

Lenguaje C

© 2005 José Luis Asín Buñuel
Dpto. de Tecnología
I.E.S. Barañain
jasin@retna.net

Índice de contenidos

1. INTRODUCCIÓN	
Orígenes	4
Características	4
Proceso de edición y compilación.....	4
2. ESTRUCTURA DEL LENGUAJE C	
Estructura de un programa	5
3. LOS DATOS EN C: VARIABLES Y CONSTANTES	
Tipos de datos	6
Identificadores de tipo.....	6
Variables	7
Constantes	7
4. OPERADORES	
Operadores fundamentales.....	8
Operadores adicionales	8
Operadores condicionales de relación	8
Operadores lógicos.....	9
Operadores de bit	9
Normas de precedencia de los operadores	9
5. FUNCIONES DE ENTRADA/SALIDA	
Función printf()	10
Función scanf()	10
6. SENTENCIAS DE CONTROL DEL PROGRAMA	
Tipos de estructuras básicas.....	12
Sentencias condicionales	12
Sentencias repetitivas.....	13
Otras sentencias de control	14
7. FUNCIONES DE MANIPULACIÓN DE CARACTERES	
Función getchar()	15
Función putchar()	15
8. EL PREPROCESADOR C	
#define.....	16
#include.....	16
Otros comandos: #undef, #if, etc	17
9. ARRAYS Y PUNTEROS	
Arrays.....	18
Inicialización de los arrays.....	18
Arrays multidimensionales	19
Punteros.....	20
Arrays con punteros	21

10. ARRAYS DE TIRAS DE CARACTERES	
gets() y puts()	23
Arrays multidimensionales con tiras.....	24
Funciones relacionadas con tiras	24
11. FUNCIONES	
Forma general de una función.....	26
Llamada por valor o por referencia.....	26
Sentencia return	27
Prototipos de funciones.....	27
Argumentos del main()	28
Recursividad	28
Declaración de listas de parámetros de longitud variable.	29
Punteros a funciones	30
12. ESTRUCTURAS Y UNIONES	
Estructuras.....	31
Acceso a un miembro de la estructura	31
Arrays de estructuras.....	31
Estructuras anidadas.....	32
Punteros a estructuras	32
Campos de bits	32
Uniones	34
Enumeraciones	34
Abreviación de los tipos de datos	35
13. FUNCIONES DE ACCESO A FICHEROS DE DISCO	
Ficheros	36
Funciones de acceso a ficheros	36
Funciones fprintf, fscanf, fgets, fputs	40
Funciones fread, fwrite, fseek, ftell	41
14. PRÁCTICAS DE C	
Enunciados de las prácticas	43
15. LISTADO DE LAS PRÁCTICAS	
Listados de las prácticas propuestas	47

1.- INTRODUCCIÓN

1.1.- ORÍGENES

Fue creado por Dennis Ritchie de los laboratorios BELL en 1972, cuando trabajaba junto a Ken Thompson, en el diseño del sistema operativo UNIX. Se deriva del lenguaje B de Thompson, que a su vez se deriva del BCPL de Martín Richards.

1.2.- CARACTERÍSTICAS

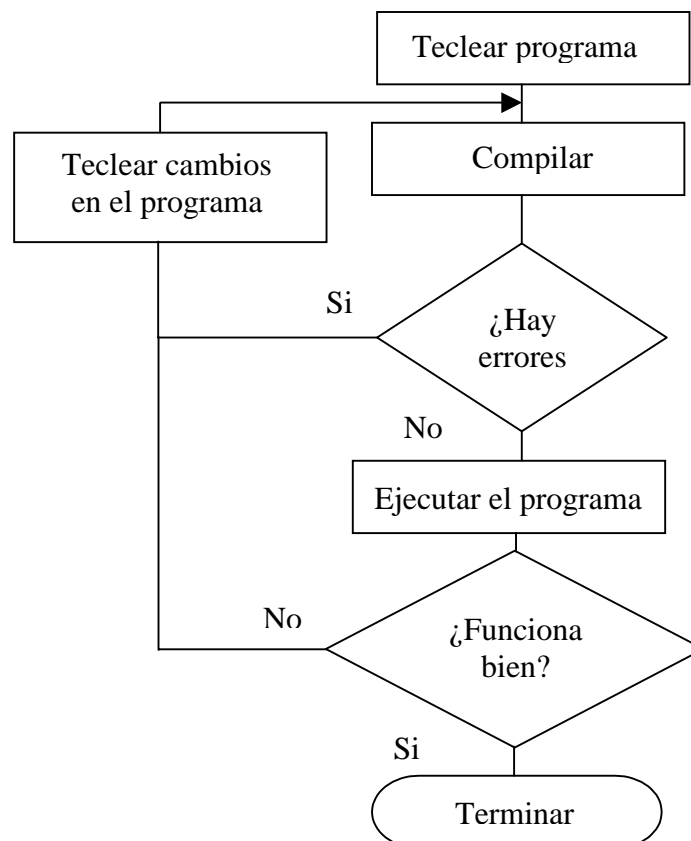
Es un lenguaje moderno de propósito general, que incorpora las características de control apuntadas como deseables por la teoría y práctica de la informática.

- Planificación escalonada.
- Programación estructurada.
- Diseño modular.
- Programas compactos.
- Rapidez de ejecución.
- Portátil.
- De relativo bajo nivel.
- Precisa compilarse.

1.3.- PROCESO DE EDICIÓN Y COMPILACIÓN

Los pasos necesarios para desarrollar un programa C son los siguientes:

- a) Edición: Utilizar un editor para escribir el programa fuente texto.
- b) Compilación: Compilar el programa fuente, es decir, traducir el programa a lenguaje máquina.
- c) Ejecución: Una vez compilado se procede a la ejecución del programa tecleando el nombre del fichero-programa.



2.- ESTRUCTURA DEL LENGUAJE C

2.1.- ESTRUCTURA DE UN PROGRAMA

Un programa en C consiste en una o más funciones, de las cuales una de ellas, debe llamarse **main()** y es la principal de todas.

- El programa comienza con la función: **main()**
- Cada función o programa, consta de un cuerpo de función delimitado por llaves de comienzo y fin **{ }**
- En el cuerpo de la función irán incluidas: sentencias, variables, funciones, etc. terminadas cada una de ellas por un punto y coma ;

```
main()           ← nombre de la función
{               ← comienzo del cuerpo de la función
variables;
sentencias;
funciones;
.....;
.....;
}               ← fin de la función
```

} ← cuerpo de la función

Programa ejemplo:

```
/* Este programa imprime un mensaje */
#include <stdio.h>
main()
{
printf("LENGUAJE C\n");
}
```

En el programa anterior, existen dos funciones: **main()** que es la principal del programa en sí y la función **printf()** que es una función de la librería estándar del lenguaje C.

Al ejecutar el programa, en pantalla aparece el texto LENGUAJE C.

Programa ejemplo:

```
/* El programa solicita el radio y muestra el */
/* valor del área del círculo y la longitud de */
/* la circunferencia */
#include <stdio.h>
#define PI 3.14159
main()
{
int r;
float l, a;
printf("Introduce radio (entero): ");
scanf("%d", &r);
l=2*PI*r; a=PI*r*r;
printf("La longitud de la circunferencia vale %f\n", l);
printf("El área del círculo vale %f\n", a);
}
```

3.- LOS DATOS EN C: VARIABLES Y CONSTANTES

3.1.- TIPOS DE DATOS

Los datos manejados en C pueden ser de cinco tipos básicos.

- INT: enteros sin decimales entre (-32768 y +32767). Ocupan en la memoria 2 bytes.
- CHAR: caracteres alfabéticos, signos especiales, etc. El rango es (0 y 255). Ocupan en la memoria 1 byte.
- FLOAT: números decimales entre (3.4E-38 a 3.4E+38) con ocho dígitos de precisión. Ocupan en la memoria 4 bytes.
- DOUBLE: números decimales entre (1.7E-308 a 1.7E+308) con 16 dígitos de precisión. Ocupan en la memoria 8 bytes.
- VOID: sin valor. No almacenar nada y por tanto no necesitan espacio físico de la memoria.

3.2.- IDENTIFICADORES DE TIPO

Todos los tipos básicos excepto *void* pueden tener modificadores. Se usan para alterar el significado de un tipo base de acuerdo con nuestras necesidades.

Los modificadores son:

signed
unsigned
long
short

Se pueden aplicar todos los modificadores para los tipos base carácter y entero. También se puede aplicar el modificador *long* a *double*. A continuación mostramos todas las posibles combinaciones de los tipos básicos y los modificadores.

Tipo	Bytes	Rango
char	1	-128 a 127
unsigned char	1	0 a 255
signed char	1	-128 a 127
int	2	-32768 a 32767
unsigned int	2	0 a 65535
signed int	2	-32768 a 32767
short int	2	-32768 a 32767
unsigned short int	2	0 a 65535
signed short int	2	-32768 a 32767
long int	4	-2147483648 a 2147483647
signed long int	4	-2147483648 a 2147483647
unsigned long int	4	0 a 4294967295
float	4	3.4E-38 a 3.4E+38
double	8	1.7E-308 a 1.7E+308
long double	8	1.7E-308 a 1.7E+308

NOTA: El espacio ocupado en la memoria por los tipos de datos aquí mostrados vendrá determinado por el tipo de compilador implementado. En cualquier caso, mediante la función `sizeof(tipo de dato)` se podrá determinar el número de bytes ocupados.

3.3.- VARIABLES

Se utilizan para almacenar datos internamente en la memoria de trabajo del ordenador, durante la ejecución de un programa.

Como nombres de variables se admiten letras y números (por norma letras minúsculas).

- El primer carácter debe ser una letra.
- No se admiten blancos pero sí el signo de subrayado.
- Sólo se admiten los ocho primeros caracteres.
- Mayúsculas y minúsculas se diferencian.
- No se pueden emplear palabras clave del lenguaje.

Las variables deben ser declaradas según el tipo de datos que van a almacenar. Bien al comienzo del programa o antes de ser usadas por primera vez.

La forma de declararlas es:

Se define el tipo de dato a almacenar.

Nombre de la variable.

Asignación de un valor inicial (no obligatorio).

Ejemplo:

```
/* Declaración de variables */
#include <stdio.h>
main()
{
    int a=3, b=7, c=0;
    c=a*b;
    printf("El resultado es: %d",c);
}
```

Ver también el programa ejemplo de la página 4.

3.4.- CONSTANTES

Son valores que no varían durante la ejecución de un programa. Pueden adoptar cualquiera de los tipos de datos declarados anteriormente. Es conveniente declararlas al principio del programa.

La sentencia **#define** es usada para definir las constantes y se coloca al principio del programa antes de la función **main()**.

```
#define PI 3.14159
#define PRIMO 13
```

Las constantes como norma se suelen escribir con mayúsculas. Ver el programa ejemplo de la página 4.

4.- OPERADORES

4.1.- OPERADORES FUNDAMENTALES

En C se utilizan operadores para representar operaciones matemáticas. Por ejemplo, el operador + hace que se sumen los valores situados a su izquierda y derecha. Veamos los que hemos dado en llamar fundamentales.

- Operador de asignación: = Asigna a la variable de la izquierda el valor de la derecha.

bmw=2002;

- Operador de adición: + Hace que los dos valores situados a su izquierda y derecha se sumen.

bmw=10+20;

- Operador de sustracción: - Hace que se reste el número situado a su derecha del situado a su izquierda.

costo=350-25;

El signo menos también se utiliza para indicar o cambiar el signo algebraico de un valor.

pepe=-12;

paco=-pepe;

- Operador de multiplicación: * Hace que se multipliquen los dos valores situados a su izquierda y derecha.

pulg=10;

*cm=2.54*pulg;*

- Operador de división: / Hace que el valor situado a su izquierda sea dividido por el que se encuentra a su derecha.

cuatro=12.0/3.0;

4.2.- OPERADORES ADICIONALES

A la lista de los operadores anteriores (que son los más comunes), vamos a añadir ahora tres operadores que resultan bastantes útiles.

- Operador módulo: % Proporciona el resto de la división entera del número situado a su izquierda entre el situado a su derecha.

pepe=15%5;

- Operador incremento: ++ Realiza la tarea de incrementar en uno el valor de su operando.

a++; modo sufijo

++a; modo prefijo

La diferencia entre ambos, reside en el preciso momento en que se realiza la operación incremento.

- Operador decremento: -- Realiza la tarea de decrementar en uno el valor de su operando.

b--; modo sufijo

--b; modo prefijo

La diferencia entre ambos, es la misma que en el caso anterior.

4.3.- OPERADORES CONDICIONALES O DE RELACIÓN

Los operadores de relación o condicionales se emplean para hacer comparaciones. A continuación, mostramos una lista completa de estos operadores en C.

<u>Operador</u>	<u>Significado</u>
<	es menor que
<=	es menor o igual que
==	es igual a
>=	es mayor o igual que
>	es mayor que
!=	es distinto de

4.4.- OPERADORES LÓGICOS

Cuando necesitamos combinar dos o más expresiones de relación, hacemos uso de los operadores lógicos. Existen tres operadores lógicos en C.

<u>Operador</u>	<u>Significado</u>
&&	and (y)
	or (o)
!	not (no)

4.5.- OPERADORES DE BIT

Las operaciones sobre bits se refieren a la comprobación, configuración o desplazamiento de los bits reales en un byte o palabra, que corresponden a los tipos de datos *char* e *int*.

No se pueden usar operaciones sobre bits en los tipos *float*, *double*, *long double* y *void*.

Los operadores que existen para el tratamiento de bits, son los siguientes:

<u>Operador</u>	<u>Significado</u>
&	AND
	OR
^	OR-EXCLUSIVE
~	NOT
>>	DESPLAZAMIENTO DERECHA
<<	DESPLAZAMIENTO IZQUIERDA

4.6.- NORMAS DE PRECEDENCIA DE LOS OPERADORES

No todos los operadores tienen la misma prioridad, lo que quiere decir, que aquellas operaciones que utilizan ciertos operadores se efectuarán antes que otras.

OPERADORES	PRIORIDAD
() [] -> .	Izquierda a derecha
! - ++ -- (tipo) * & sizeof	Derecha a izquierda
* / %	Izquierda a derecha
+ -	Izquierda a derecha
<< >>	Izquierda a derecha
> <= > >=	Izquierda a derecha
== !=	Izquierda a derecha
&	Izquierda a derecha
^	Izquierda a derecha
	Izquierda a derecha
&&	Izquierda a derecha
	Izquierda a derecha
? :	Derecha a izquierda
= += -= etc	Derecha a izquierda

5.- FUNCIONES DE ENTRADA/SALIDA

Las funciones de entrada/salida permiten al programa comunicarse con el exterior. Son utilizadas para sacar determinadas informaciones por la pantalla y capturar valores por el teclado. Son estándar de la propia librería de C por lo que no hay necesidad de definir las de nuevo.

