CAPÍTULO 2

TIPOS ESTRUCTURADOS DE DATOS

En el manejo de estructuras de datos es necesario utilizar la sentencia de control **for;** por ello, se comienza el capítulo con un análisis en profundidad de esta estructura.

2.1 La sentencia de control for

El bucle **for**, es un tipo de comando repetitivo que, en su forma más común, se utiliza para repetir una sentencia o bloque de sentencias un número determinado de veces.

Sintaxis:

```
/*(Para repetir una sentencia)*/
```

```
for( inicialización; expresión; incremento) sentencia;
```

```
/*(Para repetir un grupo de sentencias)*/
```

```
for( inicialización; expresión; incremento)
{
    grupo de sentencias
 }
```

Donde el significado de los campos es el siguiente:

Inicialización: se utiliza para dar valor inicial a la variable que controla el bucle. Se ejecuta solamente una vez, antes de que se inicie la primera iteración del bucle. (Este campo puede aparecer en blanco si la variable de control ha sido inicializada antes de comenzar el bucle).

Expresión: es cualquier expresión válida C. Se valora al principio de cada iteración del bucle. Si es evaluada como *verdad* el bucle se ejecuta. Si es evaluada como *falso* el bucle se detiene y la ejecución del programa se reanuda en la siguiente línea de código al bucle. (Recordemos que el valor cero se evalúa como falso y el resto de valores como verdadero)

Incremento: Se utiliza para cambiar el valor de la variable de control en cierta cantidad. Se ejecuta, en cada iteración, al final del bloque de sentencias y antes que la comprobación de la condición lógica.

Ejemplo:

```
#include <stdio.h>
void main( )
{
   int num;
   for (num=1; num< 11; num=num+1) printf("%d ", num);
   printf ("bucle terminado");
 }
```

(la sentencia num=num + 1 se puede sustituir por num++)

Este programa produce la siguiente salida a pantalla:

1 2 3 4 5 6 7 8 9 10 bucle terminado

El funcionamiento del bucle es el siguiente:

La variable de control num comienza con valor 1, como es menor que 11, se ejecuta el bucle. El siguiente valor de la variable de control es 2, como es menor que 11, se ejecuta el bucle, así se continúa hasta que num vale 11, momento en el que no se cumple la condición lógica y se interrumpe el bucle.

Otro ejemplo en el que se repite un grupo de sentencias:

```
#include <stdio.h>
 void main(  )
 {
        int num, sum, prod;
   sum=0;
        prod=1;
        for (num=1; num< 11; num=num+1)
   {
              sum=sum + num;
        prod= prod * num;
}
  printf("producto y suma: %d %d", prod, sum);
  }
```

En este ejemplo, en cada iteración del bucle, num varía de 1 a 10; estos valores se van sumando en sum y multiplicando en prod; por tanto, la variable sum almacena el sumatorio y prod el factorial. La salida a pantalla es:

producto y suma: 3628800 55

El comando for admite una sintaxis más general, de forma que es posible utilizar dos o más índices (variables de control) simultáneamente, separados por comas (operador coma). Véase el siguiente ejemplo:

```
for(i=1,j=1;i<6&&j<4;i+=2,j++)
printf(" %d  %d ",i,j);
```

Este bucle imprimiría:

1 1 3 2 5 3

También está permitido dejar la expresión vacía; en este caso, el bucle la evalúa siempre como verdad y, por tanto, nos encontramos ante un bucle infinito. En el próximo capítulo se analizarán comandos que permiten interrumpir un bucle desde su interior.

Ejemplo:

```
#include <stdio.h>
void main( )
{
    int i;
     for(i=1; ;i++) printf("%d",i);
}
```

En C, a diferencia de la mayoría de lenguajes de programación, es perfectamente válido alterar la variable de control del bucle desde fuera de la sección de incremento. Por ejemplo, el siguiente programa incrementa la variable de control en una instrucción del bucle:

```
#include <stdio.h>
 void main( )
 {
 int i;
    for(i=1;i<10 ;)
    {
      printf("%d",i);
       i++;
  }
 }
```

2.2 Tablas

Una tabla o *array* es una lista de valores del mismo tipo que se referencian por un nombre común. A cada una de las variables que componen la tabla se le denomina elemento de la tabla.

Las tablas constituyen un modo adecuado de gestionar grupos de datos relacionados. Por ejemplo, si se quiere guardar la nota final de 100 alumnos, parece más lógico usar un vector (tabla unidimensional) de 100 componentes que utilizar 100 variables distintas.

2.2.1 Tablas unidimensionales

Una tabla unidimensional es un vector.

