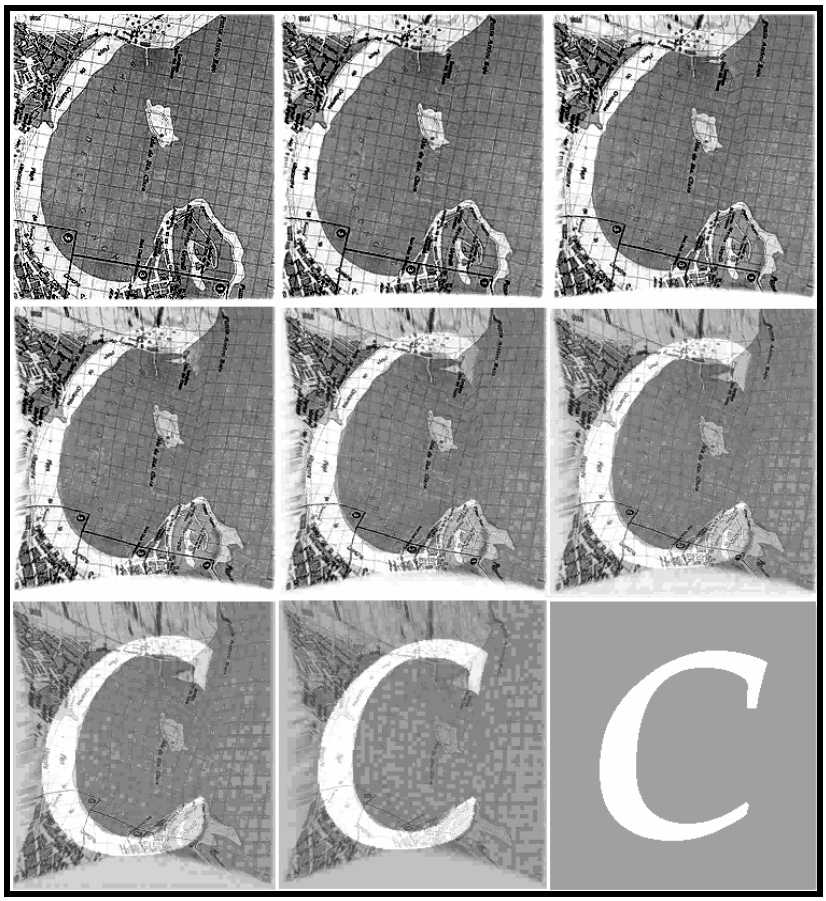
ESCUELA SUPERIOR DE INGENIEROS INDUSTRIALES
UNIVERSIDAD DE NAVARRA

INDUSTRI INJINERUEN GOIMAiLAKO ESKOLA
NAFARROAKO UNIBERTSITATEA



Practique Lenguaje ANSI C
como si estuviera en Primero

Madrid, 31 mayo de 2003

Profesores:

Javier García de Jalón de la Fuente
José Ignacio Rodríguez Garrido
Rufino Goñi Lasheras
Alfonso Brazález Guerra
Patxi Funes Martínez
Rubén Rodríguez Tamayo

ÍNDICE

INTRODUCCIÓN

Este manual recoge los ejercicios de programación en lenguaje ANSI C realizados en las prácticas de la asignatura ***Informática 1***, en el Primer Curso de la Escuela Superior de Ingenieros Industriales de San Sebastián (Universidad de Navarra), desde el curso 1993-94 al curso 1995-96.

Esta colección de ejercicios se utilizó solamente en el curso 1995-96, y nunca llegó a publicarse en Internet. Sin embargo, es lógico considerarla como el complemento imprescindible a los apuntes "Aprenda ANSI C como si estuviera en Primero", que no contienen ejemplos o ejercicios resueltos.

Aunque con cierto retraso, estos ejemplos se publican ahora en formato PDF, esperando que ayuden a aprender a programar a muchos estudiantes o simples aficionados a la informática.

Los distintos ejercicios están agrupados en "prácticas". De cada uno de ellos se incluye:

- Un enunciado que describe el programa a realizar, tal como se planteaba a los alumnos.
- El programa correspondiente al ejercicio resuelto.
- Unos breves comentarios sobre los aspectos del ejercicio resuelto a los que convenga prestar más atención.

Para facilitar la tarea a los usuarios de esta colección de ejercicios se facilita un archivo llamado ***programas.zip*** en el que se incluyen los ficheros correspondientes a todos los ejercicios resueltos, de forma que el lector no necesite teclear o escanear ningún programa. Es posible que alguno de los programas incluidos contenga algún error; se agradecerá recibir noticia de ello para corregirlo y facilitar el trabajo a los futuros lectores. De dichos programas se han eliminado los caracteres especiales (vocales acentuadas, ñ, ¿, ¡, etc.) de modo que la salida sea la misma independientemente del entorno en que se compile y ejecute el programa.

Madrid, mayo de 2003

Javier García de Jalón de la Fuente (jgjalon@etsii.upm.es)

PRACTICA 1.

EJERCICIO 1.1: EL PRIMER PROGRAMA.

Para que tu debut como programador en C sea todo un éxito, te hemos preparado un sencillo programa de bienvenida. Tecléalo en un fichero al que puedes poner por nombre ***hola.c***.

Solución comentada al Ejercicio 1.1.

```
/* fichero hola.c */
/* Este programa saluda desenfadadamente. */

#include <stdio.h>
void main(void) {
    printf("Hola! Que tal estas, futuro programador?\n");
}
```

Comentario: La primera línea de este programa es un *comentario*, y es ignorado por el compilador. La directiva ***#include*** permite utilizar la librería ***stdio***, indispensable para diferentes instrucciones de entrada/salida del lenguaje C. El fichero ***stdio.h*** contiene las *declaraciones* de las funciones de entrada/salida, así como definiciones de constantes simbólicas y algunas otras definiciones de utilidad. La palabra ***void*** es opcional; indica que la función ***main()*** no tiene *valor de retorno* ni *argumentos*.

EJERCICIO 1.2: UNA CONVERSACIÓN EN C.

Seguro que el programa del ejercicio anterior te ha dejado con ganas de responder a la pregunta que aparece en la pantalla. Para ello es necesario utilizar la función ***scanf()***. Esta función permite leer tanto números como cadenas de caracteres, pero cuando encuentra blancos, tabuladores o espacios en blanco, termina de leer. Crea el siguiente programa en un fichero llamado ***hola2.c***.

Solución comentada al Ejercicio 1.2.

```
/* fichero hola2.c */
/* Este programa saluda más personalmente */

#include <stdio.h>
void main(void) {
    char nombre[30];
    printf("Hola! Como te llamas?\n");
    scanf("%s", nombre);
    printf("Que tal estas, %s?\n", nombre);
}
```

Comentario: La sentencia ***char nombre[30]***, declara una variable llamada ***nombre*** que es una cadena de 30 caracteres (tipo *char*). Estos caracteres se numeran del 0 al 29, y deben incluir la marca de fin de cadena **'\0'**. Con la función ***scanf()***, se lee lo que será el contenido de dicha cadena por medio del formato **%s**, que es propio de las cadenas de caracteres. Como la lectura se detiene al encontrar un blanco, un carácter nueva línea o un tabulador, por medio de este programa no se puede leer una frase completa, sino sólo una palabra.

Observa que a la función ***scanf()*** hay que pasarle los argumentos *por referencia*. Como ***nombre*** es de por sí la dirección de ***nombre[0]*** no hace falta precederlo por el operador (&).

EJERCICIO 1.3: UNA PEQUEÑA OPERACIÓN ARITMÉTICA.

Estarás pensando que C ha de servir para algo más que mantener una aburrida conversación con tu PC (¿verdadero y fiel amigo?). En el siguiente programa te presentamos un avance de las "complicadas" operaciones que puede realizar el lenguaje C. Escribe el programa y sálvalo como ***marathon.c***.

Compila el programa y ejecútalo; apuntando el resultado. Después modifica el programa sustituyendo `1760.0` por `1760` en la línea que calcula el número de kilómetros. Vuelve a compilar y a ejecutar. ¿Sale lo mismo que antes? ¿Qué ha podido pasar?

Solución comentada al Ejercicio 1.3.

```
/* fichero marathon.c */
/* Un marathon tiene 26 millas y 385 yardas. */
/* Una milla tiene 1760 yardas. */
/* Calcula la distancia del marathon en kilómetros. */

#include <stdio.h>
void main(void) {
    int    millas, yardas;
    float  kilometros;

    millas=26;
    yardas=385;
    kilometros=1.609*(millas+yardas/1760.0);
    printf("\nUn marathon tiene %f kilometros.\n\n", kilometros);
}
```

Comentario: En C las constantes que incluyen un punto decimal son de tipo ***double***. La variable ***yardas*** es de tipo ***int***. Si en el denominador se pone sólo ***1760***, el resultado de ***yardas/1760*** es entero y por tanto incorrecto. Basta poner ***1760.0*** para que ***yardas*** sea promovido a ***double*** y todas las operaciones aritméticas de esa sentencia se realicen con precision ***double***.

EJERCICIO 1.4: PREGUNTAS INDISCRETAS.

En este programa vas a utilizar la función ***scanf()*** con distintos tipos de variables. De paso podrás contestar a algunas preguntas indiscretas, pero de indudable interés estadístico y social. El siguiente programa debe ser almacenado en un fichero llamado ***mascota.c***.

Si tu mascota favorita es una boa, una ardilla o una gacela, tendrás que cambiar el artículo "un" por "una", para respetar la concordancia.

Solución comentada al Ejercicio 1.4.

```
/* fichero mascota.c */

#include <stdio.h>
void main(void)
{
    int   edad;
    float sueldo;
    char  cachorro[30];

    printf("Confiesa tu edad, sueldo y mascota favorita.\n");
    scanf("%d %f",&edad, &sueldo);
    scanf("%s", cachorro);
    printf("%d %.0f pts. %s\n",edad, sueldo, cachorro);
    printf("Un %s!!", cachorro);
    printf(" Como te puede gustar un %s?\n", cachorro);
}
```

Comentario: En la función ***scanf()***, se incluye el ***operador dirección (&)*** delante de las variables escalares para pasar a la función las direcciones de dichas variables (paso de argumentos ***por referencia***). De esta forma la función ***scanf()*** puede depositar en las direcciones de memoria correctas los valores que lee desde teclado. Recordemos que para leer cadenas de caracteres basta poner el ***nombre*** de la cadena, que de por sí ya es una ***dirección***.

EJERCICIO 1.5: PROGRAMA QUE SUMA LOS CINCO PRIMEROS NÚMEROS NATURALES.

Se presenta a continuación un programa que utiliza la sentencia ***while*** para definir un bucle. El programa sumará de forma automática los cinco primeros números naturales. Sálvalo con el nombre ***sumaInt.c***.

Solución comentada al Ejercicio 1.5.

```
/* fichero sumaInt.c */
/* Programa para calcular la suma de los enteros del 1 al 5 */

#include  <stdio.h>
void main(void) {
    int i=1, suma=0;

    while (i<=5) {       /* se ejecuta el bloque mientras i<=5 */
        suma+=i;         /* equivale a  suma=suma+i; */
        ++i;             /* equivale a  i=i+1; */
    }
    printf("suma = %d\n", suma);
}
```

Comentario: El bucle ***while*** realiza la sentencia simple o compuesta que le sigue mientras la condición definida entre paréntesis sea verdadera (es decir distinta de cero). El bucle ***while*** del programa anterior podía también haberse escrito en la siguiente forma (más compacta):

```
    while (i<=5)          /* se ejecuta el bloque mientras i<=5 */
        suma += i++;      /* equivale a  suma=suma+i e i=i+1; */
```

No hacen falta las llaves porque sólo hay una sentencia simple detrás del ***while***.

EJERCICIO 1.6: MODIFICA EL PROGRAMA *MASCOTA.C*.

En este ejercicio se te pide una modificación del programa del Ejercicio 4. Edita el programa ***mascota.c*** y guardado con el nombre ***pregunta.c***, modificándolo de manera que al ejecutarse nos pregunte el número de calzado, peso y color favorito.

Solución comentada del Ejercicio 1.6.

```
/* fichero pregunta.c */

#include <stdio.h>
void main(void) {
    int   calzado;
    float peso;
    char  color[20];

    printf("Confiesa tu calzado, peso y color favorito:\n");
    printf("\n       Calzado: ");
    scanf("%d", &calzado);
    printf("\n          Peso: ");
    scanf("%f", &peso);
    printf("\nColor favorito: ");
    scanf("%s", color);
    printf("El %s!\n", color);
    printf("Como puede gustarte el %s,\n", color);
    printf("calzando un %d y pesando %6.2f Kg.?\n", calzado, peso);
}
```

Comentario: En la función ***printf()*** hay que utilizar diferentes formatos de salida para las variables que deseamos imprimir en pantalla. Así, el formato ***%6.2f*** mostrará ***peso*** en 6 espacios, de los cuales dos serán para los decimales y uno para el punto decimal. Observa cómo se pide el ***peso***, ***calzado*** y ***color favorito*** de forma que los dos puntos (:) queden alineados en la pantalla.

EJERCICIO 1.7: MODIFICA EL PROGRAMA *SUMAINT.C*.

Se trata del programa que suma los cinco primeros enteros que se presentó en el Ejercicio 1.5. En primer lugar se deberá editar este programa y salvarlo con el nombre ***sumaFor.c***.

Se te pide que modifiques la copia (***sumaFor.c***) para que el programa realice lo mismo (sumar los cinco primeros enteros), pero empleando un bucle ***for*** en lugar de un bucle ***while***.

Solución comentada del Ejercicio 1.7.

```
/* fichero sumaFor.c */

#include <stdio.h>
void main(void) {
    int i, suma=0;

    for(i=0; i<=5; i++)
        suma+=i;
    printf("La suma de los cinco primeros numeros es: %d\n", suma);
}
```

Comentario: En este fichero se ha utilizado la equivalencia directa entre los bucles ***for*** y ***while***. Por lo demás, este programa no tiene nada de particular.

EJERCICIO 1.8: SOLUCIÓN DE LA ECUACIÓN DE SEGUNDO GRADO.

Dada la ecuación de segundo grado $ax^2 + bx + c = 0$: se calcula el discriminante $discr = b^2 - 4ac$.

Se pueden presentar tres casos distintos:

- Si ***discr >0.0*** las dos raíces son reales y distintas, y valen:

$$x_1 = \frac{-b + \sqrt{discr}}{2a} \qquad x_2 = \frac{-b - \sqrt{discr}}{2a}$$

- Si ***discr = 0.0*** las dos raíces son reales e iguales, y valen:

$$x_1 = x_2 = \frac{-b}{2a}$$

- Finalmente, si ***discr<0.0*** las dos raíces son complejas conjugadas. Las partes real e imaginaria valen:

$$xr = \frac{-b}{2a} \qquad xi = \frac{\sqrt{-discr}}{2a}$$

Teclea y compila el siguiente programa para resolver la ecuación de segundo grado. Llámalo ***ecuacion2.c***. Compila y ejecuta este programa cambiando los valores de ***a***, ***b*** y ***c***, de modo que se prueben las tres opciones del programa.

Solución comentada al Ejercicio 1.8.

```
/* fichero ecuacion2.c */
/* resolución de la ecuación de segundo grado */

#include <stdio.h>
#include <math.h>       /* incluye decl. de la función sqrt() */
void main(void) {
    double a, b, c;
    double discr, x1, x2, xd, xr,xi;

    printf("Escribe los valores de los coeficientes A, B y C\n");
```

```
    scanf("%lf%lf%lf", &a, &b, &c);

    discr=b*b-4.0*a*c;
    if (discr>0.0) {
        x1=(-b+sqrt(discr))/(2.0*a);
        x2=(-b-sqrt(discr))/(2.0*a);
        printf("\nLas dos raices reales son: %12.6e y %12.6e  \n",
            x1, x2);
    } else if (discr<0.0) {
        xr=-b/(2.0*a);
        xi=sqrt(-discr)/(2.0*a);
        printf("\nRaices complejas:\n");
        printf("(%12.6e, %12.6ei) y (%12.6e,   %12.6ei)\n",
            xr, xi, xr, -xi);
    } else {
        x1 = -b/(2.0*a);
        printf("\nLas dos raices son iguales y valen: %12.6e \n", x1);
    }
}
```

Comentario: Incluyendo la librería ***math.h*** se pueden usar las funciones matemáticas tales como ***sqrt()*** para la raíz cuadrada; ***cos()*** para calcular el coseno de un ángulo, etc. La instrucción ***if...else*** permite hacer una *bifurcación*, dependiendo de la cual se realizarán diferentes actividades.

Merece la pena observar la forma utilizada para partir en varias líneas las distintas llamadas a la función ***printf()***. La idea fundamental es que, en el fichero ****.c*** la ***cadena de control*** (lo que va entre comillas como primer argumento de la función ***printf()***) no se puede partir entre dos o más líneas

EJERCICIO 1.9: PARA EXPERTOS.

Realiza un programa en C que escriba una tabla de dos columnas para la conversión entre las temperaturas en grados Fahrenheit –comprendidas entre 0 ºF y 300 ºF, según incrementos de 20 ºF– y su equivalente en grados centígrados. Se realizarán dos versiones de este programa: una llamada ***temp1.c*** que empleará un bucle ***while***. La otra versión se llamará ***temp2.c*** y utilizará un bucle ***for***. La conversión entre grados Centígrados y grados Fahrenheit obedece a la fórmula:

$$ºC = \frac{5 * (ºF - 32)}{9}$$

siendo ºC la temperatura en grados Centígrados y ºF en grados Fahrenheit.

Solución comentada al Ejercicio 1.9.

```
/* fichero temp1.c */

#include <stdio.h>
void main(void) {
    double gradosFahr, gradosCent;

    printf("grados Fahrenheit      grados Centigrados\n");
    printf("-----------------      ------------------\n\n");
    gradosFahr = 0.0;
    while (gradosFahr<=300.0) {
        gradosCent=(5*(gradosFahr-32.0))/9.0;
        printf("%17.2lf%17.2lf\n", gradosFahr, gradosCent);
        gradosFahr+=20.0;
    }
}



/* fichero temp2.c */

#include <stdio.h>
```

```
void main(void) {
    double gradosFahr, gradosCent;

    printf("grados Fahrenheit      grados Centigrados\n");
    printf("-----------------      ----------------\n\n");

    for (gradosFahr=0.0; gradosFahr<=300.0; gradosFahr+=20.0) {
        gradosCent = (5*(gradosFahr-32.0))/9.0;
        printf("%17.2lf%17.2lf\n", gradosFahr, gradosCent);
    }
}
```

Comentario: Como podrás observar la diferencia entre los dos programas está fundamentalmente en la forma de utilizar los bucles respectivos, a diferencia del ***while***, en los parámetros del bucle ***for*** está incluidas todas las condiciones necesarias para su ejecución, es decir se encuentran: la inicialización, la condición a cumplir y además el incremento del contador, lo cual simplifica mucho el programa.

PRÁCTICA 2.

EJERCICIO 2.1: VARIAS FORMAS DE UTILIZAR EL BUCLE FOR.

En el siguiente programa se muestran distintas formas de escribir un bucle for para sumar los enteros del 1 al 5. Escribe el siguiente programa y guárdalo con el nombre ***sumaFor2.c***. Compílalo, ejecútalo y observa los resultados. ¿Ves algo raro?

Solución comentada al Ejercicio 2.1.

```
/* fichero sumaFor2.c */
/* Programa para sumar los enteros del 1 al 5 */

#include <stdio.h>
void main(void) {
   int i=1, suma=0;

   for ( ; i<=5 ; ) {                    /* primera forma */
      suma += i;
      ++i;
   }
   printf("suma 1 = %d\n", suma);

   suma=0;                               /* segunda forma */
   for (i=1; i<=5; ++i)
      suma+=i;
   printf("suma 2 = %d\n", suma);

   for(i=1, suma=0; i<=5 ; ++i, suma+=i)  /* tercera forma */
      ;
   printf("suma 3 = %d\n", suma);

   for(i=1, suma=0; i<=5 ; suma+=i, ++i)  /* cuarta forma */
      ;
   printf("suma 4 = %d\n", suma);
}
```

Comentario: Para definir un bucle hace falta un ***contador*** o variable de control (que casi siempre es un entero y suele nombrarse con las letras típicas de subíndices: i, j, k, ...). Esta variable de control es la que se chequea cada vez que comienza el bucle y la que permite continuar o no realizando las operaciones de dentro del bucle.

Por otra parte, en C, el bucle ***for*** tiene tres componentes separadas por puntos y comas: la primera es una inicialización de la variable de control (u otras que pudieran afectar al bucle), la segunda es la sentencia de chequeo de la variable de control que siempre es necesaria (véase la primera forma); por último, la tercera son sentencias de actualización que se ejecutan al final del bucle (que también se podrían poner entre las llaves del bucle, detrás de las demás sentencias). Conociendo estas características, la forma más habitual de expresar un bucle es la segunda, ya que el bucle ***for*** contiene las instrucciones pertinentes a la variable de control.

La forma primera es más asimilable a un bucle ***while***, ya que la instrucción ***for*** contiene sólo el chequeo de la variable.

La diferencia entre las formas tercera y cuarta es el orden en que se ejecutan las instrucciones: la forma tercera ejecuta antes el incremento de la variable que el chequeo y la suma, por tanto cuando ***i*** vale ***6*** ya se ha sumado a la variable ***suma*** y, por eso el resultado sale ***20*** en lugar de ***15***.

EJERCICIO 2.2: MÁXIMO ELEMENTO DE UN CONJUNTO DE NÚMEROS.

Este programa calcula el máximo entre un conjunto de números enteros. Para ello se sigue el siguiente algoritmo: se crea una variable llamada ***max***, a la que se da inicialmente el valor de ***conjunto[0]***. Luego se recorren paso a paso todos los elementos del vector, comparando el valor almacenado en la posición considerada del vector con el valor de la variable ***max***. Si el valor de la posición considerada del vector es mayor que ***max***, entonces se copia (se sustituye el valor) en la

variable ***max*** este valor. De esta forma, una vez recorrido todo el vector, la variable ***max*** contendrá el máximo valor. Guarda este programa con el nombre ***maximo.c***.

Solución comentada al Ejercicio 2.2.

```
/* fichero maximo.c */
/* Programa para calcular el máximo de un conjunto de números */

#include <stdio.h>
#define SIZE 5
void main(void) {
    int i, max, imax;
    int conjunto[SIZE];

    printf("Introduce %d valores:\n", SIZE);

    for (i=0; i<SIZE; i++) {
        printf("%d: ", i+1);
        scanf("%d", &conjunto[i] );
        printf("\n");
    }

    max=conjunto[0];
    imax=0;
    for (i=0; i<SIZE; i++) {
        if (conjunto[i]>max) {
            max=conjunto[i];
            imax=i;
        }
    }

    printf("\nEl maximo valor del conjunto es: %d.\n", max);
    printf("\ny esta en la posicion %d.\n", imax+1);
}
```

Comentario: Este programa es muy sencillo, si se entiende el algoritmo. La variable ***max*** acabará siendo igual al elemento mayor del conjunto, e ***imax*** indicará la posición del máximo.

EJERCICIO 2.3: MÍNIMO VALOR ALGEBRAICO DE UN CONJUNTO DE NÚMEROS ENTEROS.

Modifica el programa anterior ***maximo.c***, de forma que calcule el mínimo valor del conjunto. Guárdalo con el nombre ***minimo.c***.

Solución comentada del Ejercicio 2.3.

```
/*fichero minimo.c */

#include <stdio.h>
#define SIZE 5
void main(void) {
    int i, min, imin;
    int conjunto[SIZE];

    printf("Introduce %d valores:\n", SIZE);

    for (i=0; i<SIZE; i++) {
        printf("%d: ", i+1);
        scanf("%d", &conjunto[i] );
        printf("\n");
    }
    min=conjunto[0];
    imin=0;
```

```
    for (i=0; i<SIZE; i++) {
        if (conjunto[i] < min) {
            min=conjunto[i];
            imin=i;
        }
    }

    printf("\nEl minimo valor del conjunto es: %d.\n", min);
    printf("\ny esta en la posicion %d.\n", imin+1);
}
```

Comentario: Este programa es prácticamente idéntico al anterior. En lugar de las variables ***max*** e ***imax*** se emplean las variables ***min*** e ***imin*** que almacenan el valor mínimo y su posición, respectivamente. Una vez hechos estos cambios de nombre de variables, la única línea del programa que varía es la correspondiente a la condición del ***if***, que como se está buscando el valor mínimo, habrá de ser:

if (conjunto[i]<min)

es decir, habrá de chequear si el elemento considerado es menor que el valor mínimo encontrado hasta ese momento.

EJERCICIO 2.4: ORDENAR UN CONJUNTO DE NÚMEROS ENTEROS.

El siguiente programa, al que llamarás ***ordena.c***, ordena un conjunto de números enteros almacenados en un vector, utilizando el siguiente algoritmo (método de la burbuja): se van recorriendo una a una todas las posiciones del vector, desde la primera hasta la penúltima. Estando en cada una de estas posiciones, se recorren, a su vez, todas las posiciones siguientes y se compara su valor con el de la posición actual. Si se encuentra un valor menor se intercambia con el de esta posición.

Para implementar este algoritmo son necesarios dos bucles: el primero, bucle ***i***, recorre el vector desde la posición ***i=0*** hasta ***i=SIZE−1***. El segundo bucle, bucle ***j***, recorre el vector desde la posición ***j=i+1*** hasta el final. Para que quede más claro, vamos a ver con un ejemplo como funciona este algoritmo. Supongamos que queremos ordenar los siguientes cinco números: 7,3,5,1,4. Estos números se almacenarán en un vector de la siguiente manera:

vector[0]	vector[1]	vector[2]	vector[3]	vector[4]
7	3	5	1	4

Vamos a recorrer las posiciones del vector desde **i=0** hasta **i=3**.

i = 0 {**7** 3 5 1 4}

Recorremos el vector desde ***j=1*** hasta ***j=4*** y comparamos ***vector[0]=7*** con ***vector[j]***. Si ***vector[j]<vector[0]*** intercambiamos los valores de posición. Vamos a ver cómo quedaría el vector inicial una vez que termina cada ***bucle j***.

j = 1 {**3** **7** 5 1 4} Se intercambia 3 con 7

j = 2 {**3** 7 **5** 1 4} No se intercambia 3 con 5

j = 3 {**1** 7 5 **3** 4} Se intercambia 1 con 3

j = 4 {**1** 7 5 3 **4**} No se intercambia 1 con 4

i = 1 {1 **7** 5 3 4}

Recorremos el vector desde ***j=2*** hasta ***j=4*** y comparamos ***vector[1]=7*** con ***vector[j]***.

j = 2 {1 **5** **7** 3 4} Se intercambia 5 con 7

j = 3 {1 **3** 7 **5** 4} Se intercambia 3 con 5

j = 4 {1 **3** 7 5 **4**} No se intercambia 3 con 4

i = 2 {1 3 **7** 5 4}

j = 3 {1 3 **5 7** 4} Se intercambia 5 con 7

j = 4 {1 3 **4** 7 **5**} Se intercambia 4 con 5

i = 3 {1 3 4 **7** 5}

j = 4 {**1 3 4 5 7**} Se intercambia 5 con 7 ¡Números ordenados!

Ya se ve que no es necesario que el ***bucle i*** llegue hasta el valor 4.

Solución comentada al Ejercicio 2.4.

```
/* fichero ordena.c*/
/* Programa para ordenar un conjunto de números enteros*/

#include <stdio.h>
#define   SIZE   7
void main(void) {
    int vector[SIZE];
    int j, i, temp;

    printf("Introduce los %d valores para ordenar:\n", SIZE);
    for(i=0; i<SIZE; i++) {
        printf("%d: ", i+1);
        scanf("%d", &vector[i]);
        printf("\n");
    }
    /* se aplica el algoritmo de la burbuja */
    for(i=0; i<(SIZE-1); i++) {
        for (j=i+1; j<SIZE; j++) {
            if(vector[j]<vector[i]) {
                temp=vector[j];
                vector[j]=vector[i];
                vector[i]=temp;
            }
        }
    }
    printf("El vector ordenado es:\n");
    for(i=0; i<SIZE ; i++) {
        printf("%d  ", vector[i]);
    }
    printf("\n");
}
```

Comentario: Para intercambiar entre sí los valores de dos variables ***no se pueden emplear*** las dos instrucciones siguientes:

```
vector[j]=vector[i];  /* esta instrucción pone en la posición j el valor de la posición i,
                         perdiéndose el valor de la posición j */
vector[i]=vector[j]; /* esta instrucción pone en la posición i el valor de la posición j,
                         pero no el original, que se ha perdido, sino el nuevo */
```

Por eso es necesaria una variable intermedia, que en este caso hemos llamado ***temp***, que guarda temporalmente el valor de la posición ***j***. Por esto se requieren tres instrucciones.