5.1.- FUNCIÓN PRINTF()

Lee datos del programa (variables, resultados de operaciones, frases, mensajes, etc.) y los envía a la pantalla. El formato es el siguiente:

printf("Cadena de control", variables, operaciones, ...);

La cadena de control se usa para escribir mensajes, identificar el tipo de dato que se va a mostrar, indicar controles tales como: salto de línea, salto de página, etc. Los identificadores empleados son los siguientes:

<u>Identificador</u>	<u>Salida</u>
%d	entero decimal
%c	carácter
%s	tira de caracteres
%f	número de punto flotante con notación decimal
%e	número de punto flotante con notación exponencial
%u	entero decimal sin signo
%o	entero octal sin signo
%x	entero hexadecimal sin signo (minúsculas)
%X	entero hexadecimal sin signo (mayúsculas)

Un número colocado entre el % y el orden de formato, actúa como especificador mínimo de ancho de campo. Este especificador completa la salida con blancos o ceros para asegurar que al menos es la longitud mínima. Si la cadena o el número a representar es mayor que el campo mínimo, el campo se amplía para imprimir la cadena o el número completo.

Ejemplos:

%5d	ajuste a la derecha, rellena con espacios
%05d	ajuste a la derecha, rellena con ceros
%10.4f	ajuste a la derecha, rellena con espacios y cuatro decimales
%-10.2f	ajuste a la izquierda, rellena con espacios y dos decimales

Caracteres de control son:

<u>Control</u>	<u>Efecto en pantalla</u>
\n	cambia de línea
\r	retorno de carro
\t	tabulador
\b	retroceso del cursor una posición
\f	cambio de página
\\	barra atrás (\)
\'	apóstrofe (')
\"	comillas (")

5.2.- FUNCIÓN SCANF()

Esta función, lee datos del teclado y los entrega al programa. El formato es el siguiente:

scanf("Cadena de control", variable, variable, ...);

La cadena de control se usa para identificar el tipo de dato que se va a introducir o capturar. Los identificadores empleados son los mismos que los empleados en la función *printf()*.

La variable o variables que se van a introducir tendrán que ir acompañadas delante del nombre por el signo (&). Si el dato que se va a introducir es una tira de caracteres tipo *char[]*, no es necesario el signo.

Ejemplo:

```
/* Programa de empleo de printf() y scanf() */  
main()  
{  
  int edad;  
  char nombre[20];  
  printf("Introduce nombre: \n");  
  scanf("%s", nombre);  
  printf("Introduce edad: \n");  
  scanf("%d",&edad);  
  printf("Tu nombre es %s y tienes %d años. \n", nombre, edad);  
}
```

6.- SENTENCIAS DE CONTROL DEL PROGRAMA

6.1.- TIPOS DE ESTRUCTURAS BÁSICAS

Las sentencias de control de los programas especifican el orden en que se han de realizar las operaciones. Existen tres formas básicas de flujo en los programas.

- Estructuras secuenciales: se componen de una serie de sentencias que se ejecutan unas tras otras en el orden escrito en el programa. Para estas estructuras no se necesita ninguna sentencia de control del flujo del programa.
- Estructuras condicionales: dentro del flujo del programa se toman decisiones para realizar unas acciones u otras dependiendo de que se cumplan una serie de condiciones.
- Estructuras repetitivas: repetición de una serie de sentencias mientras no se cumpla una determinada condición. También denominadas bucles.

6.2.- SENTENCIAS CONDICIONALES

- Sentencia *if-else*. El formato es el siguiente:

Formato 1: *if (condición)*
{ sentencias }

Formato 2: *if (condición)*
{ sentencias 1 }
else
{ sentencias 2 }

Ejemplo:

```
main()
{
float cm, pul, valor;
char unidad;
printf("Introduce número y unidad (p=pulg. o c=cm: ");
scanf("%f %c",&valor, &unidad);
if (unidad=='p')
{
cm=valor*2.54;
printf("Las %f pulgadas son %f centímetros\n",valor,cm);
}
if (unidad=='c')
{
pul=valor/2.54;
printf("Los %f centímetros son %f pulgadas\n",valor,pul);
}
if (unidad!='p' && unidad!='c')
printf("No está puesta correctamente la unidad\n");
}
```

En C existe una forma abreviada de expresar la sentencia *if-else*. Se denomina "expresión condicional" y emplea el operador condicional *? : .* El formato es el siguiente:

expresión 1 ? expresión 2 : expresión 3

El valor de la expresión total es igual al valor de la expresión 2, si expresión 1 es cierta, mientras que si es falsa toma el valor de la expresión 3.

- Sentencia **switch-case-default**. El formato es el siguiente:

```
switch (variable)
{
case etiq 1: sentencia 1
case etiq 2: sentencia 2
.....
default: sentencia 3
}
```

Ejemplo:

```
main()
{
float cm, pul, valor;
char unidad;
printf("Introduce número y unidad (p=pulg. o c=cm: ");
scanf("%f %c",&valor, &unidad);
switch(unidad)
{
case 'p': cm=valor*2.54;
pul=valor;
break;
case 'c': pul=valor/2.54;
cm=valor;
break;
default:
printf("No está puesta correctamente la unidad\n");
exit();
}
printf("\nLas %f pulgadas son %f centímetros\n", pul, cm);
}
```

6.3.- SENTENCIAS REPETITIVAS

- Bucle **for**. El formato es el siguiente:

```
for (inicio; condición; incremento)
{
sentencias
}
```

- Bucle **while**. El formato es el siguiente:

```
while ( condición)
{
sentencias
}
```

- Bucle **do-while**. El formato es el siguiente:

```
do
{
sentencias
}
while (condición);
```

Ejemplo:

```
main()
{
```

```

    long acum=1;
    int i, n;
    do
    {
        printf("Introduce número positivo: ");
        scanf("%d",&n);
    }
    while(n<0);
    for(i=n; i>1; --i)
        acum*=i;
    printf("\nEl factorial es %ld", acum);
}

```

6.4.- OTRAS SENTENCIAS DE CONTROL

- **break**: cuando el programa llega a esta sentencia, el flujo del mismo se desvía, liberándose del bucle en donde se encontraba, y dirigiéndose a ejecutar la siguiente sentencia del programa.
- **continue**: al igual que **break**, interrumpe el flujo del programa; sin embargo, en lugar de finalizar el bucle completo, hace que el resto de la iteración se evite, comenzando una nueva iteración.
- **goto**: hace que el control del programa salte a una nueva sentencia. Esta sentencia tiene dos partes: el **goto** y un nombre de etiqueta. Para que esta sentencia funcione debe haber otra sentencia que contenga la etiqueta.

Goto parte;

.....

.....

parte: printf("JLAB");

NOTA: No es recomendable su uso. En principio, no es necesario en ningún caso emplear **goto** en un programa C.

7.- FUNCIONES DE MANIPULACIÓN DE CARACTERES

7.1.- FUNCIÓN GETCHAR()

Esta función se usa para capturar caracteres uno a uno desde la entrada estándar activa (teclado, fichero de texto, etc.) y los entrega al programa. El formato es el siguiente:

```
char variable;  
variable=getchar();
```

El valor del carácter capturado se almacena en la variable. No se recomienda en entornos interactivos, ya que los caracteres tecleados son almacenados en el buffer de teclado y liberados cuando se pulsa un retorno de carro. Tiene eco de pantalla de los caracteres tecleados.

Otras funciones más interesantes en entornos interactivos son *getch()* y *getche()*. Ambas funciones no tienen almacenamiento en el buffer de teclado y se diferencian en que la primera no tiene eco de pantalla y la segunda sí.

7.2.- FUNCIÓN PUTCHAR()

Esta función se usa para mostrar caracteres uno a uno hacia la salida estándar activa, bien sea la pantalla o bien otros dispositivos como por ejemplo un fichero de texto en disco. Se usa en combinación con la función *getchar()*. El formato es el siguiente:

```
char variable;  
variable=getchar();  
putchar(variable);
```

Ejemplo:

```
#define STOP '*'  
main()  
{  
char ch;  
while ((ch=getchar())!=STOP)  
putchar(ch);  
}
```

8.- EL PREPROCESADOR C

Se encarga de leer el programa antes de que pase por la fase de compilación. Entre las tareas que realiza podemos citar: sustituir abreviaturas simbólicas por las direcciones que representan, buscar otros ficheros donde se encuentran escritos trozos de programa, tomar decisiones sobre qué partes enviar al compilador y que partes no, etc.

Las órdenes del preprocesador son sentencias cuyo primer carácter debe ser #, sin que haya ningún espacio en blanco al comienzo de la línea.

8.1.- #DEFINE

Puede aparecer en cualquier lugar del fichero fuente, y la definición es válida desde el lugar en que aparece el comando hasta el final del fichero.

Hasta ahora se ha usado para definir constantes simbólicas en los programas, pero posee nuevas aplicaciones, como se muestra a continuación.

```
/* Ejemplos sencillos del preprocesador */
#define DOS 2 /* Se pueden usar comentarios */
#define MSJ "Esto es un texto que \
continúa en otra línea."
/* Una barra atrás continúa la definición en otra línea */
#define CUATRO DOS*DOS
#define PX printf("X es %d\n", x);
#define FMT "X es %d\n"
main()
{
  int x=DOS;
  printf(FMT,x);
  printf("%s\n",MSJ);
  printf("DOS : MSJ\n");
}
```

Una macro es la definición de un símbolo de sustitución con argumentos. El proceso de cambio de la macro por su definición se llama expansión de la macro.

El parecido entre una macro y una función es solamente aparente, porque cuando una macro se llama cincuenta veces, el compilador genera otras tantas veces todas las instrucciones que la componen. De esta manera se evita el ciclo llamada-retorno de la función, cuya duración puede ser crítica en algunas circunstancias.

```
/* Macros con argumentos */
#define CUADRADO(x) x*x
#define PR(x) printf("X es %d\n", x)
main()
{
  int x=4, z;
  z=CUADRADO(x);
  PR(z);
  z=CUADRADO(2);
  PR(z);
}
```

8.2.- #INCLUDE

Cuando el preprocesador encuentra este comando, busca el fichero que atiende por el nombre que está situado detrás y lo incluye en el fichero actual.

```
#include <stdio.h> /* Busca en los directorios del sistema */
#include "fichero.h" /* Busca en el directorio que está trabajando */
#include "a:stdio.h" /* Busca en la unidad a */
```

Ejemplo:

```
#include <stdio.h>
main()
{
    char ch;
    while ((ch=getchar())!=EOF)
        putchar(ch);
}
```

El fichero *stdio.h* contiene generalmente las definiciones de *EOF*, *getchar()* y *putchar()*.

8.3.- OTROS COMANDOS: #UNDEF, #IF, #IFDEF, #IFNDEF, #ELSE Y #ENDIF

Estos comandos se emplean cuando se construyen programas de gran tamaño por medio de bloques bien diferenciados. Le permiten al programador dejar sin efecto definiciones anteriores y producir ficheros que pueden compilarse de distintas formas.

El comando *#undef* deja sin efecto la última definición de una macro.

```
#define GRANDE 3
#define ENORME 5
#undef GRANDE /*Ahora GRANDE no está definido */
#define ENORME 10 /*ENORME se redefine como 10 */
#undef ENORME /* ENORME vuelve al valor 5 */
#undef ENORME /*ENORME está ahora indefinido */
```

Los comandos restantes nos permiten realizar una compilación bajo determinadas condiciones.

```
#ifdef CONSTANTE
#include "fichero.h" /* Se realiza si CONSTANTE está definido */
#define VALOR 5
#else
#include "otro.h" /* Se realiza si no está definido */
#define VALOR 10
#endif
```

9.- ARRAYS Y PUNTEROS

9.1.- ARRAYS

Un array en lenguaje C, es una serie de variables que comparten el mismo nombre básico y se distinguen entre sí con una etiqueta numérica. También se pueden definir como una serie de posiciones de memoria contiguas que almacenan elementos de un mismo tipo.

Por ejemplo, la declaración: `int a[10]` nos indica que “a” es un array capaz de almacenar 10 valores de tipo entero. Para acceder a cada uno de los elementos del array se usa un índice. Así, en el ejemplo anterior, los distintos elementos del array serán:

`a[0], a[1], a[2], a[3],a[9]`

Los arrays pueden ser de cualquier tipo de datos.

Ejemplo:

```
/* Mostrar en pantalla los días de cada uno de los meses del año*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    static int dias[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    int ind=0;
    for(ind=0;ind<=11;++ind)
        printf("El mes %d tiene %d dias.\n", ind+1, dias[ind]);
    system("PAUSE");
    return 0;
}
```

9.2.- INICIALIZACIÓN DE LOS ARRAYS

Los arrays hay que declararlos, y en el supuesto que se quiera, inicializarlos. Así pues, las variables y arrays en general se pueden almacenar de las siguientes formas:

- **static**: las variables estáticas, son permanentes en su propia función o archivo. Son desconocidas fuera de sus funciones o archivos. Pueden ser de dos tipos: locales o globales.
 - static locales: se crea un almacenamiento permanente. Son conocidas solamente en el bloque en que se declaran. Son variables locales que retienen el valor entre llamadas de función. Por defecto se inicializan con cero.
 - static globales: se conocen solo por el archivo en el que se declaran.

Los arrays de tipo static pueden inicializarse.

`static int p[5]={6,34,56,78,9};`

- **extern**: son externas a las funciones o programas y pueden utilizarse en cualquier función una vez que han sido declaradas. Los arrays de tipo externos pueden inicializarse. Sirven para enlazar archivos múltiples.

<pre> Archivo 1 int x, y; char ch; main() { ----- ----- } fun21() { x=123; } </pre>	<pre> Archivo 2 extern int x, y; extern char ch; fun22() { x=y/10; } fun23() { y=10; ----- } </pre>
---	---

- **auto:** automáticas o locales, se declaran dentro de la propia función o programa. Solo son reconocidas por la función definida y por defecto si no se declaran de otra forma son todas automáticas. Los arrays automáticos no pueden inicializarse.

```

main()
{
int d[10], i=0;
for(i=0;i<10;++i)
scanf("%d",&d[i]);
}

```

- **register:** utilizan los registros de la CPU para su almacenamiento, siempre que se pueda. Su alcance es local. Son ideales para bucles de control. No se puede tener un número ilimitado de variables. Se aplican solo para datos de tipo *char* e *int*.

