.

Ejemplo:

TLIB USO_GENE +RATON.OBJ +GRAFICOS.OBJ

Nos crea la librería USO_GENE.H y USO_GENE.OBJ, para poder incluirla en cualquier programa nuestro.

15. ALGUNAS FUNCIONES UTILES.

Función flushall()

Con prototipo:

int flushall(void);

Limpia todos los bufferes, se suele utilizar para limpiar el buffer de escritura antes de utilizar la función scanf(), getch(), etc.

Función sprintf()

Esta función es idéntica a printf() pero en vez de imprimir en pantalla, imprime en una cadena:

Ejemplo:

```
char cadena[80];
int h=10;
sprintf(cadena, "Valor de h=%d",h);
puts(cadena);
```

Función strcpy()

Copia una cadena en otra, tiene el prototipo:

char *strcpy(char *cad1, char *cad2);

Se copia la cadena cad2 en cad1. Esta en la librería string.h.

Ejemplo:

```
char cadena[40], caden2[]={"CADENA A COPIAR"};
strcpy(cadena, caden2);
```

Función strlen()

Devuelve la longitud (sin el carácter '\0') de una cadena:

Ejemplo:

```
cadena[]={"cadena de 23 caracteres"};
printf("Tamaño de la cadena=%d",strlen(cadena));
```

Función clock()

Devuelve un valor aproximado del tiempo que ha transcurrido desde que se ha ejecutado el programa, el valor devuelto es tipo long. Cada 18.2 pulsos corresponde a un segundo.

Pertenece a la librería **time.h.**

Ejemplo:

```
long princ, final;
princ=clock();
...
final=clock();
printf("Tiempo trancurrido en segundos=%f", (final-princ)/18.2);
```

Función delay()

Realiza pausas en milisegundos, pertenece a dos.h.

Ejemplo:

```
delay(500);      Pausa de medio segundo
```

Función chdir()

Cambia el directorio actual de trabajo

Ejemplo:

```
chdir("c:\\tc\\programs");
```

Función system()

Ejecuta una instrucción del MS-DOS. Se encuentra en **stdlib.h.**

Ejemplo:

```
#include <stdlib.h>
main()
{
        system("cls");
        system("dir *.exe");
        getch();
        return 0;
}
```

16. PROGRAMA DE EJEMPLO.

Se ha desarrollado un programa de ejemplo que une los conocimientos adquiridos en todo el libro. La consulta de este listado puede ayudar al conocimiento del uso correcto de la sintaxis y organización de un programa en lenguaje Turbo C.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <dir.h>
#include <bios.h>

// PROGRAMA PARA CREAR Y MANEJAR UN FICHERO DE CLIENTES DE UN VIDEOCLUB

// (c) FRANCISCO JAVIER MARTINEZ DE PISON, 1994

#define MAXIM 100 // NUMERO DE FICHAS
#define OPCIONES 8          // OPCIONES DEL MENU
#define FICHERO "CLIENTES.DAT" // FICHERO DE CLIENTES
#define LPTN 0  // DONDE ESTA LA IMPRESORA LPT1=0

struct ficha{
            long dni;
            long numero_video;
            unsigned char nombre[45];
            unsigned char calle[40];
            unsigned char telefono[20];
            unsigned char video[40];
            };

void menu(),borra_ficha(struct ficha *), bor_fichero()
,bor_ficha();
void esc_ficha(), busc_dni(), busc_nom(), busc_dir(), busc_vid(), salir(), inicio(), busca_en_fic();
void vis_fic(),imp_fic(),imp_n_ficha();
void (*p_func[])(void)= { bor_fichero, bor_ficha
, esc_ficha, busca_en_fic, vis_fic, imp_fic ,imp_n_ficha,salir};

int numlineap=0;
struct ficha fichero[MAXIM];

main()
{
  inicio();
  menu();
  return;
}

// MENU PRINCIPAL

void menu(void)
{
  char *menus[]={"BORRAR FICHERO ENTERO","BORRAR FICHA","ESCRIBIR FICHA"
,"BUSCAR EN FICHERO","VISUALIZA FICHERO","IMPRIMIR FICHERO","IMPRIMIR UNA FICHA"
, "SALIR"};

