



Escuela Politécnica Superior de Elche

# **FUNDAMENTOS DE INFORMÁTICA**

## **1º Ingeniería Industrial**

---

**PRÁCTICA 4:**  
**Vectores, matrices y cadenas de caracteres**

---

**CURSO 2007-2008**

---

**División de Ingeniería de Sistemas y Automática**

---

## Ejemplos resueltos

**EJERCICIO 1:** Escribir un programa que lea 10 enteros, almacenándolos en un vector. A continuación, debe mostrar los valores introducidos, calcular el máximo y mostrarlo por pantalla:

```
#include <stdio.h>

// El numero de elementos a leer se define como una constante.
// De este modo, basta con cambiar el valor de TAM para que cambie
// el número de elementos con que se trabaja, no siendo necesario
// modificar el resto del programa.

#define TAM 10

void main(void)
{
    int vector[TAM];
    int i, maximo;

    // Pedimos al usuario que introduzca los valores y los guardamos en el vector:
    for(i=0; i<TAM; i++)
    {
        printf("Introduzca valor %d: ", i+1);
        scanf("%d", &vector[i]);
    }

    // Mostramos los valores por pantalla y calculamos el maximo:
    maximo = vector[0];
    printf("\nLos valores introducidos son:\n");
    for(i=0; i<TAM; i++)
    {
        printf("Valor %d = %d\n", i+1, vector[i]);
        if(vector[i]>maximo)
            maximo = vector[i];
    }

    // Mostramos el maximo:
    printf("\nEl valor maximo es %d\n\n", maximo);
    system("PAUSE");
}
```

**EJERCICIO 2:** Repetir el ejercicio anterior, pero programando una función llamada *LeerVector* para pedir al usuario que introduzca los elementos, una función llamada *MostrarVector*, y una función llamada *CalculaMaximo*. El programa debe trabajar con 5 elementos

```
#include <stdio.h>

#define TAM 5

void LeerVector(int vector[]);
void MostrarVector(int vector[]);
int CalculaMaximo(int vector[]);

void main(void)
{
    int vector[TAM], maximo;

    LeerVector(vector);
    MostrarVector(vector);
    maximo = CalculaMaximo(vector);
    printf("\nEl maximo es %d\n\n", maximo);

    system("PAUSE");
}

void LeerVector(int vector[])
{
    int i;

    for(i=0; i<TAM; i++)
    {
        printf("Introduzca valor %d: ", i+1);
        scanf("%d", &vector[i]);
    }
}

void MostrarVector(int vector[])
{
    int i;

    printf("\nLos valores introducidos son:\n");
    for(i=0; i<TAM; i++)
        printf("Valor %d = %d\n", i+1, vector[i]);
}

int CalculaMaximo(int vector[])
{
    int result, i;

    result = vector[0];
    for(i=0; i<TAM; i++)
        if(vector[i]>result)
            result = vector[i];

    return(result);
}
```

**NOTA:**

En el anterior programa, cabe destacar:

- En la definición de una función que recibe como parámetro un vector, es necesario poner corchetes después del nombre del vector. La dimensión puede omitirse. Por ejemplo, el prototipo de la función `CalculaMaximo` hubiera sido igualmente válido de la forma:
  - o `int CalculaMaximo(int vector[TAM])`
- En la llamada a una función que recibe como parámetro un vector, únicamente se debe indicar el nombre del vector. (Sin corchetes).
- El paso de un vector a una función se realiza por dirección, lo cual significa que se van a modificar las celdas de memoria correspondientes al vector original (al contrario de lo que sucedía con los tipos de variables simples). Si algún elemento del vector es modificado dentro de la función, esta modificación es reconocida también por `main` y por el resto de funciones. Por este motivo, la función `LeerVector` funciona correctamente como está programada, almacenando los valores introducidos en la zona de memoria correspondiente al vector original.

**EJERCICIO 3:** Escribir un programa que lea una secuencia de números reales, almacenándolos en un vector. Después de leer los números, se debe mostrar el valor medio, los valores introducidos y la desviación de cada valor respecto a la media (desviación = valor – media).