EJERCICIO 2.5: PROGRAMA ELECTORAL, QUE NO ELECTORALISTA.

Aprovechando las muy recientes elecciones al Parlamento Vasco, vamos a ver cómo se convierten en escaños los votos de los sufridos ciudadanos siguiendo el método d'Hont. ¿Estás familiarizado con este método? Por si acaso te lo vamos a recordar.

Supongamos que concurren 3 partidos a las elecciones y que la provincia o distrito electoral dispone de 2 escaños. El primer escaño se lo llevará el partido más votado. Para el segundo escaño se dividen los votos de cada partido entre el número de escaños obtenidos más uno (el partido que

no tenga todavía ningún escaño se dividirá entre 1) y se calculan los restos de dichas divisiones. El escaño se asigna al partido que tras esta operación tenga un resto más elevado. Vamos a verlo con un ejemplo:

Partido 1: 6000 (Se lleva el escaño 1) → restos → 6000/(1+1) = 3000

Partido 2: 4000 → restos → 4000/(1+0) = 4000 → Se lleva el escaño 2.

Partido 3: 2000 → restos → 2000/(1+0) = 2000

El programa que te presentamos a continuación es para 3 partidos y 2 escaños, pero queda a tu entera disposición para que lo particularices −si dispones de datos− para el caso de las recientes elecciones. Ten mucho cuidado porque un fallo en la programación puede hacer caer sobre ti una acusación de fraude electoral. Llamaremos al programa ***elecciones.c***.

Lo primero que hace este programa es inicializar a cero el vector ***nEsca[ ].*** En este vector se va a almacenar el número de escaños de cada partido: ***nEsca[0]*** contendrá el número de escaños del partido número 1, ***nEsca[1]*** los escaños del partido número 2, etc. Este vector se va a ir modificando a lo largo de la ejecución del programa.

A continuación se almacenan en el vector ***nVotos[ ]*** los números de votos obtenidos por cada partido, estos datos serán introducidos por el usuario a través del teclado.

La novedad de este programa es que incluye una función: ***nextEsc(long *, int *)*.** Esta función tiene como argumentos dos punteros a sendos vectores, es decir, las direcciones de memoria o los ***nombres*** de dos vectores. Cuando se llama a la función, ***nextEsc(nVotos, nEsca)*** se pasa la dirección de memoria de los vectores ***nEsca[ ]*** y ***nVotos[ ]***. El nombre de un vector (como por ejemplo ***nEsca***) es un puntero a la dirección del primer elemento. La función devuelve el número del partido que ha conseguido el escaño y se almacena en la variable ***esca***. Este número será cero si el escaño corresponde al primer partido, 1 si corresponde al segundo, etc.

Con esta pequeña explicación ya estás en condiciones de escribir el programa y comprobar cómo funciona.

Solución comentada al Ejercicio 2.5.

```
/* fichero elecciones.c*/
/* Calculo de escaños por partido (d'Hondt). */

#include <stdio.h>
#define    NESC    2           /* número de escaños en liza. */
#define    NPAR    3           /* número de partidos que concurren */
void main(void) {
    int  i, esca;
    int  nEsca[NPAR];     /* nEsca{[NPAR]= n° de escaños por partido*/
    long nVotos[NPAR];    /* nVotos[NPAR]=n° de votos por partido*/
    int  nextEsc(long *, int *);   /* declaración de función */

    for(i=0; i<NPAR; ++i)
        nEsca[i]=0;

    printf("Teclea el numero de votos de cada partido.\n");
    for (i=0; i<NPAR; ++i) {
        printf("\nPartido %d: ", i+1);
        scanf("%ld", &nVotos[i]);
    }

    for(i=0; i<NESC; ++i) {           /* asignación de escaños */
        esca = nextEsc(nVotos, nEsca);
        nEsca[esca] = nEsca[esca]+1;
    }
```

```
    for (i=0; i<NPAR; ++i)
        printf("\nEl partido %d ha obtenido %d escagnos.",
            i+1, nEsca[i]);
    printf("\n");
}

int nextEsc( long nVotos[], int nEsca[]) {
    int  imaximo=0, i;
    long maximo=0;

    for( i=0; i<NPAR; ++i) {
        if( maximo<(nVotos[i]/(nEsca[i]+1)) ) {
            maximo=nVotos[i]/(nEsca[i]+1);
            imaximo=i;
        }
    }
    return imaximo;
}
```

EJERCICIO 2.6: PRODUCTO DE MATRIZ POR VECTOR.

A estas alturas de la práctica seguro que ya les has cogido el truco a los bucles ***for***, es más, ¡te han empezado a gustar! ¿te estará pasando algo?. No te preocupes, tienes los primeros síntomas de adicción a la programación en C, y si sigues así, terminarás por ser un experto programador, enfermedad que, para tu tranquilidad, no tiene ningún tipo de efectos secundarios.

Para que no dejes sin explotar todas las posibilidades de los bules ***for,*** te proponemos este sencillo programa que multiplica una matriz por un vector (en este orden). Escribe el programa y guárdalo como ***vmatriz.c***.

Solución comentada al Ejercicio 2.6.

```
/* fichero vmatriz.c */
/* Producto de matriz por vector */

#include <stdio.h>
#define  SIZE   3

void main(void) {
    double matriz[SIZE][SIZE];
    double vector[SIZE];
    double solucion[SIZE];
    double sum;
    int    i,j;

    printf("Introduce los datos de la matriz:\n");
    for(i=0; i<SIZE ; i++) {
        for(j=0; j<SIZE; j++) {
            printf("\nElemento (%d,%d): ", (i+1), (j+1));
            scanf(" %lf", &matriz[i][j]);
        }
    }
    printf("\n\nIntroduce los datos del vector:\n");
    for(i=0; i<SIZE ; i++) {
        printf("\nElemento %d: ", (i+1));
        scanf("%lf", &vector[i]);
    }
    for(i=0; i<SIZE; i++) {
        sum=0;
        for(j=0; j<SIZE; j++) {
            sum+=matriz[i][j]*vector[j];
        }
```

```
        solucion[i]=sum;
    }
    printf("\nEl vector solucion es:\n");
    for(i=0; i<SIZE; i++) {
        printf("Elemento %d = %lf\n", i+1, solucion[i]);
    }
}
```

Comentario: Se presentan en este programa algunos pasos típicos de manejo de matrices que son muy similares en todos los lenguajes de programación.

En cuanto al *algoritmo*, es ya conocida la forma de multiplicar una matriz por un vector columna: cada elemento del vector producto es el producto escalar (la suma de los productos elemento a elemento) de la fila correspondiente de la matriz por el vector columna. Para ello hacen falta dos bucles: uno recorre las filas de la matriz y el otro realiza el sumatorio propio del producto escalar. Observa que, para realizar el sumatorio, se inicializa cada vez la variable ***sum*** a cero.

EJERCICIO 2.7: PRODUCTO DE MATRICES.

Basándote en el producto de matriz por vector anterior, haz un programa que multiplique dos matrices cuadradas y llámalo ***bimatriz.c***. Si no quieres perder el tiempo introduciendo datos con el teclado, también puedes asignar desde el programa unos valores arbitrarios a las matrices factores.

Solución comentada del Ejercicio 2.7.

```
/* fichero bimatriz.c */

#include <stdio.h>
#define SIZE 5
void main(void) {
    double matriz1[SIZE][SIZE], matriz2[SIZE][SIZE];
    double solucion[SIZE][SIZE], sum;
    int dimension, i, j, k;

    printf("Introduce la dimension de las matrices:\n");
    printf("Dimension: ");
    scanf("%d", &dimension);

    printf("Introduce los elementos de la primera matriz:\n");
    for (i=0; i<dimension; i++) {
        for (j=0; j<dimension; j++) {
            printf("a(%d,%d): ", i+1, j+1);
            scanf("%lf", &matriz1[i][j] );
        }
    }
    printf("\n");

    printf("Introduce los elementos de la segunda matriz:\n");
    for (i=0; i<dimension; i++) {
        for (j=0; j<dimension; j++) {
            printf("b(%d,%d): ", i+1, j+1);
            scanf("%lf", &matriz2[i][j] );
        }
    }
    printf("\n");

    for (i=0; i<dimension; i++) {
        for (j=0; j<dimension ; j++) {
            sum=0.0;
            for (k=0; k<dimension; k++)
                sum+=matriz1[i][k]*matriz2[k][j];
            solucion[i][j]=sum;
        }
    }

    printf("Solucion:\n\n\n");
```

```
    for (i=0; i<dimension; i++) {
        for (j=0; j<dimension; j++)
            printf("%5.2lf      ", solucion[i][j]);
        printf("\n");
    }
}
```

Comentario: Tal vez el aspecto más interesante de este programa es la necesidad de utilizar 3 bulces ***for*** para la multiplicación de dos matrices. Para verlo más claramente, sea a_{ij} un elemento de la primera matriz, b_{ij} un elemento de la segunda y c_{ij} un elemento de la matriz solución. Pensemos por un momento cómo se obtiene un elemento cualquiera de la matriz solución. El elemento c_{23}, por ejemplo, se obtiene multiplicando los elementos de la fila 2 de la primera matriz, por los elementos de la columna 3 de la segunda matriz. Suponiendo que las matrices son de dimensión 3 tendremos:

$c_{23} = a_{21}*b_{13}+a_{22}*b_{23}+a_{23}*b_{33}$ y en general:

$c_{ij} = a_{i1}*b_{1j}+a_{i2}*b_{2j}+a_{i3}*b_{3j}$

Por lo tanto, para poder acceder a los elementos de la ***fila i*** de la primera matriz y a los elementos de la ***columna j*** de la segunda matriz, y para realizar el sumatorio son necesarios tres bucles, para los que se han utilizado las variables ***i***, ***j*** y ***k***.

EJERCICIO 2.8: UN PROGRAMA PARA UN PRIMO.

El siguiente programa comprueba si un número es primo o no. Recuerda que un número primo es aquél que puede dividirse únicamente por sí mismo y por la unidad. Una manera de decidir si un número es primo o no, es dividirlo por todos los números comprendidos entre el 1 y su raíz cuadrada. Si se encuentra que el número es divisible por alguno de ellos, se deduce que ese número no es primo. Vamos a utilizar el ***operador módulo o resto de la división entera*** (%) para comprobar si la división es exacta. Este operador da como resultado el resto de la división del primer operando por el segundo. Por ejemplo, 5%2 es 1, puesto que el resto de dividir 5 entre 2 es 1.

Guarda el programa con el nombre ***primos.c***.

Solución comentada al Ejercicio 2.8.

```
/* fichero primos.c */
/* programa para determinar si un número es primo */

#include <stdio.h>
#include <math.h>

void main(void) {
    int numero, divisor;

    printf("Que numero quieres saber si es primo?\n");
    scanf("%d", &numero);
    while(numero<2) {
        printf("Lo siento, no acepto numeros menores que 2.\n");
        printf("Intentalo de nuevo\n");
        scanf("%d", &numero);
    }
    for (divisor=2; divisor<=sqrt(numero); divisor++) {
        if (numero%divisor==0) {
            printf("%d no es primo.\n", numero);
            return;
        }
    }
    printf("%d es primo.\n", numero);
}
```

Comentario: El comentario más apropiado para este ejercicio es el algoritmo para calcular el primo con el bucle ***for (divisor=2; divisor<=sqrt(numero); divisor++).*** Observa que este bucle se termina cuando se encuentra un divisor del número, en cuyo caso se escribe el mensaje de que el número no es primo y se termina la ejecución del programa con un ***return***, o cuando divisor llega a su valor límite, en cuyo caso el número es primo porque no se ha encontrado ningún divisor.

EJERCICIO 2.9: UN PROGRAMA PARA MUCHOS PRIMOS.

Basándote en el ejercicio anterior, realiza un programa que imprima todos los números primos comprendidos entre el 2 y un valor límite que se preguntará al ejecutar el programa. Guárdalo con el nombre ***primos1.c***.

Solución comentada del Ejercicio 2.9.

```
/* fichero primos1.c */

#include <stdio.h>
void main(void) {
    int numero, divisor;
    int n;

    printf("¨Hasta qué número desea conocer los primos?\n");
    printf("introduzca el valor: ");
    scanf("%d", &n);

    for (numero=2; numero<=n; numero++) {
        esPrimo=1; /* numero es primo */
        for (divisor=2; divisor<=sqrt(numero); divisor++)
            printf("%d\n", numero);
    }
}
```

Comentario: Para poder escribir todos los números primos hasta un límite, basta introducir un nuevo bucle ***for*** al programa anterior que vaya recorriendo todos los números enteros desde el 2 hasta dicho límite inclusive. En cada ejecución del bucle se comprueba si el número considerado es primo aprovechando las sentencias del programa anterior. En este caso se calcula si un número es primo mediante el bucle ***for (divisor=2; divisor<=sqrt(numero); divisor++)***. Observa que se ha definido una variable ***esPrimo*** que indica si el número es primo o no. Al principio se supone que el número es primo, y esta condición se cambia si se encuentra un divisor exacto. En esta caso, la sentencia ***break*** hace que se termine el bucle ***for*** más interno.

PRACTICA 3.

EJERCICIO 3.1: EL OPERADOR DE DIVISIÓN (/).

Ejecuta el siguiente programa para aclarar las ideas sobre cómo funciona este operador con distintos tipos de variables. En él se puede comprobar la diferencia entre la división *entera* y de *punto flotante*. Guarda el programa como ***division.c***.

Solución comentada al Ejercicio 3.1.

```
/* fichero division.c */

# include <stdio.h >
void main(void) {
    printf("division entera:   5/4   es %6d\n", 5/4);
    printf("division entera:   6/3   es %6d\n", 6/3);
    printf("division entera:   7/4   es %6d\n", 7/4);
    printf("division flotante: 7./4. es %6.3f\n", 7./4.);
    printf("division mixta:    7./4  es %6.3f\n", 7./4);
}
```

Comentario: Es importante recordar que el tipo de formato debe estar de acuerdo con el tipo del argumento en la función ***printf()***. Para el formato de salida (%6.3f) se tendrán un total de 6 espacios de salida, de los cuales tres serán para los decimales.

EJERCICIO 3.2: UN REPASO A LA FUNCIÓN *PRINTF( )*.

El siguiente programa utiliza la función ***printf( )*** para imprimir distintos tipos de variables. Como recordarás, cada tipo de variable precisa de su *carácter de conversión* precedido del carácter %, pudiendo haber entre ambos caracteres uno o varios elementos para el alineamiento por la izquierda, la anchura mínima del campo, etc. Y como una ejecución vale más que mil palabras, aquí tienes el programa para que lo puedas probar. Guárdalo con el nombre ***formatos.c***.

Solución comentada al Ejercicio 3.2.

```
/* fichero formatos.c */
#include <stdio.h>

void main(void) {
    int    x=45;
    double y=23.354;
    char   z[]="Esto es vida";

    /* utilizamos barras inclinadas (/) para ver claramente la anchura del campo de caracteres */
    printf("Voy a escribir /45/ utilizando el formato %%d: /%d/\n", x);
    printf("Voy a escribir /45/ utilizando el formato %%1d: /%1d/\n", x);
    printf("Voy a escribir /45/ utilizando el formato %%10d: /%10d/\n\n", x);

    printf("Voy a escribir /23.354/ utilizando el formato %%f:/%f/\n", y);
    printf("Voy a escribir /23.354/ utilizando el formato %%.3f: /%.3f/\n", y);
    printf("Voy a escribir /23.354/ utilizando el formato %%5.1f: /%5.1f/\n", y);
    printf("Voy a escribir /23.354/ utilizando el formato %%-10.3f:/%-10.3f/\n", y);
    printf("Voy a escribir /23.354/ utilizando el formato %%5f: /%5f/\n\n", y);

    printf("Voy a escribir /Esto es vida/ utilizando el formato %%s: /%s/\n", z);
    printf("Voy a escribir /Esto es vida/ utilizando el formato %%.7s:/%.7s/\n", z);
    printf("Voy a escribir /Esto es vida/ utilizando el formato %%-15.10s: /%-15.10s/\n", z);
    printf("Voy a escribir /Esto es vida/ utilizando el formato %%15s: /%15s/\n", z);
}
```

Comentario: La mejor forma de enteder los formatos es estudiar con mucho cuidado la salida impresa de cada una de las llamadas a la función ***printf()***. Para que se imprima el carácter % y no sea interpretado como un comienzo de formato, hay que ponerlo dos veces.

EJERCICIO 3.3: Y SEGUIMOS CON LOS BUCLES *FOR*.

Para realizar este ejercicio deberás basarte en el programa ***sumaFor.c*** de la práctica anterior. Modifica este programa de forma que realice la suma y el producto de los *n* primeros números enteros, siendo *n* un número que deberá teclear el usuario. La salida del programa deberá ser idéntica a la que se muestra un poco más adelante, es decir, debe ocupar cuatro líneas y cada salto de línea ha de realizarse en el lugar indicado. En este caso se ha particularizado para una entrada (en negrita, para indicar que es un número tecleado por el usuario) de n=12. Sálvalo como ***factorial.c*** y ¡atención a la declaración de variables!

```
Este programa suma y multiplica los n primeros numeros enteros, siendo n el numero que tu quieras.
Teclea n: 12
La suma de los 12 primeros numeros es: 78
El producto de los 12 primeros numeros, es decir, el factorial de 12 es: 479001600.
```

Solución comentada al Ejercicio 3.3.

```
/* fichero factorial.c */

#include <stdio.h>
void main(void) {
    int i, suma, n;
    long int factorial;

    printf("%s%s%s\n", "Este programa suma y multiplica ",
        "los n primeros numeros enteros, ",
        "siendo n el numero que tu quieras.\n");
    printf("Teclea n: ");
    scanf("%d", &n);
    suma=0;
    factorial=1;
    for (i=1; i<=n; i++) {
        suma+=i;
        factorial*=i;
    }
    printf("La suma de los %d primeros enteros es: %d\n", n, suma);
    printf("El producto de los %d primeros numeros, es decir, ", n);
    printf("el factorial de %d es: %ld\n", n, factorial);
}
```

Comentario: Obsserva la primera llamada a la función ***printf()*** se realiza utilizando tres cadenas de caracteres como argumentos, que se escriben con los formatos (***%s%s%s***). Este método permite acortar una cadena de control muy larga, que debe ocupar una única línea en el fichero fuente del programa. Puede ser también muy útil cuando se quieren destacar palabras, si se utiliza un número con el ***%s*** (***%20s*** escribe la cadena en 20 espacios, aunque su longitud sea menor).

El otro punto a comentar de este programa es la declaración de variables. Observa que se utiliza un ***long*** para el factorial en lugar de un ***int***. Esto es debido a que un factorial pasa con facilidad del máximo valor que puede almacenar un ***int*** (factorial de 8 ya es mayor que 32767). Incluso si se quiere obtener un valor muy alto se puede utilizar una variable de tipo ***double*** aunque los valores que maneje sean enteros.

EJERCICIO 3.4: VOLVAMOS A ORDENAR SIN OLVIDAR EL DESORDEN INICIAL.

Modifica el programa ***ordena.c*** que realizaste en la práctica anterior de manera que, además de mostrar el vector ordenado, muestre también el vector original. Además deberá mostrar la posición que cada número ocupaba en el vector inicial. Guarda este programa con nombre ***reordena.c***. Una salida posible podría ser:

```
El vector inicial es:     {7, 23, 5, 1, 8, 3, 12}
El vector ordenado es:    {1, 3, 5, 7, 8, 12, 23}
que ocupaban los lugares  {4, 6, 3, 1, 5, 7, 2}, respectivamente.
```

Solución comentada del Ejercicio 3.4.

```
/* fichero reordena.c */

#include <stdio.h>
#define SIZE 7
void main(void) {
    int vector[SIZE], copia[SIZE], orden[SIZE];
    int j, i, temp;

    printf("Introduzca los %d valores para ordenar:\n", SIZE);
    for(i=0; i<SIZE; i++) {
        printf("\n%d: ", i+1);
        scanf("%d", &vector[i]);
        copia[i]=vector[i];
        orden[i]=i+1;
    }
    for (i=0; i<(SIZE-1); i++)
        for (j=i+1; j<SIZE; j++)
            if (vector[j]<vector[i]) {
                temp=vector[j];
                vector[j]=vector[i];
                vector[i]=temp;
                temp=orden[j];
                orden[j]=orden[i];
                orden[i]=temp;
            }
    printf("\nEl vector inicial es:      { ");
    for (i=0; i<SIZE; i++)
        printf("%d, ", copia[i]);
    printf("\b\b} \n");  /* se utiliza backspace */

    printf("El vector ordenado es:     { ");
    for (i=0; i<SIZE; i++)
        printf("%d, ", vector[i]);
    printf("\b\b} \n");

    printf("que ocupaban los lugares   { ");
    for (i=0; i<SIZE; i++)
        printf("%d, ", orden[i]);
    printf("\b\b}, respectivamente.\n");
}
```

Comentario: El funcionamiento del programa de ordenación de los elementos de un vector se vió en la práctica anterior (ejercicio 2.4). Para evitar perder el vector original con el cambio, lo que se hace es una simple copia del mismo con la instrucción ***copia[i] = vector[i]***. Para saber en qué posición estaban los elementos en el vector original, se crea un nuevo vector llamado ***orden[ ]***, cuyos elementos son los números naturales 1, 2, 3, 4. ... Este vector sufre las mismas permutaciones que el vector inicial, por lo que, al final de la ordenación sus elementos indicarán que posición ocupaban al principio los elementos del vector ordenado.

Para obtener exactamente la salida impresa solicitada, hay que imprimir las ***llaves*** { } y los ***blancos*** y ***comas*** que separan los elementos de los vectores. Después de escribir cada elemento de un vector se escribe una coma y un blanco. En realidad esto no hay que hacerlo para el último elemento, que debe ir seguido por el cierre de las llaves. Sin embargo, por simplicidad se escriben también en este caso la coma y el blanco, pero luego se imprimen dos caracteres ***backspace*** ('***\b***', espacio hacia atrás), con lo cual el cierre de las llaves se imprime sobre la coma.

EJERCICIO 3.5: CÁLCULO DEL DETERMINANTE DE UNA MATRIZ 3X3.

Realiza un programa que calcule el determinante de una matriz 3×3. En el caso de que la matriz sea singular, el programa deberá advertirlo con un mensaje. Guárdalo con el nombre ***determin.c***. Recuerda que la fórmula para el cálculo del determinante es la siguiente:

$$det\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{11}a_{32}a_{23} - a_{21}a_{12}a_{33}$$

En vez de tener que introducir la matriz tecleándola cada vez, utiliza un fichero llamado ***matriz.d*** que contiene una matriz 3×3 cualquiera y haz que el programa principal lea este fichero empleando la función ***fscanf( )***.

Solución comentada al Ejercicio 3.5.

```
/* fichero determin.c */
#include <stdio.h>
#define SIZE 3
void main(void) {
    double matriz[SIZE][SIZE];
    double determinante;
    int i, j;
    FILE *fi;

    /* printf("Introduzca los elementos de la matriz 3x3:\n"); */
    fi = fopen("matriz.d", "r+");
    for (i=0; i<SIZE; i++)
        for (j=0; j<SIZE; j++) {
            fscanf(fi, "%lf", &matriz[i][j]);
        }
    fclose(fi);
    printf("\n");
    determinante =  matriz[0][0]*matriz[1][1]*matriz[2][2];
    determinante += matriz[0][1]*matriz[1][2]*matriz[2][0];
    determinante += matriz[1][0]*matriz[2][1]*matriz[0][2];
    determinante -= matriz[0][2]*matriz[1][1]*matriz[2][0];
    determinante -= matriz[0][1]*matriz[1][0]*matriz[2][2];
    determinante -= matriz[0][0]*matriz[2][1]*matriz[1][2];
    printf("La matriz introducida es:\n");
    for (i=0; i<SIZE; i++) {
        for (j=0; j<SIZE; j++)
            printf("%8.2lf", matriz[i][j]);
        printf("\n");
    }
    printf("\nY su determinante es: %12.4lf\n", determinante);
}
```

Comentario: En principio este programa no existe ninguna dificultad respecto al manejo de las matrices (ya se ha visto en programas anteriores). Lo único que es importante comentar es el algoritmo: tenemos siempre una matriz 3x3 y, por ello, podemos realizar las operaciones "a mano" (sin más complejidades en el algoritmo, es decir, sin introducir *bucles* o *bifurcaciones*). Si quisiéramos calcular el determinante de una matriz cualquiera, tendríamos que triangularizar la matriz y, una vez triangularizada, el determinante sería el producto de los elementos de la diagonal. En el ejercicio 6.6 se resolverá este problema de modo general.

La matriz se lee de un fichero de disco. Para ello es necesario abrir el fichero mediante la función ***fopen()***, que devuelve un puntero a ***FILE***. Después se van leyendo los datos de dicho fichero mediante la función ***fscanf()*** a la que se pasa como primer argumento el puntero al fichero abierto; excepto en que lee de fichero, esta función es completamente análoga a ***scanf()***. Al acabar de leer la matriz se debe cerrar el fichero con la función ***fclose()***.

Es interesante observar como se ha dividido una expresión muy larga en seis expresiones más cortas, utilizando los operadores de suma y diferencia acumulativas (+= y -=).

EJERCICIO 3.6: EL SULTÁN Y EL ESTUDIANTE.

Habrás oído hablar de la historia de un poderoso sultán que deseaba recompensar a un estudiante que le había prestado un gran servicio. Cuando el sultán le preguntó qué recompensa deseaba, éste le señaló un tablero de ajedrez y solicitó simplemente 1 grano de trigo por la primera casilla, 2 por la segunda, 4 por la tercera, 8 por la siguiente, y así sucesivamente. El sultán, que no debía andar

muy fuerte en matemáticas, quedó sorprendido por la modestia de la petición, porque estaba dispuesto a otorgarle riquezas muy superiores: al menos, eso pensaba él. En el siguiente programa se calcula el número total de granos de trigo que corresponden a cada casilla y se acumula el total. Como el número de granos no es una cantidad que se maneje habituamente, se compara también con una estimación de la producción anual mundial de trigo expresada en granos. Guarda este programa con el nombre ***trigo.c***.

Solución comentada al Ejercicio 3.6.

```
/* fichero trigo.c */

#define CUADRADOS 64   /* cuadrados del tablero */
#define COSECHA 4E15   /* cosecha mundial en granos */
#include <stdio.h>
void main(void) {
    double actual, total;
    int    cont=1;

    printf("Cuadro  granos sumados  granos totales  ");
    printf("fraccion de\n");
    printf("                                        ");
    printf(" cosecha\n");
    total=actual=1.0;              /* comenzamos con un grano */
    printf("%4d %15.2e %16.2e %12.2e\n",
        cont, actual, total, total/COSECHA);

    while (cont<CUADRADOS) {
        cont=cont+1;
        actual*=2.0;              /* duplica granos en cada cuadro */
        total+=actual;            /* actualiza total */
        printf("%4d %15.2e %16.2e %12.2e\n",
            cont, actual, total, total/COSECHA);
    }
}
```

Comentario: El interés de este ejercicio está en ver una forma de disponer los resultados de salida en columnas, formando una tabla. Primero se deben imprimir los encabezamientos de las columnas y luego las cifras correspondientes a cada línea o fila de la tabla.

EJERCICIO 3.7: INTRODUCCIÓN A LA ESTADÍSTICA.

Realiza este ejercicio basándote en el programa ***maximo.c*** de la práctica anterior y guárdalo con el nombre ***numeros.c***. Este programa deberá realizar las siguientes operaciones:

1. Preguntar al usuario con cuántos números desea trabajar.
2. Leer y almacenar los números convenientemente.
3. Hallar su media aritmética utilizando para ello una función a la que llamarás ***media( )***.
4. Hallar el máximo y el mínimo llamando a sendas funciones ***maximo( )*** y ***minimo( )***, cuya programación realizaste en la práctica anterior.
5. Hallar la semisuma de los valores máximo y mínimo mediante la función ***semisuma( )***.
6. Mostrar por pantalla los valores obtenidos: media, máximo, mínimo y semisuma.

Tanto el programa principal como las funciones deberán estar en el mismo fichero ***numeros.c***.