//,esc_fic,busc_nom,busc_dni
//busc_titu,salir};
  int h;
  char c;
  while(1)
  {
          clrscr();
          textcolor(4);
          gotoxy(15,1);
          cprintf("## MENU VIDEOCLUB ##\n\r");
```

```
                textcolor(2);
                for (h=0;h<OPCIONES;h++)
                {
                        cprintf("\n\r
%2d.- %s",h+1,menus[h]);
                }
                do  c=getch();
                        while (c<'1' ||
c>('0'+ OPCIONES));
                textcolor(15);
                p_func[c-49]();
        }
      }

      // GUARDA UNA FICHA EN EL
ARCHIVO

      void guarda_en_fichero(int ficha)
      {
        FILE *pfic;
        pfic=fopen(FICHERO,"rb+");
        if (pfic==NULL) {clrscr();

printf("ERROR ARCHIVO DE CLIENTES
        (CLIENTES.DAT)\nNO ENCON-
TRADO");getch();exit(1);}

fseek(pfic,ficha*sizeof(fichero[0]),SEEK_SET);
        fwrite(&fichero[ficha],sizeof(fiche-
ro[0]),1,pfic);
        fclose(pfic);
        return;
      }

      // GUARDA TODO EL FICHERO

      void guarda_todo_fichero()
      {
        FILE *pfic;
        pfic=fopen(FICHERO,"wb+");
        if (pfic==NULL)
{clrscr();printf("ERROR AL CREAR ARCHI-
VO DE CLIENTES (CLIENTES.DAT)");
                        getch();exit(1);}
        fwrite(&fichero,sizeof(fichero),1,pfic);
        fclose(pfic);
        return;
      }
      // RECUPERA TODO EL FICHERO

      void recup_todo_fichero()
      {
        FILE *pfic;
        pfic=fopen(FICHERO,"rb+");
        if (pfic==NULL)
{clrscr();printf("ERROR ARCHIVO DE
CLIENTES (CLIENTES.DAT)\nNO ENCON-
TRADO");getch();exit(1);}
        fread(&fichero,sizeof(fichero),1,pfic);
        fclose(pfic);
        return;
      }

      // BORRA UNA FICHA

      void borra_ficha(struct ficha *p)
      {
        p->dni=0;
        p->nombre[0]='\0';
        p->video[0]='\0';
        return;
      }
      // BORRA TODO EL FICHERO

      void bor_fichero()
      {
        int h;
        char c;
        clrscr();

printf("##################################
#######");
        printf("\n¿QUIERES BORRAR TODO
EL FICHERO? (s/n):");
```

```
printf("\n##################################
########");
        c=getch();
        if (c!='s' && c!='S') return;

        for (h=0;h<MAXIM;h++)
        {
                borra_ficha(&fichero[h]);
        }
        guarda_todo_fichero();
        printf("\n\nFICHERO BORRADO.
Pulsa una tecla para continuar");
        getch();
        return;
      }

      // PIDE FICHA PARA BORRAR

      void bor_ficha()
      {
        int fic;char cadena[20];
        clrscr();
        printf("INTRODUCE NUMERO DE
FICHA PARA BORRAR
(max=%d)=",MAXIM-1);
        cadena[0]=18;
        cgets(cadena);
        fic=atoi(cadena+2);
        if (fic<0 || fic>(MAXIM-1)) return;
        borra_ficha(&fichero[fic]);
        guarda_en_fichero(fic);

printf("\n\n##############################
##################");
        printf("\nFICHA %2d BORRADA.
Pulsa una tecla para continuar",fic);

printf("\n##################################
################");
        getch();
        return;
      }
      // CREA O RECUPERA TODO EL
FICHERO

      void inicio()
      {
        struct ffblk ffblk;
        char c;int h;
        if (findfirst(FICHERO,&ffblk,0))
                {
                printf("\nARCHIVO %s NO
ENCONTRADO\n", FICHERO);
                printf("CREANDO ARCHI-
VO NUEVO\nPULSA S PARA CREARLO");
                c=toupper(getch());
                if (c=='S') {


      for (h=0;h<MAXIM;h++)
        {
        borra_ficha(&fichero[h]);
        }
      guarda_todo_fichero();
          }
      clrscr();
      }
      else {
         recup_todo_fichero();
         }
      return;
      }