Antes de empezar a pedir los valores, se debe preguntar al usuario la cantidad de números que va a introducir.

```
#include <stdio.h>

void main(void)
{
    float lista[100], media=0, desv[100];
    int n, i;

    printf("Cuantos numeros se van a introducir? (maximo 100): ");
    scanf("%d", &n);
    while(n>100 || n<0)
    {
        printf("El numero debe positivo y menor a 100\n");
        printf("Cuantos numeros se van a introducir? (maximo 100): ");
        scanf("%d", &n);
    }

    for(i=0; i<n; i++)
    {
        printf("Introduzca valor %d: ", i+1);
        scanf("%f", &lista[i]);
        media += lista[i];
    }
    media /= n;
    printf("\nLa media es %.4f\n", media);

    for(i=0; i<n; i++)
    {
        desv[i] = lista[i] - media;
        printf("Valor %d = %.4f, Desviacion = %.4f\n", i+1, lista[i], desv[i]);
    }

    system("PAUSE");
}
```

**NOTA:**

Cuando no conocemos de antemano el número de componentes que va a tener el vector (depende del usuario), debemos declarar un vector lo suficientemente grande (en este caso, de 100 componentes). Asimismo, debemos asegurarnos que la cantidad de valores que le permitimos introducir al usuario es menor o igual a 100 (lo hacemos con un bucle while), si no, el programa fallaría al intentar guardar mas valores que el número de posiciones que hemos reservado.

Notese que en este caso, todos los bucles corren desde  $i = 0$  hasta  $i = (n-1)$ , que son las posiciones del vector que realmente van a contener valores útiles.

**EJERCICIO 4:** Repetir el programa anterior creando una función para leer el vector, otra para calcular la media y una tercera para calcular la desviación.

```
#include <stdio.h>

void LeerVector(float vector[], int dim);
float CalculaMedia(float vector[], int dim);
void CalculaDesviacion(float vector1[], float vector2[], float media, int dim);

void main(void)
{
    float lista[100], media, desv[100];
    int n, i;

    printf("Cuantos numeros se van a introducir? (maximo 100): ");
    scanf("%d", &n);
    while(n>100 || n<0)
    {
        printf("El numero debe positivo y menor a 100\n");
        printf("Cuantos numeros se van a introducir? (maximo 100): ");
        scanf("%d", &n);
    }

    LeerVector(lista, n);
    media = CalculaMedia(lista, n);
    printf("\nLa media es %.4f\n", media);

    CalculaDesviacion(lista, desv, media, n);

    for(i=0; i<n; i++)
        printf("Valor %d = %.4f, Desviacion = %.4f\n", i+1, lista[i], desv[i]);

    system("PAUSE");
}

void LeerVector(float vector[], int dim)
{
    int i;

    for(i=0; i<dim; i++)
    {
        printf("Introduzca valor %d: ", i+1);
        scanf("%f", &vector[i]);
    }
}
```

```
float CalculaMedia(float vector[], int dim)
{
    float result=0;
    int i;

    for(i=0; i<dim; i++)
        result += vector[i];

    result /= dim;

    return(result);
}

void CalculaDesviacion(float vector1[], float vector2[], float media, int dim)
{
    int i;

    for(i=0; i<dim; i++)
        vector2[i] = vector1[i] - media;
}
```

**NOTA:**

- Cuando no conocemos de antemano el número de componentes que va a tener el vector (depende del usuario), y trabajamos con funciones, dicho número de componentes se debe pasar como parámetro para que las funciones conozcan con cuántos valores deben trabajar.
- La función *CalculaDesviacion* nos debe devolver varios valores numéricos (desviación de cada elemento). En el tema de funciones se vio que una función sólo puede devolver un valor con *return*. Sin embargo, puede aprovecharse el hecho de que los vectores se pasan por dirección para calcular varios valores y almacenarlos en el vector que se pasa como parámetro, puesto que *main* reconocerá también estos valores. *CalculaDesviacion* calcula las desviaciones y las almacena en *vector2*, de forma que *main* podrá acceder a estos parámetros a través del vector *desv*. La función no devuelve nada con *return* (tipo devuelto = *void*) pero los valores calculados se encuentran en *vector2*.

**EJERCICIO 5:** Escribir un programa que pida al usuario que introduzca los componentes de dos matrices, las sume y muestre el resultado obtenido. Previamente, se debe preguntar al usuario las dimensiones de las matrices con que quiere trabajar.

```
#include <stdio.h>

void main(void)
{
    float a[100][100], b[100][100], c[100][100];
    int filas, columnas, i, j;

    printf("Introduzca numero de filas: ");
    scanf("%d", &filas);
    printf("Introduzca numero de columnas: ");
    scanf("%d", &columnas);

    printf("\nIntroduzca la matriz 1:\n");
    for(i=0; i<filas; i++)
    {
        for(j=0; j<columnas; j++)
        {
            printf("Valor[%d][%d]: ", i+1, j+1);
            scanf("%f", &a[i][j]);
        }
    }

    printf("\nIntroduzca la matriz 2:\n");
    for(i=0; i<filas; i++)
    {
        for(j=0; j<columnas; j++)
        {
            printf("Valor[%d][%d]: ", i+1, j+1);
            scanf("%f", &b[i][j]);
        }
    }

    printf("\nLa suma es: \n");
    for(i=0; i<filas; i++)
    {
        for(j=0; j<columnas; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
            printf("%6.2f ", c[i][j]);
        }
        printf("\n");
    }

    system("PAUSE");
}
```