Solución comentada al Ejercicio 3.7.

```
/* fichero numeros.c */

#include <stdio.h>
#define  SIZE 20
```

```
/* declaracion de funciones */
double media(double*, int);
double maximo(double*, int);
double minimo(double*, int);
double semisuma(double, double);
void main(void) {
    double vector[SIZE];
    int n, i;

    printf("Con cuantos valores deseas trabajar?\n");
    printf("(el numero debera ser menor que 20)\n");
    printf("Teclea n: ");
    scanf("%d", &n);
    printf("\n");

    printf("Introduce los %d valores:\n",n);
    for (i=0 ;i<n; i++) {
        printf("Valor %d: ", i+1);
        scanf("%lf", &vector[i]);
    }
    printf("\n");
    printf("media= %5.2lf \n", media(vector,n));
    printf("maximo= %5.2lf \n", maximo(vector,n));
    printf("minimo= %5.2lf \n", minimo(vector,n));
    printf("semisuma= %5.2lf \n", semisuma(maximo(vector, n), minimo(vector, n)));
}

double media(double* valores, int num) {
    double suma=0.0, med;
    int i;
    for (i=0; i<num; i++)
        suma+=valores[i];
    med=suma/num;
    return (med);
}

double minimo(double* valores, int num) {
    int i;
    double min;
    min=valores[0];
    for (i=1; i<num; i++)
        if (valores[i]<min)
            min=valores[i];
    return (min);
}

double maximo(double* valores, int num) {
    int i;
    double max;
    max=valores[0];
    for (i=1; i<num; i++)
        if (valores[i]>max)
            max=valores[i];
    return (max);
}

double semisuma(double max, double min) {
    return ((max+min)/2.0);
}
```

Comentario: En este fichero se presentan funciones para calcular la media, el máximo, el mínimo, y la semisuma del máximo y el mínimo de los elementos de un conjunto. Cada uno de estos cálculos es muy sencillo. El interés de este ejercicio está en ver cómo se definen, declaran y llaman las distintas funciones que en él aparecen. El resultado de todas estas funciones es un valor único que se devuelve como valor de retorno.

EJERCICIO 3.8: OPERACIÓN CON VECTORES.

Realiza un programa, que lea dos vectores de 3 componentes, y luego los sume, reste y multiplique, tanto escalar como vectorialmente, imprimiendo después todos estos resultados. Para ello crearás las funciones ***suma( )***, ***resta( )***, ***productoEscalar( )*** y ***productoVectorial( )***. Guárdalo con el nombre ***vectores.c***.

Solución comentada al Ejercicio 3.8.

```
/* fichero vectores.c */

#include <stdio.h>
/* declaracion de funciones */
void suma( double*, double*, double*);
void resta( double*, double*, double*);
void productoVectorial(double*, double*, double*);
double productoEscalar(double*, double*);

void main(void) {
    double vector1[3], vector2[3], vecsum[3], vecres[3];
    double escalar, prodvect[3];
    int i;
    printf("Introduce los valores del primer vector:\n");
    for (i=0; i<3; i++) {
        printf("\nElmento %d: ", i+1);
        scanf("%lf", &vector1[i]);
    }
    printf("\n");
    printf("Introduce los valores del segundo vector:\n");
    for (i=0; i<3; i++) {
        printf("\nElemento %d: ", i+1);
        scanf("%lf", &vector2[i]);
    }
    printf("\n");

    suma(vector1, vector2, vecsum);
    resta(vector1, vector2, vecres);
    productoVectorial(vector1, vector2, prodvect);

    printf("\nImprimo los resultados: \n\n");
    printf("%10s %10s %20s\n", "Suma", "Resta", "Producto Vectorial");
    for (i=1; i<=50; i++)
        printf("-");
    printf("\n");
    for (i=0; i<3; i++){
        printf("%10.2lf  %10.2lf  %10.2lf\n", vecsum[i], vecres[i],
            prodvect[i]);
        printf("\n");
    }
    printf("Producto escalar = %5.2lf\n",
        productoEscalar(vector1,vector2));
    printf("----------------\n");
}

void suma(double* vec1, double* vec2, double* sum) {
    int i;
    for (i=0; i<3; i++)
        sum[i]=vec1[i]+vec2[i];
    return;
}
```

```
void resta(double* vec1, double* vec2, double* res) {
    int i;
    for (i=0; i<3; i++)
        res[i]=vec1[i]-vec2[i];
    return;
}

void productoVectorial(double* vec1, double* vec2, double* vectorial) {
    vectorial[0] = vec1[1]*vec2[2]-vec2[1]*vec1[2];
    vectorial[1] = vec1[2]*vec2[0]-vec2[2]*vec1[0];
    vectorial[2] = vec1[0]*vec2[1]-vec2[0]*vec1[1];
    return;
}

double productoEscalar(double* vec1, double* vec2) {
    double escalar = 0.0;
    int i;
    for (i=0; i<3; i++)
        escalar+=vec1[i]*vec2[i];
    return (escalar);
}
```

Comentario: Este ejercicio es similar al anterior, pero con algunas diferencias interesantes. La más importante es que el resultado de algunas operaciones con vectores (como la suma, la diferencia o el producto vectorial), son también vectores. Por tanto el resultado de dichas operaciones no es un valor único y no se devuelve como valor de retorno. Lo que se hace es pasar todos los vectores como argumentos ***por referencia***. Como el nombre de un vector es la dirección de su primer elemento, la función es capaz de recibir datos y devolver resultados leyendo y escribiendo en las zonas de memoria adecuadas.

En el momento de imprimir resultados, se puede hacer que los letreros de ***suma***, ***resta*** y ***producto vectorial***, sean colocados a la hora de imprimir con cierto espacio, lo cual mejora la presentación de los resultados. Esto se hace con instrucciones de cadenas de caracteres reservando los espacios que consideramos necesarios (p.ej: %10s, reserva un espacio mínimo de 10 caracteres para imprimir el letrero correspondiente). Se ha incluido un bucle ***for*** que imprime una línea discontinua para subrayar los títulos. Luego, a los resultados se le da el formato indicado para que ocupen un número de espacios acorde con los títulos.

EJERCICIO 3.9: EJERCICIO DE VECTORES PLANOS.

Sean dos vectores planos (con 2 componentes cada uno) ***a[ ]*** y ***b[ ]***. Si ***a[ ]*** y ***b[ ]*** son *linealmente independientes*, cualquier vector plano ***c[ ]*** podrá ser expresado como combinación lineal de ellos:

c[ ] = alfa * a[ ] + beta * b[ ]

Realizar un programa llamado ***vector2d.c*** que lea los vectores ***a[ ]***, ***b[ ]*** y ***c[ ]*** y calcule los coeficientes ***alfa*** y ***beta***. El programa deberá distinguir 3 casos:

1. ***a[ ]*** y ***b[ ]*** son linealmente independientes (1 solución).
2. ***a[ ]*** y ***b[ ]*** son linealmente dependientes y ***c[ ]*** es colineal con ellos (infinitas soluciones). Elegir la solución de módulo mínimo.
3. ***a[ ]*** y ***b[ ]*** son linealmente dependientes y ***c[ ]*** no es colineal con ellos (no hay ninguna solución).

Solución comentada al Ejercicio 3.9.

```
/* fichero vector2d.c */

#include <stdio.h>
void main(void) {
    double a[2], b[2], c[2];
    double alfa, beta, aux;

    printf("Introduce los siguientes datos:\n");
    printf("a1: ");
    scanf("%lf", &a[0]);
    printf("a2: ");
    scanf("%lf", &a[1]);
    printf("b1: ");
```

```
    scanf("%lf", &b[0]);
    printf("b2: ");
    scanf("%lf", &b[1]);
    printf("c1: ");
    scanf("%lf", &c[0]);
    printf("c2: ");
    scanf("%lf", &c[1]);
    /* se calcula el determinante */
    aux = a[0]*b[1]-a[1]*b[0];
    if (aux!=0.0) {
        alfa = (c[0]*b[1]-c[1]*b[0])/aux;
        beta = (c[1]*a[0]-c[0]*a[1])/aux;
        printf("El problema tiene solucion:\n");
        printf("alfa=%lf\nbeta=%lf\n", alfa, beta);
    } else {
        printf("a y b son linealmente dependientes.\n");
        if (a[0]*c[1]==c[0]*a[1]){
            printf("c es colineal con a y b: infinitas soluciones.\n");
            alfa = (a[0]*c[0])/(a[0]*a[0]+a[1]*a[1]);
            beta = (a[1]*c[0])/(a[0]*a[0]+a[1]*a[1]);
            printf("La solucion de modulo minimo es:\n");
            printf("alfa=%lf\nbeta=%lf\n", alfa, beta);
        } else
            printf("No hay solucion.\n");
    }
}
```

Comentario: El algoritmo para la resolución de este programa (resolución en C de un sistema de dos ecuaciones lineales con dos incógnitas), se ha basado en la regla de Cramer y se tienen en cuenta los diferentes valores del determinante de la matriz del sistema definido por la variable ***aux***.

Respecto a la comparación numérica de variables tipo ***double*** hay que decir que es preferible realizar la comparación con números muy pequeños que nos marquen un límite, por ejemplo, la sentencia ***if(a[0]*c[1]==c[0]*a[1])*** sería más recomendable poner ***if(abs(a[0]*c[1]-c[0]*a[1])<1.0e-10)***. Esto es debido a que en el manejo de números decimales en el ordenador, siempre existe un error correspondiente al número máximo de cifras significativas que puede almacenar la variable. Por ese motivo, en este caso consideraríamos "cero" los valores absolutos inferiores a 1.0e-10.

Para calcular la solución de modulo mínimo en el caso de ser ***a*** y ***b*** colineales y ser ***c*** colineal a su vez con ***a*** y ***b***, utilizamos una operación de máximos y mínimos que da como solución:

alfa = a[1]*c[1]/(a[1]*a[1] + a[2]*a[2])

beta = a[2]*c[1]/(a[1]*a[1] + a[2]*a[2])

EJERCICIO 3.10: REPASO DE LA TABLA DE MULTIPLICAR.

¡Qué mejor momento que éste para recordar aquellos tiernos años en los que aprendimos a multiplicar! Te proponemos este programa que imprime los 21 primeros números y sus repectivos cuadrados de tres formas distintas. Para ejecutar de forma pausada el programa hemos introducido la función ***getchar( )*** que espera que pulses cualquier tecla para capturar el carácter correspondiente (en este caso no se hace nada con dicho carácter, que está disponible como valor de retorno; es un simple truco para que el ordenador espere hasta que pulsemos una tecla cualquiera) y continuar ejecutando el programa. Guarda el programa como ***cuadrados.c***.

Solución comentada al Ejercicio 3.10.

```
/* fichero cuadrados.c */

# include <stdio.h>
void main(void) {
    int num=1;

    while (num<21) {
        printf("%10d %10d\n", num, num*num);
        num=num+1;
    }
```

```
    getchar();
    num=1;
    while (num<21) {
        printf("%10d %10d\n", num, num*num);
        num+=1;
    }
    getchar();
    num=1;
    while (num<21) {
        printf("%10d %10d\n", num, num*num);
        num++;
    }
}
```

Comentario: En este programa las instrucciones: ***num=num+1***, ***num+=1*** y ***num++***, las podrás utilizar de la forma que más te convenga, es decir, que la mayor o menor utilidad de una sobre otra se dará en la medida que quieras simplificar tus programas que realizarás más adelante. Como ya hemos dicho, la macro ***getchar()*** espera a que el usario teclee un valor cualquiera para continuar ejecutando el programa. Es una forma de esperar una señal del usario para continuar con la solución del programa. Se puede observar que si se pulza un carácter cualquiera seguido de un ***Intro***, en realidad se han pulsado dos caracteres y el programa tiene suficiente para llegar hasta el final. Si se pulsa sólo el ***Intro*** hay que pulsarlo dos veces.

PRÁCTICA 4.

EJERCICIO 4.1: LEER UNA PALABRA Y ESCRIBIRLA AL REVÉS.

Para leer una palabra, contar las letras y escribirla al revés, lo primero que hay que hacer es pedir la palabra y almacenarla. Una vez que tenemos la palabra en un ***array***, para contar el número de letras se chequea letra por letra (bucle ***while***) sumando 1 cada vez. Este bucle finaliza cuando el carácter leído es el carácter '\0' que indica el final de la palabra. Este carácter, como ya se ha dicho antes, es introducido por la función ***scanf( )*** de manera automática. Para escribir la palabra al revés, basta con leerla empleando un bucle que cuente hacia atrás.

Guarda este programa como ***alreves.c***.

Solución comentada del Ejercicio 4.1.

```
/* fichero alreves.c */
/* Este programa lee una palabra y la escribe al revés */

#include <stdio.h>

void main (void) {
    char c, palabra[21];
    int i;
    printf("Teclea una palabra de menos de 20 letras:\n");
    scanf("%s", palabra);
    i=0;
    while (palabra[i++]!='\0')
        ;
    printf("%s tiene %d letras.\n", palabra, i-1);
    printf("%s escrita al reves es: ", palabra);
    while (i>0)
        printf("%c", palabra[--i]);
    printf("\n");
}
```

Comentario: La forma de almacenar el array leyéndolo del teclado es mediante la función ***scanf()***. Luego, mediante un bucle ***while*** se determina el número de letras. La forma de chequear la condición de final del bucle es por reconocimiento del carácter '**\0**' que tienen todas las cadenas de caracteres al final. El contador incluye el carácter de fin de la cadena.

La cadena se podría haber rellenado de otra manera, como se verá en el Ejercicio siguiente. También el número de caracteres de la cadena se puede determinar mediante otra forma: llamando a la función de libreria ***strlen(char*)***.

Una vez que conocemos el número de caracteres de la cadena, se puede escribir el *array* al revés mediante un bucle con un contador decreciente. Conviene utilizar el operador ***--i*** para que decrezca antes de ser utilizado.

EJERCICIO 4.2: LEER UNA FRASE (LÍNEA) Y ESCRIBIRLA AL REVÉS.

Basándote en el programa anterior, realiza un programa que lea una línea de texto y a continuación la escriba al revés. La diferencia principal con el Ejercicio anterior está en que una línea puede contener varias palabras, es decir caracteres en blanco. La función ***scanf()***, tal como se ha utilizado en el Ejercicio anterior, sólo lee la primera palabra de la frase, pues la lectura se detiene al llegar al primer carácter en blanco. En *Aprenda ANSI C como…* se sugieren dos formas de leer líneas completas, una con la función ***scanf()*** y otra con la macro ***getchar()***. Puedes utilizar cualquiera de las dos.

Te sugerimos que pruebes el programa con una de las siguientes frases, que se leen igual de derecha a izquierda y de izquierda a derecha (llamadas ***palíndromos***): "dabale arroz a la zorra el abad"; "a ti no bonita". Si utilizas la macro ***getchar()*** guarda el programa con el nombre ***frase.c*** y si utilizas ***scanf()*** guárdalo como ***frase2.c***.

Solución comentada del Ejercicio 4.2.

```
/* fichero frase.c */
#include <stdio.h>

void main(void) {
    char c, frase[100];
    int i, n;

    printf("Introduce una frase de menos de 100 letras.\n");
    printf("Para finalizar pulsa ^Z:\n");
    i=0;
    while((c=getchar())!=EOF) {
        frase[i]=c;
        i++;
    }
    frase[i]='\0';
    printf("\n");
    for (n=i; n>=0; n--)
        putchar(frase[n]);
    printf("\n");
}
```

```
/* fichero frase2.c */
#include <stdio.h>
#include <string.h>

void main(void) {
    char frase[100];
    int i, n;

    printf("Introduce una frase de menos de 100 letras.\n");
    printf("Para finalizar pulsa Intro:\n");
    i=0;
    scanf(" %[^\n]", frase);
    printf("\n");
    for (n=strlen(frase)-1; n>=0; n--)
        putchar(frase[n]);
    printf("\n");
}
```

Comentario: En este caso, la lectura del array se realiza mediante un bucle ***while*** cuya condición de ejecución es la de encontrar un carácter distinto del ***carácter fin de fichero*** (EOF). Cuando en vez de leer la frase desde un fichero, ésta se introduce desde teclado, el carácter equivalente al EOF es ***control-z***, que también se suele denotar como ***^z***. Una solución más sencilla a este ejercicio consiste en sustituir la línea ***scanf(" %s", palabra);*** del programa ***alreves.c*** por ***scanf(" %[^\n]", frase);*** El fichero ***frase2.c*** muestra esta nueva versión del programa.

Una vez almacenada la frase en el array, se procede a escribirla al revés de la misma forma que se hacía en el programa anterior. En este caso se ha utilizado la macro de librería ***putchar(c);*** en lugar de ***printf()***, de modo que la frase se escribe carácter a carácter con dicha macro.

EJERCICIO 4.3: TRANSFORMAR UN TEXTO.

El siguiente programa lee un texto cualquiera y lo escribe al revés transformando las mayúsculas en minúsculas y viceversa. Para esto último basta tener en cuenta que la diferencia entre el código ASCII de una letra mayúscula y el de la misma letra en minúscula es la misma para todas las letras del abecedario. Hay que tener cuidado para no modificar los caracteres de estilo tales como espacio en blanco (' '), tabulador ('\t'), coma (,), ... Guarda el programa como ***inverso.c***.

Como ya se ha visto, en C se utiliza la constante simbólica ***EOF*** para indicar el final de un fichero (esta constante está definida en el fichero ***stdio.h***). Su valor numérico es (−1) y equivale también a ***<control>z*** (***^z***) cuando se introduce el texto desde teclado.

Solución comentada del Ejercicio 4.3.

```
/* fichero inverso.c */
/* Este programa convierte las mayusculas en minusculas y viceversa */
/* y escribe el texto cambiado al reves */

#include <stdio.h>

void main(void) {
    int ch;
    char texto[101];        /* limite de caracteres del texto */
    int i, n, dif;

    dif='a'-'A';
    printf("Introduce un texto de no mas de 100 caracteres.\n");
    printf("Pulsa ^Z para finalizar:\n");
    i=0;
    while ((ch=getchar())!=EOF) {
       if ((ch>='a')&&(ch<='z'))      /* es minúscula */
          ch-=dif;
       else if ((ch>='A')&&(ch<='Z')) /* es mayúscula */
          ch+=dif;
       texto[i++]=ch;
    }
    texto[i]='\0';
    for (n=i; n>=0; n--)
        printf("%c", texto[n]);
    printf("\n");
}
```

Comentario: En este programa se utiliza la forma de introducir la cadena como ya se hizo en el programa ***frase.c***. Para cambiar las mayúsculas a minúsculas y viceversa hay que chequear el carácter y ver si es alfabético o no. Una vez que se sabe que es un carácter alfabético, hay que tener en cuenta que el código ASCII correspondiente a la "A" es el 65 y el correpondiente a la "a" es el 97, es decir, la diferencia entre ambos es 32. La misma diferencia se mantiene para otras letras entre la minúscula y la mayúscula

En lugar de teclear el texto cada vez, puedes crear un fichero de texto llamado ***texto1.txt*** y redireccionar la entrada del programa con el operador (<) si ejecutas el programa desde la línea de comandos, en la forma: ***>inverso < texto1.txt***.

EJERCICIO 4.4: MODIFICAR EL EJERCICIO 4.3.

Modifica el programa anterior de manera que el texto original se lea del fichero ***texto2.txt***, que inicialmente puede ser una copia del fichero ***texto1.txt*** introducido en el Ejercicio anterior. Para leer carácter a carácter puedes utilizar la función ***getf()*** y, una vez modificado el texto se puede escribir mediante ***putf()*** ó ***fprintf()***, a continuación del texto original, en el mismo fichero ***texto2.txt***. Llama a este programa ***inverso2.c***.

Solución comentada del Ejercicio 4.4.

```
/* fichero inverso2.c */

#include <stdio.h>
#include <stdlib.h>
void main(void) {
    int ch;
    char texto[100];
    int i, n, dif;
    FILE *fi;
    /* se abre el fichero */
    fi = fopen("texto2.txt", "r+");
    dif = 'a'-'A';
    i=0;
```

```
    while ((ch=getc(fi))!=EOF) {
        if ((ch>='a')&&(ch<='z'))
            ch-=dif;
        else if ((ch>='A')&&(ch<='Z'))
            ch+=dif;
        texto[i]=ch;
        i++;
    }
    texto[i]='\0';
    for (n=i; n>=0; n--)
        putc(texto[n], fi);
    fclose(fi);
}
```

Comentario: El programa es idéntico al anterior con la salvedad del manejo de ficheros. Hay que declarar un puntero al fichero mediante la declaración ***FILE *fi;***. Este puntero se utilizará a la largo del programa para designar el fichero. En primer lugar, este fichero hay que abrirlo mediante la función ***fopen()***, en la que se definen el nombre del fichero y el modo en el que se abre. En el resto del programa lo único que cambia son las instrucciones de entrada y salida: ***putc()*** sustituye a ***putchar()*** y ***getc()*** sustituye a ***getchar()***.

EJERCICIO 4.5: ORDENAR ALFABÉTICAMENTE.

El siguiente programa pide diez palabras y las ordena alfabéticamente. Guárdalo como ***ordenaPal.c***. La gran novedad que presenta este programa es la reserva dinámica de memoria mediante la función ***malloc()***.

Pensemos por un momento en el objetivo de este programa: queremos que lea diez palabras, las almacene y pueda compararlas para ordenarlas. ¿Dónde podemos almacenar una palabra? Obviamente en un ***array*** de caracteres. ¿Y diez palabras? Podríamos utilizar diez arrays, pero ya que queremos relacionarlas entre sí resulta mucho más útil emplear una ***matriz de caracteres***, que no es más que un array de arrays. Cada palabra se almacenaría en una fila de la matriz. Ahora bien, no todas las palabras tendrán el mismo número de letras, lo que nos hace intuir que no hay por qué reservar espacio para una matriz rectangular, sino para una matriz de diez filas donde cada una de ellas tendrá las posiciones de memoria necesarias para almacenar todos los caracteres que componen la palabra más uno (el carácter '\0'). Volviendo a insistir: sabemos que necesitamos una ***matriz de 10 filas pero de número de columnas variable***. El procedimiento es muy sencillo: leemos la palabra, vemos cuánto ocupa y entonces reservamos el espacio necesario. La palabra leída se almacena temporalmente en una variable array auxiliar que hemos llamado ***temp[ ]***, que permite almacenar palabras de hasta veinte letras. Para utilizar las funciones que manejan cadenas de caracteres hay que incluir la librería ***string.h***.

Veamos qué hacen exactamente las siguientes intrucciones del programa:

```
/* Esta intrucción declara cadena como un doble puntero (o puntero a puntero) */
char **cadena;
/* Esta intrucción reserva memoria para un vector de punteros, llamado también cadena[ ] */
cadena = malloc(10*sizeof(char*));
```

El vector ***cadena[ ]*** contiene diez posiciones de memoria, cada una de las cuales contiene un puntero a ***char***. La función ***malloc()***, perteneciente a ***stdlib.h***, tiene como valor de retorno un puntero al primer elemento de la zona reservada y se almacena en ***cadena*** que ya se declaró en la instrucción antes comentada. Por lo tanto ***cadena*** apunta a ***cadena[0]***, que a su vez apuntará luego al primer carácter de la primera palabra.

$$\text{Vector de punteros} : \begin{Bmatrix} cadena[0] \\ cadena[1] \\ cadena[2] \\ \\ cadena[9] \end{Bmatrix}$$

```
scanf("%s", temp); /* Esta intrucción lee una palabra y la almacena en el array temp[ ] */
cadena[i] = malloc((strlen(temp)+1)*sizeof(char));
```

La instrucción anterior reserva la memoria necesaria, en bytes, para la palabra almacenada en ***temp[ ]*** (incluyendo el carácter '\0'). El valor de retorno, que es la dirección del primer carácter de la palabra, se almacena en el puntero ***cadena[i]***.

Sólo queda por comentar que el algoritmo que se sigue para ordenar las palabras es análogo al que se utilizó para ordenar un conjunto de números.

Solución comentada al Ejercicio 4.5.

```
/* fichero ordenaPal.c */
/* Este programa pide diez palabras y las ordena por orden alfabético */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main(void) {
    char **cadena;   /* declaración de puntero a matriz de caracteres */
    int i, j;
    char temp[20];  /* declaración del array auxiliar */
    char *aux;      /* decalaracion de puntero a carácter, auxiliar */

    printf("%s%s\n", "Este programa ordena diez palabras ",
        "introducidas por teclado.");
    printf("Introduce las diez palabras:\n");
    cadena = malloc(10*sizeof(char*));
    for (i=0; i<10; i++) {
        printf("Palabra %d: ", i+1);
        scanf("%s", temp);
        cadena[i]=malloc((strlen(temp)+1)*sizeof(char));
        strcpy(cadena[i], temp);
    }
    /* se ordenan las palabras alfabéticamente */
    for (i=0; i<9; i++) {
        for(j=i+1; j<10; j++) {
            if ((strcmp(cadena[i], cadena[j]))>0) {
                aux=cadena[i];
                cadena[i]=cadena[j];
                cadena[j]=aux;
            }
        }
    }
    printf("La cadena ordenada es:\n");
    for (i=0; i<10; i++)
        printf("%s\n", cadena[i]);
}
```

Comentario: Este programa tiene un gran interés, por realizar reserva dinámica de memoria. Vamos a reservar espacio para 10 palabras, que pueden ser de longitud diferente. Por eso se lee primero la palabra, se ve qué longitud tiene con la función ***strlen()***, y entonces se le reserva memoria con un espacio adicional para el carácter fin de cadena. La cadena ***temp[ ]*** se utiliza para almecenar las palabras nada más leerlas, cuando todavía no se sabe su número de letras.

El método de ordenación utilizado es el mismo que se utilizó en una práctica anterior para ordenar números. La comparación se hace por medio de la función ***strcmp()***, que devuelve cero si las cadenas son iguales, y un número positivo o negativo según la primera cadena sea alfabéticamente posterior o anterior a la segunda cadena, respectivamente. Otro punto importante es que no se permutan las palabras, sino los punteros que apuntan a su primera letra; el resultado es el mismo, pero con mucho menos trabajo.

EJERCICIO 4.6: MODIFICAR EL EJERCICIO 4.5.

Modifica el programa anterior de manera que ordene alfabéticamente diez nombres completos de personas compuestos de primer apellido, segundo apellido y nombre. La diferencia principal está en

leer líneas completas en lugar de palabras. Guárdalo con el nombre ***ordenaNom.c***. Para probar el programa te aconsejamos que lo hagas con sólo tres o cuatro nombres.

Solución comentada del Ejercicio 4.6.

```
/* fichero ordenaNom.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(void) {
    char **cadena, *aux;
    int i, j, n=3;
    char temp[20];

    printf("%s%s\n", "Este programa ordena diez nombres ",
        "introducidos por teclado.");
    printf("Introduce los diez nombres:\n");
    cadena = (char**)malloc(10*sizeof(char*));

    for (i=0; i<n; i++) {
        printf("Nombre %d: ", i+1);
        scanf(" %[^\n]", temp);
        cadena[i]=(char*)malloc((strlen(temp)+1)*sizeof(char));
        strcpy(cadena[i], temp);
    }

    for (i=0; i<n-1; i++)
        for (j=i+1; j<n; j++)
           if ((strcmp(cadena[i], cadena[j]))>0) {
               aux=cadena[i];
               cadena[i]=cadena[j];
               cadena[j]=aux;
           }
    printf("La lista ordenada es:\n");
    for (i=0; i<n ; i++)
        printf("%s\n", cadena[i]);
    /* se libera la memoria reservada */
    for (i=0; i<n; i++)
        free(cadena[i]);
    free(cadena);
}
```

Comentario: La principal diferencia entre este ejercicio y el anterior reside en la forma de introducir la cadena de caracteres: En el ejercicio anterior se utilizó ***scanf("%s", temp)***, pero esta forma de llamar a la función se detiene al encontrar un blanco, un '**\t**' o un '**\n**'. Por ello, en este ejercicio se emplea la función ***scanf(" %[^\n]", temp)***, que se detiene sólo cuando llega al carácter '**\n**'.

Otros aspectos interesantes del programa anterior es la utilización de un ***cast***, en la forma ***(char**)***, para convertir el valor de retorno de la función ***malloc()***, que es un puntro genérico tipo ***void***, en un puntero a puntero a carácter. Observa que al final del programa se dejan las cosas ordenadas, liberando con la función ***free()*** la memoria reservada con ***malloc()*** en orden inverso al de reserva.

EJERCICIO 4.7: RECUENTO DE CARACTERES DE UN FICHERO.

El siguiente programa lee un texto y cuenta los espacios en blanco, dígitos, letras, cambios de línea y otro tipo de caracteres contiene. Para ello se emplean una serie de condiciones, comparando el carácter leído con cada uno de los caracteres anteriores. Si es igual a alguno de ellos se incrementa en uno el valor de la variable que se emplea como contador. Guarda el programa con el nombre ***recuento.c***.