Declaración de vectores. Sintaxis:

tipo nombre_vector [tamaño];

donde tipo es un tipo válido de datos C, nombre_vector es el nombre del vector y tamaño el número de elementos del vector.

Por ejemplo con la sentencia,

```
float v[20];
```

se está declarando un vector v de tipo coma flotante con 20 componentes.

A un elemento de una tabla se accede por un número entero que es el índice que le corresponda. En el lenguaje C los índices comienzan en 0, luego en el vector v, declarado antes, de 20 elementos, los índices de éstos van del 0 al 19. Para acceder a un elemento de un vector se debe indicar el nombre del vector y entre corchetes el índice correspondiente al elemento. Por ejemplo, para acceder al primer elemento y último del vector v, se utilizará: v[0], v[19].

Con el siguiente ejemplo se introduce un valor en cada una de las 5 componentes del vector i:

```
#include <stdio.h>
void main( )
{
  int i[5], j;
    for (j=0; j< 5; j++) i[j]=j*j;
}
```

La tabla i tendrá el siguiente aspecto:

i[0]	i[1]	i[2]	i[3]	i[4]	elementos
0	**1**	**4**	**9**	**16**	**Valor**

Se puede utilizar el valor de un elemento de una tabla en cualquier lugar del programa en el que se utilizaría una variable o una constante. El siguiente programa almacena en el vector cuad los cuadrados de los números del 1 al 10 y a continuación los imprime en pantalla:

```
#include <stdio.h>
void main( )
{
int cuad[10],i;
for (i=0; i< 10; i++) cuad[i]=(i+1)*(i+1);
for (i=0; i< 10; i++)
printf("La componente %d  del vector es: %d \n",i+1,cuad[i]);
}
```

La salida a pantalla es la siguiente:

```
La componente 1  del vector es: 1
La componente 2  del vector es: 4
La componente 3  del vector es: 9
La componente 4  del vector es: 16
La componente 5  del vector es: 25
La componente 6  del vector es: 36
La componente 7  del vector es: 49
La componente 8  del vector es: 64
La componente 9  del vector es: 81
La componente 10  del vector es: 100
```

Para leer por teclado un valor numérico para un elemento de una tabla, se utilizará la función scanf( ) poniendo el símbolo & delante del nombre de la componente. Por ejemplo, la siguiente sentencia lee un entero en la componente de índice 2 del vector prueba:

```
scanf ( "%d", &prueba[2]);
```

C no hace ninguna comprobación de los límites de las tablas. Esto significa que si declaro el vector a como de 5 componentes, el compilador permite acceder a su componente 10: a[9], aunque al final se produzcan fallos en la ejecución del programa o resultados inesperados. Debe ser el programador el que se asegure de no sobrepasar los límites de la tabla.

En C no se puede asignar una tabla completa a otra. El siguiente fragmento de programa es erróneo:

```
int a[10] , b[10];
……….
……..
a=b; /* es erróneo*/
```

Si se quieren copiar todos los elementos de una tabla a otra se debe realizar la operación componente a componente:

```
for (i=0; i<10; i++) a[i] = b [i];
```

Tampoco se permiten operaciones del estilo: `a=3;`

2.2.2 Cadenas

El uso más común de un vector en C es la cadena. Como se estudió en el capítulo anterior, C no incorpora un tipo de datos de cadena de caracteres; en lugar de eso, para manejar cadenas hay que definir vectores de tipo `char`.

Más concretamente, una cadena se define como un vector de caracteres con un carácter de terminación nulo. Esta última característica es fundamental para la definición de una cadena, si no termina en carácter nulo C supone que no estamos manejando una cadena sino un vector tipo `char`. Existen funciones que actúan

sobre cadenas y no sobre vectores char, por ello, la definición del carácter nulo es tan importante. En C el carácter nulo es el que tiene código 0. El hecho de que la cadena deba terminar en un carácter nulo significa que al declararla se debe incrementar en uno la longitud máxima que la cadena deba contener, para que se pueda memorizar todo más el carácter nulo.

Ejemplo de declaración de cadena:

```
char texto[81];
/*
se pueden almacenar 80 caracteres y el carácter nulo.
*/
```

La inicialización de una cadena se realiza en la sentencia de declaración asignándola una constante de cadena.

En una constante de cadena (texto entre doble comilla inicial y final) el compilador pone automáticamente un carácter nulo al final de la cadena, por ello el programador no debe añadir el carácter nulo en la inicialización.

Ejemplo de declaración e inicialización de una cadena:

```
char texto[81]="esto es una cadena";
```

Tal como se adelantó en el capítulo precedente, para leer una cadena por teclado (toda de una vez), se puede utilizar la función gets( ). Pertenece al archivo de cabecera stdio.h. Funciona leyendo una cadena de caracteres hasta el primer salto de carro; el salto de carro no lo almacena sino que lo sustituye por el carácter nulo. Se utiliza introduciendo el nombre del *array* como argumento en la función.