9.3.- ARRAYS MULTIDIMENSIONALES

El lenguaje C permite crear arrays multidimensionales y por tanto permite los arrays bidimensionales (filas y columnas), también conocidas como matrices.

Ejemplo de array bidimensional de 4 filas y 8 columnas:

```

int a[4][8];
a[0][0], a[0][1], a[0][2],.....a[0][7]
a[1][0], a[1][1], a[1][2],.....a[1][7]
a[2][0], a[2][1], a[2][2],.....a[2][7]
a[3][0], a[3][1], a[3][2],.....a[3][7]

```

Para acceder al elemento de la fila 2 y columna 3 haremos referencia al `a[1][2]`

Ejemplo: el programa supone que el profesor tiene 3 clases y un máximo de 30 alumnos por clase. Guardar el grado numérico de cada estudiante en cada clase.

```

#include<stdio.h>
#include<stdlib.h>
int grado[3][30];
main()
{
char ch;
for(;;)
{
do

```

```

    {
        borra_pantalla();
        puts("(I)ntroducir grados\n");
        puts("(L)istar grados\n");
        puts("(S)alir\n");
        printf("Seleccione opcion: ");
        ch=toupper(getchar());
    }
while(ch!='T' && ch!='L' && ch!='S');
switch(ch)
{
    case 'T': entrar_grados(); break;
    case 'L': listar_grados(); break;
    case 'S': exit(0);
}
}
}
entrar_grados()
{
    int i, t;
    borra_pantalla();
    for(t=0;t<3;t++)
    {
        printf("Clase # %d:\n", t+1);
        for(i=0;i<30;i++)
        {
            printf("Estudiante %d: ", i+1);
            scanf("%d",&grado[t][i]);
        }
    }
}
listar_grados()
{
    int i, t;    char c;
    borra_pantalla();
    for(t=0;t<3;t++)
    {
        printf("Clase # %d:\n", t+1);
        for(i=0;i<30;i++)
            printf("Estudiante %d es %d\n",i+1, grado[t][i]);
    }
    printf("\nPulse una tecla para continuar...");c=getch();
}
borra_pantalla()
{    system("cls");    }

```

9.4.- PUNTEROS

Un puntero es una variable que contiene una dirección, que es a su vez la posición de otra variable de memoria, es decir, apunta a una posición de memoria.

Un puntero es una variable que contiene la dirección de otra variable. Los punteros contienen direcciones de memoria.

La principal ventaja de usar punteros es la mayor velocidad de tratamiento de los datos y sobre todo para manejar arrays y tiras de caracteres.

Se utilizan dos operadores especiales:

* nos proporciona el contenido de una variable.

& nos proporciona la dirección de una variable.

Ejemplo:

Si p es un puntero

$p = \&x$ ----> guarda en p la dirección de la variable x , por lo que podemos decir que p apunta a la variable x .

$y = *p$ ----> asigna a la variable y el valor de donde apunta el puntero p .

Lo visto anteriormente es lo mismo que:

$y = x$ ----> asignación a y del valor de x .

Los punteros antes de ser usados han de ser declarados de la siguiente forma:

*tipo de dato al que apuntan *nombre del puntero*
`int *p;`

9.5.- ARRAYS CON PUNTEROS

Existe una fuerte relación entre punteros y arrays, ya que para acceder a cada uno de los elementos del array se puede hacer con punteros aumentando de esta forma la velocidad.

Para acceder a los elementos de un array existen dos formas:

- Indexando con un índice normal para acceder a cada uno de los elementos.
- Con punteros con lo que se consigue una mayor velocidad de acceso a los elementos del array.

Ejemplo:

`int *pu` ----> declaración de un puntero a entero.

`int a[10]` ----> declaración de un array con 10 elementos.

`pu = &a[0]` ----> el puntero apunta al primer elemento.

`x = *pu` ----> la x toma el valor del primer elemento.

`y = *(pu+1)` ----> la y toma el valor del segundo elemento.

`z = *(pu+2)` ----> la z toma el valor del tercer elemento.

Hay que hacer notar que $pu = a$ es lo mismo que $pu = \&a[0]$.

Programa ejemplo: inicializar un array entero con 10 elementos y mostrarlos en pantalla usando punteros.

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    static int pila[10]={5,6,4,3,2,1,8,9,7,0};
    int *pu, i;
    pu = &pila[0];
    for(i=0;i<10;i++)
        printf("%d ", *(pu+i));
    system("PAUSE");
    return 0;
}
```

Es lo mismo hacer:

`printf("%d ",pila[5]);` que `printf("%d ",*(pu+5));` y en general
`pila[i]==*(pu+i)`

10.- ARRAYS DE TIRAS DE CARACTERES

Una tira de caracteres es un array tipo *char* terminado con un carácter nulo “\0” y todas las operaciones con cadenas se realizan indexando con punteros que apuntan a cada uno de los caracteres de la cadena o tira.

Ejemplo:

```
static char a[10]="CADENAS";
Char *p;
p=&a[0];    ----> el puntero apunta a la C
p+1        ----> el puntero apunta a la A
p+2        ----> el puntero apunta a la D
etc.
```

El compilador es el encargado de añadir el carácter nulo al final de la tira sin preocuparse de ello el programador.

10.1.- GETS() Y PUTS()

La función *gets()* es utilizada para capturar tiras de caracteres desde el teclado y asignarlas a los arrays de caracteres. La función captura caracteres hasta que se pulsa el retorno de carro. El retorno de carro no forma parte de la cadena capturada, en su lugar se coloca el carácter nulo ‘\0’.

Ejemplo: capturar un nombre y mostrarlo en pantalla.

```
main()
{
char nombre[35];
printf("Introduzca nombre: "); gets(nombre);
printf("Hola %s ", nombre);
}
```

También la función *cgets()* lee una cadena desde el teclado. El formato es el siguiente:

```
char nombre[30]
cgets(nombre)
nombre[0], podría contener la máxima longitud de la cadena a leer.
nombre[1], contiene el número de caracteres introducidos.
nombre[2], contiene donde realmente comienza la cadena.
```

La función devuelve la dirección donde comienza la cadena. Es decir, devuelve *&nombre[2]*.

Ejemplo: capturar un nombre y mostrarlo en pantalla.

```
main()
{
char nombre[20];
char *p;
printf("Introduzca nombre: ");p=cgets(nombre);
printf("\n El nombre es %s\n", p);
}
```

La función *puts()* es utilizada para mostrar tiras de caracteres por la pantalla.

Ejemplo: mostrar un nombre capturado por *gets()*.

```
main()
{
char nombre[35];
```

```

printf("Introduzca nombre: ");
gets(nombre);
puts("Hola ");
puts( nombre);
}

```

También al función *cputs()* sirve para mostrar cadenas en la ventana de texto activa.

10.2.- ARRAYS MULTIDIMENSIONALES CON TIRAS DE CARACTERES

Para tratar una tira de caracteres, solo es necesario usar un array tipo *char* unidimensional. Para tratar varias tiras de caracteres a la vez, es necesario utilizar arrays tipo *char* de dos dimensiones en las que:

```
char nombre[10][30];
```

filas: las 10 diferentes cadenas o nombres.

columnas: los 30 posibles caracteres de cada cadena.

En el ejemplo anterior necesitamos un total de 10 punteros para cada una de las cadenas o nombres.

Ejemplo: introducir varios nombres por el teclado y posteriormente sacarlos en pantalla.

```

main()
{
char nombres[3][20]; int i=0;
for(i=0;i<3;++i)
{ printf("Nombre %d: ", i+1); gets(nombres[i]); }
for(i=0;i<3;++i)
printf("El nombre %d es %s\n", i+1, nombres[i]);
}

```

Un empleo muy usual de un array de punteros a tiras de caracteres, es el de guardar los punteros a los mensajes de error.

Ejemplo: crear una función que sacará un mensaje dando su número de código.

```

void error(int num)
{
static char*err[]={
"No se puede abrir el fichero\n",
"Error de lectura\n",
"Error de escritura\n",
"Fallo del sistema\n",
"Fallo de la unidad\n"
};
printf(" %s", err[num]);
}

```

10.3.- FUNCIONES RELACIONADAS CON TIRAS DE CARACTERES

El lenguaje C incorpora una serie de funciones para la manipulación de tiras de caracteres.

- ***strlen()***: devuelve la longitud de una tira de caracteres.

```

int largo; char *cadena;
-----
largo=strlen(cadena);

```

En la variable *largo* dispondremos del número de caracteres que componen la cadena.

- **strcat()**: concatena dos cadenas.

```
char *p="Hola", nombre[10], *final;
```

```
-----
```

```
final=strcat(nombre,p);
```

En *nombre* se almacena el resultado de la concatenación y *final* apunta al primer carácter de *nombre*.

- **strcmp()**: compara dos cadenas.

```
int res;
```

```
char *p, *q;
```

```
-----
```

```
res=strcmp(p,q);
```

Compara la cadena *p* y *q* y nos devuelve tres posibles valores:

0: si *p* y *q* son iguales.

-1: si *p* es menor que *q*.

Valor positivo: si *p* es mayor que *q*.

La comparación se establece según el código ASCII de los caracteres que componen *p* y *q*.

- **strcpy()**: copia una cadena en otra.

```
char *final, *una, *otra;
```

```
-----
```

```
final=strcpy(otra,una);
```

Nos devuelve un puntero de tipo *char* que apunta a la primera posición de *otra*.

- **strupr()**: convierte la cadena a mayúsculas.

```
char *p, cadena[30];
```

```
-----
```

```
p=strupr(cadena);
```

Nos devuelve un puntero de tipo *char* que apunta al primer carácter de la cadena en mayúsculas.

- **strlwr()**: convierte la cadena a minúsculas.

```
char *p, cadena[30];
```

```
-----
```

```
p=strlwr(cadena);
```

Nos devuelve un puntero de tipo *char* que apunta al primer carácter de la cadena en minúsculas.

11.- FUNCIONES

Las funciones son los bloques constructores del C y el lugar donde se centra toda la actividad del programa. Son una de las características más importantes del C.

11.1.- FORMA GENERAL DE UNA FUNCIÓN

La forma general de una función es:

```
tipo nombre_función(lista_parámetros)
{
cuerpo_de_la_función
}
```

Siendo *tipo*, el tipo de dato devuelto por la función. Si no se especifica ningún tipo, el compilador interpreta que se devuelve un entero. *Nombre_función* representa el nombre dado a la función. Y *lista_parámetros* son los argumentos que toma la función.

Hay dos formatos a la hora de definir una función:

- Formato clásico.

```
char nombre_función(a,b,c)
int a;
float b;
char c;
{
cuerpo_de_función;
}
```

- Formato moderno.

```
char nombre_función(int a, float b, char c)
{
cuerpo_de_función;
}
```

11.2.- LLAMADA POR VALOR O POR REFERENCIA

Si una función usa argumentos (parámetros), estos se comportan como variables locales de la función.

Se pueden pasar argumentos a las funciones de dos formas: por valor o por referencia.

- Por valor: Este método copia el valor de un argumento en el parámetro formal de la función. De esta forma, los cambios en los parámetros de la función no afectan a las variables que se usan en la llamada.

```
main()
{
int t=10;
printf(“%d %d”, cuad(t), t);
}
int cuad(int x)
{
x=x*x;
return (x);
}
```

- Por referencia: Este método copia la dirección del argumento en el parámetro de la función. De esta forma, los cambios hechos a los parámetros afectan a la variable usada en la llamada.

```
main()
{
    int x=10, y=20;
    printf("X=%d e Y=%d",x, y);
    inter(&x,&y);
    printf("X=%d e Y=%d",x,y);
}
void inter(int *p, int *q)
{
    int temp;
    temp=*p; *p=*q; *q=temp;
}
```

Cuando se usa un array como argumento de una función, solo se pasa la dirección del array.

Cuando se llama a una función con un nombre de array, se pasa a la función un puntero al primer elemento del array.

```
main()
{
    int i, t[10];
    for(i=0;i<10;++i) t[i]=i;
    mostrar(t);
}
void mostrar(int *num)
{
    int i;
    for(i=0;i<10;++i)
        printf("%d",num[i]);
}
```

11.3.- SENTENCIA RETURN

La sentencia *return* tiene dos usos importantes. Primero, fuerza la salida de la función donde se encuentra. O sea, hace que la ejecución del programa vuelva al código que llamó a la función. En segundo lugar, se puede utilizar para devolver un valor.

Se vuelve de una función por: haber llegado al final del cuerpo de la función o porque hemos encontrado una sentencia *return*.

Todas las funciones, excepto las de tipo *void*, devuelven un valor.

11.4.- PROTOTIPOS DE FUNCIONES

Las normas ANSI recomiendan el uso del prototipado de funciones. Estos prototipos informan al compilador que se va a usar una función que tiene cierto nombre, que toma unos ciertos parámetros y devuelve un cierto valor.

La forma general de un prototipo de función es:

tipo nombre_función(tipo para1, tipo para2, tipo paraN);

El uso de los nombres de los parámetros es opcional. Sin embargo, permiten que el compilador identifique cualquier discordancia de tipo por su nombre, por lo que es bueno incluirlos.