Solución comentada al Ejercicio 4.7.

```
/* fichero recuento.c */
/* Este programa cuenta caracteres de un fichero */

#include <stdio.h>
void main(void) {
    int nBlancos=0, c, nDigitos=0;
    int nLetras=0, nNulineas=0, nOtros=0;
     printf("Introduce un texto con dos o mas lineas. Termina con Intro, ^z e Intro:\n");
    while ((c=getchar()) != EOF)
        if (c==' ')
            ++nBlancos;
        else if (c>='0' && c<='9')
            ++nDigitos;
        else if (c>='a' && c<='z' || c>='A' && c<='Z')
            ++nLetras;
        else if (c=='\n')
            ++nNulineas;
        else
            ++nOtros;

    printf("%10s%10s%10s%10s%10s%10s\n\n",
         "blancos", "digitos", "letras", "lineas", "otros", "total");
    printf("%10d%10d%10d%10d%10d%10d\n\n",
          nBlancos, nDigitos, nLetras, nNulineas, nOtros,
          nBlancos+nDigitos+nLetras+nNulineas+nOtros);
}
```

EJERCICIO 4.8: MODIFICAR EL EJERCICIO 4.7 PARA CONTAR LAS PALABRAS DE UN FICHERO.

Se trata de realizar un programa que cuente el número de palabras contenidas en un fichero de texto. El programa se llamará ***contador.c*** y deberá leer el contenido del fichero y mostrar por pantalla el número de palabras encontrado.

Para contar las palabras, se puede suponer en principio que los ***separadores de palabras*** son un único espacio en blanco, o un único tabulador, o un único salto de línea. El programa irá leyendo carácter a carácter e incrementará en una unidad una variable cada vez que se encuentre una letra precedida por uno de estos caracteres. Hay que tener en cuenta que la primera palabra del fichero puede no estar precedida por alguno de estos caracteres.

Puedes desarrollar una versión más complicada de este programa, que tenga en cuenta que entre cada dos palabras puede haber uno o más espacios en blanco, o un espacio en blanco y un tabulador, o −en general− cualquier combinación de espacios en blanco, tabuladores y saltos de línea. Para ayudarte en la realización de este programa tan difícil, te proponemos el siguiente algoritmo escrito en seudocódigo:

```
while (caracter leido distinto de EOF) do {
    if (estoy dentro de una palabra)
        then [if (encuentro un separador) then (salgo fuera)
                 else (no hago nada)]
    else if [(encuentro separador) then (no hago nada)
             else (entro dentro de una palabra, palabra++)]
}
```

Construye tu propio fichero de comprobación de este programa que contenga un mínimo de 20 palabras en un mínimo de 4 líneas. Deberá contener también algún tabulador. Este fichero se llamará ***contador.c*** y contará las palabras de un fichero llamado ***palabras.txt***.

Solución comentada del Ejercicio 4.8.

```
/* fichero contador.c */

#include <stdio.h>
void main(void) {
    char c;
    int dentro=0, numPal=0;
    FILE *fi;
    fi=fopen("palabras.txt", "r+");
    while ((c=getc(fi))!=EOF) {
        if (((c>='a')&&(c<='z')) || ((c>='A')&&(c<='Z'))) {
            if (dentro==0) {
                numPal+=1;
                dentro=1;
            }
        } else if (dentro==1)
            dentro=0;
    }
    printf("El numero de palabras del texto es %d\n", numPal);
}
```

Comentario: La dificultad de este programa reside casi exclusivamente en la comprensión del algoritmo.

Para contar las palabras se define una variable llamada ***numPal*** que indicará el número de palabras que se van contando. Se define también otra variable que se llama ***dentro*** que indica si nos encontramos dentro o fuera de una palabra; si esta variable vale 1 nos encontramos *dentro* y si vale cero nos encontramos *fuera*. El algoritmo que sigue este programa es:

1 Leo el carácter

2 Si el carácter es alfabético, se chequea si estoy dentro o fuera de la palabra (***dentr=1*** ó ***dentro=0***).

  2.1 Si ***dentro==0*** significa que estaba fuera de palabra y he entrado en una palabra nueva, por lo que hacemos ***numPal+=1*** (incrementamos una palabra en el contador) y ***dentro=1*** (volvemos a estar dentro de palabra).

  2.2 Si ***dentro==1*** significa que estábamos metidos en una palabra y que seguimos dentro de ella. No hacemos nada.

3 Si el carácter no es alfabético, chequeo si he salido o no de una palabra (chequeo ***dentro***).

  3.1 SI ***dentro==1*** significa que el carácter anterior era alfabético y, por tanto, he salido de una palabra: hacemos ***dentro=0*** porque ahora nos hemos salido de ella.

  3.2 Si ***dentro==0*** significa que no estábamos en ninguna palabra y que seguimos sin estar, por tanto no hacemos nada.

EJERCICIO 4.9: UN FICHERO SECRETO.

El siguiente programa lee un texto y lo escribe en clave. Escribirlo en clave no es más que sumar una cantidad al número en código ASCII que corresponde a cada carácter. Esta cantidad o clave se ha de teclear en cada ejecución del programa. Llama al fichero de este programa ***secreto.c***.

Solución comentada al Ejercicio 4.9.

```
/* fichero secreto.c */
/* Este programa encripta un texto */

#include <stdio.h>

void main(void) {
    char ch;
    int n;

    printf("Introduce la clave (un numero entero entre 1 y 10): ");
    scanf("%d", &n);
    getchar(); /* para eliminar el \n del buffer de entrada */
    printf("Introduce los caracteres del texto.\n");
    printf("Pulsa Intro+^Z para finalizar:\n");
```

```
    while((ch=getchar())!=EOF) {
        if (ch=='\n')
            printf("%c", ch);
        else
            printf("%c", ch+n);
    }
}
```

Comentario: El comportamiento de este programa pude depender del compilador concreto de C con el que se compile y ejecute. Este comentario se refiere al compilador Visual C/C++ de Microsoft. Las entradas/salidas de este programa se ven afectadas por los "buffers". Por ejemplo, al leer la clave (un número entero) con la instrucción ***scanf()*** queda el carácter \n en el buffer de entrada; la instrucción ***getchar()*** se introduce para vaciar dicho buffer, es decir para eliminar de la entrada el '***\n***' tecleado después de introducir la clave. Se teclea un conjunto de caracteres y se pulsa ***Intro***: entoces se escriben los caracteres en "clave". Para que el programa termine es necesario teclear ***ctrl.-z*** e ***Intro***.

EJERCICIO 4.10: CREAR FUNCIONES ANÁLOGAS A STRLEN(), STRCPY(), STRCAT() Y STRCMP().

Realiza un programa principal que utilice varias cadenas de caracteres y llame a unas funciones programadas por ti que realicen las mismas tareas que ***strlen( )***, ***strcpy***( ), ***strcat***( ) ***strcmp( )***. Deberás llamar a tus funciones con un nombre distinto a las que existen en la librería de C. Te sugerimos los siguientes nombres:

strlen( ) ⇒ cuentaCaracteres( )

strcpy( ) ⇒ copiaCadenas( )

strcat( ) ⇒ concatenaCadenas( )

strcmp( ) ⇒ comparaCadenas( )

Guarda estas funciones en un fichero análogo a ***string.h*** y al que llamarás ***misfunc.h*** e incluye en el encabezamiento del programa principal la instrucción:

```
#include "misfunc.h"
```

El programa principal se llamará ***cadenas.c***.

Solución comentada del Ejercicio 4.10.

```
/* fichero cadenas.c */
#include <stdio.h>
#include "misfunc.h"

unsigned cuentaCaracteres(const char*);
char* copiaCadenas(char*, const char*);
char* concatenaCadenas(char*, const char*);
int comparaCadenas(const char*, const char*);

void main(void) {
    char car1[100] = "Esto es una cadena";
    char car3[100] = "Esto es otra cadena";
    char car2[100] = "";

    printf("car1 es: \"%s\"\n", car1);
    printf("car2 es: \"%s\"\n", car2);
    printf("car3 es: \"%s\"\n", car3);
    printf("No de caracteres de car1: %d\n", cuentaCaracteres(car1));
    copiaCadenas(car2, car1); /* se copia car1 en car2 */
    printf("car2 ahora es: \"%s\"\n", car2);
    printf("car3+car1 es: \"%s\"\n", concatenaCadenas(car3,car1));
    printf("car1-car2 es %d\n", comparaCadenas(car1,car2));
    printf("car2-car3 es %d\n", comparaCadenas(car2,car3));
}
```

Comentario: El programa principal crea tres cadenas de caracteres y luego opera con ellas de forma análoga a como lo hacen las funciones ***strlen()***, ***strcpy()***, ***strcat()*** y ***strcmp()***. Previamente al comienzo de la función ***main()*** se ha incluido la declaración de las

funciones que luego van a ser utilizadas. Esta declaración no es necesaria porque se ha incluido el fichero ***misfunc.h***, en el que están definidas las funciones posteriormente declaradas. La definición puede sustituir en este caso a la declaración. Observa que los argumentos de las funciones que no van a ser modificados se declaran como ***const***.

```
/* fichero misfunc.h */

unsigned cuentaCaracteres(const char* carac) {
    unsigned i=0;
    while (carac[i]!='\0')
        i++;
    return i;
}
```

Comentario: La función ***cuentaCaracteres()*** cuenta los caracteres de la cadena detectando el carácter o marca de final de cadena. Por lo demás, esta función es muy sencilla.

```
char* copiaCadenas(char* carac1, const char* carac2) {
    int i=0;
    while ((carac1[i]=carac2[i])!='\0')
        i++;
    return carac1;
}
```

Comentario: La función ***copiaCadenas()*** anterior tiene como punto interesante el copiar el carácter antes de compararlo con la marca de fin de cadena. Para ello la asignación debe ir entre paréntesis, de modo que se realice antes de comparar con el carácter de fin de cadena. De este modo se asugura que dicho carácter es también copiado por la función. Se supone que ***carac1*** tiene espacio suficiente.

```
char* concatenaCadenas(char* carac1, const char* carac2) {
    int i=-1, j=0;
    while (carac1[++i]!='\0')
        ;
    while (carac2[j]!='\0')
        carac1[i++]=carac2[j++];
    carac1[i]='\0';
    return carac1 ;
}
```

Comentario: Se detecta primero el final de la cadana ***carac1[ ]***. Después se van copiando a continuación todos los caracteres de ***carac2[ ]*** y finalmente se añade el carácter fin de cadena.

```
int comparaCadenas(const char* carac1, const char* carac2) {
    int i=0, dif;
    while (carac1[i]!='\0') {
        dif=carac1[i]-carac2[i];
        if (dif==0)
            i++;
        else
            return (dif);
    }
    if (carac2[i]=='\0')
        return(0);
    else
        return(-carac2[i]);
}
```

Comentario: Para comparar las dos cadenas, lo que hacemos es recorrer la primera de ellas (***carac1***), hasta que encontramos el caracter ***\0***. Dentro de este bucle se restan los caracteres en posiciones equivalentes de las dos cadenas. Recuérdese que un carácter se puede manejar como carácter o se puede considerar su equivalencia ASCII. Así, si restamos los dos caracteres, lo que en realidad estamos haciendo es restar su número ASCII. Por tanto, si la diferencia entre los dos caracteres es cero, significa que son iguales. El bucle continúa hasta que encuentra dos caracteres distintos, con lo cual no se cumple la sentencia ***if (dif==0)*** y se ejecuta ***return(dif);*** con lo que se devuelve un valor distinto de cero y, por tanto, las cadenas son distintas. Si se llega al final de la primera cadena y en esa posición se tiene que la segunda cadena acaba también (***carac2[i]='\0'***) es que las cadenas son iguales y se devuelve un cero. Si se llega al final de la primera cadena antes de llegar al final de la segunda se devuelve el primer carácter diferente de ésta cambiado de signo.

PRÁCTICA 5.

EJERCICIO 5.1: EVALUACIÓN DE LA FUNCIÓN EXP(X) POR DESARROLLO EN SERIE.

Según la fórmula de Taylor, podemos expresar la función exponencial e^x mediante la siguiente serie de potencias de *x*:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

En el programa que te presentamos a continuación, las variables ***z*** y ***numero***, *argumentos formales* de la función, reciben respectivamente una copia de ***x*** y ***terminos***, que son los *argumentos actuales*. La variable ***frac*** va almacenando la fracción a sumar en cada término del sumatorio. Así para cada nuevo sumando no hay más que multiplicar el anterior valor por ***z/i***:

$$\frac{z^4}{4!} = \frac{z^3}{3!} * \frac{z}{4}$$

Guarda el programa en un fichero con el nombre ***serie.c***.

Solución comentada al Ejercicio 5.1.

```
/* fichero serie.c */

#include <stdio.h>
#include <math.h>
void main(void) {
    int terminos;
    double x;
    double serie(double ,int );

    printf("%s%s",
        "Este programa calcula \"e\" elevado a \"x\" por desarrollo ",
        "en serie\ncon el numero de terminos que quieras\n");
    printf("Teclea x: ");
    scanf("%lf",&x);
    printf("Teclea el numero de terminos: ");
    scanf("%d", &terminos);
    printf("%s%lf con %d %s%16.6lf\n",
        "El resultado de e elevado a ", x, terminos, "terminos es ",
        serie(x, terminos));
    printf("Error cometido: %16.6lf\n", serie(x, terminos)-exp(x));
}

double serie(double z, int numero) {
    double miserie=1, frac;
    int i;
    frac=z;
    miserie+=frac;
    i=2;
    if (numero==1) {
        miserie=1;
        return miserie;
    }
    while (i<=numero) {
        frac=frac*z/i++;
        miserie+=frac;
    }
    return miserie;
}
```

Comentario: La complejidad de este tipo de programas reside en el algoritmo concreto que se utilice. En este caso, la condición que tiene el bucle para finalizar la iteración es el número de términos, que se ha introducido por el teclado como dato. La forma de calcular la serie es añadiendo una nueva fracción a la serie. Para el cálculo de esta nueva fracción utilizamos el valor que tenía la fracción en la iteración anterior, según explica el enunciado del ejercicio. En este tipo de sumas los errores son más pequeños sumando primero los términos más pequeños y luego los más grandes, pues al sumar un número pequeño a uno grande unas cuantas cifras del número pequeño se pierden de modo irremediable. Además, es posible que un número pequeño sea despreciable frente a uno grade, pero la suma de muchos números pequeños puede no ser tan insignificante. Si cada número pequeño se suma al número grande individualmente, el resultado puede ser diferente que si todos los número pequeños se suman entre sí primero, y luego el resultado se suma al número grande.

EJERCICIO 5.2: OTRA FORMA DE TERMINAR UN BUCLE

En el programa anterior el bucle ***while*** terminaba al llegar al número de términos solicitado por el usuario. Vamos a emplear ahora otro criterio para terminar ese bucle.

El programa que se presenta a continuación tiene tres variables importantes: ***serie1***, ***serie2*** y ***frac***. Al final del bucle ***while*** tenemos dos variables que serán distintas: ***serie1*** y ***serie2***. La diferencia entre ambas está en ***frac***. Como ***frac*** va disminuyendo su valor, habrá un momento en que los decimales de ***serie1*** y ***serie2*** coincidirán (el ordenador no puede tener información de todos los decimales). Por esta razón, lo que marca el límite del bucle es la comparación entre ***serie1*** y ***serie2***. Este programa se ha realizado sin emplear ninguna función. Guárdalo como ***serie2.c***.

Solución comentada al Ejercicio 5.2.

```
/* fichero serie2.c */

#include <stdio.h>
#include <math.h>
void main(void) {
    double x, serie1, serie2, frac;
    int    i;

    printf("Teclea el valor de x: ");
    scanf("%lf", &x);
    serie1=1.0;
    frac=x;
    serie2=serie1+frac;
    i=2;
    while (serie1!=serie2) {
        serie1=serie2;
        frac=frac*x/i++;
        serie2=serie1+frac;
    }
    printf("Valor estimado: %20.16e\n", serie2);
    printf("Valor exacto: %20.16e.\n", exp(x));
    printf("Numero de terminos utilizados: %d\n", i);
}
```

Comentario: Este ejercicio es similar al anterior. En este caso la diferencia estriba en la condición de finalización de las iteraciones. Para este ejercicio utilizamos un bucle ***while*** y dos series (con los nombres ***serie1*** y ***serie2***) cuya diferencia mutua esta en un término, es decir, ***serie2*** tiene un término más que ***serie1***. La razón de esta forma de calcular la serie la encontramos una vez más en la capacidad de almacenamiento de las variables tipo ***double***: la fracción que calculamos y que añadimos a la serie es cada vez más pequeña y, por tanto, la diferencia entre las dos series también. Llega un momento en el que las cifras significativas que puede almacenar la variable no son apreciables y la fracción que sumamos no representa cambio en las variables: en ese momento las series toman el mismo valor, se igualan y, por tanto, se termina la ejecución del bucle.

EJERCICIO 5.3: DESARROLLO EN SERIE DE *SEN(X)*.

Realiza un programa análogo al anterior que emplee una función de C para calcular ***sen(x)***, cuyo desarrollo en serie es:

$$senx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Guarda el programa como ***serie3.c***.

Solución comentada al Ejercicio 5.3.

```
/* fichero serie3.c */

#include <stdio.h>
#include <math.h>
void main(void) {
    int    i, n;
    double x, serie1, serie2, frac=1;
    double fac(int);
    double pot(double, int);

    printf("Este programa calcula el valor del seno.\n");
    printf("Introduce el valor del angulo en radianes: ");
    scanf("%lf", &x);
    serie2=x;
    serie1=0.0;
     frac=x;
    i=2;
    while (serie1!=serie2) {
        serie1=serie2;
        n=2*i-1;
        frac=-frac*(x*x/(n*(n-1)));
        serie2+=frac;
        i++;
    }
    printf("\nValor estimado: %20.16e\n", serie2);
    printf("Valor exacto:   %20.16e\n", sin(x));
    printf("Numero de terminos utilizados: %d\n", i);
}
```

Comentario: En este programa cada fracción que se añade a la serie se calcula también a partir del anterior. Se tiene en cuenta que los distintos términos cambian de signo alternativamente, y que cada uno es el anterior multiplicado por ***x*** al cuadrado, y dividido por ***(2*i-1)*(2*i-2)***.

EJERCICIO 5.4: MÁXIMO DE UN CONJUNTO DE TRES NÚMEROS.

Realiza un programa principal que lea tres números enteros por teclado, los almacene en tres variables (***x***, ***y***, ***z***) y llame a una función llamada ***maximo()***, con tres argumentos, que devuelva el máximo de estos tres valores. Guarda el programa como ***maxval.c***.

Solución comentada del Ejercicio 5.4.

```
/* fichero maxval.c */

#include <stdio.h>
void main(void) {
    int x, y, z;
    int maximo(int, int, int);

    printf("Introduce tres valores:\n");
    printf("primero: ");
    scanf("%d", &x);
    printf("segundo: ");
    scanf("%d", &y);
    printf("tercero: ");
```

```
    scanf("%d", &z);
    printf("El maximo de los tres valores introducidos es: ");
    printf("%d\n", maximo(x, y, z));
}

int maximo(int a, int b, int c) {
    int max;
    max=a;
    if (b>max)
        max=b;
    if (c>max)
        max=c;
    return max;
}
```

Comentario: Como la función ***maximo()*** devuelve un valor entero. Las variables ***a***, ***b***, y ***c***, argumentos formales de la función, reciben una copia de los argumentos actuales***: x***, ***y***, ***z***. La forma de calcular el máximo de una lista de números es ya conocida.

EJERCICIO 5.5: POTENCIAS

Realiza un programa que calcule las potencias de la 2 a la 10 del número PI y la raíz cuadrada de dicha potencia. Para ello construye una función que calcule la potencia n-ésima de un número cualquiera y utiliza después la función ***sqrt*** de la librería matemática de C. Guarda el programa como ***potencia.c***.

Solución comentada del Ejercicio 5.5.

```
/* fichero potencia.c */

#include <stdio.h>
#include <math.h>
#define PI 3.14159265359

void main(void) {
    int i;
    double poten(double, int);

    for (i=2; i<=10; i++) {
        printf("PI elevado a %d es %5.2lf y su raiz es %5.2lf\n",
            i, poten(PI, i), sqrt(poten(PI, i)) );
    }
}

double poten(double a, int b) {
    int    i;
    double resul=1.0;
    for (i=1; i<=b; i++)
        resul*=a;
    return resul;
}
```

Comentario: Para calcular las potencias de la 2 a la 10 del número PI, se utiliza un bucle ***for*** que va de i=2 a i=10, llamando en cada ejecución a la función ***poten()***. A esta función se le pasa una constante double, que es el número **PI**, y el exponente entero ***i***. La función ***poten()*** sirve para calcular la potencia i-ésima de cualquier número. En este caso, la variable ***a*** de la función toma el valor de **PI** en todas las llamadas, mientras que la variable ***b*** copia el valor de ***i*** que será distinto en cada una de las llamadas.

EJERCICIO 5.6: UNA FUNCIÓN QUE NO TIENE ARGUMENTOS.

Pues sí, hay funciones −y personas− que tienen argumentos. El siguiente programa calcula la diferencia en minutos entre dos horas distintas. La función ***minutos()*** acepta el tiempo que el usuario introduce en horas y minutos y lo convierte a minutos devolviendo este valor entero. La función presenta la particularidad de no tener argumentos. Guarda el programa como ***tiempo.c***.

Otra novedad de este programa es la existencia del carácter ***dos puntos*** (:) entre los dos especificadores de formato, %d y %d, de la instrucción ***scanf()***. Esto hace que el usuario deba introducir los dos puntos entre los números que conforman el formato del tiempo. Estos números son capturados por la función mediante los especificadores %d. Los dos puntos sustituyen a los caracteres de espaciado tradicionales (espacio, tabulador, nueva línea). Así el usuario puede introducir el tiempo en el formato estándar de horas:minutos, utilizando los dos puntos para separarlos.

Solución comentada al Ejercicio 5.6.

```
/* fichero tiempo.c */
/* calcula la diferencia entre dos horas distintas */

#include <stdio.h>
void main(void) {
    int minutos1, minutos2;
    int minutos(void);

    printf("Escribe la primera hora (formato 3:22): ");
    minutos1=minutos();        /* obtiene los minutos */
    printf("Escribe la segunda hora (posterior): ");
    minutos2=minutos();        /* obtiene los minutos*/
    printf("La diferencia es %d minutos.\n", minutos2-minutos1);
}

int minutos(void) {
    int horas, mins;
    scanf("%d:%d", &horas, &mins);
    return (horas*60+mins);
}
```

EJERCICIO 5.7: MODIFICAR EL EJERCICIO 5.6.

No podemos dejar pasar la ocasión de modificar un programa tan sencillo como el anterior. Modifica el programa ***tiempo.c*** de manera que permita introducir las dos horas, incluyendo los segundos, utilizando el formato 3:22:17 y calcule la diferencia en horas, minutos y segundos. Además deberá avisar en el caso de que la segunda hora sea anterior a la primera o si se teclea una hora errónea, es decir, que el número correspondiente a las horas sea mayor que 24 ó el de los minutos mayor que 60, etc. Guarda este programa como ***tiempo2.c***.

Solución comentada al Ejercicio 5.7.

```
/* fichero tiempo2.c */

#include <stdio.h>
void main(void) {
    int horas, minutos;
    long diferencia, segundos1, segundos2;
    long segs(void);

    printf("Escribe la primera hora (formato 3:22:14): ");
    segundos1=segs();
    do {
        printf("Escribe la segunda hora (posterior): ");
        segundos2=segs();
    } while (segundos2<=segundos1);
    diferencia=segundos2-segundos1;
    horas=(int)(diferencia/3600);
    diferencia=diferencia%3600;
    minutos=(int)(diferencia/60);
    diferencia=diferencia%60;
```

```
    printf("La diferencia es %d hora(s), %d minuto(s), %d segundo(s).\n",
        horas, minutos, diferencia);
}

long segs(void) {
    int horas, minutos, segundos, pass = 0;
    long total;
    do {
        pass=0;
        scanf("%d:%d:%d", &horas, &minutos, &segundos);
        if (horas>24)
            pass=1;
        if (minutos>60)
            pass=1;
        if (pass==1) {
            printf("Hora incorrecta!!!!\n");
            printf("Escribe la hora correcta: ");
        }
    } while(pass==1);
    total=(long)horas*3600;
    total+=(minutos*60+segundos);
    return total;
}
```

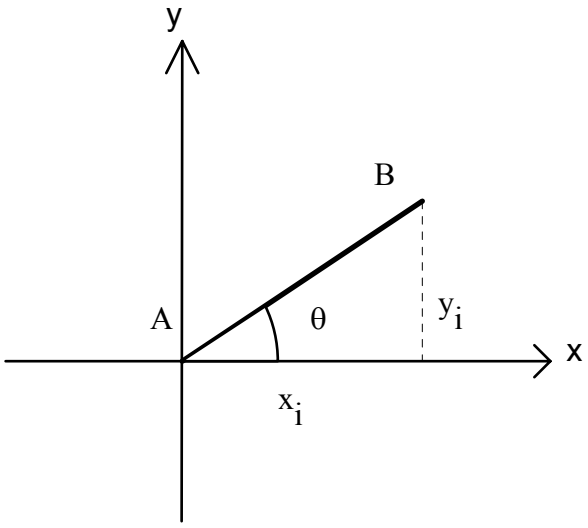
Comentario: Para calcular la diferencia entre las dos horas es necesario convertirlas antes en segundos, para así poder restarlas. Después hay que convertir este número, en segundos, a horas, minutos y segundos. Para obtener las ***horas*** se divide el número entre 3600 segundos/hora, quedándonos con el número entero que resulta. El resto de esta división, es decir, el resto de los segundos se convertirán a minutos dividiendo por 60 segundos/minuto. El número entero resultante de esta división serán los ***minutos*** y el resto los ***segundos***. Dentro de la funcion ***segs()*** aparece un bulce ***do while***, que como ya sabemos siempre se ejecuta al menos una vez. En nuestro caso el bucle se ejecutará sólo una vez, si el usuario ha tecleado correctamente la hora. De lo contrario, es decir, si se equivoca al teclear el número correspondiente a las horas, los minutos o los segundos, la variable ***pass*** tomará el valor 1, que es la condición para que el bucle se vuelva a ejecutar. De esta forma se pueden teclear otra vez los datos.

Obsérvese que la función ***segs()*** devuelve un valor de tipo ***long***, esto es debido a que si la diferencia entre horas es 24 (máximo valor posible). Este valor en segundos (86400) supera la capacidad de una variable de tipo ***int***.

Dentro de ***main()*** también hay otro bucle ***do while***, que se ejecuta más de una vez en el caso de que la segunda hora sea anterior a la primera, dando así la posibilidad de volver a teclear la hora.

EJERCICIO 5.8: GIRO DE UNA BARRA EXTENSIBLE.

En la siguiente figura se muestra una barra AB extensible, es decir, de longitud variable, que gira alrededor de su extremo fijo A. Las coordenadas (x_i, y_i) de su extremo móvil B son conocidas en diez instantes del movimiento.



Las coordenadas del extremo B son las siguientes:

(2, 0), (1.5, 3), (-1, 3), (-1.5, 2), (-2.5, 0), (-2.5, -1), (-1.5, -1.5), (0, -1), (1, -2), (1.5, -1.5)

Estas coordenadas se almacenarán en el fichero ***datos.d***.

Realiza un programa que calcule el ángulo formado por la barra con la horizontal y su longitud, en cada uno de los diez instantes de tiempo. El ángulo se calculará leyendo los datos del fichero ***datos.d***, y utilizando la función arcotangente.

La librería matemática de C dispone de dos funciones para calcular el arcotangente: ***atan()*** y ***atan2()***, cuyos prototipos son:

double atan(double a) que calcula el arcotangente de **a** entre (-π/2, π/2)

double atan2(double b, double a) que calcula el arcotangente de un ángulo cuyo seno es proporcional a **b** y cuyo coseno es proporcional a **a**. De esta forma puede detectarse el cuadrante y devolver un ángulo entre –π y π radianes.

Te pedimos que programes una función similar a ***atan2()*** y que llamarás ***atan2a()*** (empleando para ello ***atan()***), de manera que el ángulo que devuelva esté comprendido entre (-π, π).

La salida del programa deberá ser similar a la siguiente:

```
    En el instante 1 la barra tiene una longitud de 2 y forma un angulo de 0
radianes o de 0 grados con la horizontal.

    En el instante 2 la barra tiene una longitud de 1.765 y forma un angulo de
0.17823 radianes o 10.21 grados con la horizontal.

    En el instante 3 ...
```

(Los valores que se muestran son inventados y no han de concidir con los que calcules).