      // PONE FICHA EN PANTALLA

      void pon_ficha(int fic)
      {
        textcolor(10);

cprintf("NOMBRE:%s",fichero[fic].nombre);
        cprintf("\n\rDNI:%ld",fichero[fic].dni);
        cprintf("\n\rDIRECCION:%s",fiche-
ro[fic].calle);
```

```
cprintf("\n\rTELEFONO:%s",fichero[fic].telefono);
        cprintf("\n\r\n\rVIDEO ALQUILADO:%s" ,fichero[fic].video);
        cprintf("\n\rCODIGO DEL VIDEO=%ld", fichero[fic].numero_video);
        textcolor(15);
        return;
      }
      // PONE FICHA ENCONTRADA

      int pon_ficha2(int h)
      {
        char c,cad[50];
        textcolor(3);
        cprintf("\n\r--------------------------------------------------------------------------------");
        cprintf("\n\rFICHA ENCONTRADA:%d",h);
        cprintf("\n\r");
        pon_ficha(h);
        printf("\n\n¿Cambiar pelicula (c)? Otra tecla para continuar");
        c=toupper(getch());

        if (c=='C')
                {

printf("\n##################################################");
                printf("\nINTRODUCE TITULO DE LA PELICULA:");
                cad[0]=45;
                cgets(cad);

strcpy(fichero[h].video,cad+2);
                printf("\nINTRODUCE NUMERO DE LA PELICULA:");
                cad[0]=10;
                cgets(cad);
fichero[h].numero_video=atol(cad+2);
                printf("\n--------------------------------------------------------------------------------\n");
                pon_ficha(h);
                printf("\n--------------------------------------------------------------------------------\n");
                }
        clrscr();
        return 0;
      }

      // VISUALIZA TODO EL FICHERO

      void vis_fic()
      {
        int h;
        clrscr();

        for (h=0;h<MAXIM;h++)
        {
                if (fichero[h].nombre[0]!='\0') pon_ficha2(h);
        }
        return;
      }

      // INTRODUCE FICHERO

      void esc_ficha()
      {
        int h;char c;char cadena[50];
        clrscr();
        for (h=0;h<MAXIM;h++)
        {
                if (fichero[h].nombre[0]=='\0')
                        {
                        clrscr();
                        printf("FICHA Nº=%d\n\n",h);
                        printf("INTRODUCE NOMBRE:");
```

```
                    cadena[0]=45;
                    cgets(cadena);

strcpy(fichero[h].nombre,cadena+2);
                    if
(strlen(fichero[h].nombre)==0)return;

printf("\nINTRODUCE DIRECCION:");
                    cadena[0]=40;
                    cgets(cadena);

strcpy(fichero[h].calle,cadena+2);

printf("\nINTRODUCE TELEFONO:");
                    cadena[0]=20;
                    cgets(cadena);

strcpy(fichero[h].telefono,cadena+2);

printf("\nINTRODUCE DNI:");
                    cadena[0]=10;
                    cgets(cadena);

fichero[h].dni=atol(cadena+2);

printf("\n\nINTRODUCE TITULO DE LA
PELICULA:");
                    cadena[0]=40;
                    cgets(cadena);

strcpy(fichero[h].video,cadena+2);

printf("\nINTRODUCE CODIGO DE LA PELI-
CULA:");
                    cadena[0]=10;
                    cgets(cadena);
                    fichero[h].nume-
ro_video=atol(cadena+2);
                    printf("\n————
————————————————————————————————
\n");
                    guarda_en_fiche-
ro(h);
                    pon_ficha(h);
                    printf("\n————
————————————————————————————————
\n");
                    printf("————
PULSA UNA TECLA PARA CONTINUAR—
———");
                    getch();
                    }
          }
          printf("\n¡¡¡FICHERO
LLENO!!!.Pulsa una tecla para continuar");
          getch();
          return;
        }