```

void main(void)
{
int x;
void cuad(int *);
x=10;
cuad(x);
}
void cuad(int *i)
{
*i=*i * *i;
}

```

Antes de usar cualquier función que devuelve un dato *float* o *void*, se debe declarar su prototipo. De lo contrario se produce un error.

11.5.- ARGUMENTOS DEL MAIN()

Algunas veces es útil pasar información al programa cuando se ejecuta. El método general es pasar información a la función *main()* mediante el uso de argumentos en la línea de órdenes. Un argumento de la línea de órdenes es la información que sigue al nombre del programa en la línea de órdenes del sistema operativo.

Hay dos argumentos ya incorporados, *argc* y *argv* que se utilizan para recibir los argumentos de la línea de órdenes.

El parámetro *argc* contiene el número de argumentos de la línea de órdenes y es un entero. Como mínimo vale 1 ya que el nombre del programa es el primer argumento.

El parámetro *argv* es un puntero a un array de punteros a caracteres. Cada elemento del array apunta a un argumento de la línea de órdenes. Todos los argumentos de la línea de órdenes son cadenas.

Ejemplo:

```

void main(it argc, char *argv[])
{
if(argc!=2)
{
printf("Ha olvidado su nombre\n");
exit(1);
}
printf("Hola %s",argv[1]);
}

```

11.6.- RECURSIVIDAD

En C, las funciones pueden llamarse a sí mismas. Si una expresión en el cuerpo de una función llama a la propia función, se dice que ésta es recursiva. La recursividad es el proceso de definir algo en términos de sí mismo, a veces se llama definición circular.

Veamos un ejemplo de una definición de una función en dos modalidades, recursiva y no recursiva.

<pre> long fact(int n) { int resp; if(n==1) return (1); resp=fact(n-1)*n; return (resp); } </pre>	<pre> long fact(int n) { int t, resp=1; for(t=1;t<n;t++) resp=resp*t; return(resp); } </pre>
---	--

11.7.- DECLARACIÓN DE LISTAS DE PARÁMETROS DE LONGITUD VARIABLE

En C, se puede especificar una función con un número variable de parámetros y tipos. El ejemplo más común es *printf()*. Para indicar al compilador que se va a pasar a una función un número y unos tipos desconocidos de parámetros, se debe terminar la declaración de sus parámetros con tres puntos.

func(int a, int b, ...);

La declaración anterior indica que *func()* tiene dos parámetros enteros y un número desconocido de parámetros siguientes.

Cualquier función que tenga un número de parámetros variable, debe tener al menos un argumento concreto.

Las macros *va_arg()*, *va_start()* y *v_end()* (definidas en el fichero cabecera *stdarg.h*) trabajan juntas para permitir pasar un número variable de argumentos a la función.

El procedimiento general para crear una función que pueda tomar un número variable de argumentos es el siguiente:

1. La función debe tener al menos un parámetro previo a la lista de parámetros variables, pero puede tener más.
2. El parámetro conocido más a la derecha se llama último parámetro.
3. Antes de que cualquiera de los parámetros de longitud variable puedan ser utilizados, el puntero del argumento debe estar inicializado por medio de una llamada a *va_start()*.
4. Después los parámetros se devuelven a través de llamadas a *va_arg()*.
5. Finalmente antes de volver se debe hacer una llamada a *va_end()* para asegurar que la pila será reconstruida apropiadamente.

Ejemplo:

```
#include<stdio.h>
#include<stdlib.h>
#include<stdarg.h>
main()
{
double d; double suma_serie(int, ...);
d=suma_serie(4, 0.5, 0.125, 0.0625, 0.03125); /* el 4 indica el número */
printf("La suma de la serie es %f\n",d);
system ("PAUSE"); return (0);
}
double suma_serie(int num, ...)
{
double suma=0.0, t;
va_list argptr; /* va_list es el tipo de puntero pasado a una función que acepta
una lista de argumentos */
va_start(argptr, num); /* inicializa argptr */
for(;num;num--) /* suma la serie */
{
t=va_arg(argptr, double); /* acceso a la lista de argumentos */
suma+=t; /* devuelve el argumento apuntado */
}
va_end(argptr); /* acceso al ultimo argumento */
return (suma);
}
```

11.8.- PUNTEROS A FUNCIONES

La dirección de la función es el punto de entrada de la función. Por ello, se puede usar un puntero a función para llamar a una función. Esto también permite pasar funciones como argumentos a otras funciones.

La dirección de la función se obtiene utilizando el nombre de la función sin paréntesis ni argumentos.

Ejemplo:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
void (*p) (int i); /* Puntero a función */
void lineav(), lineah(); /* Prototipos */
p=lineav; /* Asigna a p la dirección de lineav() */
(*p) (4); /* Llama a lineav() con un argumento 4 */
p=lineah; /* Asigna a p la dirección de lineah() */
(*p) (3); /* Llama a lineah() con un argumento 3 */
system ("PAUSE"); return (0);
}
void lineav(int i)
{
for(;i;i--) printf("-"); printf("\n");
}
void lineah(int i)
{
for(;i;i--) printf("\n");
}
```

12.- ESTRUCTURAS Y UNIONES

El lenguaje C permite crear tipos de datos nuevos de cinco formas. La primera, combinando muchas variables en una variable conglomerada llamada *estructura*. La segunda es, los *campos de bits* que permite acceder fácilmente a los bits dentro de una palabra. La tercera es, utilizando una *unión* para permitir que muchas variables compartan una misma parte de memoria. La cuarta es la *enumeración* que es una lista de símbolos. La quinta y última forma es, utilizar *typedef* que define un nuevo nombre para un tipo ya existente.

12.1.- ESTRUCTURAS

Una estructura es una colección de una o más variables, posiblemente de tipos diferentes, agrupadas bajo un mismo nombre de variable para una cuestión de manejo más eficiente. Una estructura organiza tipos de datos distintos, de manera que se puede hacer referencia a ellos como si fuera una sola unidad.

La definición de estructura forma una plantilla que puede utilizarse para crear variables de estructura. Las variables que componen la estructura se llaman elementos de la estructura.

```
struct nombre {
    def. var. 1;
    def. var. 2;
    -----
    -----
};
```

Una estructura solo se puede inicializar si es estática o externa. Para crear una variable de estructura se sigue el siguiente formato:

```
struct nombre_de_la_estructura nombre_de_la_variable;
```

12.2.- ACCESO A UN MIEMBRO DE LA ESTRUCTURA

Para acceder a un elemento o miembro de la estructura se referencia con el nombre de la estructura seguido por un punto y el nombre del elemento.

```
nombre_de_la_estructura.nombre_del_elemento
```

Ejemplos:

```
cliente.dp=31010;          scanf("%d", &individuo.edad);
printf("Código: %d", cliente.dp);
```

12.3.- ARRAYS DE ESTRUCTURAS

Una de las aplicaciones más comunes de las estructuras son los arrays de estructuras. Para declarar un array de estructuras, primero hay que definir una estructura y luego declarar una variable array de ese tipo.

Ejemplo:

```
struct dir {
    char nombre[30];
    char calle[35];
    char ciudad[20];
    char prov[15];
    int dp;
};
struct dir lista[100];
```

Para acceder a un elemento o miembro del array de estructura se referencia:

variable[número].elemento

Así por ejemplo, para asignar un valor al elemento *dp* del array número 5 del ejemplo anterior:

```
lista[5].dp=31010;
printf("Código: %d", lista[5].dp);
```

12.4.- ESTRUCTURAS ANIDADAS

Un elemento de una estructura puede ser a su vez otra estructura. Así pues, podemos declarar:

```
struct persona {
char nombre[30];
char dirección[30];
int edad;
};

struct empleado {
long dni;
long numss;
struct persona individuo;
};
```

Definimos la variable:

```
struct empleado emp;
```

Entonces para acceder a un elemento de una estructura anidada sería:

```
emp.individuo.nombre;
```

sencillamente se utiliza el operador "." dos veces.

12.5.- PUNTEROS A ESTRUCTURAS

Existen punteros a estructuras. Los motivos de su uso son tres fundamentalmente:

1. Son más sencillos de manejar que las propias estructuras.
2. Una estructura no puede pasarse como argumento a una función pero el puntero a estructura sí.
3. Existen muchos datos elaborados que son estructuras que contienen punteros a otras estructuras.

Se declaran de la siguiente forma:

```
struct nombre_patrón *nombre_puntero_a_estructura
```

Ejemplo:

```
struct persona *ser;
```

Para acceder a un elemento de un puntero a estructura se realiza mediante el operador flecha "->".

```
nombre_puntero_a_estructura -> nombre_elemento
```

Ejemplo:

```
mes=(ser->edad)*12;
```

12.6.- CAMPOS DE BITS

En el lenguaje C, se puede acceder a un bit individual dentro de un byte. Esto puede ser útil por diversas razones:

- Si el almacenamiento es limitado, se pueden almacenar varias variables lógicas (verdadero/falso) en un byte.
- Hay dispositivos que transmiten la información codificada en los bits dentro de bytes.
- Ciertas rutinas de cifrado necesitan acceder a los bits dentro de los bytes.

Todo esto se puede hacer utilizando los bytes y los operadores a nivel de bits, pero un campo de bits relaja la estructura.

Un campo de bits es un tipo especial de elemento de estructura que define su tamaño en bits. El formato de definición es el siguiente:

```
struct etiqueta {
    tipo nombre1: longitud;
    tipo nombre2: longitud;
    -----
    tipo nombreN: longitud;
} lista_de_variables;
```

Cada campo de bits debe declararse como *int*, *unsigned* o *signed*. Los campos de longitud deben declararse como *unsigned*.

Los campos de bits se utilizan frecuentemente cuando se analiza la entrada de un dispositivo hardware. Por ejemplo: el puerto serie de comunicaciones.

<u>BIT</u>	<u>SIGNIFICADO CUANDO ESTÁ ACTIVO</u>
0	cambio en la línea listo_para_enviar
1	cambio en datos_listos
2	detección de final
3	cambio en la línea de recepción
4	listo para enviar
5	datos listos
6	llamada telefónica
7	señal recibida

Se puede representar esta información en un byte de estado mediante el siguiente campo de bits:

```
struct tipo_estado {
    unsigned delta_cts: 1;
    unsigned delta_dsr: 1;
    unsigned tr_final: 1;
    unsigned delta_rec: 1;
    unsigned cts: 1;
    unsigned dsr: 1;
    unsigned ring: 1;
    unsigned linea: 1;
} estado;
```

Se puede usar una rutina como la que sigue para permitir que un programa determine si puede enviar o recibir datos.

```
estado=obt_estado_puerto();
if(estado.cts) printf("Listo para enviar");
if(estado.dsr) printf("Datos listos");
```

Para acceder a los elementos de un campo de bits se utiliza el operador "." (punto). Si se referencia la estructura de campo de bits a través de un puntero, se usará el operador "->" (flecha).

No es necesario nombrar cada campo de bits. Se puede llegar al bit que se desea pasando por alto los que no se usan. Por ejemplo, si solo interesan los bits *cts* y *dsr*:

```
struct tipo_estado {
    unsigned : 4;
    unsigned cts: 1;
    unsigned dsr: 1;
} estado;
```

Por último, se pueden mezclar elementos de estructura normales con elementos de campo de bits.

```
struct emp {
    struct dir dirección;
    float paga;
    unsigned baja: 1; /* De baja o activo */
    unsigned por_horas: 1; /* Por horas o en nómina */
    unsigned deducciones: 3; /* Deducciones de IRPF */
};
```

12.7.- UNIONES

Una unión es un instrumento que permite almacenar tipo de datos diferentes en el mismo espacio de memoria. Un modelo de unión podría ser:

```
unión etiqueta {
    def. 1;
    def. 2;
    -----
    -----
    def. n;
};
```

La definición de variables de una unión se realiza según el siguiente formato:

```
unión etiqueta variable;
```

Para acceder a los diferentes elementos de una unión se utiliza el operador “.” (punto). También se puede emplear el operador -> (flecha) con uniones, de igual manera a como se hace con estructuras.

12.8.- ENUMERACIONES

El estándar ANSI incluye la enumeración como una extensión del lenguaje C. Una enumeración es un conjunto de constantes enteras con nombre, que especifica todos los valores válidos que una variable de este tipo puede tener.

Las enumeraciones se definen de forma parecida a las estructuras. La forma general es:

```
enum etiqueta { lista_de_enumeraciones };
```

Por ejemplo:

```
enum moneda { penique, niquel, diez_centavos, cuarto, medio_dólar, dólar };
```

Para declarar variables enumeraciones, es similar a las variables estructuras.

```
enum moneda dinero;
```

La clave para entender la enumeración es que cada uno de los símbolos corresponde a un valor entero. Cada símbolo recibe un valor que es mayor en uno que el símbolo que le precede. El valor del primer símbolo de la enumeración es cero.

Se puede especificar el valor de cada uno o más símbolos utilizando un inicializador. Esto se hace siguiendo el símbolo con un signo igual y un valor entero. A los símbolos que siguen a uno inicializado, se les asignan valores mayores que el valor previo de inicialización.

Por ejemplo:

```
enum moneda { penique, niquel, diez_centavos, cuarto=100,
              medio_dólar, dólar };
```

Ahora los valores de los símbolos son:

```
penique 0    niquel 1    diez_centavos 2
cuarto 100   medio_dólar 101   dólar 102
```

12.9.- ABREVIACIÓN DE LOS TIPOS DE DATOS

La palabra clave *typedef* permite crear un tipo de dato con un nombre arbitrario otorgado por el usuario.

Realmente no se crea una nueva clase de datos, sino que se define un nuevo nombre para un tipo existente.

El alcance depende de su definición:

- Definición interna: alcance a la propia función.
- Definición externa: alcance global.

Por convenio se suelen emplear letras mayúsculas para las definiciones.

Ejemplos:

```
typedef flota REAL
typedef char *CHAR
typedef struct COMPLEJO {
                                float real;
                                float imag;
                                };
```

13.- ESTRUCTURAS Y UNIONES

13.1.- FICHEROS

Las funciones que componen el sistema de entrada/salida de C pueden ser agrupadas en tres categorías:

- E/S por consola
- E/S de archivo sin buffer al estilo UNIX (nivel bajo)
- E/S de archivo usando buffer (nivel alto)

Las E/S por consola ya han sido tratadas con anterioridad y las E/S de archivos sin buffer intermedio al estilo UNIX no forman parte del estándar ANSI, por lo que dedicaremos el estudio al tema de E/S a ficheros de disco de alto nivel, es decir, usando un buffer intermedio para la gestión de datos.

Para nosotros, un fichero es una porción de almacenamiento, generalmente en disco, con un nombre.

Para el compilador C, un fichero es una estructura que se reconoce por el nombre FILE. Luego, para definir un fichero, nuestro programa debe incorporar la sentencia:

```
FILE *nombre_de_la_variable_que_designa_al_fichero
```

Como vemos se declara como un puntero a estructura de fichero. Esta estructura está definida en el fichero cabecera `<stdio.h>` por lo que será necesario incluir al principio la sentencia:

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

13.2.- FUNCIONES DE ACCESO A FICHEROS

Las funciones más comunes relacionadas con el tratamiento de ficheros con buffer son las siguientes. Estas funciones requieren que el archivo de cabecera antes mencionado `<stdio.h>` sea incluido en todos los programas que vayan a ser usadas:

<u>Nombre</u>	<u>Función</u>
<code>fopen()</code>	Abrir un archivo
<code>fclose()</code>	Cerrar un archivo
<code>putc()</code>	Escribir un carácter en archivo
<code>getc()</code>	Leer un carácter de archivo
<code>putw()</code>	Escribir un entero en archivo
<code>getw()</code>	Leer un entero de archivo
<code>fseek()</code>	Posiciona puntero en archivo
<code>ftell()</code>	Da la posición del puntero
<code>fwrite()</code>	Escribe datos en archivo
<code>fread()</code>	Lee datos de archivo
<code>fprintf()</code>	Escribe con formato en archivo
<code>fscanf()</code>	Lee de disco con formato
<code>feof()</code>	Detecta final de archivo
<code>ferror()</code>	Devuelve error de archivo
<code>rewind()</code>	Coloca puntero al principio de archivo
<code>fgets()</code>	Lectura de cadenas de archivo
<code>fputs()</code>	Escritura de cadenas en archivo

- `fopen()`: es usada para abrir un fichero y poder acceder a sus datos.

```
FILE *puntero
Puntero=fopen(fichero.modos)
```

puntero: declara puntero a fichero
 fichero: nombre del fichero a abrir
 modo: modo de apertura

Diferentes modos de apertura:

r: abrir archivo de texto para lectura
 w: crear archivo de texto para escritura
 a: añadir en archivo de texto
 rb: abrir archivo binario para lectura
 wb: crear archivo binario para escritura
 ab: añadir en archivo binario
 r+: abrir archivo de texto para lectura/escritura
 w+: crear archivo de texto para lectura/escritura
 a+: añadir en archivo de texto para lectura/escritura
 rb+: abrir archivo binario para lectura/escritura
 wb+: crear archivo binario para lectura/escritura
 ab+: añadir en archivo binario para lectura/escritura

La diferencia entre modo “texto” y “binario” radica en que en el primero la secuencia de un retorno de carro (CR) y salto de línea (LF) se almacenan en el fichero como tal, sin embargo en el modo binario no se almacenan.