Ejemplo:

```
#include <stdio.h>
 void main( )
 {
 char cad[80];
 int i;
 printf("Introduce una cadena de caracteres (menos de 80)\n");
 gets (cad); /*leemos la cadena hasta el salto*/
   for (i=0; cad[i];i++) printf ("%c",cad[i]);
 }
```

Obsérvese la forma de escribir en pantalla la cadena de caracteres. Mediante un bucle se escribe carácter a carácter (cad[i]) hasta que la expresión sea falsa (valor 0); la expresión es cad[i], es decir el código de cada carácter, y será falsa cuando el código del carácter sea nulo, en este caso, cuando se llegue al último carácter almacenado de la cadena.

Para imprimir un solo carácter existe la alternativa de utilizar la función putchar(c), siendo c la variable tipo char a escribir:

```
printf ("%c",cad[i]);
```

es equivalente a

```
putchar(cad[i]);
```

Una forma más sencilla de escribir la cadena es utilizando la función **printf( )**:

```
printf(cad); /* se utiliza la función con un único argumento */
printf ("%s \n", cad); /* cuando se pongan dos argumentos, entonces es
                          necesario especificar formato de cadena %s */
```

También puede escribirse una cadena con la función **puts ( )**:

```
puts(cad); /* es la única forma de utilizar esta función */
```

Se puede leer una cadena carácter a carácter, para ello se deben utilizar las funciones **getchar( )** o **getche( )**.

1) Lectura con getchar( )

Es una función de la librería stdio.h
Aparece a continuación un primer ejemplo con el que se analiza el funcionamiento de getchar:

```
# include <stdio.h>
void main ()
{
  int i;
  char c;
  printf("Teclea cualquier caracter, X para finalizar\n");
  for (i=0;c!='X';i++){
                    c=getchar( );
            putchar (c);
   }
            printf("\n Fin del programa");
 }
```

En este ejemplo, se repite mediante un bucle la lectura de un carácter que se escribe a continuación en pantalla; cuando el usuario escriba 'X' el programa finaliza. Si se ejecuta el programa, quizás no funciona como creeríamos en un principio. Esto es debido a que getchar no lee los caracteres hasta que se pulse un salto de línea, es entonces cuando lee y deja que se ejecuten las instrucciones que aparecen por detrás en el programa. El salto de línea también lo toma como un carácter introducido. Así si introducimos por teclado:

Abc
NhXcvb

La salida será:

Teclea cualquier carácter, X para finalizar
Si el usuario introduce: Abc
el *programa responde*: Abc

Si el usuario introduce: NhXcvb
el programa responde: NhX

Fin del programa

A continuación se utiliza getchar para la lectura de una cadena:

```
#include<stdio.h>
void main( )
{
 char nombre[30];
       int i;
 for(i=0;(nombre[i]=getchar( )) != '\n';i++)  ;
 nombre[i]=0;/* se introduce el carácter nulo en
      la posición siguiente a la última leida*/
 puts(nombre);
}
```

El lector debe prestar especial atención a la expresión del bucle:

(nombre[i]=**getchar( )**) != '\n'

en ella se efectúa la lectura y la comprobación de si el carácter leído es un salto de línea. Una vez finalizada la lectura de los caracteres de la cadena se debe memorizar el nulo:

nombre[i]=0

2) Utilización de getche

Es una función que se encuentra en la librería <conio.h>
Lee el carácter inmediatamente después que se ha escrito, sin esperar a un salto de línea. Por tanto, una vez introducido el carácter se lee y se pasa a la siguiente sentencia del programa. A continuación se analizan los mismos ejemplos del punto anterior empleando la función getche( ):

```
# include <stdio.h>
# include <conio.h>
void main (  )
{
 int i;
 char c;
  printf("Teclea cualquier caracter, X para finalizar");
  for (i=0;c!='X';i++)
  {
                c=getche( );
                putchar (c);
  };
                printf("\n Fin del programa");
 }
```

Si se introduce por teclado: abcX la salida será:

Teclea cualquier caracter, X para finalizar
aabbccXX (de estas parejas de caracteres el primero lo escribe el usuario y el segundo es la respuesta del programa)
Fin del programa

La función getche ( ) no lee el carácter fin de línea como tal, por ello no permite su utilización en un programa similar al visto con getchar( ) para lectura de una cadena.
Para leer una cadena carácter a carácter es necesario utilizar getchar( ).