        // BUSCA EN EL ARCHIVO

        void busca_en_fic()
        {
          char c;
          clrscr();
          textcolor(9);
          flushall();
          cprintf("\n\n\rINTRODUCE:\n\r
0=BUSCAR DNI\n\r  1=BUSCAR
NOMBRE\n\r  2=BUSCAR DIRECCION\n\r
3=BUSCAR PELICULA:");
          textcolor(15);
          do c=getch();
          while (c<48 || c>51);
          switch(c)
          {
                case '0':
                      busc_dni();
                      break;
                case '1':
                      busc_nom();
                      break;
                case '2':
                      busc_dir();
```

```
                        break;
                case '3':
                        busc_vid();
                        break;
        }
        return;
}

// BUSCA DENTRO DEL DNI

void busc_dni()
{
  int h;char c;char cadena[20];
  long dni2;
  clrscr();
  printf("INTRODUCE DNI PARA
BUSCAR:");
  cadena[0]=10;
  cgets(cadena);
  dni2=atoi(cadena+2);

  for (h=0;h<MAXIM;h++)
  {
        if (fichero[h].nom-
bre[0]!='\0')
                {
                if
(dni2==fichero[h].dni)
                        {
                        if
(pon_ficha2(h)) return;
                        }
                }
  }
  printf("\n¡¡¡FINAL DEL FICHE-
RO!!!.Pulsa una tecla para continuar");
  getch();
  return;
}

// BUSCA EN NOMBRE

void busc_nom()
{
  int h;
  char cadena[40];
  clrscr();
  printf("INTRODUCE CADENA
PARA BUSCAR EN NOMBRE:");
  cadena[0]=45;
  cgets(cadena);
  for (h=0;h<MAXIM;h++)
  {
        if (fichero[h].nom-
bre[0]!='\0')
                {
                if
(strstr(strupr(fichero[h].nombre),strupr(cade-
na+2)))
                        {
                        if
(pon_ficha2(h)) return;
                        }
                }
  }
  printf("\n¡¡¡FINAL DEL FICHE-
RO!!!.Pulsa una tecla para continuar");
  getch();
  return;
}

// BUSCA EN DIRECCION

void busc_dir()
{
  int h;
  char cadena[42];
  clrscr();
  printf("INTRODUCE CADENA
PARA BUSCAR EN DIRECCION:");
  cadena[0]=40;
  cgets(cadena);
  for (h=0;h<MAXIM;h++)
  {
```

```
                if (fichero[h].nom-
bre[0]!='\0')
                        {
                        if
(strstr(strupr(fichero[h].calle),strupr(cadena+2)))
                                {
                                if
(pon_ficha2(h)) return;
                                }
                        }
        }
        printf("\n¡¡¡FINAL DEL FICHE-
RO!!!.Pulsa una tecla para continuar");
        getch();
        return;
      }

      // BUSCA EN VIDEO

      void busc_vid()
      {
        int h;
        char cadena[42];
        clrscr();
        printf("INTRODUCE CADENA
PARA BUSCAR EN PELICULA:");
        cadena[0]=40;
        gets(cadena);
        for (h=0;h<MAXIM;h++)
        {
                if (fichero[h].nom-
bre[0]!='\0')
                        {
                        if
(strstr(strupr(fichero[h].video),strupr(cadena+2)))
                                {
                                if
(pon_ficha2(h)) return;
                                }
                        }
        }
        printf("\n¡¡¡FINAL DEL FICHE-
RO!!!.Pulsa una tecla para continuar");
        getch();
        return;
      }


      // OPCION PARA SALIR

      void salir()
      {
        char c;
        clrscr();
        printf("QUIERES SALIR DE VER-
DAD (s/n)");
        do      {
                c=getch();
                if (c=='n' || c=='N') return;
                }
        while (c!='s' && c!='S');
        exit(0);
      }

      // IMPRIME EN LA IMPRESORA UNA
CADENA

      void pprint2(char *txtc)
      {
        while(*txtc)
{biosprint(0,*txtc,LPTN);txtc++;}
        return;
      }

      // IMPRIME LINEAS Y MIRA SI SE
SALE DE PAGINA

      void pprint(char *txtc)
      {

        pprint2(txtc);
        biosprint(0,10,LPTN);
        biosprint(0,13,LPTN);
        numlineap++;
```

```
        if (numlineap>58) {numlineap=0;bios-
print(0,12,LPTN);}
        error();
        return;
      }