Si se va a utilizar el modo de escritura (w), los datos empezarán a escribirse desde el principio del fichero borrando cualquier dato que en él hubiese, por lo tanto antes de utilizar esta operación es conveniente determinar el tipo de error, si se ha producido el error al abrirlo, si está protegido contra escritura, disco lleno, etc. En general, cuando al intentar abrirlo devuelve la función un valor (NULL) cero quiere decir que la operación ha sido errores. Nótese que la constante NULL está definida en el fichero `<stdio.h>`. Por lo tanto:

```
if((puntero=fopen("fichero","modo"))==NULL)
{
puts("No se puede abrir el fichero");
exit(0);
}
```

La operación realizada anteriormente es necesaria también para evitar problemas tales como:

- Fichero no existente y se quiere leer.
 - Nombre del fichero erróneo.
 - Algún otro tipo de error.
 - Etc.
- *fclose()*: se usa para cerrar un fichero que se ha abierto mediante *fopen()*. Siempre hay que cerrar todos los ficheros abiertos antes de terminar un programa.

close(puntero)

El puntero le dice a *fclose()* el nombre del fichero que se va a cerrar. Otra forma de cerrar cualquier fichero que estuviese abierto es salirse del programa con la función *exit()*.

- *putc()*: se usa para escribir caracteres en un fichero de disco que se ha abierto utilizando la función *fopen()* en modo escritura. Siempre escribe byte a byte.

putc(carácter,puntero)

El puntero le dice a *putc()* en que fichero se va a escribir.

- *getc()*: se usa para leer caracteres en un fichero de disco que se ha abierto utilizando la función *fopen()* en modo lectura. Siempre lee byte a byte.

carácter=getc(puntero)

El puntero le dice a *getc()* en que fichero se va a leer. Cuando se lee un fichero, para detectar el final del mismo se utiliza el carácter EOF.

- *putw()*: se usa para escribir datos enteros en un fichero de disco que se ha abierto utilizando la función *fopen()* en modo escritura. Siempre escribe palabra a palabra.

putw(int,puntero)

- *getw()*: se usa para leer datos enteros en un fichero de disco que se ha abierto utilizando la función *fopen()* en modo lectura. Siempre lee palabra a palabra.

int=getw(puntero)

- *feof()*: se usa para detectar final de archivo. Devuelve verdad (distinto de cero) si se ha alcanzado el final del archivo, en cualquier otro caso devuelve un cero.

int=feof(puntero)

- *ferror()*: se usa para detectar error durante la última operación sobre el fichero. Devuelve verdad (distinto de cero) si se ha producido un error, en cualquier otro caso devuelve un cero.

int=ferror(puntero)

- *rewind()*: se usa para localizar el puntero del archivo al principio del mismo.

rewind(puntero)

Veamos a continuación cuatro ejemplos sobre el manejo de estas funciones.

```

/* Ejemplo de escritura de caracteres en fichero secuencial */
#include<stdio.h>
#include<stdlib.h>
main()
{
    char c;
    FILE *fp;
    if((fp=fopen("fichero","w"))==NULL)
        { printf("ERROR"); exit (0); }
    while(c!=EOF)
        { c=getchar(); putc(c,fp); }
    fclose(fp);
    system("PAUSE");
    return 0;
}

```

```
/* Ejemplo de lectura en fichero secuencial */
#include<stdio.h>
#include<stdlib.h>
main()
{
    char c;
    FILE *fp;
    if((fp=fopen("fichero","r"))==NULL)
        { printf("ERROR"); exit (0); }
    while(!feof(fp))
        { c=getc(fp); putchar(c); }
    fclose(fp);
    system("PAUSE");
    return 0;
}
```

```
/* Ejemplo de adición en fichero secuencial */
#include<stdio.h>
#include<stdlib.h>
main()
{
    char c;
    FILE *fp;
    if((fp=fopen("fichero","a"))==NULL)
        { printf("ERROR"); exit (0); }
    while(c!=EOF)
        { c=getchar(); putc(c,fp); }
    fclose(fp);
    system("PAUSE");
    return 0;
}
```

```

/* Ejemplo de copia de un fichero de cualquier tipo */
#include<stdio.h>
#include<stdlib.h>
main(int argc, char *argv[])
{
    char c;
    FILE *fe, *fs;
    if(argc!=3)
        { printf("Ha olvidado introducir nombre de archivo");
          getch(); exit (0);
        }
    if((fe=fopen(argv[1], "rb"))==NULL)
        { printf("ERROR"); exit (0); }
    if((fs=fopen(argv[2], "wb"))==NULL)
        { printf("ERROR"); exit (0); }
    while(!feof(fe)) putc(getc(fe),fs);
    fclose(fe);
    fclose(fs);
    system("PAUSE");
    return 0;
}

```

13.3.- FUNCIONES FPRINTF(), FSCANF(), FPUTS(), FGETS()

Estas funciones son exactamente iguales que las *printf()*, *scanf()*, *gets()* y *puts()* pero con la diferencia que sirven para acceder a la lectura y escritura de los ficheros. Se usan para entrada y salida de líneas formateadas.

Las dos primeras tienen como argumento primero al puntero del fichero y las dos últimas tienen como argumento último al puntero del fichero.

Modo de acceso secuencial.

<i>fprintf()</i>	<i>fprintf(puntero, "control", variables)</i>
<i>fscanf()</i>	<i>fscanf(puntero, "control", &variable)</i>
<i>fgets()</i>	<i>fgets(cadena, longitud, puntero)</i>
<i>fputs()</i>	<i>fputs(cadena, puntero)</i>

Veamos unos ejemplos de su uso.

```

/* Ejemplo de fprintf() y fscanf() */
#include<stdio.h>
#include<stdlib.h>
main()
{
    int edad;
    FILE *fi;
    fi=fopen("fichero", "r");
    fscanf(fi, "%d", &edad);
    fclose(fi);
    fi=fopen("datos", "a");
    fprintf(fi, "Tiene %d años.\n", edad);
    fclose(fi);
    system("PAUSE");
    return 0;
}

```

```
/* Ejemplo de fgets() */
#include<stdio.h>
#include<stdlib.h>
main()
{
    FILE *fi;
    char *tira[80];
    fi=fopen("fichero","r");
    while(fgets(tira,80,fi)!=NULL)
        puts(tira);
    system("PAUSE");
    return 0;
}
```

13.4.- FUNCIONES FREAD(), FWRITE(), FSEEK(), FTELLS()

Estas funciones permiten la lectura y escritura de datos en archivos, bloque a bloque; es decir, tratan los datos agrupados en registros de longitud fija. Están orientadas al tratamiento de los ficheros con organización directa.

Modo de acceso directo.

<i>fread()</i>	<i>fread(buffer,num_bytes,cont,puntero)</i>
<i>fwrite()</i>	<i>fwrite(buffer,num_bytes,cont,puntero)</i>
<i>fseek()</i>	<i>fseek(puntero,desplazamiento,origen)</i>
<i>ftell()</i>	<i>ftell(puntero)</i>

buffer: es un puntero a la zona de memoria en donde están o recibirán los datos del archivo.

num_bytes: número de bytes que se leen o escriben.

cont: cuántos elementos de *num_bytes* se leerán o escribirán.

puntero: puntero a fichero.

La función *fseek()* se usa para posicionar el puntero dentro del archivo. En donde cada uno de los parámetros son:

puntero: puntero a fichero.

desplazamiento: el número de bytes a desplazar el puntero.

origen: referencia para hacer el desplazamiento, siendo cada uno de los valores siguientes:

- 0 desde el principio del archivo
- 1 desde la posición actual
- 2 desde el final del archivo hacia atrás

La función *ftell()* da la posición actual del fichero.

A continuación se expone un ejemplo de gestión de ficheros directos usando las funciones descritas anteriormente.

```
/* Ejemplo de lectura en fichero aleatorio */
#include<stdio.h>
#include<stdlib.h>
main()
{
    long reg=0;
    FILE *fi;
    int mode=0, dato=99;
    char c, ap[20], di[20], lo[12], te[9];
    if((fi=fopen("d:directo","r"))==NULL) exit(0);
    while(dato!=0)
    {
        printf("Numero dato: "); scanf("%d",&dato);
        if(dato==0) exit(0);
        reg=dato*60-59;
        fseek(fi,reg,mode); printf("%ld", ftell(fi));
        fscanf(fi,"%20s %20s ", ap, di);
        fscanf(fi,"%12s %9s ", lo, te);
        printf("Apellidos: %20s \n", ap);
        printf("Direccion: %20s \n", di);
        printf("Localidad: %12s \n", lo);
        printf("Telefono: %9s \n", te);
        getch();
    }
    system("PAUSE");
    return 0;
}
```

14.- PRÁCTICAS DE C

1. Realizar un programa que guarde los siguientes datos en variables y los escriba en pantalla: -40, '5', 123450, 3.141592, 123456.7890
2. Mostrar en pantalla el tamaño en bytes de los tipos de datos existentes en C. Utilizar la función `sizeof()` de la librería estándar.
3. Introducir un carácter por teclado y mostrar en pantalla:
 - El carácter introducido.
 - El valor decimal del carácter introducido.
 - El valor hexadecimal del carácter introducido.
 - El valor octal del carácter introducido.
4. Pedir una cantidad de segundos a introducir por el teclado y convertirlos en horas, minutos y segundos.
5. Solicitar un número mayor que 1 y menor que 100 y realizar la suma desde 1 hasta ese número mediante los siguientes bucles:
 - un bucle *while*
 - un bucle *for*
 - un bucle *do while*
6. Indicar la cantidad de números a introducir, proceder a leerlos y calcular:
 - La suma de todos ellos.
 - La media.
 - El valor máximo.
 - El valor mínimo.
7. Mostrar la lista de números primos hasta un límite determinado introducido por teclado.
8. Mostrar un texto en pantalla introducido por teclado y contar:
 - El número de caracteres.
 - El número de líneas.
 - El número de palabras.
9. Hacer un cronómetro de minutos y segundos que cuente hasta una cantidad introducida por teclado.
10. Diseñar una macrofunción para obtener el máximo de dos valores. Utilizarla en un programa de aplicación.
11. Diseñar una macrofunción para obtener el valor absoluto de un número. Utilizarla en un programa de aplicación.
12. Realiza un conversor de euros a pesetas y viceversa.
13. Inicializar un array entero con 10 elementos y posteriormente mostrarlo en el orden de entrada FIFO y en el orden inverso LIFO.

14. Introducir por teclado una lista de 5 valores y a continuación obtener el máximo, mínimo y media aritmética de los valores introducidos en el array.
15. Pedir valores para una matriz de 3x3 elementos y realizar la suma de filas y columnas. Mostrar en pantalla la matriz y los resultados
16. Utilizando el ejercicio 14, modificarlo para usar punteros.
17. Dados dos arrays entrada e[10] y salida s[10] inicializarlos con valores numéricos enteros. Introducir un valor por teclado y compararlo con el primer elemento de la pila de entrada, si coincide sacar el primer elemento de la pila de salida. Solicitar un nuevo dato por teclado y volver a repetir el proceso, en este caso con el segundo elemento. Así hasta que se hayan acabado los arrays.
18. Inicializar un array numérico entero y ordenarlo de menor a mayor.
19. En el mismo programa anterior dar opción con pregunta para ver si la ordenación se va a realizar en sentido ascendente o descendente.
20. Hacer un programa sencillo para jugar a “tres en raya”. El adversario nuestro será el ordenador.
21. Usar la función strcat() para concatenar dos cadenas.
22. Introducir dos frases por teclado y compararlas si son iguales o no.
23. Utilizando arrays multidimensionales generar un programa para capturar 10 nombres y posteriormente sacarlos ordenados.
24. Realizar un programa que nos convierta números decimales en números escritos en el sistema de numeración romano.
25. Diseñar un programa, que mediante un menú tengamos las siguientes opciones:
 - Introducir nombres.
 - Listar nombres.
 - Ordenar nombres.
 - Buscar nombres.
 - Salir al sistema.
26. Hacer un programa que mediante un menú tengamos las posibilidades de:
 - Introducir nombres de clientes.
 - Listar nombres de clientes en el orden:
 - ✓ En que se han introducido.
 - ✓ En orden alfabético ascendente.
 - ✓ En orden alfabético descendente.
 - Salir al sistema.
27. Diseñar una función para obtener el máximo de tres valores. Utilizarla en un programa de aplicación.

-
28. Diseñar una función para ordenar una lista de valores enteros. Utilizarla en un programa de aplicación.
29. Diseñar una librería con las siguientes funciones:
- ✓ Suma de dos números.
 - ✓ Resta de dos números.
 - ✓ Multiplicación de dos números.
 - ✓ División de dos números.
 - ✓ Valor absoluto de un número.
 - ✓ Factorial de un número.
 - ✓ Suma de los elementos de un array.
 - ✓ Media de los elementos de un array.
 - ✓ Valor máximo de los elementos de un array.
 - ✓ Valor mínimo de los elementos de un array.
30. Realizar un programa que mediante menús sea capaz de calcular áreas y perímetros de cuadrados, triángulos y círculos dándole los oportunos parámetros.
31. Realizar un contador indicando desde la línea de órdenes el valor final del contador y la posibilidad o no de visualizarlo en la pantalla.
32. Crear una estructura que contenga: nombre, edad y estatura en metros de un individuo. Diseñar una función que pregunte estos datos y los guarde en una variable del tipo estructura definida.
33. Hacer que los datos del programa anterior se vayan guardando en un fichero.
34. Diseñar un programa para gestionar a través de un menú, las direcciones de una serie de personas. Las opciones son: introducir, listar y borrar direcciones. El tamaño máximo de la lista será de 100. El programa antes de introducir una nueva dirección, comprobará si la lista no está completamente llena.
35. Diseñar un programa para gestionar por menú las siguientes opciones:
- Introducir dirección.
 - Borrar dirección.
 - Listar el fichero.
 - Buscar dirección.
 - Almacenar el fichero
 - Cargar el fichero.
 - Salir al sistema.
- El tamaño de la lista se incrementará en función de las direcciones introducidas. La lista estará doblemente enlazada.
36. Leer un fichero secuencial de caracteres preguntando por el nombre del fichero. Detectar si no existe el fichero fuente.
37. Diseñar un programa capaz de copiar ficheros. Preguntará por el nombre del fichero fuente y destino. Detectará error si no existe el fichero fuente.