La biblioteca estándar de C suministra muchas funciones relacionadas con cadenas. Las cuatro más importantes son strcpy( ), strcat( ), strcmp( ) y strlen( ). Se estudiarán en el capítulo 5.

2.2.3 Tablas multidimensionales

Se pueden crear tablas de dos o más dimensiones, generalizando lo estudiado para una dimensión. Por ejemplo, para declarar una matriz entera llamada m de 3 filas y 4 columnas se utiliza la sentencia:

```
int m[3][4];
```

En la memoria principal los elementos de la matriz se almacenan de forma lineal, ordenados por filas, es decir, variando más rápido el índice derecho.

Véase un ejemplo en el que se declara una matriz entera de 4x5, y se almacena en cada elemento el producto de sus índices, al final se escriben en pantalla los elementos de la matriz por filas (cada fila en una línea de pantalla).

```
#include <stdio.h>
void main( )
{
   int mat [4][5];
   int i,j;
 for (i=0; i<4; i++) for (j=0;  j<5; j++) mat[i][j]=i*j;
 for (i=0; i<4; i++)
            {
            for (j=0;  j<5; j++) printf("%d ", mat[i][j]);
            printf("\n");
            }
}
```

La salida a pantalla es:

```
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
```

Para crear tablas de 3 basta con añadir el tamaño de la dimensión adicional. Ejemplo:

```
float tri [3][5][4];
```

Se pueden crear tablas de más de 3 dimensiones, pero no suele hacerse en la práctica debido a la gran cantidad de memoria que consumen.

2.2.4 Inicialización de tablas

A los elementos de una tabla se les puede asignar valores iniciales especificando una lista de valores en su declaración según la siguiente sintaxis:

tipo nombre_tabla [tamaño]= {lista de valores};

La lista de valores es una lista de constantes separadas por comas de tipo compatible con el tipo de la tabla. La primera constante se coloca en la primera posición de la tabla, la segunda en la segunda, y así sucesivamente. Ejemplos:

```
int a[5] = {1,4,9,16,25};
char c[3]= {'A','B','C'};
```

Si se trata de inicializar un vector de caracteres que va a contener una cadena, se realiza de la forma ya analizada:

```
char nombre[5]="Pepe";
```

Se observa que en este caso de inicialización no se incluyen las llaves. Puesto que las cadenas en C deben terminar con un carácter nulo, el vector se debe declarar lo suficientemente largo para incluir todos los caracteres a memorizar más el nulo. (Cuando se utiliza una constante de cadena, el compilador suministra automáticamente el carácter de terminación nulo).

Las tablas multidimensionales se inicializan del mismo modo que las unidimensionales:

```
void main( )
{
  int mat [3][3]= {1,2,3,4,5,6,7,8,9};
  int j,i;
  for (j=0; j<3; j++)
  {
    for (i=0; i<3; i++) printf("%d ", mat[i][j]);
    printf("\n");
   }
 }
```

En el ejemplo anterior se inicializa la tabla de dimensión 2 separando entre comas los valores de sus 9 elementos. Estos valores se deben introducir entre las llaves por filas ya que así están dispuestos los elementos de mat en la memoria. Mediante la utilización de los bucles for se escriben los elementos de la matriz, en este caso, por columnas (cada columna en una línea de pantalla). La salida a pantalla es:

```
1 4 7
2 5 8
3 6 9
```

Cuando se está inicializando una tabla unidimensional se puede dejar sin especificar su tamaño, dejando vacío el interior de los corchetes. En ese caso, el compilador cuenta el número de constantes de inicialización y utiliza ese valor como tamaño. Ejemplo:

```
float tab[ ]= {1,3,5,7,9,11};
```

El compilador crea una tabla de 6 elementos.

Las tablas que no tienen sus dimensiones especificadas explícitamente se denominan *tablas de tamaño indeterminado*. Son útiles, ya que el tamaño cambia automáticamente cambiando la inicialización y no hay que contar los elementos para dar el tamaño; es especialmente útil en la inicialización de cadenas. Ejemplo:

```
char prompt[ ] = "Introduzca su nombre: ";
```

Para inicializar las tablas multidimensionales con tamaño indeterminado, se deben fijar todas las dimensiones salvo la de más a la izquierda:

```
int mat [ ][3]= {1,2,3,4,5,6,7,8,9};
```

De esta forma, se puede alargar o acortar la tabla sin cambiar las dimensiones.
Con la siguiente inicialización, el compilador genera una matriz de 4x3 con los dos últimos elementos de valor 0.

```
int mat [ ][3]= {1,2,3,4,5,6,7,8,9,10};
```

2.3 Estructuras

2.3.1 Estructuras simples

Una estructura es un tipo de datos definido por el usuario que está compuesto por dos o más elementos relacionados. Los elementos de la estructura pueden ser de tipos distintos.