Guarda el programa como ***barra.c***.

Solución comentada del Ejercicio 5.8.

```
/* fichero barra.c */

#include <stdio.h>
#include <math.h>
#define PI 3.141592654

void main(void) {
    int i;
    FILE *fi;
    double x, y;
    double radianes, grados, longitud;
    double calculaLongitud(double, double);
    double atan2a(double, double);

    fi = fopen("datos.d", "r");
    for (i=1; i<=10; i++) {
        fscanf(fi, "%lf", &x);
        fscanf(fi, "%lf", &y);

        longitud=calculaLongitud(x,y);
        radianes=atan2a(x,y);
        grados=radianes*180.0/PI;

        printf("En el instante %d la barra tiene una longitud de %lf\n",
            i, longitud);
        printf("y forma un angulo de %lf radianes ", radianes);
        printf("o %lf grados con la horizontal.\n", grados);
        getchar();
    }
```

```
    fclose(fi);
}

double calculaLongitud(double x, double y) {
    double resul;
    resul=sqrt(x*x+y*y);
    return(resul);
}

double atan2a(double x, double y) {
    double resul, aux;
    if (x==0.0)
        if (y>0.0)
            resul=PI/2;
        else
            resul=-PI/2;
    else
        aux=atan(y/x);

    if ((x<0.0)&&(x!=0.0))
        if (y>=0.0)         /*si el angulo esta en el segundo cuadrante */
            resul=aux+PI;
        else
           resul=aux-PI;    /* si el angulo esta en el tercer cuadrante */
    else if (x!=0.0)
        resul=aux;          /* si el angulo esta en el primer o cuarto cuadrante */
    return resul;
}
```

Comentario: Este programa va leyendo las coordenadas ***x*** e ***y*** para cada posición de la barra y llama ***calculaLongitud()*** y ***atan2a()*** respectivamente a las funciones que calculan la longitud de la barra y que devuelve el ángulo que forma la barra en radianes.

En cuanto a la función ***atan2a()***, queremos que devuelva un ángulo comprendido entre $-\pi$ y π radianes basándonos en la función ***atan()*** que devulve un ángulo comprendido entre $-\pi/2$ y $\pi/2$. Para ello hay que tener en cuenta que:

Si x>0 e y>0: el ángulo pertenece al primer cuadrante, por lo tanto ***atan2a()=atan()***.

Si x>0 e y<0: el ángulo pertenece al cuarto cuadrante, por lo tanto ***atan2a()=atan()***.

Si x<0 e y>0: el ángulo pertenece al segundo cuadrante, por lo tanto ***atan2a()=atan()+π***.

Si x<0 e y<0: el ángulo pertenece al tercer cuadrante, por lo tanto ***atan2a()=-atan()-π***.

Hay que tener cuidado cuando ***x=0***, pues se realiza una división por cero que produce un error al ejecutar el programa. Esto se puede evitar fácimente, ya que en este caso la barra se encuentra en posición vertical, formando un ángulo de $\pi/2$ (y>0), o $-\pi/2$ (y<0).

EJERCICIO 5.9: MODIFICAR EL EJERCICIO 5.8.

Con el mismo programa principal anterior sustituye la función ***atan2a()*** por la función ***atan2b()***, también programada por ti, ¡cómo no!, que devuelva un ángulo comprendido entre $(0, 2\pi)$ radianes.

Solución comentada del Ejercicio 5.9

```
/* fichero barra2.c */
#include <stdio.h>
#include <math.h>
#define PI 3.141592654
double calculaLongitud(double, double);
double atan2b(double, double);

void main(void) {
   FILE *fi;
   double x, y;
   int i;
   double radianes;
   double grados;
   double longitud;
```

```
   fi=fopen("datos.d", "r");
   for (i=1; i<=10; i++) {
      fscanf(fi, "%lf", &x);
      fscanf(fi, "%lf", &y);
      longitud=calculaLongitud(x,y);
      radianes=atan2b(x,y);
      grados=radianes*180.0/PI;

      printf("En el instante %d la barra tiene una longitud de %lf\n", i, longitud);
      printf("y forma un angulo de %lf radianes o %lf grados con la horizontal.\n",
             radianes, grados);
      getchar();
   }
   fclose(fi);
}

double calculaLongitud(double x, double y) {
   double resul;
   resul=sqrt(x*x+y*y);
   return(resul);
}

double atan2b(double x, double y) {
   double resul, aux;
   if (x==0.0)
      if (y>0.0)
         resul=PI/2;
      else
         resul=3*PI/2;
   else
      aux=atan(y/x);

   if (x<0.0)
      resul=aux+PI;
   else if (x>0.0)
      if (y<0.0)
         resul=2*PI+aux;
      else
         resul=aux;
   return (resul);
}
```

Comentario: El programa es idéntico al anterior, con la salvedad del cálculo de la tangente que se hace de forma similar al programa anterior.

EJERCICIO 5.10: MÁXIMO DE UN CONJUNTO DE TRES NÚMEROS.

No, no se trata de repetir el Ejercicio 2, sino de realizarlo de otro modo. Realiza este nuevo programa, ***maxref.c***, que lea también tres números enteros, los almacene en un vector y llame a una función ***maximo()*** que devuelva el máximo valor de entre los tres. En este caso el vector se pasará por referencia.

Solución comentada del Ejercicio 5.10.

```
/* fichero maxref.c */
#include <stdio.h>

void main(void) {
    int vector[3];
    int maximo(int*);

    printf("Introduce tres valores:\n");
    printf("primero: ");
    scanf("%d", &vector[0]);
```

```
    printf("segundo: ");
    scanf("%d", &vector[1]);
    printf("tercero: ");
    scanf("%d", &vector[2]);
    printf("El maximo de los tres valores introducidos es: ");
    printf("%d\n", maximo(vector));
}

int maximo(int *a) {
    int max;
    max=a[0];
    if (a[1]>max)
        max=a[1];
    if (a[2]>max)
        max=a[2];
    return max;
}
```

Comentario: Este programa es muy fácil de entender y no requiere ningún comentario particular.

EJERCICIO 5.11: OPERACIONES CON VECTORES (EJERCICIO 9 DE LA PRÁCTICA 3)

El siguiente programa es el mismo que se sugirió en la Práctica 3, pero puestos a escribirlo otra vez se ha introducido una novedad: la sentencia ***switch***.

La sentencia ***switch*** funciona de la siguiente manera: se evalúa la expresión que se encuentra a la derecha del ***switch*** y entre paréntesis. A continuación se rastrean las etiquetas hasta que se encuentra una que corresponda a dicho valor; entonces se transfiere el control del programa a dicha línea. Si ninguna de las etiquetas encaja se transfiere el control a la línea marcada con la etiqueta ***default***. La sentencia ***break***, que es opcional, hace que el programa salga del ***switch*** y se dirija a la sentencia situada inmediatamente después del mismo. De no utilizar el ***break***, se ejecutarían todas las sentencias situadas entre la etiqueta correspondiente y el final del ***switch***. Las etiquetas de un ***switch*** deben ser constantes de tipo entero (incluyendo ***char***) o bien expresiones de constantes. ¡Nunca se pueden emplear variables!. Guarda este programa como ***vectors2.c***.

Solución comentada al Ejercicio 5.11.

```
/* fichero vectors2.c */

#include <stdio.h>
void main(void) {
    double vector1[3], vector2[3], vector3[3];
    int i,numero;
    void suma(double *, double *, double *);
    void resta(double *, double *, double *);
    double productoEscalar(double *, double *);
    void productoVectorial(double *, double *, double *);

    printf("Este programa realiza las siguientes operaciones:");
    printf("con dos vectores:\n");
    printf("1.- Suma\n\b2.- Resta\n\b");
    printf("3.- Multiplica escalarmente\n\b");
    printf("4.- Multiplica vectorialmente\n");
    printf("Que operacion deseas realizar? ");
    printf("Teclea su numero correspondiente: ");
    scanf("%d", &numero);
    printf("\nDame las componentes del primer vector:\n");
    for (i=0; i<3; i++) {
        printf("vector1[%d]=", i+1);
        scanf("%lf", &vector1[i]);
    }
    printf("\nDame las componentes del segundo vector:\n");
```

```
    for(i=0; i<3; i++) {
        printf("vector2[%d]=", i+1);
        scanf("%lf", &vector2[i]);
    }
    printf("\n");
    switch (numero) {
        case 1:
            suma(vector1, vector2, vector3);
            printf("La suma de los dos vectores es el vector:\n");
            printf("{ ");
            for (i=0; i<3; i++)
                printf("%lf ",vector3[i]);
            printf("}\n");
            break;
        case 2:
            resta(vector1, vector2, vector3);
            printf("La resta de los dos vectores es el vector:\n");
            printf("{ ");
            for (i=0; i<3; i++)
                printf("%lf ",vector3[i]);
            printf("}\n");
            break;
        case 3:
            printf("El producto escalar de los dos vectores es: %lf\n",
                productoEscalar(vector1, vector2));
            break;
        case 4:
            productoVectorial(vector1, vector2, vector3);
            printf("El producto vectorial de los dos vectores es:\n");
            printf("{ ");
            for(i=0; i<3; i++)
                printf("%lf ", vector3[i]);
            printf("}\n");
            break;
    }
}

void suma(double *a, double *b, double *c) {
    int i;
    for (i=0; i<3; i++)
        c[i]=a[i]+b[i];
}

void resta(double *a, double *b, double *c) {
    int i;
    for (i=0; i<3; i++)
        c[i]=a[i]-b[i];
}

double productoEscalar(double *a, double *b) {
    int i;
    double escalar=0;
    for (i=0; i<3; i++)
        escalar+=a[i]*b[i];
    return(escalar);
}

void productoVectorial(double *a, double *b, double *c) {
    c[0]=a[1]*b[2]-a[2]*b[1];
    c[1]=-a[0]*b[2]+a[2]*b[0];
    c[2]=a[0]*b[1]-a[1]*b[0];
}
```

EJERCICIO 5.12: UNA FUNCIÓN QUE SE LLAMA A SÍ MISMA (FUNCIÓN RECURSIVA): N!.

Este programa calcula el factorial de un número entero *n* utilizando la siguiente fórmula recursiva:

$$n! = n * (n\text{-}1)!$$

teniendo en cuenta que: 1! = 1. Guarda el programa en el disco con el nombre ***factor2.c***.

Solución comentada al Ejercicio 5.12.

```
/* fichero factor2.c */

#include <stdio.h>
void main(void) {
    int n;
    double factor(int);

    printf("Teclea un numero entero positivo: ");
    scanf("%d", &n);
    printf("\nEl factorial de %d es %lf\n", n, factor(n));
}

double factor(int n) {
    if(n<=1)
        return 1.0;
    return (n*factor(n-1));
}
```

Comentario: Las funciones recursivas se llaman a sí mismas, y es muy importante que este proceso tenga un final. Para ello es fundamental que esté presente el caso de terminación, que en esta función es el caso ***n<=1***. Para este valor, la función devuelve valor sin llamarse a sí misma. Observa que el valor de retorno de la función factorial es ***double***, pues esta función crece muy rápidamente y por encima de 20 ya no se puede representar con variables enteras, ni siquiera ***long***.

EJERCICIO 5.13: OTRA FUNCIÓN RECURSIVA: LA SUMA DE LOS *N* PRIMEROS NÚMEROS ENTEROS

Seguro que con la experiencia adquirida en el ejercicio anterior no tienes ninguna dificultad en desarrollar tu propia función recursiva para hallar la suma de los *n* primeros números enteros impares. Llama ***sum2.c*** al fichero correspondiente. En este caso, la función se puede basar en la fórmula recursiva siguiente:

$$suma(n) = (2*n-1) + suma(n-1)$$

Solución comentada al Ejercicio 5.13.

```
/* fichero sum2.c */
#include <stdio.h>

void main(void) {
    int n;
    double suma(int);
    printf("Teclea un numero entero positivo: \n");
    scanf("%d", &n);
    printf("La suma de los %d primeros impares es %lf\n", n, suma(n));
}

double suma(int n) {
    if(n<=1)
        return 1.0;
    return ((2*n-1)+suma(n-1));
}
```

Comentario: Sólo reseñar que no se trata de la suma de los impares hasta el número ***n*** que se teclea, sino que es la suma de los ***n*** primeros números impares, es decir, si por ejemplo tecleamos 3, el resultado será 9 que es la suma de 1 más 3 más 5.

PRÁCTICA 6.

EJERCICIO 6.1: NORMA SUB-INFINITO DE UNA MATRIZ.

El módulo de un vector es una forma de indicar su tamaño mediante una único número. Existe algo parecido para las matrices y es la ***norma*** de una matriz. Se pueden encontrar distintas definiciones para la norma de una matriz. Una de ellas, particularmente sencilla, es la llamada ***norma sub-infinito***, que es el máximo valor de la suma de los valores absolutos de cada fila de la matriz. Dicho de otra forma: se calcula la suma de los valores absolutos de los elementos de cada fila, y se elige el valor más grande de entre todos los obtenidos.

$$\|\mathbf{A}\|_\infty = \max_i \left(\sum_j |a_{ij}| \right)$$

Crea una función llamada ***normaInf()*** que reciba como argumentos los números de filas y de columnas de la matriz, así como la propia matriz, que será de tipo ***double***. Como valor de retorno se devolverá la norma sub-infinito de la matriz. Haz un programa principal que lea los datos y llame a la función ***normaInf()***. Guarda todo en un fichero llamado ***norma.c***.

Solución comentada del Ejercicio 6.1.

La solución de este ejercicio se va a presentar conjuntamente con la del Ejercicio 6.2.

EJERCICIO 6.2: FUNCIÓN PARA CREAR UNA MATRIZ CON RESERVA DINÁMICA DE MEMORIA.

Es muy frecuente el tener que utilizar matrices con ***reserva dinámica de memoria***. Por eso puede ser útil que crees tu propia función para realizar esta tarea. Llámala ***maAlloc()***. Sus argumentos serán los números de filas y de columnas de la matriz (que podrán ser diferentes, en el caso de las matrices rectangulares). El valor de retorno será un puntero al primer elemento del vector de punteros de la matriz. La matriz será de tipo *double*. Guarda esta función junto con una función ***main()*** que te permita probarla en un fichero llamado ***maAlloc.c***.

Solución comentada de los Ejercicios 6.1 y 6.2.

A continuación se presenta un programa principal que llama a las funciones ***maAlloc()*** y ***normaInf()***. La primera de ellas reserva dinámicamente memoria para una matriz *double* de ***m*** filas y ***n*** columnas. La segunda calcula la norma sub-infinito de dicha matriz. En aras de la brevedad del programa, se han suprimido los ***printf()*** que piden los datos al usuario.

```
/* fichero norma.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void main(void) {       /* funcion o programa principal */
    int i, j, m, n;
    double **a;
    double **maAlloc(int m, int n);
    double normaInf(int m, int n, double **a);
    printf("Teclea m y n: ");
    scanf("%d%d", &m, &n);
    a=maAlloc(m, n);
    printf("\nTeclea los elementos de la matriz por filas\n");
```

```
    for(i=0; i<m; i++)
        for(j=0; j<n; j++)
            scanf(" %lf", &a[i][j]);
    printf("\nLa norma sub-infinito es: %lf\n", normaInf(m, n, a));
}

double **maAlloc(int m, int n) {
    double **matrix;
    int i;
    matrix=calloc(m, sizeof(double *));
    matrix[0]=calloc(m*n, sizeof(double));
    for (i=1; i<m; i++)
        matrix[i]=matrix[i-1] + n;
    return matrix;
}

double normaInf(int m, int n, double **a) {
    double norma=0.0, suma;
    int i, j;
    for (i=0; i<m; i++) {
        suma=0.0;
        for (j=0; j<n; j++)
            suma+=fabs(a[i][j]);
        if (norma<suma)
            norma=suma;
    }
    return norma;
}
```

Comentario: En la función ***main()*** no se reserva espacio para toda la matriz, sino sólo para el nombre de la matriz ***a***, que como es sabido es un *puntero a puntero a double*. La función de reserva dinámica de memoria ***maAlloc()*** devuelve también un *puntero a puntero a double*, valor de retorno que será asignado al nombre de la matriz cuando esta función sea llamada. En este ejemplo, la matriz puede ser rectangular, con ***m*** filas y ***n*** columnas.

Es interesante observar cómo se declara y cómo se llama la función ***normaInf()***. Su valor de retorno es de tipo *double*, pues va a devolver el valor de la norma. En la declaración no es necesario poner los nombres de las variables que constituyen los argumentos (basta poner los tipos; el compilador ignora los nombres). En concreto, hay que poner el tipo del nombre de la matriz, que es (*double* **); en la llamada, basta pasar el nombre de la matriz (***a***).

La función ***maAlloc()*** reserva memoria para una matriz de tipo *double*, de nombre ***matrix***, con ***m*** filas y ***n*** columnas. Lo primero que se hace es declarar el nombre de la matriz como *puntero a puntero a double*. Después se reserva memoria para los ***m*** elementos del vector de *punteros a double*. El valor de retorno se asigna al nombre de la matriz. Después, con un bucle ***for*** se podría reservar memoria para cada una de las filas de la matriz. El espacio para los ***n*** elementos de cada fila se reservaría con una nueva llamada a ***calloc()***, por lo que no habría garantía de que las filas se guarden de modo contiguo (una a continuación de la otra).

La versión presentada de la función ***maAlloc()*** garantiza que las filas de la matriz están contiguas en la memoria del ordenador. Para ello, se ha reservado memoria para toda la matriz en una sola llamada a ***calloc()***. El valor de retorno se asigna al primer elemento del vector de punteros, que debe contener la dirección del primer elemento de la primera fila. Después, se asigna a cada elemento restante del vector de punteros la dirección del primer elemento de la fila correspondiente, que está ***n*** lugares más atrás que el primer elemento de la fila anterior. Esta operación se realiza teniendo en cuenta las propiedades de la aritmética de punteros. Finalmente, mediante el valor de retorno se devuelve la dirección del primero elemento del vector de punteros.

La función ***normaInf()*** calcula el máximo de la suma de los valores absolutos de los elementos de cada fila. Se podría crear un vector que contuviera dichas sumas de valores absolutos de cada fila, pero no es necesario si lo único que se quiere es hallar la suma máxima: se calcula la suma de los valores absolutos de cada fila y se compara con ***norma***, que contiene la suma máxima encontrada hasta ese momento. Si ***norma*** es más pequeña que la última suma se actualiza ***norma*** y se sigue la comparación con la suma correspondiente a la fila siguiente. Al final se devuelve ***norma*** como valor de retorno.

EJERCICIO 6.3: RESOLUCIÓN DE UN SISTEMA DE ECUACIONES LINEALES.

Este ejercicio te lo vamos a poner más fácil. Crea o copia del disco del servidor el fichero ***resolve.c*** que contiene un programa para resolver sistemas de hasta 10 ecuaciones lineales. El programa está basado en el uso de varias funciones distintas para leer los datos, hacer la triangularización, hacer la vuelta atrás, imprimir los resultados e incluso calcular el valor absoluto de una variable *double*. Fíjate muy bien en cómo son los ***argumentos formales*** (definición y declaración de la función) y los ***argumentos actuales*** (llamada a la función), en el caso de las ***matrices*** y los ***vectores***.

Podrás observar que la función ***triang()*** se asegura de que los pivots sean diferentes de cero. Comrueba con dos o tres ejemplos sencillos de pequeño tamaño.

Solución comentada del Ejercicio 6.3.

```
/* fichero resolve.c */
#include <stdio.h>
#include <stdlib.h>
#define   N     10
#define   EPS   1.e-12

void main(void) {
     int    n, vRet;
     double a[N][N], x[N], b[N];
     void   lect(int *n, double a[][N], double b[]);
     int    triang(int n, double (*a)[N], double *b);
     void   sust(int n, double a[][N], double b[], double x[]);
     void   escribir(int n, double x[]);

     /* Lectura de datos */
     lect(&n, a, b);

     /* Triangularizacion de la matriz */
     vRet=triang(n, a, b);
     if (vRet!=1) {
          printf("Ha aparecido un pivot nulo o muy pequeño. Agur! \n");
          exit(1);
     }
     /* Vuelta atras */
     sust(n, a, b, x);

     /* Escritura de resultados */
     escribir(n, x);
}

/* funcion para leer los elementos de una matriz */
void lect(int *n, double mat[][N], double b[])  {
     int i, j;
     printf("Numero de ecuaciones: ");
     scanf("%d", n);
     printf("\nMatriz del sistema:\n");
     for (i=0; i<*n; i++)
          for (j=0; j<*n; j++) {
               printf("a(%d, %d): ", i+1, j+1);
               scanf("%lf", &mat[i][j]);
          }
     printf("\n\nTermino independiente:\n");
     for (i=0; i<*n; i++) {
          printf("b(%d): ", i+1);
          scanf("%lf", &b[i]);
     }
     return;
}

/* funcion para triangularizar una matriz */
int triang(int nec, double a[][N], double b[]) {
    int i, j, k, error;
    double fac;
    double va(double);
    for (k=0; k<nec-1; k++)
        for (i=k+1; i<nec; i++) {
            if (va(a[k][k])<EPS)
                return error=-1;
```

```
            fac=-a[i][k]/a[k][k];
            for (j=k; j<nec; j++)
                a[i][j]+=a[k][j]*fac;
            b[i]+=b[k]*fac;
        }
    return error=1;
}

/* funcion para resolver un sistema triangular superior */
void sust(int n, double a[][N], double b[], double sol[]) {
    int i, k;
    double sum;
    for (k=n-1; k>=0; k--) {
        sum=0.0;
        for (i=k+1; i<n; i++)
            sum+=a[k][i]*sol[i];
        sol[k]=(b[k]-sum)/a[k][k];
    }
}

/* funcion para escribir los resultados */
void escribir(int n, double solucion[]) {
    int i;
    printf("\nEl vector solucion es: \n");
    for (i=0; i<n; i++)
         printf("solucion(%d)= %lf\n", i, solucion[i]);
}

/* valor absoluto de una variable */
double va(double u) {
    if (u<0.0)
        return -u;
    else
        return u;
}
```

Comentario: En este caso se ha reservado memoria para la matriz ***a*** y para los vectores ***b*** y ***x*** de modo convencional, con un tamaño fijo de N=10. Es interesante observar cómo se realiza en este caso la declaración de las funciones que tienen vectores y matrices como argumentos formales. Para poder utilizar adecuadamente la fórmula de direccionamiento dentro de la función es necesario pasarle la información del número de columnas de la matriz (número de elementos que tiene cada fila); no es necesario pasar el número de filas porque no interviene en la fórmula citada. Por ejemplo, la función ***lect()*** se declara en la forma:

```
void lect(int *n, double a[][N], double b[]);
```

donde el número de ecuaciones ***n*** se pasa por referencia pues es un valor que la función debe leer y devolver al programa principal. Obsérvese que el primer par de corchetes de la matriz ***a*** está vacío, lo mismo que los del vector ***b***, pues dicha información es innecesaria para el direccionamiento. La función ***triang()*** se ha declarado de una forma distinta, igualmente correcta.

```
int  triang(int nec, double (*a)[N], double *b);
```

En este caso es equivalente declarar ***b*** como un vector *double* o como un puntero a *double*. La variable **a** puede ser declarada como una matriz con N elementos por fila o como un puntero a un vector de N elementos (el paréntesis es necesario para que no se declare como un vector de N punteros a *double*). En este ejemplo se han presentado pues las dos formas posibles de introducir vectores y matrices como argumentos en las declaraciones de las funciones

Observa que el número de ecuaciones ***n*** se ha pasado a la función ***lect()*** por referencia. Esto quiere decir que dentro de esta función ***n*** es la dirección del número de ecuaciones. Cuando se quiere hacer uso de dicho número, ***n*** debe ir precedido por el operador *indirección*, en la forma (****n***). Por otra parte, para pasar a ***scanf()*** la dirección del número de ecuaciones, ya no hace falta preceder ***n*** por el operador ***&***.

EJERCICIO 6.4: MODIFICA EL PROGRAMA ANTERIOR.

Modifica el programa anterior para que trabaje con reserva dinámica de memoria. Podrías utilizar la función que has programado en el ***Ejercicio 6.2***. Compruébala con los mismos ejemplos que antes y asegúrate de que obtienes el mismo resultado. Guarda el resultado de este ejercicio en un fichero llamado ***resolve2.c***.

Solución comentada del Ejercicio 6.4.

La solución de este ejercicio se presentará conjuntamente con la del Ejercicio 5.

EJERCICIO 6.5: RESOLUCIÓN DE SISTEMAS DE ECUACIONES LINEALES CON PIVOTAMIENTO POR COLUMNAS.

En este ejercicio te pedimos que vuelvas a modificar el programa anterior. Como habrás podido comprobar, los programas anteriores fallan si se encuentra un pivot nulo o muy pequeño. Ya has estudiado en *Álgebra Lineal* una forma de solventar esta dificultad: hacer pivotamiento. También sabes que si se permutan dos ecuaciones de un sistema, la solución no varía.

El pivotamiento por columnas consiste en buscar el máximo elemento en valor absoluto de cada columna y utilizarlo como pivot. Al hacer esta búsqueda, no se deben considerar los elementos de esa columna situados en filas en las que han aparecido los pivots anteriores (así pues, excepto en el caso de la primera columna en que no hay pivots anteriores, no todos los elementos de la columna pueden ser pivots).

Te sugerimos dos formas de hacer este ejercicio. La primera de ellas consiste en guardar la posición (la fila) en que ha aparecido cada pivot. La segunda, que aprovecha mejor las características del lenguaje C, consiste en permutar filas de forma que el pivot siempre esté sobre la diagonal; de esta forma puedes aprovechar mejor el código del ***Ejercicio 6.3***. Recuerda que para permutar las filas de una matriz basta con permutar el valor de los punteros que apuntan al primer elemento de cada fila. Recuerda también que para permutar ecuaciones correctamente debes permutar también los elementos del vector de términos independientes.

Te sugerimos que construyas una función llamada ***pivot()*** que calcule el máximo elemento en valor absoluto de una columna de la matriz, teniendo en cuenta los elementos de las filas donde han aparecido pivots en las columnas anteriores. El valor de retorno será la posición del pivot en la columna.

Guarda este ejercicio en un fichero llamado ***resolve3.c***.

Solución comentada de los Ejercicios 6.4 y 6.5.

```
/* fichero resolve3.c */

#include  <stdio.h>
#include  <stdlib.h>
#include  <math.h>
#define   EPS   1.e-12
void main(void) {
    int    i, j, n, vRet;
    double **a, *x, *b;
    double **crearMatriz(int m, int n);
    int    triang(int n, double **a, double *b);
    int    pivot(int n, int k, double **a);
    void   sust(int n, double **a, double *b, double *x);
    void   escribir(int n, double *x);

    /* Lectura de datos y creaci¢n dinamica de matrices y vectores */
    printf("Numero de ecuaciones: ");
    scanf("%d", &n);
    a=crearMatriz(n, n);
    printf("\nMatriz del sistema:\n");
```

```
    for (i=0; i<n; i++)
        for(j=0; j<n; j++) {
            printf("a(%d, %d): ", i+1, j+1);
            scanf(" %lf", &a[i][j]);
        }
    b=calloc(n, sizeof(double));
    x=calloc(n, sizeof(double));
    printf("\nTermino independiente:\n");
    for (i=0; i<n; i++) {
        printf("b(%d): ", i+1);
        scanf(" %lf", &b[i]);
    }
    /* Triangularizacion de la matriz */
    vRet=triang(n, a, b);
    if (vRet!=1) {
        printf("Ha aparecido un pivot nulo o muy pequeño. Agur! \n");
        exit(1);
    }

    /* Vuelta atras */
    sust(n, a, b, x);

    /* Escritura de resultados */
    escribir(n, x);
}   /* Fin de main() */

/* Funcion para hallar el pivot de la columna k */
int  pivot(int n, int k, double **a) {
    int i, imaximo;
    double maximo=0.0;
    double va(double);
    for (i=k; i<n; i++) {
        if (maximo<va(a[i][k])) {
            maximo=va(a[i][k]);
            imaximo=i;
        }
    }
    return imaximo;
}

/* Funcion para realizar la triangularizacion de una matriz */
int triang(int nec, double **a, double *b)  {
    int i, j, k, ipivot, error;
    double fac, *temp;
    double va( double );
    for (k=0; k<nec-1; k++) {
        ipivot=pivot(nec, k, a);
        /* intercambio de los punteros a las filas k e imaximo */
        temp=a[k];
        a[k]=a[ipivot];
        a[ipivot]=temp;
        fac=b[k];
        b[k]=b[ipivot];
        b[ipivot]=fac;
        for (i=k+1; i<nec; i++) {
            if (va(a[k][k]) < EPS)
                return error=-1;
            fac=-a[i][k]/a[k][k];
            for (j=k; j<nec; j++)
                a[i][j]+=a[k][j]*fac;
            b[i]+=b[k]*fac;
        }
    }
    return error=1;
}
```

```
void sust(int n, double **a, double *b, double *sol) {
    int i, k;
    double sum;
    for (k=n-1; k>=0; k--) {
        sum=0.0;
        for (i=k+1; i<n; i++)
            sum+=a[k][i]*sol[i];
        sol[k]=(b[k]-sum)/a[k][k];
    }
}

double **crearMatriz(int m, int n) {
    double **matriz, *mat;
    int i;
    matriz=calloc(m, sizeof(double *));
    matriz[0]=mat=calloc(m*n, sizeof(double));
    for(i=1; i<m; i++)
        matriz[i]=mat+n*i;
    return matriz;
}

void escribir(int n, double *solucion) {
    int i;
    printf("\nEl vector solucion es: \n");
    for (i=0; i<n; i++)
        printf("solucion(%d)= %lf\n", i, solucion[i]);
}

double va(double u) {
    if (u<0.0)
        return -u;
    else
        return u;
}
```

EJERCICIO 6.6: CÁLCULO DEL DETERMINANTE DE UNA MATRIZ *NXN*.