**NOTA:**

- De nuevo, como no conocemos de antemano la dimensión de las matrices, es necesario declararlas con unas dimensiones lo suficientemente grandes. Aunque no se ha hecho, como en el caso anterior, sería necesario comprobar que las dimensiones introducidas no exceden el valor de 100 en este caso.



**EJERCICIO 6:** Repetir el ejercicio anterior usando una función para leer las matrices, otra para mostrarlas y otra para realizar la suma:

```
#include <stdio.h>

void LeerMatriz(float matriz[][100], int f, int c);
void MostrarMatriz(float matriz[][100], int f, int c);
void SumarMatrices(float matriz1[][100], float matriz2[][100], float matriz3[][100], int f, int c);

void main(void)
{
    float a[100][100], b[100][100], c[100][100];
    int filas, columnas;

    printf("Introduzca numero de filas: ");
    scanf("%d", &filas);
    printf("Introduzca numero de columnas: ");
    scanf("%d", &columnas);

    printf("\nIntroduzca la matriz 1:\n");
    LeerMatriz(a, filas, columnas);
    printf("\nIntroduzca la matriz 2:\n");
    LeerMatriz(b, filas, columnas);
    SumarMatrices(a, b, c, filas, columnas);
    printf("\nLa suma es: \n");
    MostrarMatriz(c, filas, columnas);

    system("PAUSE");
}

void LeerMatriz(float matriz[][100], int f, int c)
{
    int i, j;

    for(i=0; i<f; i++)
    {
        for(j=0; j<c; j++)
        {
            printf("Valor[%d][%d]: ", i+1, j+1);
            scanf("%f", &matriz[i][j]);
        }
    }
}

void MostrarMatriz(float matriz[][100], int f, int c)
{
    int i, j;

    for(i=0; i<f; i++)
    {
        for(j=0; j<c; j++)
            printf("%6.2f ", matriz[i][j]);
        printf("\n");
    }
}
```

```
void SumarMatrices(float matriz1[][100], float matriz2[][100], float matriz3[][100], int f, int c)
{
    int i, j;

    for(i=0; i<f; i++)
        for(j=0; j<c; j++)
            matriz3[i][j] = matriz1[i][j] + matriz2[i][j];
}
```

**NOTA:**

- En la definición de una función que toma como parámetro una matriz, se debe indicar una matriz con un par de corchetes detrás. La primera dimensión se puede omitir, pero no la segunda.
- La función *SumarMatrices* vuelve a aprovechar el paso por dirección de los vectores. Los resultados calculados en *matriz3* son vistos por la función *main*, puesto que estarán en la matriz *c*. Dicho de otro modo, al llamar a la función *SumarMatrices*, *matriz3* y *c* son exactamente la misma zona de memoria.

**EJERCICIO 7:** Escribir un programa que pida dos cadenas de caracteres al usuario y a continuación muestre estas cadenas, diga su longitud e indique si son iguales o no. Para ello, se debe programar una función llamada *CalculaLongitud* y otra llamada *CadenasIguales*:

```
#include <stdio.h>

short CalculaLongitud(char cad[]);
short CadenasIguales(char cad1[], char cad2[]);

void main(void)
{
    char cadena1[100], cadena2[100];
    short long1, long2, igual;

    printf("Introduzca la primera cadena de caracteres: \n");
    fflush(stdin);
    gets(cadena1);
    printf("Introduzca la segunda cadena de caracteres: \n");
    fflush(stdin);
    gets(cadena2);

    printf("\nLa primera cadena es:");
    puts(cadena1);
    long1 = CalculaLongitud(cadena1);
    printf("Su longitud es: %d\n", long1);

    printf("\nLa segunda cadena es:");
    puts(cadena2);
    long2 = CalculaLongitud(cadena2);
    printf("Su longitud es: %d\n", long2);

    igual = CadenasIguales(cadena1, cadena2);
    if(igual == 1)
        printf("\nLas cadenas son iguales\n");
    else
        printf("\nLas cadenas son diferentes\n");

    system("PAUSE");
}

short CalculaLongitud(char cad[])
{
    short i=0;

    while(cad[i] != '\0')
        i++;

    return(i);
}
```

```
short CadenasIguales(char cad1[], char cad2[])
{
    short i;

    if(CalculaLongitud(cad1) != CalculaLongitud(cad2))
        return(0);

    for(i=0; cad1[i] != 0; i++)
        if(cad1[i] != cad2[i])
            return(0);

    return(1);
}
```