Sintaxis:

```
struct nombre {
 tipo elemento1;
 tipo elemento2;
     .............
     .............
     .............
 tipo elementoN;
}lista_de_variables;
```

donde:

nombre es el nombre de la estructura, es decir, el nombre del nuevo tipo de datos creado.

tipo es un tipo de datos C válido.

lista_de_variables es el conjunto de variables que se declaran de tipo estructura nombre.

A los elementos de la estructura se les denomina *campos*.

Ejemplo:

```
struct alumno {
        char nombre[20];
        char apellidos [40];
        int matricula;
        float nota;
} curso;
```

alumno es el nombre de la estructura, es decir, el nombre del nuevo tipo de datos creado, NO ES EL NOMBRE DE UNA VARIABLE. La única variable definida es curso, que se almacenará en memoria de la siguiente forma:

curso			
nombre (20 bytes)	apellidos (40 bytes)	matricula (4 bytes)	nota (4 bytes)
curso.nombre	curso.apellidos	curso.matricula	curso.nota

Para acceder a un campo de una variable estructura hay que especificar tanto el nombre de la variable como el del campo, separados por un punto, por ejemplo

```
curso.nota = 8.3
```

Una vez que se ha definido la estructura, se pueden crear más variables del tipo estructura en cualquier parte del programa restante con sentencias de la forma:

```
struct nombre lista_de_variables;
```

por ello no es necesario definir todas las variables cuando se define el tipo estructura.

Por ejemplo, si se quieren definir tres variables c1, c2, c3 de tipo *estructura alumno*:

```
struct alumno c1,c2,c3;
```

Además, se pueden definir tablas de tipo estructura:

```
struct alumno metodos[100];
```

metodos es un vector de 100 componentes de tipo *estructura alumno*.

Para acceder a una estructura individual de la tabla se utilizan los índices. Por ejemplo, la siguiente sentencia accede a la primera estructura :

```
metodos[0]
```

Para acceder a un campo específico:

```
metodos[8].matricula=22987
```

/* asigna el valor 22987 al campo matricula de la novena estructura del vector metodos*/

Se puede asignar el contenido de una variable de estructura a otra variable de estructura del mismo tipo, tal como se aplica en el siguiente programa:

```
# include <stdio.h>
void main(  )
{
 struct prueba {
      int a;
      float b;
 } v1,v2;
 v1.a=7;
 v1.b=8.3;
 v2=v1;
printf("%d %f", v2.a,v2.b);
 }
```

La salida a pantalla es: 7 8.300000

La lectura y escritura de datos de una estructura tiene que realizarse campo a campo, ya que se debe especificar el formato de todos los datos. Por ello en el ejemplo anterior aparece la sentencia:

```
printf("%d %f", v2.a,v2.b);
```

y no sería válido sustituirla por:

```
printf("%d %f", v2);
```

Los nombres de los campos de la estructura no entran en conflicto con otras variables del programa que se llamen igual, debido a que el nombre del campo, por sí solo, no es una variable ya que está vinculado al de la variable de tipo estructura.

Véase el siguiente ejemplo:

```
# include <stdio.h>
void main(  )
{
    struct prueba {
                int a;
                float b;
     } c;
   int a;
   a=5;
   c.a=7;
   c.b=10.8;
    printf("%d %d %f", a, c.a,c.b);
}
```

Se tiene la variable entera a, y el campo a de la estructura prueba vinculado a la variable estructura c. No hay conflicto entre la variable a y el campo a.

La salida a pantalla es:

```
5  7  10.800000
```

2.3.2 Estructuras anidadas

Hasta ahora se ha trabajado con estructuras cuyos elementos están formados por tipos básicos C; sin embargo, cabe la posibilidad de que los elementos de la estructura sean también otras estructuras.

En el siguiente ejemplo se ha creado una estructura alumno formada por tipos simples, además se ha creado una estructura metodos en la que uno de los

campos es la primera estructura. La variable estructura que se define es el vector lista de 100 elementos. Ni alumno ni metodos son variables.

```
#include <stdio.h>
void main ( )
{
        struct alumno {
                char nombre[20];
                char apellidos[40];
                int matricula;
        };

        struct metodos {
                float parcial1;
                float parcial2;
                float junio;
                struct alumno grupo_a;
        } lista[100];
        lista[0].grupo_a.matricula=23456;
        printf("El numero de matricula del primer alumno es: %d",
lista[0].grupo_a.matricula);
}
```

Un elemento de lista aparecerá en memoria de la siguiente forma:

<table>
<tr><td colspan="3"></td><td>nombre</td><td>apellidos</td><td>matricula</td></tr>
<tr><td>parcial1</td><td>parcial2</td><td>junio</td><td colspan="3">grupo_a</td></tr>
<tr><td colspan="6">lista[i]</td></tr>
</table>

Tiene 4 campos, y el cuarto campo a su vez tiene 3 sub-campos.