Como sabes, el determinante de una matriz no cambia si se sustituye una ecuación por esa misma ecuación a la que se suma otra de las ecuaciones multiplicada por un factor cualquiera. Esto quiere decir que la triangularización de Gauss no cambia el valor del determinante. Ya sabes también que en una matriz triangular el determinante es el producto de los elementos de la diagonal.

Por otra parte, si se permutan dos filas de una matriz, el determinanate cambia de signo. Si quieres calcular el determinanate de una matriz mediante la triangularización de Gauss con pivotamiento, tendrás que llevar cuenta de las permutaciones de filas que has realizado.

Haz un programa principal llamado ***determin.c*** que lea una matriz cuadrada de cualquier tamaño y llame a la función ***determinante()*** que devuelve el valor de su determinante. El programa principal deberá ofrecer al usuario dos opciones para introducir los datos: bien por teclado, bien leyéndolos de un fichero, en cuyo caso se pedirá el nombre del fichero que contiene la matriz.

Solución comentada del Ejercicio 6.6.

```
/* fichero determin.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define   EPS   1.e-12
```

```
void main(void) {
    int    i, j, n, vRet, signo=1;
    double **a, det;
    double **crearMatriz(int m, int n);
    int    triang2(int n, double **a, int *signo);
    int    pivot(int n, int k, double **a);

    /* Lectura de datos y creacion dinamica de matrices y vectores */
    printf("Numero de filas y de columnas: ");
    scanf("%d", &n);
    a=crearMatriz(n, n);
    printf("\nMatriz del sistema:\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            printf("a(%d, %d): ", i+1, j+1);
            scanf("%lf", &a[i][j]);
        }
    /* Triangularizacion de la matriz */
    vRet=triang2(n, a, &signo);
    if (vRet!=1) {
        printf("El determinante es cero o muy pequeño. Agur! \n");
        exit(1);
    }
    /* Calculo del determinante */
    det=1.0;
    for (i=0; i<n; i++)
         det*=a[i][i];
    det*=signo;
    printf("\n\nEl determinante de la matriz es %lf\n", det);
}

/* Funcion para hallar el pivot de la columna k */
int  pivot(int n, int k, double **a) {
    int    i, imaximo;
    double maximo=0.0;
    double va(double);
    for (i=k; i<n; i++) {
        if (maximo<va(a[i][k])) {
            maximo=va(a[i][k]);
            imaximo=i;
        }
    }
    return imaximo;
}

/* Funcion para realizar la triangularizacion de una matriz */
int triang2(int nec, double **a, int *signo) {
    int i, j, k, ipivot, error;
    double fac, *temp;
    double va(double);
    for (k=0; k<nec-1; k++) {
        ipivot=pivot(nec, k, a);
        /* intercambio de los punteros a las filas k e imaximo */
        if (ipivot!=k) {
            temp=a[k];
            a[k]=a[ipivot];
            a[ipivot]=temp;
            *signo=-*signo;
        }
        for (i=k+1; i<nec; i++) {
            if (va(a[k][k])<EPS)
                return error=-1;
            fac=-a[i][k]/a[k][k];
            for (j=k; j<nec; j++)
                a[i][j]+=a[k][j]*fac;
```

```
        }
     }
     return error=1;
}

double **crearMatriz(int m, int n) {
    double **matriz, *mat;
    int i;
    matriz=malloc(m*sizeof(double *));
    matriz[0]=mat=malloc(m*n*sizeof(double));
    for (i=1; i<m; i++)
        matriz[i]=mat+n*i;
    return matriz;
}

double va(double u) {
    if (u<0.0)
        return -u;
    else
        return u;
}
```

EJERCICIO 6.7: RESOLUCIÓN DE SISTEMAS DE ECUACIONES LINEALES POR EL MÉTODO ITERATIVO DE GAUSS-SEIDEL.

El método de eliminación Gauss no es el único método para resolver sistemas de ecuaciones lineales (aunque sí el más eficiente en la mayor parte de los casos). El mismo Gauss inventó otro método para resolver sistemas de ecuaciones lineales de modo iterativo, al parecer con un amigo llamado Seidel del que no tenemos muchos más datos.

El método de Gauss-Seidel procede del siguiente modo: consideremos un sistema de tres ecuaciones con tres incógnitas:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

Si suponemos que los elementos de la diagonal son mucho mayores que los demás elementos de cada fila, una primera aproximación de la solución podría ser la siguiente:

$$x_1^{(1)} = b_1 / a_{11}$$
$$x_2^{(1)} = b_2 / a_{22}$$
$$x_3^{(1)} = b_3 / a_{33}$$

donde con el superíandice (1) indicamos que se trata de la primera aproximación.

La segunda aproximación se podría calcular a partir de la primera utilizando ya el sistema original. Para ello haríamos:

$$x_1^{(2)} = (b_1 - a_{12}x_2^{(1)} - a_{13}x_3^{(1)}) / a_{11}$$
$$x_2^{(2)} = (b_2 - a_{21}x_1^{(2)} - a_{23}x_3^{(1)}) / a_{22}$$
$$x_3^{(2)} = (b_3 - a_{31}x_1^{(2)} - a_{32}x_2^{(2)}) / a_{33}$$

Observa que $x_1^{(2)}$ (segunda aproximación de x_1) se obtiene a partir de la primera aproximación de x_2 y x_3; $x_2^{(2)}$ se obtiene a partir de la segunda aproximación de x_1 (que ya está disponible) y de la primera aproximación de x_3 (pues aún no se dispone de la segunda); la segunda aprox.-mación de x_3 se obtiene a partir de la segunda aproximación de x_1 y x_2, que ya están disponibles.

En general:

$$x_1^{(i+1)} = (b_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)}) / a_{11}$$
$$x_2^{(i+1)} = (b_2 - a_{21}x_1^{(i+1)} - a_{23}x_3^{(i)}) / a_{22}$$
$$x_3^{(i+1)} = (b_3 - a_{31}x_1^{(i+1)} - a_{32}x_2^{(i+1)}) / a_{33}$$

Generaliza estas expresiones para el caso de sistemas de *n* ecuaciones con *n* incógnitas y haz un programa que realice estas iteraciones hasta que el error relativo entre dos iteraciones consecutivas sea menor que una constante EPS, que harás igual a 10^{-8}. El error relativo lo tendrás que medir del siguiente modo:

$$\text{abs}\left(\frac{\text{módulo}\left\{\mathbf{x}^{(i+1)}\right\} - \text{módulo}\left\{\mathbf{x}^{(i)}\right\}}{\text{módulo}\left\{\mathbf{x}^{(i+1)}\right\}}\right) < \text{EPS}$$

Guarda el programa en un fichero llamado ***gseidel.c*** y pruébalo con el siguiente sistema de ecuaciones:

$$15x_1 + 3x_2 - 6x_3 = 3$$
$$-2x_1 + 9x_2 - 3x_3 = 7$$
$$5x_1 + 2x_2 + 10x_3 = 39$$

Comprueba que obtienes el mismo resultado que con el método que inventó Gauss solo (sin el subsodicho Seidel…).

Solución comentada del Ejercicio 6.7.

```
/* fichero gseidel.c */
#include <stdio.h>
#include <math.h>
#define  EPS   10.e-12

void main(void) {
    double a[10][10], x[10], b[10], xn;
    double error, modulo, suma;
    int n, i, j, nit=0;

    printf("\nNumero de ecuaciones: ");
    scanf("%d", &n);

    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            printf("Elemento a(%d,%d): ", i+1, j+1);
            scanf(" %lf", &a[i][j]);
        }
    }
    printf("\n");
    for (i=0; i<n; i++) {
        printf("Elemento b(%d): ", i+1);
        scanf(" %lf", &b[i]);
    }

    /* valores para iteración inicial */
    for (i=0; i<n; i++)
        x[i] = b[i]/a[i][i];

    do {
        error=0.0;
        modulo=0.0;
        nit++;
```

```
        for(i=0; i<n; i++) {
            suma=0.0;
            for (j=0; j<n; j++)
                if (j!=i)
                    suma+=a[i][j]*x[j];
            xn=(b[i]-suma)/a[i][i];
            error+=(xn-x[i])*(xn-x[i]);
            x[i]=xn;
            modulo+=xn*xn;
        }
    } while (fabs(error/modulo)>EPS);

    printf("\nNumero de iteraciones: %d", nit);
    printf("\nEl vector solucion es:");
    for (i=0; i<n; i++)
        printf("\nx(%d) = %lf", i+1, x[i]);
    printf("\n");
}
```

Comentario: Se ha utilizado la función ***fabs()*** para calcular el valor absoluto de una variable ***double***.

EJERCICIO 6.8: DIRECCIONA UN VECTOR DE 1 A *N* (Y NO DE 0 A *N-1*).

El lenguaje C tiene algunas características un poco incómodas −como por ejemplo numerar los *n* elementos de un vector de 0 a *n*-1− pero es casi siempre lo suficientemente versátil como para permitir superar esos inconvenientes. En este caso, una forma de hacerlo podría ser reservar espacio para *n*+1 elementos (del 0 al *n*) y no utilizar el primero. Aparte de que esta solución no optimiza el uso de la memoria, hay soluciones más elegantes que se basan en la ***aritmética de los punteros***: Si disminuimos en una unidad el valor del puntero que apunta al primer elemento del vector (es decir, el nombre del vector, que supondremos es ***vector***), cuando utilicemos ***vector[1]*** estamos direccionando igual que si hiciésemos ****(vector+1)***, con lo cual apuntamos al primer elemento del vector. La siguiente función indica cómo se crea un vector de estas características. Crea una programa ***main()*** que reserve espacio para dos vectores llamando a ***crearVector()*** y halle su producto escalar. Guarda el programa resultante en un fichero llamado ***vector1n.c***.

Solución comentada del Ejercicio 6.8.

```
/* fichero vector1n.c */
#include <stdio.h>
#include <stdlib.h>

void main(void) {
    double *crearVector(int n);
    double *v, *w;
    double pe=0.0;
    int i;
    v=crearVector(3);
    w=crearVector(3);
    v[1]=1;  v[2]=2; v[3]=3;
    w[1]=-1; w[2]=2; w[3]=-3;
    for (i=1; i<=3; i++)
        pe+=v[i]*w[i];
    printf("El producto escalar es: %lf\n", pe);
}

double *crearVector(int n) {
    double *vector;
    vector=calloc(n, sizeof(double));
    return --vector;
}
```

EJERCICIO 6.9: DIRECCIONA UNA MATRIZ DE 1 A *N*, Y DE 1 A *M* (Y NO DE 0 A *N-1*, Y DE 0 A *M-1*).

También pueden crearse matrices de forma que sus elementos se direccionen a partir de 1 y no a partir de 0. La función ***crearMatriz()*** lo hace de esta forma. La función ***destruirMatriz()*** libera el espacio de memoria correspondiente cuando ya no es necesario. Crea un fichero llamado ***mat1n1m.c*** que contenga estas funciones y un programa ***main()*** que las utilice.

Solución comentada del Ejercicio 6.9.

```
/* fichero mat1n1m.c */
#include <stdio.h>
#include <stdlib.h>

double **crearMatriz(int m, int n) {
    int    i;
    double **matriz;
    matriz=calloc(m, sizeof(double *));
    --matriz;
    for (i=1; i<=m; i++) {
        matriz[i]=calloc(n, sizeof(double));
        --matriz[i];
    }
    return matriz;
}
```

Comentario: La idea es decrementar el puntero que es el nombre de la matriz, de forma que quede apuntando al valor anterior al primer elemento del vector de punteros. Para acceder a este primer elemento habrá que utilizar el subíndice 1. De la misma forma hay que decrementar (asimismo según la aritmética de los punteros) los elementos del vector de punteros que apuntan a los primeros elementos de cada fila de la matriz. Para acceder a dichos elementos habrá que utilizar el subíndice incrementado en una unidad.

```
void destruirMatriz(double **matriz, int m) {
    int i;
    for (i=1; i<=m; i++)
        free(++matriz[i]);
    free(++matriz);
}
```

Comentario: De forma análoga, hay que incrementar los punteros antes decrementados para liberar correctamente la zona de memoria correspondiente.

```
void main(void) {
    double **a;
    int    n, i, j;

    printf("\nTeclea el valor de n: ");
    scanf("%d", &n);
    a=crearMatriz(n, n);
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++) {
            printf("Teclea a(%d, %d): ", i, j);
            scanf(" %lf", &a[i][j]);
        }
    printf("\n");
    for (i=1; i<=n; i++) {
        for (j=1; j<=n; j++){
            printf("   a(%d, %d) = ", i, j);
            printf("%lf ", a[i][j]);
        }
        printf("\n");
    }
    destruirMatriz(a, n);
}
```

Comentario: La función ***main()*** simplemente lee la matriz y la vuelve a escribir, para comprobar que el direccionamiento comenzando por 1 funciona correctamente..

PRÁCTICA 7.

Se presentan resueltos a continuación los 9 ejercicios –3 por cada grupo– que constituyeron la Práctica nº 7 (práctica evaluada). Algunas de las soluciones presentadas son más complicadas de lo estrictamente necesario y de lo que se exigía en dicha práctica. Se hace así con objeto de presentar diversas técnicas de programación que puedan ser útiles en la realización de otros ejercicios de dificultad similar o superior.

EJERCICIO 7.1: CONTAR LAS VOCALES ACENTUADAS DE UN FICHERO.

Este ejercicio consta de dos partes[1]:

1.- Deberás realizar un programa que te permita conocer el número según el código ASCII de las vocales minúsculas acentuadas (á, é, í, ó, ú), tanto en MS-DOS como en Windows. Guarda este programa con el nombre de ***acentos.c***

2.- Después, deberás realizar un programa que lea un fichero con vocales acentuadas llamado ***acentos.d***, indique el número de cada una de ellas e imprima el contenido con la codificación original y con la adecuada para MS-DOS. Guarda este programa como ***cacentos.c***

Solución comentada del Ejercicio 7.1.

```
/* fichero acentos.c */
/* Código ASCII de las vocales minúsculas acentuadas */

#include <stdio.h>

void main(void) {
    int i=0, j=0;
    printf("Código ASCII para Windows de las vocales minúsculas acentuadas");
    printf("\nLa a acentuada en MSW (%c) es el: %d.", 'á', 'á');
    printf("\nLa e acentuada en MSW (%c) es el: %d.", 'é', 'é');
    printf("\nLa i acentuada en MSW (%c) es el: %d.", 'í', 'í');
    printf("\nLa o acentuada en MSW (%c) es el: %d.", 'ó', 'ó');
    printf("\nLa u acentuada en MSW (%c) es el: %d.", 'ú', 'ú');

    printf("\n\nTabla de numeros ASCII de los caracteres MS-DOS");
    for (i=0; i<=255; i++) {
        if (i<=127)
            j=i;
        else
            j=i-256;
        printf("\n %4d, %4d: %c ", i, j, i);
    }
}
```

Comentario: Para conocer el número ASCII asociado con un carácter cualquiera basta escribir dicho carácter con formato %d, propio de los números enteros. Si por el contrario se utiliza el formato %c, propio de los caracteres, se imprime el carácter correspondiente. En la primera parte del ejemplo anterior cada carácter acentuado se escribe con los dos formatos. Puede comprobarse que al ejecutar el programa en MS-DOS los caracteres acentuados no se scriben correctamente.

En la segunda parte del programa anterior se escriben los 256 caracteres del código ASCII ampliado, lo que proporciona una tabla completa de los caracteres imprimibles en MS-DOS.

---

[1] Este Ejercicio se ha modificado como consecuencia del cambio que ha sufrido el entorno de programación C/C++ de Microsoft. Cuando se redactó originalmente las aplicaciones tipo "console" se ejecutaban en una ventana especial que mantenía correctamente los caracteres ASCII extendidos (por ejemplo, las vocales acentuadas). En las últimas versiones, Visual Studio ejecuta estas aplicaciones en una ventana MS-DOS que utiliza una codifiocación distinta para los caracteres especiales. Este Ejercicio pone de manifiesto estas diferencias.

```
/* fichero cacentos.c */
/* Número de vocales acentuadas en un determinado fichero */
#include <stdio.h>
#include <stdlib.h>

void main(void) {
    char ch;
    char txt[200];
    int i=0;
    int nas=0, nes=0, nis=0, nos=0, nus=0;
    FILE* fi;
    fi=fopen("acentos.d", "r");

    while ((txt[i]=getc(fi))!=EOF) {
        ch=txt[i];
        printf("%c", ch);
        switch (ch) {
            case 'á':
                nas++;
                txt[i]='\240';
                break;
            case 'é':
                nes++;
                txt[i]='\202';
                break;
            case 'í':
                nis++;
                txt[i]='\241';
                break;
            case 'ó':
                nos++;
                txt[i]='\242';
                break;
            case 'ú':
                nus++;
                txt[i]='\243';
        }
        i++;
    } /* fin del bucle while */
    txt[i]='\0';
    printf("\nEl numero de '\240' es: %d", nas);
    printf("\nEl numero de '\202' es: %d", nes);
    printf("\nEl numero de '\241' es: %d", nis);
    printf("\nEl numero de '\242' es: %d", nos);
    printf("\nEl numero de '\243' es: %d", nus);
    printf("\n\nEl texto transformado para MS-DOS es:\n,\n%s\n", txt);
    fclose(fi);
}
```

Comentario: Éste es un ejemplo típico de uso de la sentencia ***switch/case***. Cada vez que se encuentra una vocal acentuada se incrementa el contador correspondiente. Es muy importante la presencia de los ***break***, pues si no se incrementarían también todos los demás contadores que aparecen hasta el final del ***switch***. La clave de este programa es la detección de las vocales acentuadas y su sustitución por el código ASCII que da el resultado correcto en MS-DOS, de acuerdo con la numeracón obtenida en el Ejercicio anterior. Es importante recordar que los códigos ASCII se deben expresar en base ocho (números octales).

EJERCICIO 7.2: EVALUACIÓN DE UNA FORMA CUADRÁTICA.

Este ejercicio consiste en evaluar una forma cuadrática del tipo:

$$f = \mathbf{x}^T \mathbf{A} \mathbf{x}$$

Para ello debes crear una función a la que pases por referencia el vector **x**, su tamaño y la matriz **A** y que devuelva *f*, que es un escalar. Para crear las distintas variables deberás usar reserva dinámica de memoria.

El tamaño del vector **x** debe ser una variable que introduzca el usuario durante la ejecución del programa. Por otra parte se considera que el vector **x** es una matriz con una única columna. Guarda el fichero como ***formcuad.c***.

Solución comentada del Ejercicio 7.2.

```
/* fichero formcuad.c */
/* cálculo de una forma cuadrática */
#include <stdio.h>
#include <stdlib.h>

void main(void) {
    double *x, **a;
    int    n, i, j;
    double fc(int n, double **a, double *x);

    printf("Teclea el numero de filas y de columnas: ");
    scanf("%d", &n);
    a=calloc(n, sizeof(double*));
    for (i=0; i<n; i++)
        a[i]=calloc(n, sizeof(double));
    x=calloc(n, sizeof(double));
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            printf("Teclea a(%d, %d): ", i+1, j+1);
            scanf(" %lf", &a[i][j]);
        }
    printf("\n");
    for (i=0; i<n; i++) {
        printf("Teclea x(%d): ", i+1);
        scanf(" %lf", &x[i]);
    }
    printf("\nLa forma cuadratica vale: %lf\n", fc(n, a, x));
}

/* función para calcular el valor de la forma cuadrática */
double fc(int n, double **a, double *x) {
    int    i, j;
    double fac=0.0;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fac+=x[i]*a[i][j]*x[j];
    return (fac);
}
```

Comentario: Este Ejercicio se ha resuelto con reserva dinámica de memoria para la matriz y el vector con los que se calcula la forma cuadrática. La forma cuadrática se calcula por medio de una función llamada ***fc()***. Esta función utiliza directamente la fórmula del resultado de la forma cuadrática, basada en un doble sumatorio.

EJERCICIO 7.3: ENCONTRAR UN NÚMERO EN UNA LISTA E INSERTARLO SI NO ESTÁ EN ELLA.

A partir de una lista ordenada de ***n*** números, realizar un programa que detecte sin un número determinado forma parte de la lista; en caso negativo, tendrás que insertarlo en el lugar correspondiente para que la lista siga conservando el orden inicial.

La impresión final del programa mostrará la nueva lista incluyendo al nuevo número. Guarda este programa en un fichero llamado ***listord.c***.

Solución comentada del Ejercicio 7.3.

```
/* fichero listord.c */
/* Comprobar si un número está en un vector ordenado,
   y si no está, incluirlo */
```

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

void main(void) {
   int i, j, n, num, temp, vector[100];

   printf("Teclea el numero de elementos: ");
   scanf("%d", &n);

   /* lectura de los elementos del vector */
   for (i=0; i<n; i++) {
       printf("vector[%d]: ", i+1);
       scanf("%d", &vector[i]);
   }

   /* ordenación de menor a mayor */
   for (i=0; i<n-1; i++)
      for (j=i+1; j<n; j++)
         if (vector[i]>vector[j]) {
            temp=vector[i];
            vector[i]=vector[j];
            vector[j]=temp;
         }

   /* Escritura del vector ordenado */
   printf("\n\nEl vector ordenado es:\n");
   for (i=0; i<n; i++)
       printf("%6d", vector[i]);

   printf("\n\nTeclea el numero deseado: ");
   scanf("%d", &num);
   /* Comprobar si está en la lista. Búsqueda binaria */
   if (num<vector[0]) {             /* está antes del primero */
       i=0;
       j=1;
   } else if (num>vector[n-1]) {    /* está después del último */
       i=n-2;
       j=n-1;
   } else {
       i=0;
       j=n-1;
       while (j>i+1) {
           temp=(j+i)/2;
           if (num==vector[temp]) {
               printf("\n\nEl numero %d pertenece al vector, en posicion %d.",
                      num, temp+1);
               printf("\n");
               exit(0);
           }
           else if (num<vector[temp])
               j=temp;
           else
               i=temp;
       } /* fin bucle while */

       if (num==vector[i]) {
           printf("\n\nEl numero %d pertenece al vector, en posicion %d.",
               num, i+1);
           printf("\n");
           exit(0);
       } else if (num==vector[j]) {
           printf("\n\nEl numero %d pertenece al vector, en posicion %d.",
               num, j+1);
           printf("\n");
           exit(0);
       }
```

```
   }
   /* Insertar num en vector[] */
   if (num<vector[i]) {               /* se inserta delante de i */
      for (j=n-1; j>=i; j--)
         vector[j+1]=vector[j];
      vector[i]=num;
   } else if (num<vector[j]) {
      for (j=n-1; j>i; j--)           /* se inserta detrás de i */
         vector[j+1]=vector[j];
      vector[i+1]=num;
   } else                             /* se inserta detrás de j */
      vector[j+1]=num;
   n++;
   /* Escritura del nuevo vector ordenado */
   printf("\n\nEl nuevo vector ordenado es:\n");
   for (i=0; i<n; i++)
      printf("%6d", vector[i]);
   printf("\n");
} /* fin de main() */
```

Comentario: En primer lugar se lee el vector, y si no está ordenado se ordena. Después se lee el número que hay que chequear. Una posible forma de ver si este número ya pertenece al vector es irlo comparando con todos los elementos del vector, al menos hasta que se encuentre un elemento del vector diferente y mayor que el número leído (en ese momento podría pararse la búsqueda, teniendo en cuenta que los elementos del vector han sido ordenados previamente). En este programa se ha seguido un método diferente y más rápido para averiguar si el número leído pertenece al vector: la llamada ***búsqueda binaria***. Se parte de un intervalo definido por los índices ***i*** y ***j***, que iniciamente corresponden al primer (***0***) y último elementos del vector (***n-1***). A continuación se calcula el elemento central de este intervalo y se compara con el número; de esta forma sabemos en cual de los dos subintervalos ***[i, central]*** o ***[central, j]*** está el número, y –según en cual esté– actualizamos los valores de ***i*** o de ***j***. Seguimos haciendo esto sucesivas veces, con intervalos cada vez más pequeños hasta que se encuantra el número, o se encuentran dos elementos consecutivos del vector entre los que está el número. Hay que tener también en cuenta los casos particulares de que el número sea menor que el primer elemento o mayor que el último.

EJERCICIO 7.4: TRASPONER UNA MATRIZ RECTANGULAR

Realiza un programa en el cual, a partir de una matriz rectangular, por medio de una función se calcule su matriz transpuesta.

Es decir, si tenemos una matriz $m \times n$, tendrás primero que crear esta matriz con dichas dimensiones con reserva dinámica de memoria; a continuación habrá que pedir los datos de la matriz y finalmente llamar a una función que se encargará de transponer la matriz inicial a una matriz de $n \times m$.

La salida del programa mostrará las dos matrices, tanto la inicial como la transpuesta. Guarda el programa en un fichero llamado ***trasp.c***.

Solución comentada del Ejercicio 7.4.

```
/* fichero trasp.c */
/* Trasponer una matriz con reserva dinámica de memoria */
#include <stdio.h>
#include <stdlib.h>

void main(void) {
    double **a, **b;
    int  m, n, i, j;
    void *crearMatriz(int m, int n);
    void trasponer(int m, int n, double **a, double **b);

    printf("Teclea los numeros de filas y de columnas: ");
    scanf("%d%d", &m, &n);
    a=crearMatriz(m, n);
    b=crearMatriz(n, m);
```

```
    /* Lectura de la matriz */
    for (i=0; i<m; i++)
        for (j=0; j<n; j++) {
            printf("Elemento a(%d, %d): ", i+1, j+1);
            scanf(" %lf", &a[i][j]);
        }

    /* Impresión de la matriz original */
    printf("\n\nLa matriz original es: ");
    for (i=0; i<m; i++) {
        printf("\n");
        for (j=0; j<n; j++)
            printf("%10.4lf", a[i][j]);
    }

    /* se traspone la matriz leida */
    trasponer(m, n, a, b);

    /* Impresión de la matriz traspuesta */
    printf("\n\nLa matriz traspuesta es: ");
    for (i=0; i<n; i++) {
        printf("\n");
        for (j=0; j<m; j++)
            printf("%10.4lf", b[i][j]);
    }
    printf("\n");
}

/* función crearMatriz() */
void *crearMatriz(int m, int n) {
    double** a;
    int i;
    a=calloc(m, sizeof(double*));
    for (i=0; i<m; i++)
        a[i]=calloc(n, sizeof(double));
    return a;
}

/* función trasponer() */
void trasponer(int m, int n, double **a, double **b) {
    int i, j;
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            b[j][i]=a[i][j];
}
```

Comentario: Este Ejercicio se ha resuelto con reserva dinámica de memoria y con dos funciones ***crearMatriz()*** y ***trasponer()*** cuya aplicación resulta obvia. Por lo demás, el programa no tiene mucho de particular: se crean las dos matrices –la original y la traspuesta–, se leen los datos, se escribe la matriz original, se traspone y se escribe la matriz traspuesta.

EJERCICIO 7.5: INTERSECCIÓN DE UNA RECTA CON UNA CIRCUNFERENCIA.

En este programa deberás estudiar la intersección de una recta con una circunferencia, ambas contenidas en el plano XY, cuyas ecuaciones generales son:

$$y = mx + c \tag{1}$$

$$(x-a)^2 + (y-b)^2 = r^2 \tag{2}$$

Para hallar la solución basta con sustituir la ecuación (1) en la (2) y resolver la ecuación de segundo grado que resulta:

$$(1+m^2)x^2 + (2mc - 2a - 2mb)x + (a^2 + b^2 + c^2 - 2bc - r^2) = 0$$

Los tres casos posibles son:

1. Dos raíces reales distintas: La recta es secante −corta− a la circunferencia.
2. Una raíz doble. La recta es tangente a la circunferencia en un punto.
3. Dos raíces imaginarias: La recta y la circunferencia no se cortan.

En los casos 1. y 2. el programa deberá proporcionar el punto o puntos de intersección, con sus dos coordenadas.

Los datos de entrada al programa serán los coeficientes ***m***, ***c***, ***a***, ***b*** y ***r***.

Guarda el programa en un fichero llamado ***recta.c***.

Solución comentada del Ejercicio 7.5.