      // MIRA EL ESTADO DE LA IMPRE-
SORA

      int estado_prin(void)
      {
        int estado=0;
        estado=biosprint(2,estado,LPTN);
        if (estado & 0x08) {printf("ERROR
I/O.          ");return(255);}
        if (estado & 0x01) {printf("LA
IMPRESORA NO RESPONDE.");return(255);}
        if (!(estado & 0x10)) {printf("PON LA
IMPRESORA ON-LINE.");return(255);}
        if (estado & 0x20) {printf("NO HAY
PAPEL.        ");return(255);}
        return 0;
      }

      // MIRA SI HAY ERROR

      int error(void)
      {
        char c;
        while    (estado_prin()==255)
                 {
                 printf("PULSA UNA
TECLA,ESC SALIR.");
                 c=getch();
                 if (c==27) return 255;
                 }
        return 0;
      }

      // INICIALIZA IMPRESORA
      int ini_imp()
      {
        biosprint(1,0,LPTN);
        numlineap=0;
        return (0);
      }

      // FIN DE IMPRESORA
      void fin_imp()
      {

        biosprint(0,12,LPTN);
        biosprint(0,3,LPTN);
        biosprint(0,4,LPTN);
        return;
      }

      void imp_ficha(int fic)
      {
        char cadena[70];

pprint("==========================
==================");
        sprintf(cadena,"FICHA NUME-
RO=%d",fic);
        pprint(cadena);

pprint("==========================
==================");
        sprintf(cadena,"NOMBRE:%s",fiche-
ro[fic].nombre);
        pprint(cadena);

sprintf(cadena,"DNI:%ld",fichero[fic].dni);
        pprint(cadena);

sprintf(cadena,"DIRECCION:%s",fichero[fic].ca
lle);
        pprint(cadena);

sprintf(cadena,"TELEFONO:%s",fichero[fic].tel
efono);
        pprint(cadena);
        sprintf(cadena,"VIDEO ALQUILA-
```

```
        DO:%s",fichero[fic].video);
          pprint(cadena);
          sprintf(cadena,"CODIGO DEL
VIDEO=%ld",fichero[fic].numero_video);
          pprint(cadena);
          pprint("-------------------------
-------------------------");
          return;
        }

        void imp_fic()
        {
          int h,k;
          clrscr();
          printf("SACAR FICHERO POR LA
IMPRESORA\nPULSA UNA TECLA");
          getch();
          k=error();
          if (k==255) return;
        // ini_imp();

          for (h=0;h<MAXIM;h++)
          {
                  if (fichero[h].nom-
bre[0]!='\0') imp_ficha(h);
          }
          fin_imp();
          return;
        }

        // PIDE FICHA PARA IMPRIMIR
        void imp_n_ficha()
        {
          int fic;char cadena[20];
          clrscr();
          printf("INTRODUCE NUMERO DE
FICHA PARA IMPRIMIR
(max=%d)=",MAXIM-1);
          cadena[0]=18;
          cgets(cadena);
          fic=atoi(cadena+2);
          if (fic<0 || fic>(MAXIM-1)) return;
          imp_ficha(fic);
        // fin_imp();
          getch();
          return;
        }
```

BIBLIOGRAFIA

Bibliografía.

1.- Herbert Schildt (1992). *Turbo C/C++ Manual de Referencia*,. Mc Graw Hill, Madrid

2.- Scott Zimmerman, Beverly B.Zimmerman (1989). *La Biblia del Turbo C*. Anaya multimedia.

3.- Pilar Lasala y Alberto Lekuona.(1989):*Lenguaje de programación C*. Universidad de Zaragoza, Zaragoza.

4.- Michel Tischer.(1993): *PC interno*. Marcombo.

5.- David J.Kruglinski.(1994): *Progrese con Visual C++*. Mc Graw Hill. Madrid.

6.- Peter Norton (1987). *El IBM PC a fondo*. Anaya multimedia. Madrid

7.- Peter Norton, John Socha.(1988): *Guía del programador en Ensamblador para IBM, PC, XT AT y Compatibles*,.Anaya multimedia. Madrid

8.- Lee Adams.(1991) *Programación gráfica en C*. Anaya multimedia. Madrid.

UNIVERSIDAD DE LA RIOJA
SERVICIO DE PUBLICACIONES