38. Diseñar un programa para comparar dos ficheros de texto. Preguntará por el nombre de los dos ficheros detectando si existen.
39. Realizar un programa que a través de menú tenga acceso a inicializar, escribir y leer de un fichero secuencial.
Menú: 0.- Iniciar.
1.- Escribir.
2.- Leer.
3.- Fin de la sesión.
40. Leer un fichero secuencial carácter a carácter y mostrar su contenido en pantalla. El nombre del archivo a leer se introducirá por el teclado con pregunta.
41. Escribir en un fichero secuencial carácter a carácter y para terminar pulsar (^Z). El nombre del archivo a escribir se introducirá por el teclado con pregunta.
42. Copiar secuencialmente de un fichero original hacia otro de destino. El nombre de ambos ficheros se introducirán por el teclado.

15.- LISTADOS DE LAS PRÁCTICAS

```
/* Práctica 1*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    char letra='5';
    int numero=-40;
    long grande=123450;
    float pi=3.141592;
    double n=123456.7890;
    printf("%d %c %ld %f %f\n", numero, letra, grande, pi, n);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 2*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int a; long b; short c; float d; double e; char f;
    printf("Los INT ocupan %d\n", sizeof(a));
    printf("Los LONG ocupan %d\n", sizeof(b));
    printf("Los SHORT ocupan %d\n", sizeof(c));
    printf("Los FLOAT ocupan %d\n", sizeof(d));
    printf("Los DOUBLE ocupan %d\n", sizeof(e));
    printf("Los CHAR ocupan %d\n", sizeof(f));
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 3*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    char valor;
    printf("Introduzca caracter: ");
    scanf("%c", &valor);
    printf("Caracter= %c\n", valor);
    printf("Valor decimal= %d\n", valor);
    printf("Valor hexadecimal= %x\n", valor);
    printf("Valor octal= %o\n", valor);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 4*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int total, horas, min, seg;
    printf("Cantidad total de segundos: ");
    scanf("%d", &total);
    horas=total/3600;
    min=(total%3600)/60;
    seg=total%60;
    printf("Tiempo= %2d horas %2d minutos %2d segundos\n", horas, min, seg);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 5*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int n, n2;
    long suma=0;
    do
    {
        printf("Introduce un numero mayor que 1 y menor que 100: ");
        scanf("%d",&n);
    }
    while(n<=1 || n>=100);
    n2=n;
    printf("La suma de 1 a %d es ", n2);
    while(n>0)
    suma+=n--;
    printf("%ld\n", suma);
    printf("La suma de 1 a %d es ", n2);
    for(n=n2, suma=0; n>0; suma+=n--);
    printf("%ld\n", suma);
    printf("La suma de 1 a %d es ", n2);
    n=n2;
    suma=0;
    do
    suma+=n--;
    while(n>0);
    printf("%ld\n", suma);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 6*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int total, i;
    long numero, suma, maximo, minimo;
    float media;
    printf("Cantidad de numeros a procesar: ");
    scanf("%d",&total);
    for(i=0, suma=0; i<total; i++)
    {
        printf("Numero %d: ", i+1);
        scanf("%ld",&numero);
        if(i==0) maximo=minimo=numero;
        if(numero>maximo) maximo=numero;
        if(numero<minimo) minimo=numero;
        suma+=numero;
    }
    media=(float)suma/total;
    printf("Suma= %10d\n", suma);
    printf("Maximo= %10d\n", maximo);
    printf("Minimo= %10d\n", minimo);
    printf("Media= %10.2f\n", media);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 7*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int lim, num, div;
    printf("Introduzca limite: ");
    scanf("%d",&lim);
    printf("1 ");
    for(num=2; num<=lim; ++num)
    {
        for(div=2; num%div!=0; ++div);
        if(div==num) printf("%d ", num);
    }
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 8*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int ch=0, nc=0, nl=0, np=0, palabra=0;
    while(ch!=EOF)
    {
        ch=getchar(); putchar(ch); ++nc;
        if(ch=='\n') ++nl;
        if(ch!=' ' && ch!='\n' && palabra==0) { ++np; palabra=1; }
        if((ch==' ' || ch=='\n') && palabra==1) palabra=0;
    }
    printf("Numero de caracteres: %d\n", nc);
    printf("Numero de lineas: %d\n", nl);
    printf("Numero de palabras: %d\n", np);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 9*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int camin, caseg, min=0, seg=0;
    printf("Dame los minutos para los que suena la alarma: "); scanf("%d",&camin);
    printf("Dame los segundos para los que suena la alarma: "); scanf("%d",&caseg);
    printf("%02d:%02d",min,seg);
    while(1)
    {
        for(seg=0;seg<=59;seg++)
        {
            printf("\r%02d:%02d",min,seg);
            sleep(100);
            if((min==camin)&&(seg==caseg)) goto ya;
        }
        min++;
    }
    ya: printf("\nEs el tiempo de parar el reloj\n");
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 10*/
#include<stdio.h>
#include<stdlib.h>
#define MAX(X,Y) ((X) > (Y) ? (X) : (Y))
main()
{
    float x, y;
    printf("Primer valor: "); scanf("%f", &x);
    printf("Segundo valor: "); scanf("%f", &y);
    printf("El maximo de %f y %f es: %f\n", x, y, MAX(x,y));
    system("PAUSE");
    return 0;
}

/* Práctica 11*/
#include<stdio.h>
#include<stdlib.h>
#define ABS(X) ((X) <0 ? -(X) : (X))
main()
{
    float x;
    printf("Valor: "); scanf("%f", &x);
    printf("El valor absoluto de %f es: %f\n", x, ABS(x));
    system("PAUSE");
    return 0;
}

/* Práctica 12*/
#include<stdio.h>
#include<stdlib.h>
#define euro 166.386
main()
{
    float n, x;
    int opcion;
    printf("La cantidad: "); scanf("%f", &n);
    printf("1-Ptas a Euros 2-Euros a Ptas "); scanf("%d",&opcion);
    switch(opcion) {
        case 1: x=n/euro;
            printf("%0.2f pesetas son %0.2f euros\n",n,x); break;
        case 2: x=n*euro;
            printf("%0.2f euros son %0.2f pesetas\n",n,x); break;
        default: printf("incorrecta la opcion tecleada\n");
    }
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 13*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    static int pila[10]={0,1,2,3,4,5,6,7,8,9};
    int i=0;
    printf("FIFO: "); for(i=0;i<10;++i) printf("%d ", pila[i]);
    printf("\nLIFO: "); for(i=9; i>=0;--i) printf("%d ", pila[i]);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 14*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    float lista[5], suma=0, max=0, min=0;
    int i=0;
    for(i=0;i<5;++i)
    { printf("Valor: "); scanf("%f",&lista[i]); }
    max=lista[0]; min=lista[0];
    for(i=0;i<5;++i)
    {
        suma=suma+lista[i];
        if(lista[i]>max) max=lista[i];
        if(lista[i]<min) min=lista[i];
    }
    printf("Media %5.2f\n", suma/5);
    printf("Maximo %5.2f\n", max);
    printf("Minimo %5.2f\n", min);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 15*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    int tabla[4][4], i, j;
    for(i=0;i<4;++i) tabla[i][3]=tabla[3][i]=0;
    printf("Introduce la tabla de 3x3 elementos:\n\n");
    for(i=0;i<3;++i)
        for(j=0;j<3;++j)
        {
            printf("Elemento (%d,%d): ",i+1, j+1);
            scanf("%d",&tabla[i][j]);
            tabla[i][3]+=tabla[i][j];
            tabla[3][j]+=tabla[i][j];
            tabla[3][3]+=tabla[i][j];
        }
    for(i=0;i<4;++i)
        for(j=0;j<4;++j)
            printf("%3d%c", tabla[i][j], (j==3) ? '\n' : ' ');
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 16*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    float lista[5], suma=0, max=0, min=0;
    float *p; int i=0;
    for(i=0;i<5;++i)
        { printf("Valor: "); scanf("%f",&lista[i]); }
    p=&lista[0]; max=lista[0]; min=lista[0];
    for(i=0;i<5;++i)
        {
            suma=suma+*(p+i);
            if(*(p+i)>max) max=*(p+i);
            if(*(p+i)<min) min=*(p+i);
        }
    printf("Media %5.2f\n", suma/5);
    printf("Maximo %5.2f\n", max);
    printf("Minimo %5.2f\n", min);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 17*/
#include<stdio.h>
#include<stdlib.h>
int e[10]={1,2,3,4,5,6,7,8,9,0};
int s[10]={11,22,33,44,55,66,77,88,99,00};
main()
{
    int *ee, *ss, dato=0, x=0;
    ee=&e[0]; ss=&s[0];
    for(x=0;x<10;++x)
    {
        do
        {
            printf("Valor de entrada: "); scanf("%d",&dato);
        }
        while(dato!=*(ee+x));
        printf("\nSalida= %d\n",*(ss+x));
    }
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 18*/
#include<stdio.h>
#include<stdlib.h>
int n[20]={3,4,5,1,2,8,9,6,7,8,12,10,11,2,1,0,4,3,2,15};
main()
{
    int i,j,k=0;
    int *po; po=&n[0];
    printf("Lista original: ");
    for(i=0;i<20;++i) printf("%d ", *(po+i));
    for(i=0;i<19;++i)
    {
        for(j=i+1;j<20;++j)
            if(*(po+i)>*(po+j))
            {
                k=*(po+i); *(po+i)=*(po+j); *(po+j)=k;
            }
    }
    printf("\nLista ordenada: ");
    for(i=0;i<20;++i) printf("%d ", *(po+i));
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 19*/
#include<stdio.h>
#include<stdlib.h>
int lista[20]={4,6,7,8,10,5,3,2,7,0,10,12,15,4,2,0,1,3,2,1};
```

```
int i,j,k=0;
main()
{
    int c; int *p; p=&lista[0];
    borra_pantalla();
    printf("Array original: ");
    for(i=0;i<20;++i) printf("%d ", *(p+i)); printf("\n\n");
    printf("1.- Ordenar array en sentido ascendente.\n");
    printf("2.- Ordenar array en sentido descendente.\n");
    printf("  Seleccionar opcion "); scanf("%d", &c);
    switch(c)
    {
        case 1: ascender(); break;
        case 2: descender(); break;
        default: printf("Error de seleccion ");
    }
    system("PAUSE");
    return 0;
}

ascender()
{
    int *p; p=&lista[0];
    for(i=0;i<19;++i)
    {
        for(j=i+1;j<20;++j)
            if(*(p+i)>*(p+j))
                { k=*(p+i); *(p+i)=*(p+j); *(p+j)=k; }
    }
    printf("\nArray ordenado en sentido ascendente: \n");
    for(i=0;i<20;++i)
        printf("%d ",*(p+i));
}

descender()
{
    int *p; p=&lista[0];
    for(i=0;i<19;++i)
    {
        for(j=i+1;j<20;++j)
            if(*(p+i)<*(p+j))
                { k=*(p+i); *(p+i)=*(p+j); *(p+j)=k; }
    }
    printf("\nArray ordenado en sentido descendente: \n");
    for(i=0;i<20;++i)
        printf("%d ",*(p+i));
}

borra_pantalla()
{
    system("cls");
}
```

```

/* Práctica 20*/
#include<stdio.h>
#include<stdlib.h>
char matriz[3][3]={' ',' ',' ',' ',' ',' ',' ',' ',' '};
main()
{
    char done;
    borra_pantalla();
    printf("Este es el juego de tres en raya. \n");
    printf("Se jugara frente a la computadora. \n");
    done=' ';
    do
    {
        disp_matriz();
        coge_mov_jugador();
        done=check();
        if(done!=' ')break;
        coge_mov_computa();
        done=check();
    }
    while(done==' ');
    if(done=='X') printf("Usted gana GRRRR.... !\n");
    else printf("Yo gano ..... !\n");
    disp_matriz();
    system("PAUSE");
    return 0;
}
borra_pantalla() { system("cls"); }
disp_matriz()
{
    int i, t;
    for(t=0;t<3;t++)
    {
        printf(" %c | %c | %c ",matriz[t][0], matriz[t][1], matriz[t][2]);
        if(t!=2) printf("\n---|---|---\n");
    }
    printf("\n");
}
coge_mov_jugador()
{
    int x,y;
    int ok=0;
    printf("Introduce las coordenadas para X: ");
    do
    {
        scanf("%d %d",&x,&y);
        x--;y--;
        if(matriz[x][y]!=' ')
            printf("Movimiento invalido, intentelo otra vez.\n");
        else

```

```

        {
            matriz[x][y]='X';
            ok=1;
        }
    }
    while(!ok);
}
check()
{
    int t;
    char *p;
    for(t=0;t<3;t++)
    {
        p=&matriz[t][0];
        if(*p==*(p+1) && *(p+1)==*(p+2)) return *p;
    }
    for(t=0;t<3;t++)
    if(matriz[0][t]==matriz[1][t] && matriz[1][t]==matriz[2][t]) return matriz[0][t];
    if(matriz[0][0]==matriz[1][1] && matriz[1][1]==matriz[2][2]) return matriz[0][0];
    if(matriz[0][2]==matriz[1][1] && matriz[1][1]==matriz[2][0]) return matriz[0][2];
    return ' ';
}
coge_mov_computa()
{
    int i, t;
    for(t=0;t<3;t++)
    {
        for(i=0;i<3;i++)
        if(matriz[t][i]==' ') break;
        if(matriz[t][i]==' ') break;
    }
    if(t*i==9)
    {
        printf("Tablas.....\n");
        exit(0);
    }
    else matriz[t][i]='O';
}

/* Práctica 21*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    char final[60], *primer, *segund;
    primer="Todos para Uno ";    segund="y Uno para Todos";
    strcat(final,primer); strcat(final,segund);    puts(final);
    system("PAUSE");    return (0);
}