```
/* fichero recta.c */
#include <stdio.h>
#include <math.h>

void main(void) {
    double m, c, a, b, r;
    double a1, b1, c1;
    double discr;
    double sol1x, sol2x;
    double sol1y, sol2y;

    printf("Interseccion de una circunferencia con una recta.\n\n");
    printf("Introduce los datos de la circunferencia:\n");
    printf("Centro: coordenadas x e y: ");
    scanf("%lf%lf", &a, &b);
    printf("Radio: ");
    scanf("%lf", &r);
    printf("\nIntroduce los datos de la recta:\n");
    printf("Pendiente m y ordenada en el origen b: ");
    scanf("%lf%lf", &m, &c);

    a1=1+m*m;
    b1=2*m*c-2*a-2*m*b;
    c1=a*a+c*c+b*b-2*b*c-r*r;

    discr=b1*b1-4*a1*c1;
    if (discr>0.0) {
        printf("\nExisten dos soluciones:\n");
        sol1x=(-b1+sqrt(discr))/(2*a1);
        sol2x=(-b1-sqrt(discr))/(2*a1);
        sol1y=m*sol1x+c;
        sol2y=m*sol2x+c;

        printf("primer punto:\n%lf %lf\n", sol1x, sol1y);
        printf("segundo punto:\n%lf %lf\n", sol2x, sol2y);
    } else if (discr<0.0)
        printf("\n\nNo hay solucion.\n");
    else {
        printf("\n\nSolo hay una solucion:\n");
        sol1x=-b1/(2*a1);
        sol1y=m*sol1x+c;
        printf("%lf %lf\n", sol1x, sol1y);
    }
}
```

Comentario: Este Ejercicio es muy sencillo. Lo único que hay que hacer es formar la ecuación de segundo grado a partir de los datos leídos y resolverla, teniendo en cuanta los tres casos citados que se pueden presentar.

EJERCICIO 7.6: DAR LA VUELTA A CADA UNA DE LAS PALABRAS DE UN TEXTO.

Se trata de realizar un programa que lea una frase y, conservando el orden de las palabras que ha leído, escriba cada una de estas palabras al revés. Por ejemplo:

"La casa de la pradera"

se transformaría en:

"aL asac ed al aredarp"

Guarda este programa con el nombre de ***alreves1.c***

Nota: Este Ejercicio se va a resolver junto con el siguiente.

EJERCICIO 7.7: ESCRIBIR LAS PALABRAS DE UN TEXTO EN ORDEN INVERSO.

Se trata de realizar un programa que lea un fichero y cambie el orden de las palabras al escribirlo de tal manera que la primera palabra sea la última y viceversa. Por ejemplo:

"La casa de la pradera"

se transformaría en:

"pradera la de casa La"

Guarda este programa con el nombre de ***alreves2.c***

Solución comentada del Ejercicio 7.7.

```
/* fichero alreves2.c */
/* Se lee cada palabra y se almacena de modo dinamico */
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#define MAXFRA 10

void main(void) {
    char c, temp[20];
    char **frase;
    int n, j=0, i, npal=0;

    /* reservar espacio para el vector de punteros a las palabras */
    frase=malloc(MAXFRA*sizeof(char*));

    printf("Introduce una frase, y pulsa \"Intro\" para finalizar.\n");
    /* Lectura de las palabras */
    do {
        c=getchar();
        if (c!=' '&&c!='\n')
            temp[j++]=c;
        else {
            temp[j++]='\0';
            frase[npal]=malloc(j*sizeof(char));
            strcpy(frase[npal++], temp);
            j=0;
        }
    } while (c!='\n');

    printf("\nLas palabras de la frase son:");
    for (j=0; j<npal; j++)
        printf("\n%s ", frase[j]);
```

```
    printf("\n\nLa frase escrita al reves es:\n");
    n=npal-1;
    for (i=n; i>=0; i--)
        printf("%s ", frase[i]);

    printf("\n\nLas palabras al reves son:\n");
    for (i=0; i<npal; i++) {
        for (j=strlen(frase[i])-1; j>=0; j--)
            printf("%c", frase[i][j]);
        printf(" ");
    }
    printf("\n");
}
```

Comentario: La idea principal de este Ejercicio es ir leyendo cada una de las palabras y guardándolas en cadenas de caracteres. La variable ***frase*** es un puntero a un vector de punteros también llamado ***frase***, que apuntan a las primeras letras de cada palabra. Con esta estructura es bastante fácil escribir las palabras en orden inverso y también escribir en orden inverso las letras de cada palabra. Se utiliza rererva dinámica de memoria.

EJERCICIO 7.8: DESCOMPOSICIÓN DE UNA MATRIZ CUADRADA CUALQUIERA EN SUMA DE UNA MATRIZ SIMÉTRICA Y OTRA ANTISIMÉTRICA.

En este ejercicio tienes que descomponer una matriz cuadrada cualquiera en la suma de una matriz simétrica y otra antisimétrica.

Para cada par de elementos de la misma posición (Simétrica [i][j] y Antisimétrica [i][j]) tendrás que aplicar las siguientes fórmulas:

$$\text{Simétrica [i] [j]} = (\text{matriz [i] [j]} + \text{traspuesta [i][j]})/2$$

$$\text{Antisimétrica [j] [i]} = (\text{matriz [i] [j]} - \text{traspuesta [i][j]})/2$$

Recuerda que como comprobación de la solución te puede servir saber que los elementos de la diagonal principal de la matriz antisimétrica han de ser ceros.

El tamaño de la matriz a descomponer tiene que ser una variable introducida por el usuario durante la ejecución por lo que tienes que usar reserva dinámica de memoria.Guarda el programa en un fichero con el nombre ***antisim.c***.

Solución comentada del Ejercicio 7.8.

```
/* fichero antisim.c */
#include <stdio.h>
#include <malloc.h>

void main(void) {
    double **a;
    double **sim;
    double **antisim;
    int    i, j, n;

    printf("Descomposicion de una matriz cuadrada cualquiera en \n");
    printf("suma de simetrica y antisimetrica.\n");
    printf("\nIntroduce la dimension de la matriz a descomponer:\n");
    printf("Dimension: ");
    scanf("%d", &n);
    printf("Introduce los valores de la matriz:\n");
    a=malloc(n*sizeof(double*));
    sim=malloc(n*sizeof(double*));
    antisim=malloc(n*sizeof(double*));
```

```
    for (i=0; i<n; i++) {
        a[i]=malloc(n*sizeof(double));
        sim[i]=malloc(n*sizeof(double));
        antisim[i]=malloc(n*sizeof(double));
    }

    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            printf("%d,%d: ", i+1, j+1);
            scanf("%lf", &a[i][j]);
        }

    for (i=0; i<n; i++)
        for (j=0; j<n; j++) {
            sim[i][j]=(a[i][j]+a[j][i])/2;
            antisim[i][j]=(a[i][j]-a[j][i])/2;
        }

    printf("\nLa matriz simetrica es:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++)
            printf("%lf  ", sim[i][j]);
        printf("\n");
    }
    printf("\n");
    printf("La matriz antisimetrica es:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++)
            printf("%lf   ", antisim[i][j]);
        printf("\n");
    }
    printf("\n");
    printf("La matriz original debe ser:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++)
            printf("%lf   ", sim[i][j]+antisim[i][j] );
        printf("\n");
    }
    printf("\n");
}
```

Comentario: Cualquier matriz cuadrada puede descomponerse de esta forma utilizando las fórmulas dadas en el enunciado del Ejercicio. En el programa presentado se utiliza reserva dinámica de memoria para estas tres matrices. Se lee la matriz inicial y se hallan las otras dos. Al final se suman para comprobar que se obtinne de nuevo la matriz original.

EJERCICIO 7.9: CALCULAR *SEN(X)* POR INTERPOLACIÓN A PARTIR DE UNA TABLA

Realiza un programa principal que calcule el ***seno*** de un ángulo introducido por teclado (en grados), interpolando entre los valores de una tabla. Esta tabla se encuentra en el fichero ***tabla.d*** del disco, y está formada por una columna con los valores del seno de los ángulos comprendidos entre 0º y 90º, ambos inclusives. El ángulo cuyo seno se quiere calcular se introducirá por teclado con un formato tal y como 35.22.

Guarda el programa como ***interpol.c***.

Solución comentada del Ejercicio 7.9.

```
/* fichero interpol.c */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define  PI 3.14159265358979
```

```
void main(void) {
    double vector[91];
    int    x1, x2;
    double y1, y2;
    double x, y, a, b, radianes;
    FILE   *fi;
    int    i;

    fi=fopen("tabla.d", "r");
    for (i=0; i<=90; i++)
        fscanf(fi, " %lf", &vector[i]);
    printf("Este programa calcula el valor del seno de un angulo\n");
    printf("por interpolacion entre dos valores.\n");
    printf("\nIntroduce el valor del angulo en grados:\n");
    printf("valor: ");
    scanf("%lf", &x);
    printf("\n");

    /* Valores entre los que interpolar */
    x1=(int)x;
    x2=x1+1;
    y1=vector[x1];
    y2=vector[x2];

    /* Cálculo del valor interpolado */
    a=(y2-y1)/(x2-x1);
    b=(y1*x2-y2*x1)/(x2-x1);
    y=a*x+b;
    printf("El valor del seno de %6.2lf es %18.15lf.\n", x, y);
    radianes=x*PI/180.0;
    printf("%s%6.2lf es %18.15lf%s",
         "El valor del seno de ", x,
         sin(radianes), ", calculado con math.h\n");
}
```

Comentario: La interpolación entre una serie de valores almacenados en memoria es una de las posibles formas de evaluar funciones con expresiones matemáticas complicadas o incluso que no responden a ninguna expresión analítica concreta. En este ejemplo se interpola linealmente para hallar el valor de la función ***seno(x)*** y el resultado se compara con el que se obtiene con la función ***sin()*** definida en ***math.h***. Puede observarse que que con la interpolación lineal sólo se obtienen unas pocas cifras exactas.

PRÁCTICA 8: EXAMEN FINAL

Como última práctica se presentan resueltos a continuación los 12 ejercicios –4 por cada grupo– que constituyeron el examen final de febrero de 1995.

EJERCICIO 8.1: PROGRAMA QUE REALIZA LA MISMA FUNCIÓN QUE EL COMANDO *COPY* DE MS-DOS

Se trata de realizar un programa que copie el contenido de un fichero de texto en otro fichero con nombre distinto. El programa pedirá en primer lugar el nombre de un fichero ya existente; una vez introducido dicho nombre pedirá otro nombre que será el del nuevo fichero donde irá la copia. Con estos datos, el programa realizará una copia del texto contenido en el primer fichero en otro fichero con el segundo nombre en el mismo directorio. El nombre del programa será ***copiar.c***.

Para nota: se trata de realizar un programa que realice la copia de ficheros pero dando los nombres de los programas en la línea de comandos, es decir, pasando los nombres como argumentos de la función ***main()***. Este programa se llamará ***copiar1.c***

Solución comentada del Ejercicio 8.1.

```
/* fichero copiar.c */
#include <stdio.h>
#include <stdlib.h>

void main() {
    int ch;
    char file1[13], file2[13];
    FILE *f1, *f2;

    printf("Teclea el nombre de fichero original: ");
    scanf("%s", &file1);
    printf("y ahora el nombre de fichero de copia: ");
    scanf("%s", &file2);

    f1=fopen(file1, "r");
    f2=fopen(file2, "w");
    if(!f1)
        printf("El fichero %s no existe!!", file1);
    else
        while ((ch=getc(f1))!=EOF)
            putc(ch,f2);
    fclose(f1);
    fclose(f2);
}
```

Comentario: La resolución del problema es muy sencilla: basta con abrir dos ficheros: uno de lectura y otro de escritura, que contengan respectivamente el fichero original y el fichero copia. Una vez abiertos estos ficheros, se leen los caracteres del fichero original y se copian en el fichero copia. Para ello se emplea el bucle ***while***.

EJERCICIO 8.2: CENTRO DE MASAS DE UN SISTEMA DE MASAS PUNTUALES

Se pide realizar un programa que calcule el centro de masas de un sistema de n masas puntuales. Para ello el programa pedirá conjuntos de tres números reales: la coordenada x del punto, la coordenada y y su masa m. El número n de masas se lee de antemano. El programa imprimirá por pantalla como resultado las coordenadas xg e yg del centro de masas del conjunto de puntos que se han introducido. El programa se llamará ***cdg.c***. Las fórmulas que dan la posición del centro de gravedad son las siguientes:

$$xg = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i} \qquad yg = \frac{\sum_{i=1}^{n} m_i y_i}{\sum_{i=1}^{n} m_i}$$

Solución comentada del Ejercicio 8.2.

```
/* fichero cdg.c */
/* centro de gravedad de un sistema de masas puntuales */
#include<stdio.h>
#define SIZE 20

void main(void) {
    double sumat1, sumat2, sumat3;
    double xg, yg, x[SIZE], y[SIZE], m[SIZE];
    int  i=0, np=0;
    char c='a';    /* iniciar a un carácter cualquiera */

    printf("Calculo del cdg de un sistema discreto de masas\n");
    printf("Teclea el numero de puntos: ");
    scanf(" %d", &np);
    for (i=0; i<np; i++) {
        printf("Coordenadas(x,y) y masa del punto %d: ", i+1);
        scanf("%lf %lf %lf", &x[i], &y[i], &m[i]);
    }
    sumat1=sumat2=sumat3=0;
    for (i=0; i<np; i++){
        sumat1+=m[i]*x[i];
        sumat2+=m[i];
        sumat3+=m[i]*y[i];
    }
    xg=sumat1/sumat2;
    yg=sumat3/sumat2;
    printf("\n\nEl centro de gravedad es (%5.2lf, %5.2lf)\n", xg, yg);
}
```

Comentario: Mientras el carácter leído no se corresponda con el de fin de fichero el programa lee las coordenadas **x** e **y**, y la masa **m** de cada punto.

EJERCICIO 8.3: CÁLCULO DE UNA RAÍZ DE UNA FUNCIÓN POR EL MÉTODO DE BISECCIÓN

Según el teorema de Bolzano, si una función continua tiene valores de distinto signo en los extremos ***a*** y ***b*** de un intervalo, la función tiene al menos un cero (una raíz) en ese intervalo. Este teorema se utilizará como base para un método de cálculo de raíces de funciones, llamado ***método de la bisección***. Se parte de un par de puntos ***a*** y ***b*** en los que se cumple la condición ***f(a)*f(b)<0*** (la función tiene signos opuestos, luego existe al menos una raiz en ***[a, b]***). A continuación se estudia la función en el punto ***c***, que es el punto medio del intervalo (***c=(a+b)/2***), y se estudia el signo de ***f(c)***. Si ***f(c)*** tiene el mismo signo que ***f(a)*** se hace ***a=c*** y se vuelve a comenzar (se tiene un nuevo intervalo ***[a, b]*** en el que está la raíz y que es la mitad que el anterior); si ***f(c)*** tiene el mismo signo que ***f(b)*** se hace ***b=c*** y se vuelve a comenzar. El proceso prosigue con intervalos ***[a, b]*** cada vez más pequeños hasta que se cumple una de las siguientes condiciones:

- ***f(c)=0*** ***c*** es la solución buscada
- ***f(c) < eps1*** ***c*** es la solución aproximada buscada
- ***b−a < eps2*** ***c=(b−a)/2*** es la solución aproximada buscada

Aplicar este método para encontrar la raiz de la función ***f(x)=cos(x)−x***, en el intervalo [1, 4]. Probar con valores ***eps1=10e−08*** y ***eps2=10e−12***. Este programa se llamará ***bisec.c***.

Solución comentada del Ejercicio 8.3.

```
/* fichero bisec.c */
/* cálculo de una raíz por el método de la bisección */
#include <stdio.h>
#include <math.h>
#define EPS1 10E-8
#define EPS2 10E-12

void main(void) {
    double a=0, b=4, c;
    double func(double);

    c=(a+b)/2.0;
    while (fabs(func(c))>EPS1 && (b-a)>EPS2) {
        c=(a+b)/2;
        if (func(c)*func(a)>0)
            a=c;
        else
            b=c;
    }
    printf("El valor de la raiz es: %12.6e\n", c);
}

double func(double x) {
    return (cos(x)-x);
}
```

Comentario: Mientras se cumplan las dos condiciones (que el valor absoluto de ***func(c)*** sea mayor que ***EPS1*** y que la distancia entre **a** y **b** sea mayor que ***EPS2***), el programa continúa haciendo aproximaciones de la solución. Conviene darse cuenta de que la condición de que ***func(c)=0*** es más fuerte que la de que sea menor que ***EPS1***, por lo que no es necesario ponerla. Al salir del bucle ***while*** se imprime la solución que ya se considera suficientemente aproximada.

EJERCICIO 8.4: GIRAR 90º UNA MATRIZ RECTANGULAR

Se trata de realizar un programa que gire una matriz rectangular (mxn) 90 grados en sentido antihorario. A continuación se ilustra con un ejemplo lo que se pretende realizar.

Matriz original: $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

Matriz girada 90 grados: $\begin{bmatrix} 3 & 6 \\ 2 & 5 \\ 1 & 4 \end{bmatrix}$

Si se vuelve a girar 90 grados resultará:

$$\begin{bmatrix} 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

Pensar bien, a la vista de estos ejemplos, como queda modificada la matriz con un giro de 90º en sentido antihorario. Hacer que el programa pueda aplicar varios giros consecutivos, de forma que se pueda comprobar que con cuatro giros se vuelve a la posición inicial. Llama a este programa ***rotamat.c***.

Solución comentada del Ejercicio 8.4.

```
/* fichero rotamat.c */
/* girar una matriz rectangular */
#include<stdio.h>
#include<stdlib.h>
double **crearMatriz(int, int);

void main(void) {
    int numfil, numcol;
    int i, j, num, giro;
    double **a;

    printf("Introduce el numero de filas: ");
    scanf("%d", &numfil);
    printf("Introduce el numero de columnas: ");
    scanf("%d", &numcol);
    a=crearMatriz(numfil, numcol);
    for (i=0; i<numfil; i++)
        for(j=0; j<numcol; j++) {
            printf("Elemento (%d,%d):  ", i+1, j+1);
            scanf("%lf", &a[i][j]);
        }
    printf("\nLa matriz original es: ");
    for (i=0; i<numfil; i++) {
        printf("\n");
        for (j=0; j<numcol; j++)
            printf("%12.4lf  ", a[i][j]);
    }
    printf("\n\nCuantas veces deseas girar la matriz?: ");
    scanf("%d", &num);
    giro=num%4;
    if (giro==1) {
        printf("\nLa matriz girada %d veces es:", num);
        for (j=numcol-1; j>=0; j--) {
            printf("\n");
            for(i=0; i<numfil; i++)
                printf("%12.4lf  ", a[i][j]);
        }
    } else if(giro==2) {
        printf("\nLa matriz girada %d veces es:", num);
        for (i=numfil-1; i>=0; i--) {
            printf("\n");
            for (j=numcol-1; j>=0; j--)
                printf("%12.4lf  ", a[i][j]);
        }
    } else if(giro==3) {
        printf("\nLa matriz girada %d veces es:",num);
        for (i=numcol-1; i>=0; i--) {
            printf("\n");
            for (j=numfil-1; j>=0; j--)
                printf("%12.4lf  ", a[j][i]);
        }
    } else {
        printf("\nLa matriz girada %d veces es:", num);
        for(i=0; i<numfil; i++) {
            printf("\n");
            for (j=0; j<numcol; j++)
                printf("%12.4lf  ", a[i][j]);
        }
    }
    printf("\n");
}
```

```
double **crearMatriz(int m, int n) {
    double **matriz;
    int i;
    matriz=calloc(m, sizeof(double *));
    matriz[0]=calloc(m*n, sizeof(double));
    for(i=0; i<m; i++)
        matriz[i]=matriz[0]+n*i;
    return matriz;
}
```

Comentario: Se guarda memoria para la matriz y se pregunta el número de veces que se desea girar. Girar 7 veces es equivalente a girar 3 veces, y por eso se calcula el resto de la división entre 4.

En primer lugar se imprime la matriz original y luego la que corresponde a cada giro. Hay que darse cuenta de que no es necesario reservar memoria para una nueva matriz.

EJERCICIO 8.5: CAMBIAR LA FORMA DE DE UNA MATRIZ, MANTENIENDO EL ORDEN (POR FILAS) DE SUS ELEMENTOS

Una matriz con m filas y n columnas tiene $m \times n$ elementos, y en C se almacena en la memoria por filas. Se pide realizar un programa que haga lo siguiente: 1) leer m, n y los elementos de una matriz $m \times n$. 2) Leer unos valores p y q, y con los mismos elementos de la matriz $m \times n$ leída previamente, crear otra matriz de p filas y q columnas, que tenga el mismo número de elementos que la anterior (comprobar que $p \times q = m \times n$, y si no se cumple volver a leer p y q), de modo que los elementos estén almacenados en la memoria en el mismo orden, o dicho de otra forma, que el orden por filas se mantenga. El siguiente ejemplo ilustra lo que se pretende realizar.

Dada la siguiente matriz 3×4:

$$\begin{bmatrix} 1 & 3 & 8 & -1 \\ 6 & 4 & 2 & 9 \\ 6 & -3 & 1 & 5 \end{bmatrix}$$

hallar una matriz 2×6 que tenga sus elementos en el mismo orden en la memoria.

Solución: 3×4 es igual que 2×6, luego los números son correctos. Los elementos de la matriz original están en la memoria ordenados por filas, es decir, su orden es:

$$\begin{bmatrix} 1 & 3 & 8 & -1 & 6 & 4 & 2 & 9 & 6 & -3 & 1 & 5 \end{bmatrix}$$

Reorganizándolos como matriz 2×6 se obtiene:

$$\begin{bmatrix} 1 & 3 & 8 & -1 & 6 & 4 \\ 2 & 9 & 6 & -3 & 1 & 5 \end{bmatrix}$$

Se pide pues realizar un programa que pida una matriz (las filas, las columnas y los elementos), pida otros valores de filas y columnas y presente la matriz de la nueva forma. El programa deberá chequear, previamente, si los nuevos valores de filas y columnas que se introducen dan el mismo producto *filas*×*columnas* de la matriz original. Llámese a este programa ***cambio.c***

Solución comentada del Ejercicio 8.5.

```
/* fichero cambio.c */
#include<stdio.h>
#include<stdlib.h>

void main(void) {
    int m, n, p, q;
    int i, j, k, l;
    double **a, **b;
```

```
    double **crearMatriz(int, int);

    printf("Numero de filas y de columnas de la matriz: ");
    scanf("%d%d", &m, &n);
    a=crearMatriz(m,n);

    printf("\nIntroduce los elementos de la matriz.\n");
    for (i=0; i<m; i++)
        for(j=0; j<n; j++){
            printf("Elemento (%d,%d): ", i+1, j+1);
            scanf("%lf", &a[i][j]);
        }
    do {
        printf("\nNumero de filas y columnas de la nueva matriz: ");
        scanf("%d%d", &p, &q);
    } while(p*q != m*n);

    b=crearMatriz(p, q);
    for (i=0, k=0; i<m; i++)
        for (j=0, l=0; j<n; j++){
            k=(i*n+j)/q;
            l=(i*n+j)-k*q;
            b[k][l]=a[i][j];
        }
    for (i=0; i<p; i++){
        printf("\n");
        for (j=0; j<q; j++)
            printf("%5.1lf", b[i][j]);
    }
    printf("\n");
}

double **crearMatriz(int m, int n) {
    double **matriz;
    int i;
    matriz=calloc(m, sizeof(double *));
    matriz[0]=calloc(m*n, sizeof(double));
    for(i=0; i<m; i++)
        matriz[i]=matriz[0]+n*i;
    return matriz;
}
```

Comentario: El problema se resuelve introduciendo primeramente una matriz (reservando memoria de modo dinámico e introduciendo los datos). Luego se introducen las dimensiones de otra matriz de tal forma que el producto de filas por columnas de la nueva matriz sea igual al de la matriz anterior. Una vez que hemos reservado sitio para la segunda matriz, se va rellenando sabiendo que la equivalencia de índices entre las dos matrices es: ***k=(i*n+j)/q*** y ***l=(i*n+j)-k*q***, siendo ***q*** el número de filas de la segunda matriz y ***n*** el número de filas de la primera. Los índices de la primera matriz son ***i*** y ***j*** y los de la segunda ***k*** y ***l***.

EJERCICIO 8.6: CÁLCULO DE UNA RAÍZ DE UNA FUNCIÓN POR EL MÉTODO DE NEWTON

El método de Newton permite hallar de modo iterativo raíces (ceros) de funciones no lineales. Este método está basado en el desarrollo en serie de Taylor de la función ***f(x)*** en el punto x_i, siendo x_i una cierta aproximación de la solución ***x*** que se desea calcular. El desarrollo en serie de Taylor produce:

$$f(x) = f(x_i) + (x - x_i) f'(x_i) + \ldots = 0 \qquad (1)$$

Utilizando sólo los dos primeros términos del desarrollo en serie (es decir, sustituyendo la función por la tangente en el punto x_i), se tiene la siguiente ecuación lineal:

$$f(x_i) + (x - x_i) f'(x_i) = 0 \qquad (2)$$

de donde se puede despejar el valor de la ***x*** que anula esta expresión (la raíz de la ecuación linealizada), es decir, una nueva aproximación a la verdadera solución. A esta nueva aproximación le llamaremos x_{i+1}. Este valor se puede calcular a partir de (2) con la expresión:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{3}$$

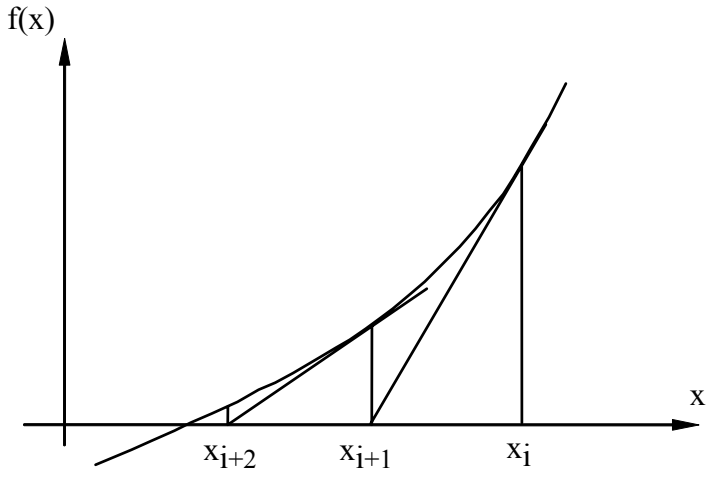
Ahora volvemos a aplicar el mismo procedimiento, sustituyendo la función no lineal por los dos primeros términos (de nuevo la tangente) de su desarrollo en serie alrededor del punto x_{i+1}, y calculando una nueva y mejor aproximación x_{i+2}:

$$x_{i+2} = x_{i+1} - \frac{f(x_{i+1})}{f'(x_{i+1})} \tag{4}$$

expresión equivalente a la (3) con los subíndices actualizados. El proceso prosigue de la misma manera hasta que el valor absoluto de la diferencia relativa entre dos aproximaciones consecutivas sea menor que un determinado número *epsilon*,

$$\frac{|x_{i+1} - x_i|}{|x_{i+1}|} < epsilon \tag{5}$$

La siguiente figura ilustra el significado geométrico del método de Newton para hallar la raíz de una función no lineal: se parte de un primer punto en el que se linealiza la función (se sutituye la función por la recta tangente) y se halla la raíz de la función linealizada. En esa raíz se vuelve a linealizar la función original y se halla una nueva aproximación a la verdadera solución. El proceso prosigue hasta que la diferencia entre dos aproximaciones consecutivas es suficientemente pequeña.



Aplica este método a la función $x^3-3x^2-x+3=0$, comenzando con un valor inicial $x=10$. Guarda este programa en un fichero que se llame ***newton.c***.

Solución comentada del Ejercicio 8.6.