Para referenciar un campo de una estructura encajada se empieza por la más externa continuando hacia la más interna.

2.4 Enunciados de las prácticas

TABLAS

Ejercicio 1. Declaración de tablas e inicialización

Determinar la salida a pantalla del siguiente programa C analizando los valores que toman todas las componentes de las tablas.

```
#include <stdio.h>

void main( )
{
        float a[3][3]={1,2,3,4,5,6,7,8,9};
        int b[4]={10,9,8,7};
        char cadena[ ]="Estructuras de datos";
        int c [ ][3]= {1,0,0,0,0,1};

        printf("%f\n", a[2][2]);
        printf("%d\n", b[1]);
        printf("%c\n", cadena[2]);
        printf("%d\n", c[1][1]);
}
```

¿Cuáles son las dimensiones de las tablas de tamaño indeterminado que aparecen en el programa?

Ejercicio 2. Cadenas de caracteres. Lectura con getchar ( )

La función getchar( ) actúa leyendo un carácter por teclado.

Diseñar un programa C que, mediante un bucle for, lea una cadena por teclado utilizando la función getchar( ). La cadena introducida terminará cuando el usuario pulse el salto de línea y se debe almacenar en la variable *cadena* de 80 bytes.
Además, se debe imprimir en pantalla la cadena leída después de la frase,

La cadena que has escrito es:

Ejercicio 3. Matrices

Las tablas de dos dimensiones, se pueden "identificar" con el concepto matemático de matriz: el primer índice hace referencia a lo que denominamos filas y el segundo a las columnas. Hay que tener en cuenta que la memoria del ordenador es una estructura "lineal", es decir, los valores, almacenados en binario, se sitúan unos a continuación de otros; en el caso de las matrices, las distintas componentes ocupan posiciones consecutivas de memoria, y se almacenan por filas, es decir, primero la primera fila, después la segunda y así sucesivamente. De hecho se suele pensar que una matriz es un vector de vectores.
Se pide:

1. Escribir un programa C para leer los valores de una matriz P de tipo entero con 3 filas y 4 columnas, por columnas; una vez leídos estos datos se imprimirán en pantalla por filas.
2. Resolver el problema inverso: leer la misma matriz por filas e imprimirla por columnas.

Realizar los dos apartados de todas las formas posibles: leyendo e imprimiendo elemento a elemento, fila a fila, columna a columna,....

Ejercicio 4.

Observar con atención el siguiente programa C, y determinar la salida a pantalla:

```
#include <stdio.h>

#define FILAS      4
#define COLUMNAS   2

void main(  )
{
   int fila, columna;
   int datos[4][2] = {11, 22, 33, 44, 55, 66, 77, 88 };

   for (fila = 0; fila < FILAS; ++fila)
      for (columna = 0; columna < COLUMNAS; ++columna)
         printf("datos[%d][%d] = %d\n",fila, columna, datos[fila][columna]);
}
```

Ejercicio 5.

Diseñar un programa que lea por teclado las temperaturas medias de cada día de un mes cualquiera del año (como número entero) almacenándolas en el vector *temp* y que calcule la temperatura media del mes (como número real). La salida a pantalla debe ser:
La temperatura media del mes es .....

A continuación se debe obtener la desviación de la temperatura de cada día respecto a la media, almacenándola en el vector *desv* e imprimiendo en pantalla el resultado de la forma:
En el día 1 la desviación ha sido de 0.6 grados respecto a la media
En el día 2 .....

ESTRUCTURAS

Ejercicio 6 . Vector de estructuras

Sea el siguiente programa C:

```
#include <stdio.h>

void main(  )
{
   struct alumno {
                char nombre[20];
                char apellido [40];
                int nota;
         } metodos [4];

   printf("Introducir los datos del alumno 1: nombre y apellidos\n");
   gets (metodos[0].nombre);
   gets (metodos[0].apellido);
   metodos[0].nota=7;
   printf("El alumno %s %s tiene una calificacion de %d\n",
                metodos[0].nombre, metodos[0].apellido,metodos[0].nota);
}
```

Se pide realizar un dibujo de la memoria principal para situar las cuatro estructuras del vector METODOS, y determinar la salida suponiendo que se introdujesen los datos:

MARIA (nombre)
MARTÍN PÉREZ (apellidos)

Ejercicio 7. Estructuras anidadas

Escribir un programa C que realice las siguientes tareas:

- Diseñar una estructura de nombre ACADEMICO que contenga cuatro campos: uno de tipo carácter de longitud 30 llamado *titulacion*, otro de tipo carácter de longitud 30 llamado *especialidad*, otro entero llamado *fechatit* y un último campo entero llamado *master* (para almacenar información de tipo lógica).
- Diseñar una estructura de nombre FICHAS que contenga cuatro campos: uno de tipo carácter de longitud 10 llamado *nombre*, otro de tipo carácter de longitud 20 llamado *apellidos*, otro entero llamado *fechanac* y un último campo llamado *datosacad* de tipo estructura ACADEMICO.
- Declarar el vector estructura EMPLEADOS de 100 componentes de tipo FICHAS.
- Escribir una sentencia de lectura para leer del teclado todos los datos referentes al empleado número 3: nombre, apellidos, fecha de nacimiento, titulación, especialidad, año en que la obtuvo y si ha hecho algún master.
- Escribir una sentencia de escritura para imprimir en pantalla todos los datos introducidos en el apartado anterior.

2.5 Soluciones a las prácticas del capítulo 2

TABLAS

Ejercicio 1. Declaración de tablas e inicialización

La salida a pantalla es la siguiente:

```
9.000000
9
t
0
```

El vector cadena tiene una longitud de 21 bytes (20 caracteres más el nulo). La matriz c cuenta con 2 filas y 3 columnas (6 elementos).

Ejercicio 2. Cadenas de caracteres. Lectura con getchar ( )

Un programa adecuado es el siguiente:

```
#include <stdio.h>

void main(  )
{
        char cadena[80];
        int i;

        for (i=0;(cadena[i]=getchar())!='\n'; i++);
        cadena[i]=0;
        printf ("La cadena que has escrito es: %s\n", cadena);

}
```

Este programa repite el bucle mientras la condición lógica sea verdadera, es decir, mientras el carácter leído no sea un salto de línea. Cuando el usuario introduce un salto de línea la condición lógica vale falso y el bucle deja de ejecutarse. Fíjense como la condición lógica: (cadena[i]=getchar( ))!='\n' lleva implícita la lectura del carácter, que se almacena en cadena[i] analizando a continuación si es distinto de '\n'.

La sentencia posterior a la finalización del bucle es la que sustituye el carácter '\n' por el carácter nulo.

A continuación aparece otra solución que hace uso de una variable auxiliar tipo char:

```
#include <stdio.h>

void main( )
{
        char cadena[80], aux;
        int i;

        for (i=0;aux!='\n'; i++) aux=cadena[i]=getchar( );
        cadena[i-1]=0;
        printf ("La cadena que has escrito es: %s\n", cadena);

}
```

Otro bucle válido para este último caso sería:

```
        for (i=0; cadena[i-1]!='\n'; i++)cadena[i]=getchar( );
```

Ejercicio 3. Matrices

I. Se lee por columnas y se escribe por filas

I.a Todas las componentes seguidas

```
#include <stdio.h>

void main( )
{
        int p [3][4], i, j;

        /*se pide al usuario que introduzca todas las componentes seguidas*/
        printf ("Introduce las componentes de la matriz por columnas:\n");
        for (j=0; j<4;j++)
                for(i=0;i<3;i++)
                        scanf("%d", &p[i][j]);
```

```
        printf ("Las componentes de la matriz ordenadas por filas son:\n");
                /*se escriben seguidas*/
        for(i=0;i<3;i++)
                for (j=0; j<4;j++)
                      printf("%d ", p[i][j]);
}
```

I.b. Las componentes una a una

```
#include <stdio.h>

void main(  )
{
        int p [3][4], i, j;
        /*se piden al usuario las componentes una a una*/
        for (j=0; j<4;j++)
             for(i=0;i<3;i++)
             {
                 printf ("Introduce la componente [%d][%d] de la matriz:\n",i,j);
                 scanf("%d", &p[i][j]);
              }
        printf ("Las componentes de la matriz por filas son (una a una):\n");
/*se escribe cada componente en una línea de pantalla*/
        for(i=0;i<3;i++)
             for (j=0; j<4;j++)
                  printf("%d\n", p[i][j]);
}
```

I.c. Las componentes fila a fila (columna a columna en lectura)

```
#include <stdio.h>

void main(  )
{
  int p [3][4], i, j;

  /* se pide al usuario las componentes de cada columna*/
  for (j=0; j<4;j++)
  {
   printf ("Introduce las componentes de la columna %d de la matriz:\n",j+1);
   for(i=0;i<3;i++)   scanf("%d", &p[i][j]);
   }
```

```
  printf ("Las componentes de la matriz por filas son (fila a fila):\n");
  /* cada fila de la matriz se escribe en una línea de pantalla*/
  for(i=0;i<3;i++)
  {
        for (j=0; j<4;j++)  printf("%d ", p[i][j]);
        printf("\n");
  }

}
```