```

```
/* Práctica 22*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    char frase1[60], frase2[60];
    printf("Frase 1: "); gets(frase1);
    printf("Frase 2: "); gets(frase2);
    if(strcmp(frase1,frase2)>0)
        printf("-%s- es mayor que -%s- ", frase1, frase2);
    else if(strcmp(frase1,frase2)<0)
        printf("-%s- es menor que -%s- ", frase1, frase2);
    else printf("Son iguales\n");
    system("PAUSE");
    return (0);
}
```

```
/* Práctica 23*/
#include<stdio.h>
#include<stdlib.h>
main()
{
    char frase[10][35];
    int i=0, j=0;
    char *punt[10], *k;
    for(i=0;i<10;++i)
    {
        printf("?:");
        gets(frase[i]);
        punt[i]=frase[i];
    }
    for(i=0;i<9;++i)
    {
        for(j=i+1;j<10;++j)
            if(strcmp(punt[i],punt[j])>0)
                { k=punt[i]; punt[i]=punt[j]; punt[j]=k; }
    }
    printf("\n\nLista ordenada: \n");
    for(i=0;i<10;++i)
        puts(punt[i]);
    system("PAUSE");
    return (0);
}
```

```
/* Práctica 24*/
#include<stdio.h>
#include<stdlib.h>
main()
{
static char *uni[10]={ "", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX" };
static char *dec[10]={ "", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC" };
static char *cen[10]={ "", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM" };
static char *mil[4]={ "", "M", "MM", "MMM" };
char **indice[4];
int num, lon, lonfi, b;
char numero[5], otro[3];
indice[0]=&uni[0];
indice[1]=&dec[0];
indice[2]=&cen[0];
indice[3]=&mil[0];
borra_pantalla();
printf("Introduce un numero decimal y lo veras en el sistema de numeracion romano: ");
scanf("%d",&num);
lon=strlen(itoa(num,numero,10));
lonfi=lon;
for(;lon>0;lon--)
{
otro[0]=numero[lonfi-lon];
b=atoi(otro);
printf("%s", *(indice[lon-1]+b));
}
printf("\n\n");
system("PAUSE");
return (0);
}
borra_pantalla() { system("cls"); }
```

```
/* Práctica 25*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char nombre[20][20];
main()
{
char ch;
for(;;)
{
borra_pantalla();
gotoxy(30,10); puts("(I)ntroducir nombres.");
gotoxy(30,0);puts("(L)istar nombres.");
gotoxy(30,0);puts("(O)rdenar nombres.");
gotoxy(30,0);puts("(B)uscar nombre.");
gotoxy(30,0);puts("(S)alir al sistema.");
gotoxy(30,2); puts("Seleccione opcion: ");
do ch=toupper(getch());
while(ch!='I' && ch!='L' && ch!='O' && ch!='B' && ch!='S');
switch(ch)
{
case 'I': intro(); break;
case 'L': listar(); break;
case 'O': orden(); break;
case 'B': buscar(); break;
case 'S': exit(0);
}
}
system("PAUSE");
return (0);
}
borra_pantalla()
{
system("cls");
}
gotoxy(int x, int y)
{
for(;y-->0) printf("\n");
for(;x-->0) printf(" ");
}
intro()
{
int i;
borra_pantalla();
for(i=0;i<20;++i)
nombre[i][0]='\0'; i=0;
puts("Introducir nombres. Pulse INTRO para terminar.");
puts("-----");
while(i<20)
{
```

```
printf("Nombre: "); gets(nombre[i]);
if(!nombre[i][0]) return (0);
strupr(nombre[i]);
i++;
}
}
listar()
{
int i=0;
borra_pantalla();
puts("Litar los nombres introducidos.");
puts("-----");
while(i<20)
{
puts(nombre[i]);
i++;
}
puts("Pulse una tecla para continuar");
getch();
}
orden()
{
borra_pantalla();
qsort(nombre,20,20,strcmp);
puts("Pulse una tecla para continuar");
getch();
}
buscar()
{
char busca[20], *p;
qsort(nombre,20,20,strcmp);
borra_pantalla();
printf("Nombre a buscar: ");gets(busca); strupr(busca);
p=(char *)bsearch(busca,nombre,20,20,strcmp);
if(*p==*busca)
puts("Localizado");
else
puts("No localizado");
puts("Pulse una tecla para continuar");
getch();
}
```

```
/* Práctica 26*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char ap[10][20], pu[10][20];
main()
{
int t;
while(1)
{
borra_pantalla();
puts("(C)argar lista (L)istar (X)Final");
puts("=====");
do
{
t=getch(); t=toupper(t);
} while(t!='C' && t!='L' && t!='X');
switch(t)
{
case 'C': cargar(); break;
case 'L': listar(); break;
case 'X': exit(0);
}
}
system("PAUSE");
return (0);
}
borra_pantalla() { system("cls"); }
cargar()
{
int i;
borra_pantalla();
for(i=0;i<10;++i)
{
printf("Nombre Cliente: ? ");
gets(ap[i]); strupr(ap[i]);
memcpy(pu[i],ap[i],20);
}
}
listar()
{
int t;
while(1)
{
borra_pantalla();
puts("(O)riginal (A)scendente (D)escendente (X)Retorna");
puts("-----");
do {
t=getch(); t=toupper(t);
} while(t!='O' && t!='A' && t!='X' && t!='D');
```

```
switch(t)
{
  case 'O': lorigi(); break;
  case 'A': lascen(); break;
  case 'D': ldesce(); break;
  case 'X': return (0);
}
}
}
lorigi()
{
  int i;
  puts("APELLIDOS CLIENTES");
  puts("=====");
  for(i=0;i<10;i++)
  puts(ap[i]);
  getch();
}
lascen()
{
  int i;
  qsort(pu,10,20,strcmp);
  puts("APELLIDOS CLIENTES");
  puts("=====");
  for(i=0;i<10;++i)
  puts(pu[i]);
  getch();
}
ldesce()
{
  int i;
  qsort(pu,10,20,strcmp);
  puts("APELLIDOS CLIENTES");
  puts("=====");
  for(i=9;i>=0;--i)
  puts(pu[i]);
  getch();
}
```

```
/* Práctica 27*/
#include<stdio.h>
#include<stdlib.h>
main()
{
int i, n[3];
printf("Introduce tres numeros enteros\n");
for(i=0;i<3;i++)
    { printf("Numero %d: ",i+1); scanf("%d",&n[i]); }
printf("El mayor es %d \n", max(n[0],n[1],n[2]));
system("PAUSE");
return (0);
}
int max(int a, int b, int c)
{
int m=0;
if((a>b)&(a>c)) m=a;
else
if((b>a)&(b>c)) m=b;
else m=c;
return (m);
}

/* Práctica 28*/
#include<stdio.h>
#include<stdlib.h>
int numeros[20]={3,4,5,1,2,8,9,6,7,8,6,5,11,2,1,0,4,3,2,8};
main()
{
int a=0, tope=20;
printf("Lista Original: "); for(a=0;a<tope;a++) printf("%d ",numeros[a]);
ordena(tope,numeros);
printf("\nLista Ordenada: "); for(a=0;a<tope;++a) printf("%d ",numeros[a]);
printf("\n");
system("PAUSE");
return (0);
}
ordena(int num, int lista[100])
{
int i, j, k=0;
int *po; po=&lista[0];
for(i=0;i<num-1;++i)
    {
for(j=i+1;j<num;++j)
if(*(po+i)>*(po+j))
    { k=*(po+i); *(po+i)=*(po+j); *(po+j)=k; }
}
return (0);
}
```

```
/* Práctica 29*/
float suma(float a, float b) /* Función suma(a,b) */
{ return (a+b); }
float resta(float a, float b) /* Función resta(a,b) */
{ return (a-b); }
float multi(float a, float b) /* Función multi(a,b) */
{ return (a*b); }
float divi(float a, float b) /* Función divi(a,b) */
{ return (a/b); }
float abso(float a) /* Función abso(a) */
{ return (a<0 ?-a:a); }
int fact(int a) /* Función fact(a) */
{
int resul;
if(a==1) return (1);
resul=fact(a-1)*a;
return (resul);
}
float sumlista(int num, float *lista) /* Función sumlista(num,lista) */
{
int i; float resul=0.0;
for(i=0;i<num;++i)
resul=resul+lista[i];
return (resul);
}
float medlista(int num, float *lista) /* Función medlista(num,lista) */
{
int i; float resul=0.0;
for(i=0;i<num;++i)
resul=resul+lista[i];
return (resul/num);
}
float maxlista(int num, float *lista) /* Función maxlista(num,lista) */
{
int i; float max=lista[0];
lista=&lista[0];
for(i=0;i<num;++i)
{ if(*(lista+i)>max) max=*(lista+i); }
return (max);
}
float minlista(int num, float *lista) /* Función minlista(num,lista) */
{
int i; float min=lista[0];
lista=&lista[0];
for(i=0;i<num;++i)
{ if(*(lista+i)<min) min=*(lista+i); }
return (min);
}
```

```
/* Práctica 30*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
char c=' ';
while(c!='s')
{
menu1();
c=getch();fflush(stdin);
switch(c)
{
case 'p': menu2(); peri(); break;
case 'a': menu2(); area(); break;
case 's': exit(0);
}
}
system("PAUSE");
return 0;
}
menu1()
{
borra_pantalla();
printf("(P)Perimetros\n");
printf("(A)Areas\n");
printf("(S)Salir\n");
}
menu2()
{
borra_pantalla();
printf("(C)Cuadrado\n");
printf("(T)Triangulo\n");
printf("(O)Circulo\n");
printf("(R)Retornar\n");
}
peri()
{
char c;
c=getch();fflush(stdin);
switch(c)
{
case 'c': pecua(); break;
case 't': petri(); break;
case 'o': pecir(); break;
case 'r': return(0);
}
}
area()
{
char c;
```

```
c=getch();fflush(stdin);
switch(c)
{
    case 'c': arecua(); break;
    case 't': aretri(); break;
    case 'o': arecir(); break;
    case 'r': return(0);
}
}
pecua()
{
float lado;
borra_pantalla();
printf("Dame el lado del cuadrado: "); scanf("%f",&lado);
printf("EL perimetro del cuadrado de lado %0.2f es %0.2f\n", lado, 4*lado);
printf("Pulse una tecla para continuar...");
getch();
}
petri()
{
int i;
float a[3];
borra_pantalla();
for(i=0;i<3;i++)
{
    printf("Dame el lado %d: \n", i+1);
    scanf("%f",&a[i]);fflush(stdin);
}
printf("El perimetro del triangulo de lados %0.2f %0.2f y %0.2f es
%0.2f\n",a[0],a[1],a[2],a[0]+a[1]+a[2]);
printf("Pulse una tecla para continuar...");
getch();
}
pecir()
{
float radio;
borra_pantalla();
printf("Dame el radio del circulo: "); scanf("%f",&radio);fflush(stdin);
printf("EL perimetro del circulo de radio %0.2f es %0.2f\n", radio, 2*3.141592*radio);
printf("Pulse una tecla para continuar...");
getch();
}
arecua()
{
float lado;
borra_pantalla();
printf("Dame el lado del cuadrado: "); scanf("%f",&lado);fflush(stdin);
printf("EL area del cuadrado de lado %0.2f es %0.2f\n", lado, lado*lado);
printf("Pulse una tecla para continuar...");
getch();
```

```

}
aretri()
{
float base, altura;
borra_pantalla();
printf("Dame la base del triangulo: "); scanf("%f",&base);fflush(stdin);
printf("Dame la altura del triangulo: "); scanf("%f",&altura);fflush(stdin);
printf("El area del triangulo de base %0.2f y altura %0.2f es
%0.2f\n",base,altura,base*altura/2);
printf("Pulse una tecla para continuar...");
getch();
}
arecir()
{
float radio;
borra_pantalla();
printf("Dame el radio del circulo: "); scanf("%f",&radio);fflush(stdin);
printf("EL area del circulo de radio %0.2f es %0.2f\n", radio, 3.141592*radio*radio);
printf("Pulse una tecla para continuar...");
getch();
}
borra_pantalla() { system("cls"); }

```

```

/* Práctica 31 */
#include<stdio.h>
#include<stdlib.h>
main(int argc,char *argv[])
{
int disp, cont, i;
if(argc<2)
{
printf("Se debe introducir la longitud del contador\n");
printf("en la linea de ordenes. Intentelo de nuevo.\n");
exit(0);
}
if(argc==3 && !strcmp(argv[2],"visualiza")) disp=1;
else disp=0;
cont=atoi(argv[1]);
for(i=0;i<=cont;++i)
{if(disp) printf("%d\r", i); sleep (1000); }
printf("%c",7);
system("PAUSE");
return 0;
}