### **NOTA:**

- Como no se conoce de antemano la longitud de las cadenas que va a introducir el usuario, se definen dos *strings* (vectores de caracteres) de una longitud lo suficientemente grande (en este caso, 100).
  - Para la lectura de cadenas desde teclado se recomienda utilizar la función *gets*, tal cual aparece en el ejemplo. Otra opción, hubiera sido utilizar la función *scanf* en una de las dos siguientes formas:
    - o *scanf("%s", &cadena1[0]);*
    - o *scanf("%s", cadena1);*
- Sin embargo, la función *scanf* no permite introducir cadenas que contengan espacios en blanco. En este caso, se debe utilizar obligatoriamente *gets*. Tanto cuando se utilice *gets* como cuando se utilice *scanf* para leer una cadena, se debe limpiar antes el buffer de entradas mediante la función *fflush(stdin)*.
- Para mostrar una cadena por pantalla se ha utilizado la función *puts*. Se podría haber utilizado *printf*, con el mismo resultado, teniendo en cuenta que el indicador de *strings* es *%s*:
    - o *printf("%s\n", cadena1);*
  - Cuando se tenga que recorrer una cadena para realizar alguna comprobación sobre ella, hay que tener en cuenta que la condición de finalización del bucle es que el carácter actual sea `'\0'`, dado que todas las cadenas finalizan con este carácter especial.
  - Por último, cabe tener en cuenta que un *string* no es más que un *array* de caracteres (finalizado por `'\0'`), por lo tanto, sigue las mismas reglas que los vectores en el paso a funciones (formato de la definición y formato de la llamada).

**EJERCICIO 8:** Escribir un programa que pida cadenas de caracteres hasta que se introduzca la cadena 'salir'. Para cada cadena, se debe contar el número de veces que aparece la letra 'a' (tanto en mayúscula como en minúscula), se deben convertir todas las letras a mayúscula y se debe mostrar el resultado. A continuación se debe pedir al usuario una nueva cadena, realizar las mismas operaciones y así sucesivamente hasta que se introduzca la cadena 'salir'.

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

void ConvierteMayusculas(char cad[]);
int CuentaA(char cad[]);

void main(void)
{
    char cadena[100];
    int numA;

    while(1)
    {
        printf("\nIntroduzca una cadena de caracteres\n");
        fflush(stdin);
        gets(cadena);
        if(strcmp(cadena, "salir") == 0)
            break;

        numA = CuentaA(cadena);
        printf("La letra A aparece %d veces\n", numA);
        ConvierteMayusculas(cadena);
        printf("La cadena convertida es: %s\n", cadena);
    }

    system("PAUSE");
}

void ConvierteMayusculas(char cad[])
{
    int i=0;

    while(cad[i] != '\0')
    {
        if((cad[i] >= 'a') && (cad[i] <= 'z'))
            cad[i]-=32;
        i++;
    }
}

int CuentaA(char cad[])
{
    int resultado = 0, i;

    for(i=0; cad[i] != '\0'; i++)
        if((cad[i] == 'a') || (cad[i] == 'A'))
            resultado++;

    return(resultado);
}
```

**NOTA:**

- La función *ConvierteMayusculas* aprovecha el hecho de que los *arrays* se pasan por dirección para almacenar el resultado sobre la misma cadena de entrada, dado que este resultado será visible también por *main*, y por tanto, cuando saquemos la cadena por pantalla en *main* aparecerá la cadena modificada (en mayúsculas).
- Para conocer si la cadena introducida es "salir" (para saber si se debe finalizar el programa) se podría haber utilizado la función *CadenasIguales* del ejercicio anterior. Sin embargo, se ha utilizado la función *strcmp*, que ya está definida en la librería *string.h* (por eso se ha tenido que agregar una referencia a esta librería al principio del programa). Al final de la práctica se aporta información sobre otras funciones de librerías que se pueden usar en los programas.

## Ejercicios Propuestos

### EJERCICIO 1

Escribir un programa que realice cálculos con vectores. El programa debe pedir en primer lugar la dimensión con la que se va a trabajar y a continuación se pedirán dos vectores. Con estos datos, el programa debe calcular y mostrar por pantalla la suma, el producto escalar y el módulo de los dos vectores. Los vectores pueden tener 10 componentes como máximo.

A continuación se muestra un ejemplo de funcionamiento. Los datos introducidos por el usuario aparecen en **negrita**. Todos los resultados se deben mostrar con dos decimales de precisión.