```
/* fichero newton.c */
/* cálculo de una raíz de un polinomio por el método de Newton */
#include<stdio.h>
#include<math.h>
#define EPS 10e-6
```

```
void main(void) {
    double x;
    double raiz=10.0, error=5.0;
    double f(double);
    double fd(double);
    double absoluto(double);

    while (error>EPS) {
        x=raiz;
        raiz=x-f(x)/fd(x);
        error=absoluto((raiz-x)/raiz);
    }
    printf("El valor de la raiz es: %9lf\n",raiz);
}

double f(double x) {
    double y;
    y=x*x*x-3*x*x-x+3;
    return y;
}

double fd(double x) {
    double y;
    y=3*x*x-6*x-1;
    return y;
}

double absoluto(double x) {
    if (x>=0)
        return x;
    else
        return -x;
}
```

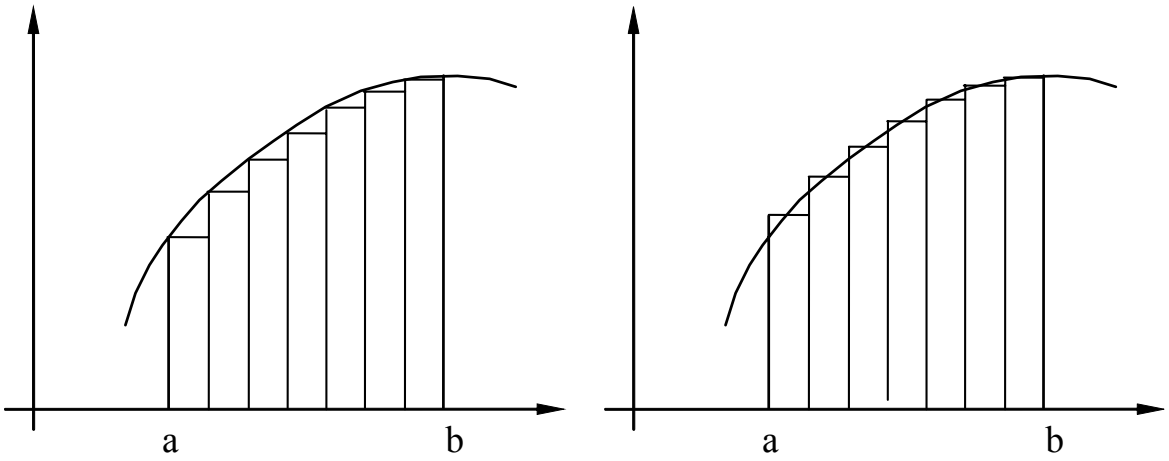
Comentario: La función ***f()*** evalúa el valor de la función ***f(x)*** en un punto dado. La función ***fd()*** evalúa el valor de la derivada de la función en ese mismo punto. La función ***absoluto()*** devuelve el valor absoluto de la variable pasada como argumento. Mientras el error sea mayor que el permitido, se calculan iterativamente los nuevos valores de ***x***.

EJERCICIO 8.7: INTEGRACIÓN NUMÉRICA DE UNA FUNCIÓN

La integral definida entre los puntos ***a*** y ***b*** de una función continua y acotada ***f(x)*** representa el área comprendida debajo de esa función. En ocasiones es necesario calcular integrales (áreas) de modo numérico, es decir, sin conocer la integral de la función ***f(x)***. Existen varios posibles métodos para calcular esta área. Quizás el más sencillo sea sustituir el área por un conjunto de ***n*** rectángulos elementales de base ***h=(b−a)/n*** y altura ***f(a+ih)***, ***i=0, 1, 2, ..., n−1***. El área sería:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} f(a+ih)h \qquad (1)$$

La representación gráfica de esta forma de aproximar la integral se presenta en la figura siguiente (parte izquierda). Resulta que si ***n*** se hace muy grande (***h*** muy pequeño) el error será pequeño.



Otro método algo más complicado (representado en la figura de la derecha) consiste en sustituir el área por un conjunto de ***n*** trapecios elementales de base ***h*** y lados ***f(a+ih)*** y ***f(a+(i+1)h)***, o lo que es lo mismo, por ***n*** rectángulos de altura ***(f(a+ih)+f(a+(i+1)h))/2***. Es obvio que con este segundo método los errores van a ser más pequeños. En este caso, si llamamos f_i a ***f(a+ih)***, la fórmula resulta ser:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1}(f_i + f_{i+1})h/2 = \frac{f_0 + f_n}{2}h + \sum_{i=1}^{n-1} f_i h \qquad (2)$$

A la vista de estos métodos, se pide realizar un programa que calcule el área de la función ***f(x)=x^3+x^2–5x+3*** entre 0 y 3, ***por el segundo de los métodos*** explicados. El programa deberá pedir el valor de ***n***. Llama a este programa ***integra.c***

Solución comentada del Ejercicio 8.7.

```
/* fichero integra.c */
/* cálculo de la integral definida de una función */
#include<stdio.h>

void main(void) {
    int n, k;
    double area=0.0, h;
    double f(double);

    printf("Introduce el numero de trapecios de la aproximacion: ");
    scanf("%d", &n);
    h=3./n;
    for (k=0; k<n; k++)
        area+=((f(k*h)+f((k+1)*h))*h)/2;
    printf("El area calculada es %9.3lf\n", area);
}

double f(double x) {
    double y;
    y=x*x*x+x*x-5*x+3;
    return y;
}
```

Comentario: La función ***f()*** evalúa el valor de la función en un punto dado. Una vez determinado el número de trapecios que se van a utilizar para calcular el área, ésta se determina mediante la expresión (2) del enunciado.

EJERCICIO 8.8: CREAR EL COMANDO *MORE*

Existe un comando en MS-DOS y en UNIX llamado ***more*** que permite presentar los archivos grandes por pantalla de forma "paginada", es decir, de modo que la presentación del archivo se detiene al llegar a un número determinado de líneas (normalmente 2 ó 3 líneas menos de las que caben en la pantalla), y prosigue con las líneas siguientes al pulsar una tecla cualquiera (distinta de la "q", que tiene un uso especial como ahora se verá).

Se pide realizar un programa que sea capaz de presentar cualquier archivo de texto de forma paginada: el programa pedirá el nombre del archivo a presentar y escribirá tantas líneas como permita el monitor del PC (en este caso, 18 líneas). Al cabo de la primera página, aparecerá la palabra "***more?***" de tal forma que si se pulsa la letra "***q***" se detiene la ejecución y se acaba el programa, pero si se pulsa cualquier otra letra, se continúa mostrando el fichero con la siguiente página (18 líneas) donde se repetirá el mismo proceso. La ejecución del programa terminará cuando se haya presentado el fichero en su totalidad o se haya pulsado la tecla "***q***", como ya se ha señalado. El programa se llamará ***masmore.c***. Pruébalo con un fichero que tenga al menos 50 líneas.

Solución comentada del Ejercicio 8.8.

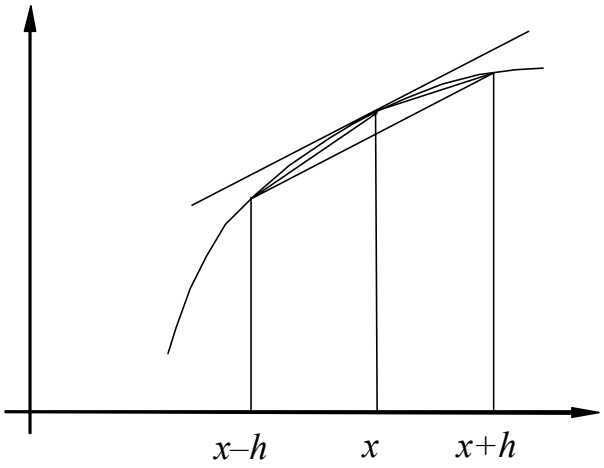
```
/* fichero masmore.c */
#include<stdio.h>
#include<stdlib.h>

void main(void) {
    char archivo[13];
    FILE *fp;
    char c[10]="", c2='a';
    int li=0;

    printf("Introduce el nombre del fichero: ");
    scanf("%s", archivo);
    fp=fopen(archivo, "r");

    for ( ; c2!=EOF; ) {
        if (li==18) {
            printf("more?");
            scanf(" %s", c); /* para retirar del input el carácter '\n' */
            if ((c[0]=='Q')||(c[0]=='q'))
                exit(1);     /* se termina el programa */
            else {
                li=0;        /* se imprimen 18 lineas mas */
            }
        }
        c2=fgetc(fp);        /* se leen caracteres del fichero */
        printf("%c", c2);
        if (c2=='\n')
            li++;
    } /* fin del bucle for */
}
```

Comentario: Se leen e imprimen todos los caracteres hasta que se acabe el fichero, y se suma 1 a un contador cada vez que se salta de línea. En el caso de llegar a la línea nº 18 sin haber acabado el fichero, se imprime la pregunta ***more?***. Si se responde con un carácter cualquiera se continuan imprimiendo las 18 líneas siguientes del fichero; si la respuesta es el carácter '***q***' ó '***Q***' el programa termina.

Se produce un cierto problema al leer un carácter desde la consola de entrada, puesto que después de dicho carácter se pulsa ***Intro*** y dicho carácter se queda en el buffer de entrada, y se lee la siguiente vez que se pretenda leer un carácter desde teclado. Para evitar esto se utiliza la función ***scanf()*** que lee todo lo que esté en la línea de entrada incluyendo el final de línea.

EJERCICIO 8.9: DIFERENCIACIÓN NUMÉRICA

La derivada de una función ***y=f(x)*** con respecto a ***x*** se representa geométricamente mediante la pendiente angular de la recta tangente a la función en el punto considerado. Supongamos que tenemos una función ***f(x)*** cuyos valores podemos calcular, pero que no conocemos la función derivada ***f'(x)***. En este caso la derivada se puede aproximar numéricamente de varios modos, dos de los cuales se muestran gráficamente en la siguiente figura.



Una primera forma de aproximar el valor de ***f'(x)*** es utilizar la pendiente de una de las dos secantes correspondientes a los intervalos $[x-h, x]$ y $[x, x+h]$, resultando:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \tag{1}$$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{2}$$

pero es mucho más preciso (se ve a simple vista) el utilizar la siguiente fórmula, correspondiente a la secante definida entre los puntos $[x-h, x+h]$:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{3}$$

Se pide realizar un programa que comience calculando los 101 valores de una función ***seno(x)*** desde ***x=0.0*** hasta ***x=1.0***, con intervalos de ***h=0.01***, almacenándolos en un vector ***f[ ]***. A continuación se creará otro vector ***d[ ]*** que contenga las derivadas numéricas de la función ***f[ ]***, calculadas con las fórmulas anteriores. En principio se utilizará la ecuación (3) que es la más exacta, pero para el primer valor de ***d[ ]*** habrá que utilizar la expresión (2) que calcula la derivada en ***x*** sin necesidad de valores anteriores a ***x***, y para el último valor de ***d[ ]*** se utilizará la ecuación (1), que aproxima la derivada sin necesidad de valores de ***f[ ]*** posteriores a ***x***. Finalmente se escribirán en cuatro columnas con al menos 8 decimales los valores de ***x***, de la función ***f[i]***, de la derivada aproximada ***d[i]*** y de la derivada exacta (lógicamente ***cos(f[i])***). Guarda este programa con el nombre de ***derivada.c***.

Solución comentada del Ejercicio 8.9.

```
/* fichero derivada.c */
/* cálculo numérico de la derivada de una función en un punto */
#include<stdio.h>
#include<math.h>
#define SIZE 101
```

```
void main(void) {
    int i;
    double h=0.01;
    double x[SIZE], f[SIZE], d[SIZE], e[SIZE];

    x[0]=0.0;
    for (i=1; i<SIZE; i++)
        x[i]=i*h;
    for (i=0; i<SIZE; i++)
        f[i]=sin(x[i]);                /* función seno(x) */
    d[0]=(f[1]-f[0])/h;
    for (i=1; i<SIZE-1; i++)
        d[i]=(f[i+1]-f[i-1])/(2*h);    /* derivada numérica */
    d[SIZE-1]=(f[SIZE-1]-f[SIZE-2])/h;
    for (i=0; i<SIZE; i++)
        e[i]=cos(x[i]);                /* derivada exacta */
    printf("           x           seno(x)       deriv. num.   deriv. exacta\n");
    for (i=0; i<SIZE; i++)
        printf("%15.8lf %15.8lf %15.8lf %15.8lf\n",x[i], f[i], d[i], e[i]);
}
```

Comentario: Se reserva espacio en memoria para los vectores que se van a utilizar: ***x*** será un vector donde se van a guardar los puntos en los que se va a calcular la función y su derivada.; ***f*** el valor de la función ***seno*** y ***d*** el valor de la aproximación de la derivada. Por último, ***e*** será el vector donde se almacene el valor exacto de la derivada que es la función ***coseno***.

EJERCICIO 8.10: CALCULADORA ELEMENTAL INTERACTIVA

Se pide preparar un programa que realice, de modo interactivo, las cuatro operaciones básicas: ***suma***, ***resta***, ***multiplicación*** y ***división***. Al arrancar, el programa presentará un menú principal con cinco opciones: *suma*, *resta*, *multiplicación*, *división* y *salida* (para terminar la ejecución). Estas opciones se elegirán mediante un número o una letra indicada en el propio menú. Una vez elegida una de las cuatro primeras opciones, se pedirán dos números –los operandos– y el programa presentará el resultado de la operación. Una vez realizado esto, el programa volverá al menú principal y se podrá realizar una nueva operación, hasta que se utilice la opción de ***salida***.

El programa se llamará ***calcula.c***.

Solución comentada del Ejercicio 8.10.

```
/* fichero calcula.c */
/* calculadora elemental interactiva */
#include <stdio.h>
#include <stdlib.h>

void main(void) {
    int fin=1;
    double a, b;
    char c;
    double suma(double, double);
    double resta(double, double);
    double multiplicacion(double, double);
    double division(double, double);
    void valor(double *, double *);
    void menu(void);

    do {
        menu();
        c=getchar();
```

```
        if (c!='\n' && (c<'1'||c>'5')) {
            printf("%s%s", "\nNo es una opcion valida",
                " introduce de nuevo el numero");
            printf("\nPulsa una tecla y repite el proceso");
            getchar();
            getchar();
        }
        switch(c) {
            case '1':
                valor(&a, &b);
                suma(a, b);
                break;
            case '2':
                valor(&a, &b);
                resta(a, b);
                break;
            case '3':
                valor(&a, &b);
                multiplicacion(a, b);
                break;
            case '4':
                valor(&a, &b);
                division(a, b);
                break;
            case '5':
                fin=0;
                break;
        }
    } while(fin);
}

void menu(void) {
    system("cls");
    printf("\n\n\n\tBienvenido al programa matematico.\n");
    printf("\tIntroduce una de las siguientes opciones:");
    printf("\n\n\t\t1.- Sumar numeros");
    printf("\n\n\t\t2.- Restar numeros");
    printf("\n\n\t\t3.- Multiplicar numeros");
    printf("\n\n\t\t4.- Dividir numeros");
    printf("\n\n\t\t5.- Salir\n\n");
}

double suma(double a, double b) {
    printf ("\n\nLa suma introducida es la siguiente:\n");
    printf ("%.2lf + %.2lf = %.2lf", a, b, (a+b));
    getchar();
    getchar();
    return (a+b);
}

double resta(double a, double b) {
    printf ("\n\nLa resta introducida es la siguiente:\n");
    printf ("%.2lf - %.2lf = %.2lf", a, b, (a-b));
    getchar();
    getchar();
    return (a-b);
}

double multiplicacion (double a, double b) {
    printf ("\n\nLa multiplicacion introducida es la siguiente:\n");
    printf ("%.2lf * %.2lf = %.2lf", a, b, (a*b));
    getchar();
    getchar();
    return (a*b);
}
```

```
double division(double a, double b) {
    printf ("\n\nLa division introducida es la siguiente:\n");
    printf ("%.2lf / %.2lf = %.2lf", a, b, (a/b));
    getchar();
    getchar();
    return (a/b);
}

void valor(double *a, double*b) {
    system ("cls");
    printf("Introduce el valor del primer termino: ");
    scanf("%lf", a);
    printf("Introduce el valor del segundo termino: ");
    scanf("%lf", b);
}
```

Comentario: Mientras la variable ***fin*** no valga cero (inicialmente vale 1) se despliega el menú inicial y, dependiendo del valor introducido, se llama a la función correspondiente que, una vez leídos los valores de los dos operandos, efectúa la operación que se desee. Si en el menú inicial se elige la opción "5", la variable ***fin*** vale 0 y termina la ejecución.

EJERCICIO 8.11: TABLA DE LOGARITMOS CON FORMATOS ADECUADOS POR COLUMNAS

Se pide realizar un programa que calcule los logaritmos decimal y neperiano de los números comprendidos entre 1 y 100, con intervalos de 5 en 5. Para calcular los logaritmos se empleará la librería matemática ***math.h***. La función que calcula el logaritmo decimal es ***log10(d)*** siendo ***d*** un número cualquiera, y la que calcula el logaritmo neperiano es ***log(d)***.

La salida por pantalla que debe dar este programa **deberá ser exactamente** la siguiente:

```
Este programa calcula el logaritmo decimal
y el logaritmo neperiano de los numeros entre el
uno y el 100, de 5 en 5.
    numero                log                   ln
-----------------------------------------------------
         1            0.000000            0.000000
         5            0.698970            1.609438
        10            1.000000            2.302585
        15            1.176091            2.708050
        20            1.301030            2.995732
        25            1.397940            3.218876
        30            1.477121            3.401197
        35            1.544068            3.555348
        40            1.602060            3.688879
        45            1.653213            3.806662
        50            1.698970            3.912023
        55            1.740363            4.007333
        60            1.778151            4.094345
        65            1.812913            4.174387
        70            1.845098            4.248495
        75            1.875061            4.317488
        80            1.903090            4.382027
        85            1.929419            4.442651
        90            1.954243            4.499810
        95            1.977724            4.553877
       100            2.000000            4.605170
```

Se pondrá atención especial en la alineación de las columnas, a los encabezamientos con su subrayado y a los puntos decimales. El nombre del programa será ***logarit.c***

Solución comentada del Ejercicio 8.11.

```
/* fichero logarit.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
void main(void) {
    int i;
    system("cls");
    printf("Este programa calcula el logaritmo decimal");
    printf("\ny el logaritmo neperiano de los numeros entre el\n");
    printf("uno y el 100, de 5 en 5.\n");
    printf ("%10s%20s%20s\n", "numero", "log", "ln");
    for (i=0; i<51; i++)
        putchar('-');
    printf ("\n%10d%20lf%20lf", 1, log10(1), log(1));
    for (i=5; i<=100; i=i+5)
        printf ("\n%10d%20lf%20lf", i, log10(i), log(i));
    printf("\n");
}
```

Comentario: En este ejercicio sólo hay que tener cuidado con el formato de la salida de los datos. También se debe de tener en cuenta que, al incrementarse de forma diferente, el primer término debe de ir fuera del bucle.

EJERCICIO 8.12: MÁXIMO VALOR Y VECTOR PROPIO POR ITERACIÓN DIRECTA

Existe un método iterativo muy sencillo para calcular el mayor valor propio de una matriz y su vector propio correspondiente. Este método se llama método de la ***iteración directa***. Este método procede de la siguiente forma:

Se parte de un vector arbitrario $\overline{\mathbf{y}}^0$ (por ejemplo, con todos los elementos con un mismo valor) que cumple la condición de tener módulo unidad (esta condición está indicada por la raya encima del nombre del vector). A partir de este vector, mediante el producto por la matriz **A**, hallamos un nuevo vector $\mathbf{y}^1$ (sin raya porque no será unitario):

$$\mathbf{y}^1 = \mathbf{A}\overline{\mathbf{y}}^0 \tag{1}$$

que normalizamos (es decir, hacemos unitario) dividiendo cada uno de sus elementos por el módulo:

$$\overline{\mathbf{y}}^1 = \frac{\mathbf{y}^1}{\left|\mathbf{y}^1\right|} \tag{2}$$

A partir de $\overline{\mathbf{y}}^1$ hallamos mediante el producto por la matriz **A** un nuevo vector $\mathbf{y}^2$ que normalizamos a continuación:

$$\mathbf{y}^2 = \mathbf{A}\overline{\mathbf{y}}^1 \tag{3}$$

$$\overline{\mathbf{y}}^2 = \frac{\mathbf{y}^2}{\left|\mathbf{y}^2\right|} \tag{4}$$

Proseguimos este proceso con pasos sucesivos basados en las fórmulas iterativas:

$$\mathbf{y}^i = \mathbf{A}\overline{\mathbf{y}}^{i-1} \tag{5}$$

$$\overline{\mathbf{y}}^i = \frac{\mathbf{y}^i}{\left|\mathbf{y}^i\right|} \tag{6}$$

Se puede demostrar que para *i* suficientemente elevado, el módulo $|\mathbf{y}^i|$ tiende al valor propio de máximo valor absoluto, que supondremos es λ_1 (supondremos que los valores propios están ordenados de modo decreciente $|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$). Al mismo tiempo, el vector $\overline{\mathbf{y}}^i$ tiende al vector propio correspondiente $\boldsymbol{x}^1$.

Se pide hacer un programa que lea una matriz cuadrada de dimensión *n* y calcule el máximo valor propio y el vector propio asociado con él. Para garantizar que ambos –el valor y el vector propio son reales– se considerará que la matriz **A** es simétrica. Se supondrá también que los valores propios son positivos. Una vez leída la matriz se formará un vector arbitrario $\mathbf{y}^0$ (por ejemplo, todo "1.0") que se normalizará para que tenga módulo unitario. A partir de ese vector, comenzará un proceso iterativo basado en las expresiones (5) y (6) que terminará cuando dos estimaciones consecutivas del mayor valor propio ($|\mathbf{y}^i|$) sean suficientemente próximas en valor relativo, o diciéndolo de otra forma, cuando se cumpla la condición:

$$abs\left(\frac{\left|\mathbf{y}^{i+1}\right|-\left|\mathbf{y}^{i}\right|}{\left|\mathbf{y}^{i+1}\right|}\right)<epsilon \tag{7}$$

donde *epsilon* es un número pequeño, definido por el usuario (por ejemplo 10e-06). Una vez calculados el valor y vector propio, se ecribirán en la pantalla y terminará la ejecución del programa. Guarda este programa con el nombre ***maxvalp.c***.

NOTA: Para satisfacer la curiosidad de los aficionados a las demostraciones algebraicas se incluye un breve esbozo de la demostración que está detrás del metodo de la ***iteración directa***. La demostración que sigue se separa un poco del algoritmo, sobre todo en el tema de la normalización de los vectores. Desde luego, no es en absoluto necesaria para hacer el programa. Sea cual sea el vector arbitrario elegido inicialmente, si la matriz **A** es simétrica dicho vector siempre se podrá expresar como combinación lineal de los *n* vectores propios $\mathbf{x}^i$ (con coeficientes desconocidos α_i)

$$\mathbf{y}^0=\sum_{i=1}^{n}\alpha_i\mathbf{x}^i \tag{i}$$

Por otra parte, definiendo el vector $\mathbf{y}^{i+1} = \mathbf{A}\ \mathbf{y}^i$, tendremos que al aplicar *m* veces la fórmula iterativa llegamos a:

$$\mathbf{y}^m=\mathbf{A}\mathbf{y}^{m-1}=\mathbf{A}^2\mathbf{y}^{m-2}=\mathbf{A}^m\mathbf{y}^0 \tag{ii}$$

Sustituyendo (i) en (ii) y utilizando las propiedades de los vectores propios:

$$\mathbf{y}^m=\mathbf{A}^m\mathbf{y}^0=\mathbf{A}^m\sum_{i=1}^{n}\alpha_i\mathbf{x}^i=\sum_{i=1}^{n}\alpha_i\mathbf{A}^m\mathbf{x}^i=\sum_{i=1}^{n}\alpha_i\lambda_i^m\mathbf{x}^i \tag{iii}$$

Si sacamos factor común λ_1^m y tenemos en cuenta que por ser el mayor valor propio, todos los cocientes λ_1/λ_i serán menores que 1, y para *m* suficientemente elevado todos los términos del sumatorio serán despreciables frente al primero, y se podrá escribir:

$$\mathbf{y}^m=\sum_{i=1}^{n}\alpha_i\lambda_i^m\mathbf{x}^i=\lambda_1^m\left(\sum_{i=1}^{n}\alpha_i\frac{\lambda_i^m}{\lambda_1^m}\mathbf{x}^i\right)\approx\alpha_1\lambda_1^m\mathbf{x}^1 \tag{iv}$$

Para la siguiente la iteración:

$$\mathbf{y}^{m+1}\approx\alpha_1\lambda_1^{m+1}\mathbf{x}^1 \tag{v}$$

de donde se deduce que el vector $\mathbf{y}^m$ tiende realmente al vector propio buscado. Por otra parte, dividiendo miembro a miembro los módulos de las ecuaciones (v) y (iv) se obtiene:

$$\lambda_1=\frac{\left|\mathbf{y}^{m+1}\right|}{\left|\mathbf{y}^{m}\right|}$$

En esta demostración se ha supuesto que el coeficiente α_1 era distinto de cero. Eso es lo más probable si el vector inicial se ha escogido de modo arbitrario, pero aunque no fuera así el método también funciona, aunque la demostración nos llevaría un poco lejos...

Solución comentada del Ejercicio 8.12.

```
/* fichero maxvalp.c */
/* cálculo del máximo valor propio por itaración directa */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void main(void) {
    int i, j, nit=0, orden;
    double vpn, vpv;
    double **a, *vectn, *vectv;

    double **maAlloc(int, int);
    double vabs(double);
    void normalizar(double*, int);
    void matvec(double**, double*, double*, int);

    printf ("Introduce el orden de la matriz: ");
    scanf("%d", &orden);
    a=maAlloc(orden, orden);
    vectv=malloc(orden*sizeof(double));
    vectn=malloc(orden*sizeof(double));

    printf("\nIntroduce los datos de la matriz:\n");
    for (i=0; i<orden; i++)
        for (j=0;j<orden;j++) {
            printf("Elemento (%d,%d): ", (i+1), (j+1));
            scanf("%lf", &a[i][j]);
        }
    for (i=0; i<orden; i++)   /* aproximación inicial */
        vectn[i]=1.0/sqrt(orden);
    vpn=1;
    do {
        vpv=vpn;
        for (j=0; j<orden; j++)
            vectv[j]=vectn[j];
        matvec(a, vectv, vectn, orden);
        vpn=0.0;
        for (j=0; j<orden; j++)
            vpn+=vectn[j]*vectn[j];
        vpn=sqrt(vpn);
        for (j=0; j<orden; j++)
            vectn[j]=vectn[j]/vpn;
        nit++;
    } while (vabs((vpn-vpv)/vpn)>1e-12);

    printf("\nEl vector buscado es el:\n(");
    for (i=0; i<orden; i++)
        printf("%lf ", vectn[i]);
    printf("\b)");
    printf("\nEl valor propio es: %lf", vpn);
    printf("\nNumero de iteraciones: %d\n", nit);
}
```

```
double **maAlloc(int filas, int columnas) {
    int i;
    double **a;
    a=malloc(filas*sizeof(double *));
    for (i=0; i<filas; i++)
        a[i]=malloc(columnas*sizeof(double));
    return a;
}

void matvec(double **matriz, double *vector, double *vector2, int orden) {
    int i,j;
    double sum;
    for (i=0; i<orden; i++) {
        sum=0;
        for (j=0; j<orden; j++)
            sum+=matriz[i][j]*vector[j];
        vector2[i]=sum;
    }
}

double vabs(double a) {
    if (a<0.0)
        return (-a);
    else
        return a;
}
```

Comentario: Este programa calcula iterativamente el mayor valor propio de una matriz. Para eso, mientras no se cumpla la condición de error de la fórmula (7), continúa calculando nuevos valores con más iteraciones (bucle ***do while***). Las funciones que utiliza son las siguientes: ***maAlloc()*** para reservar dinámicamente memoria para una matriz, ***vabs()*** para calcular el valor absoluto y ***matvect()*** para multiplicar una matriz por un vector.

La iteración mantiene la nueva y la antigua aproximación del valor y del vector propio. Los nuevos valores se convierten en los antiguos al comenzar una nueva iteración. La variable ***nit*** cuenta el número de iteraciones, valor que es impreso al final del programa.