```

```
/* Práctica 32*/
#include<stdio.h>
#include<stdlib.h>
struct persona
{
    char nombre[40];
    int edad;
    float talla;
};
main()
{
    struct persona variable;
    pregunta(&variable);
    printf("\nNombre: %s", variable.nombre);
    printf("\nEdad: %d", variable.edad);
    printf("\nAltura %0.2f", variable.talla);
    system("PAUSE");
    return 0;
}
pregunta(struct persona *p)
{
    printf("Nombre: ");
    gets(p->nombre);fflush(stdin);
    printf("Edad: ");
    scanf("%d",&p->edad);fflush(stdin);
    printf("Altura: ");
    scanf("%f",&p->talla);fflush(stdin);
}

/* Práctica 33*/
#include<stdio.h>
#include<stdlib.h>
struct persona
{
    char nombre[40];
    char edad[12];
    char altura[12];
};
main()
{
    struct persona variable;
    FILE *fi;
    char fin='S', fichero[20];
    int i=0;
    printf("Generacion del fichero\n");
    printf("=====\n");
    printf("Nombre del fichero de datos: ");
    gets(fichero);fflush(stdin);
    if((fi=fopen(fichero,"a"))==NULL)
    printf("ERROR, ha sido imposible crear el fichero");
```

```
else
{
while(fin=='S' || fin=='s')
{
printf("Persona %d: ",++i);
pregunta(&variable);
fwrite(&variable,sizeof variable,1,fi);
printf("\nOtra persona (S/N)");
do
{
fin=getchar();fflush(stdin);
}
while(fin!='N' && fin!='n' && fin!='S' && fin!='s');
}
fclose(fi);
}
system("PAUSE");
return 0;
}
pregunta(struct persona *p)
{
printf("Nombre: ");
gets(p->nombre);fflush(stdin);
printf("Edad: ");
gets(p->edad);fflush(stdin);
printf("Altura: ");
gets(p->altura);fflush(stdin);
}

/* Práctica 34*/
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
struct dir
{
char nombre[40];
char calle[40];
char ciudad[30];
char prov[20];
unsigned long cp;
} info_dir[MAX];
main()
{
char opcion;
inic_lista();
for(;;) {
opcion=menu();
switch(opcion)
{
case 1: intro(); break;
```

```
        case 2: borrar(); break;
        case 3: listar(); break;
        case 4: exit(0);
        }
    }
    system("PAUSE");
    return 0;
}
inic_lista()
{
    register int t;
    for(t=0;t<MAX;t++) info_dir[t].nombre[0]='\0';
}
menu()
{
    char s[3];
    int c;
    borra_pantalla();
    gotoxy(18,7);puts("*****");
    gotoxy(18,0);puts("      MENU PRINCIPAL      ");
    gotoxy(18,0);puts("*****");
    gotoxy(18,0);puts("    1.- Introducir un nombre.    ");
    gotoxy(18,0);puts("    2.- Borrar un nombre.        ");
    gotoxy(18,0);puts("    3.- Listar.                  ");
    gotoxy(18,0);puts("    4.- Salir.                   ");
    gotoxy(18,0);puts("*****");
    do {
        gotoxy(18,0); printf("Introduzca opcion: ");
        gets(s);fflush(stdin);
        c=atoi(s);
    }
    while(c<0 || c>4);
    return c;
}
intro()
{
    int sitio;
    char s[8];
    borra_pantalla();
    sitio=busca_libre();
    if(sitio==-1) { printf("\nLista llena."); return (0); }
    printf("Introduzca nombre: ");
    gets(info_dir[sitio].nombre);fflush(stdin);
    printf("Introduzca calle: ");
    gets(info_dir[sitio].calle);fflush(stdin);
    printf("Introduzca ciudad: ");
    gets(info_dir[sitio].ciudad);fflush(stdin);
    printf("Introduzca provincia: ");
    gets(info_dir[sitio].prov);fflush(stdin);
    printf("Introduzca codigo postal: ");
```

```
gets(s);fflush(stdin);
info_dir[sitio].cp=strtoul(s,'\0',10);
}
busca_libre()
{
register int t;
for(t=0;info_dir[t].nombre[0] && t<MAX;t++)
if(t==MAX) return -1;
return t;
}
borrar()
{
register int sitio;
char s[8];
borra_pantalla();
printf("Introduzca numero de registro: ");
gets(s);fflush(stdin);
sitio=atoi(s);
if(sitio>=0 && sitio<MAX) info_dir[sitio].nombre[0]='\0';
}
listar()
{
register int t;
borra_pantalla();
for(t=0;t<MAX;t++)
{
if(info_dir[t].nombre[0])
{
printf("%s\n",info_dir[t].nombre);
printf("%s\n",info_dir[t].calle);
printf("%s\n",info_dir[t].ciudad);
printf("%s\n",info_dir[t].prov);
printf("%lu\n",info_dir[t].cp);
}
}
printf("\nPulse una tecla para continuar");
getch();
}
borra_pantalla() { system("cls"); }
gotoxy(int x, int y) { for(;y-->0) printf("\n"); for(;x-->0) printf(" "); }
```

```

/* Práctica 35 */
#include<stdio.h>
#include<stdlib.h>
struct dir
{
    char nombre[30];
    char calle[40];
    char ciudad[20];
    char prov[3];
    char dp[10];
    struct dir *siguiente;
    struct dir *anterior;
} lista_entrada;
struct dir *principio;
struct dir *ultimo;
menu_sel()
{
    char s[80];
    int c;
    borra_pantalla();
    gotoxy(18,7);puts("*****");
    gotoxy(18,0);puts("          MENU PRINCIPAL          ");
    gotoxy(18,0);puts("*****");
    gotoxy(18,0);puts("    1.- Introducir un nombre.    ");
    gotoxy(18,0);puts("    2.- Borrar un nombre.       ");
    gotoxy(18,0);puts("    3.- Listar el fichero.      ");
    gotoxy(18,0);puts("    4.- Buscar.                 ");
    gotoxy(18,0);puts("    5.- Almacenar el fichero.   ");
    gotoxy(18,0);puts("    6.- Cargar el fichero.      ");
    gotoxy(18,0);puts("    7.- Salir al sistema.       ");
    gotoxy(18,0);puts("*****");
do {
    gotoxy(18,0); printf("Introduzca opcion: ");
    gets(s);fflush(stdin);
    c=atoi(s);
    }
while(c<0 || c>7);
return c;
}
introducir()
{
    struct dir *info;
    char *x;
do {
    info=malloc(sizeof(lista_entrada));
    if(info==0) {
        puts("No hay memoria");
        return (0);
    }
    borra_pantalla();

```

```

inputs("Introducir nombre: ",info->nombre,30);
if(!info->nombre[0]) break;
inputs("Introducir calle: ",info->calle,40);
inputs("Introducir ciudad: ",info->ciudad,20);
inputs("Introducir provincia: ",info->prov,3);
inputs("Introducir DP: ",info->dp,10);
if(principio!=0) {
    ultimo->siguiente=info;
    info->anterior=ultimo;
    ultimo=info;
    ultimo->siguiente=0;
}
else {
    principio=info;
    principio->siguiente=0;
    ultimo=principio;
    principio->anterior=0;
}
} while(1);
}
inputs(prompt,h,cuenta)
char *prompt;
char *h;
int cuenta;
{
char p[255];
do {
    printf(prompt);
    gets(p);
    if(strlen(p)>=cuenta) puts("Demasiado largo");
} while(strlen(p)>=cuenta);
strcpy(h,p);
}
borrar()
{
struct dir *info, *encontrar();
char s[255];
borra_pantalla();
inputs("Introducir nombre: ",s,30);
info=encontrar(s);
if(info) {
    if(principio==info) {
        principio=info->siguiente;
        principio->anterior=0;
    }
    else {
        info->anterior->siguiente=info->siguiente;
        if(info!=ultimo)
            info->siguiente->anterior=info->anterior;
        else

```

```
        ultimo=info->anterior;
    }
    free(info);
}
}
struct dir *
encontrar(char *nombre)
{
    struct dir *info;
    info=principio;
    while(info) {
        if(!strcmp(nombre,info->nombre)) return info;
        info=info->siguiente;
    }
    printf("Nombre no encontrado\n");
    return (0);
}
listar()
{
    register int t;
    struct dir *info;
    info=principio;
    borra_pantalla();
    while(info) {
        visualiz(info);
        info=info->siguiente;
    }
    printf("\n\n");
    getch();
}
visualiz(struct dir *info)
{
    printf("%s\n",info->nombre);
    printf("%s %s\n",info->calle,info->ciudad);
    printf("%s %s\n",info->dp,info->prov);
    printf("\n\n");
}
buscar()
{
    char nombre[30];
    struct dir *info, *encontrar();
    borra_pantalla();
    inputs("Introducir nombre a encontrar: ",nombre,30);
    if(!(info=encontrar(nombre))) printf("\n");
    else visualiz(info); getch();
}
almacenar()
{
    register int t, tamano;
    struct dir *info;
```

```
char *p;
FILE *fp;
if((fp=fopen("mlist","w"))==NULL)
    { printf("No puede abrirse el fichero\n"); exit (0); }
borra_pantalla();
printf("\nAlmacenado el fichero\n");getch();
tamano=sizeof(lista_entrada);
info=principio;
while(info) {
    p=info;
    for(t=0;t<tamano;++t)
        putc(*p++,fp);
    info=info->siguiente;
}
putc(EOF,fp);
fclose(fp);
}
cargar()
{
register int t, tamano;
struct dir *info, *temp;
char *p, *d;
FILE *fp;
if((fp=fopen("mlist","r"))==NULL)
    { printf("No puede abrirse el fichero\n"); exit (0); }
borra_pantalla();
printf("\nCargado el fichero\n");getch();
tamano=sizeof(lista_entrada);
d=malloc(tamano);
principio=d;
if(!principio)
    { printf("Fuera de memoria\n"); return (0); }
info=principio;
p=info;
while((*p++=getc(fp))!=EOF)
    {
    for(t=0;t<tamano-1;++t)
        *p++=getc(fp);
    d=malloc(tamano);
    info->siguiente=d;
    if(!info->siguiente)
        { printf("Fuera de memoria\n"); return (0); }
    info->anterior=temp;
    temp=info;
    info=info->siguiente;
    p=info;
    }
free(temp->siguiente);
temp->siguiente=0;
ultimo=temp;
}
```

```
principio->anterior=0;
fclose(fp);
}
borra_pantalla() { system("cls"); }
gotoxy(int x, int y) { for(;y-->0) printf("\n"); for(;x-->0) printf(" "); }
```

```
main()
{
    int eleccion;
    principio=0;
    do {
        eleccion=menu_sel();
        switch(eleccion) {
            case 1: introducir(); break;
            case 2: borrar(); break;
            case 3: listar(); break;
            case 4: buscar(); break;
            case 5: almacenar(); break;
            case 6: cargar(); break;
            case 7: exit(0);
        }
    } while(1);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 36 */
#include<stdio.h>
#include<stdlib.h>
main()
{
    FILE *fi;
    char fuente[25];
    int c=0;
    printf("Fichero fuente: "); gets(fuente);
    if((fi=fopen(fuente,"r"))==NULL)
        { printf("ERROR no existe fuente"); getch(); exit (0); }
    while((c=getc(fi))!=EOF)
        putchar(c);
    fclose(fi);
    system("PAUSE");
    return 0;
}
```

```
/* Práctica 37 */
#include<stdio.h>
#include<stdlib.h>
main()
{
    FILE *fi, *fo;
    char fuente[25], destino[25];
    int c=0;
    printf("Fichero fuente: "); gets(fuente);
    printf("Fichero destino: "); gets(destino);
    if((fi=fopen(fuente,"r"))==NULL)
        { printf("ERROR no existe fuente"); getch(); exit (0); }
    fo=fopen(destino,"w");
    while((c=getc(fi))!=EOF)
        { putc(c,fo); putchar(c); }
    fclose(fi);
    fclose(fo);
    system("PAUSE");
    return 0;
}

/* Práctica 38 */
#include<stdio.h>
#include<stdlib.h>
main()
{
    FILE *fi, *fo;
    char fuente[25], destino[25];
    int c=0, c1, c2, fallo=0;
    printf("Fichero fuente: "); gets(fuente);
    printf("Fichero destino: "); gets(destino);
    if((fi=fopen(fuente,"r"))==NULL)
        { printf("ERROR no existe fuente"); getch(); exit (0); }
    if((fo=fopen(destino,"r"))==NULL)
        { printf("ERROR no existe destino"); getch(); exit (0); }
    while((c1=getc(fi))!=EOF && (c2=getc(fo))!=EOF)
        { if(c2!=c1) ++fallo; }
    if(fallo==0) { printf("Son iguales\n"); getch(); }
    else { printf("Difieren en %d caracteres\n", fallo); getch(); }
    fclose(fi);
    fclose(fo);
    system("PAUSE");
    return 0;
}
```

```

/* Práctica 39 */
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int car, tecla, yy;
char fuente[12];
FILE *fi;
main()
{
int c;
fichero();
conforme();
if(yy=='N' || yy=='n') exit (0);
while(1)
{
borra_pantalla();
puts(" < MENU DE FICHEROS SECUENCIALES > ");
puts(" ===== ");
puts(" ");
puts(" (0) Inicializar el fichero. ");
puts(" (1) Entrada de datos al fichero. ");
puts(" (2) Lectura de datos del fichero. ");
puts(" (3) Fin de la sesion. ");
puts(" ");
puts(" ===== ");
printf(" Seleccione opcion: "); scanf("%d",&c);
switch(c)
{
case 0: iniciar(); break;
case 1: escribir(); break;
case 2: leer(); break;
case 3: exit(0);
default: printf("ERROR de seleccion....."); pulse();
}
}
system("PAUSE");
return 0;
}
fichero()
{ printf("Nombre del fichero: "); scanf("%s",fuente); }
conforme()
{ printf("Conforme con el nombre ? (S/N) "); yy=getch(); }
iniciar()
{
if((fi=fopen(fuente,"w"))==NULL) { error(); exit(0); }
fclose(fi);
}
error()
{ printf("ERROR de acceso a fichero...."); pulse(); }
pulse()

```

```

{ puts("\nPulse una tecla...."); tecla=getch(); }
escribir()
{
if((fi=fopen(fuente,"a"))==NULL) { error(); exit(0); }
borra_pantalla();
puts("Entrada de datos (EOF)=Fin");
puts("=====");
while((car=getchar())!=EOF)
putc(car,fi);
fclose(fi);
}
leer()
{
if((fi=fopen(fuente,"r"))==NULL) { error(); exit (0); }
borra_pantalla();
puts("Lectura de datos ");
puts("===== ");
while((car=getc(fi))!=EOF)
putchar(car);
fclose(fi);
pulse();
}
borra_pantalla()
{ system("cls"); }

/* Práctica 40 */
#include<stdio.h>
#include<stdlib.h>
FILE *fi;
main()
{
char nombre[10];
int ch=0;
borra_pantalla();
printf("Nombre de fichero a leer: "); gets(nombre);
if((fi=fopen(nombre,"r"))==NULL)
{ printf("ERROR apertura"); getch(); exit (0); }
while(!feof(fi)) putchar(getc(fi));
fclose(fi);
system("PAUSE");
return 0;
}
borra_pantalla() { system("cls"); }

```

```
/* Práctica 41 */
#include<stdio.h>
#include<stdlib.h>
FILE *fi;
main()
{
    char nombre[12];
    int ch=0;
    borra_pantalla();
    printf("Nombre de archivo a escribir: "); gets(nombre);
    borra_pantalla();
    if((fi=fopen(nombre,"w"))==NULL)
        { printf("ERROR apertura"); getch(); exit (0); }
    while(ch!=EOF)
        { ch=getchar();putc(ch,fi); }
    fclose(fi);
    system("PAUSE");
    return 0;
}
borra_pantalla() { system("cls"); }
```

```
/* Práctica 42 */
#include<stdio.h>
#include<stdlib.h>
FILE *f1, *f2;
main()
{
    char origen[10], destino[10];
    borra_pantalla();
    printf("Nombre del archivo origen: "); gets(origen);
    printf("Nombre del archivo destino: "); gets(destino);
    if((f1=fopen(origen,"r"))==NULL)
        { puts("ERROR apertura"); getch(); exit (0); }
    if((f2=fopen(destino,"w"))==NULL)
        { puts("ERROR apertura"); getch(); exit (0); }
    while(!feof(f1)) putc(getc(f1),f2);
    fclose(f1); fclose(f2);
    system("PAUSE");
    return 0;
}
borra_pantalla() { system("cls"); }
```