II. Ahora se lee en orden filas y se escribe por columnas

II.a. Todas las componentes seguidas

```
#include <stdio.h>

void main(  )
{
    int p [3][4], i, j;
    printf ("Introduce las componentes de la matriz por filas:\n");
        for (i=0; i<3;i++)
                for(j=0;j<4;j++)
                        scanf("%d", &p[i][j]);

    printf ("Las componentes de la matriz por columnas son (todas
seguidas):\n");

        for(j=0; j<4;j++)
                for (i=0;i<3;i++)  printf("%d ", p[i][j]);
}
```

II.b. Las componentes una a una

```
#include <stdio.h>

void main(  )
{
    int p [3][4], i, j;

    for (i=0; i<3;i++)
         for(j=0;j<4;j++)
         {
                  printf ("Introduce la componente [%d][%d] de la matriz:\n",i,j);
                  scanf("%d", &p[i][j]);
         }

    printf ("Las componentes de la matriz ordenadas por columnas son :\n");

    for(j=0; j<4;j++)
                  for (i=0;i<3;i++)  printf("%d \n", p[i][j]);
}
```

II.c. Las componentes columna a columna (fila a fila en lectura)

```
#include <stdio.h>

void main(  )
{
    int p [3][4], i, j;
    for (i=0; i<3;i++)
   {
           printf ("Introduce las componentes de la fila %d de la matriz:\n",i+1);
           for(j=0;j<4;j++)     scanf("%d", &p[i][j]);
    }

    printf ("Las componentes de la matriz por columnas son:\n");
/* cada columna se escribe en una línea de pantalla */
         for(j=0; j<4;j++)
         {
                  for (i=0;i<3;i++)  printf("%d ", p[i][j]);
                  printf("\n");
         }

}
```

Ejercicio 4.

La salida a pantalla es la siguiente:

datos[0][0] = 11
datos[0][1] = 22
datos[1][0] = 33
datos[1][1] = 44
datos[2][0] = 55
datos[2][1] = 66
datos[3][0] = 77
datos[3][1] = 88

Ejercicio 5.

Un posible programa es el siguiente:

```
# include <stdio.h>

void main(  )
{
        int temp[31], n_dias;
        int i;
        float desv[31],media=0.;
        printf("Introduce el numero de dias del mes:");
        scanf("%d", &n_dias);
        for (i=0; i<n_dias; i++)
        {       printf("Introduce la temperatura media del día %d:",i+1);
                scanf("%d", &temp[i]);
                media=media+temp[i];
        }
        media=media/n_dias;
        printf("La temperatura media del mes es %f\n", media);

        for (i=0; i<n_dias; i++)
        {
          desv[i]=temp[i]-media;
          printf("En el dia %d la desviacion ha sido de %f respecto a la media\n",
           i+1,desv[i]);
         }
}
```

ESTRUCTURAS

Ejercicio 6 . Vector de estructuras

La salida a pantalla es la indicada a continuación:

Introducir los datos del alumno 1: nombre y apellidos
MARIA
MARTIN PEREZ
El alumno MARIA MARTIN PEREZ tiene una calificacion de 7

Ejercicio 7. Estructuras anidadas

Seguidamente se ofrece un programa válido:

```
void main()
{
        struct academico {
                char titulacion[30];
                char especialidad [30];
                int fechatit;
                int master;
        };

        struct fichas {
                char nombre[10];
                char apellidos [20];
                int fechanac;
                struct academico datosacad;
        } empleados[100];

        printf("Introducir los datos del empleado 3: nombre,apellidos, fecha de nacimiento, titulacion, especialidad, fecha de titulacion y si ha hecho un master\n");
scanf ("%[^\n] %[^\n] %d %[^\n] %[^\n] %d %d",
                        &empleados[2].nombre,&empleados[2].apellidos,
                &empleados[2].fechanac, &empleados[2].datosacad. titulacion,
                &empleados[2].datosacad.especialidad,
                &empleados[2].datosacad.fechatit,
                &empleados[2].datosacad.master);
```

```
printf("El empleado %s %s, nacido el %d tiene la titulación de %s y especialidad %s obtenida en %d. Tiene un master:(si (1), no(0))........%d\n",
            empleados[2].nombre,empleados[2].apellidos,
            empleados[2].fechanac, empleados[2].datosacad. titulacion,
            empleados[2].datosacad.especialidad,
            empleados[2].datosacad.fechatit,
            empleados[2].datosacad.master)
```