```
Introduzca dimension: 3
Introduzca los elementos del vector 1:
  Elemento 1: 2
  Elemento 2: 3.2
  Elemento 3: -4.5
Introduzca los elementos del vector 2:
  Elemento 1: 5.1
  Elemento 2: 2.25
  Elemento 3: 0.8

El vector suma es: (7.10, 5.45, -3.70)
El producto escalar es: 13.80
El modulo del vector 1 es: 5.87 y el del vector 2 es: 5.63
```

Para resolver el problema, se deben crear las siguientes funciones:

- **LeerVector**: Recibe como parámetro el vector donde almacenar los datos y la dimensión. Se encarga de pedir los elementos del vector y almacenarlos en las posiciones correspondientes.
- **CalculaSuma**: Toma como parámetros tres vectores y su tamaño. Calcula la suma de los dos primeros y la almacena en el tercero.
- **CalculaProducto**: Toma como parámetros dos vectores y su tamaño. Devuelve como resultado el producto escalar de los vectores.
- **CalculaModulo**: Toma como parámetro un vector y su tamaño. Devuelve como resultado el módulo del vector.

Por último, se debe crear una función **main** desde la que se pida la dimensión al usuario, se llame a las funciones necesarias y finalmente se muestren los resultados. Todos los resultados se mostrarán en **main**, no en las funciones anteriores.

#### Ficheros a entregar:

Se entregará el fichero fuente impreso y comentado (*ejercicio1.c*) y el fichero ejecutable con el nombre *ejercicio1.exe* en formato electrónico.

## EJERCICIO 2

### **Objetivo:**

*Estudiar la utilización de bucles de control para la captura de datos, el uso de vectores para almacenar información y la definición de funciones para estructurar un programa.*

### **Resultado que debe obtenerse:**

Programa que lee números reales (con un máximo de 50) por el teclado hasta que se teclee la palabra "salir" en mayúsculas o minúsculas. Se deberá controlar si se leen números y no otro tipo de entrada (caracteres no numéricos o secuencias de caracteres que no tengan sentido como números reales). Si se introduce alguna entrada que no sea un número correcto, el programa deberá ignorar esa entrada y pasar a leer la siguiente.

Después de leer todos los números, se mostrará por pantalla la siguiente información:

- Cantidad total de entradas
- Cantidad de entradas válidas
- Lista de valores válidos
- Valor promedio de los valores válidos
- Varianza (de los valores válidos)
- Desviación típica (de los valores válidos)

Todos los datos reales se imprimirán con 2 cifras decimales.

Veamos dos ejemplos de ejecución de este programa (en negrita aparecen los datos introducidos por el usuario):

### Ejemplo 1:

```
Introducir los numeros (escribir salir para terminar)
8
10
1.5
3
salir

Cantidad total de entradas: 5
Cantidad de entradas validas: 4
Lista de valores validos:
8.00 10.00 1.50 3.00
Valor promedio de los valores introducidos: 5.63
Varianza: 12.17
Desviacion tipica: 3.49
```



Ejemplo 2:

```
Introducir los numeros (escribir salir para terminar)
```

```
7
```

```
5
```

```
pepe
```

```
4.5
```

```
juan
```

```
8
```

```
9.5
```

```
salir
```

```
Cantidad total de entradas: 8
```

```
Cantidad de entradas validas: 5
```

```
Lista de valores validos:
```

```
7.00 5.00 4.50 8.00 9.50
```

```
Valor promedio de los valores introducidos: 6.80
```

```
Varianza: 3.46
```

```
Desviacion tipica: 1.86
```

**Ficheros a entregar:**

Se entregará el fichero fuente impreso y comentado (*ejercicio2.c*) y el fichero ejecutable con el nombre *ejercicio2.exe* en formato electrónico.

**Orientaciones:**

Se sugieren las siguientes tareas para realizar el programa anterior, con carácter orientativo:

1. Crear un proyecto en Dev C++ según los pasos descritos en prácticas anteriores. El proyecto deberá llamarse ejercicio2.
2. Crear un archivo de código C dentro del proyecto que implemente la función **main()**, y otras dos funciones adicionales, llamadas **Media()** y **Varianza()**.
3. Puesto que el usuario puede introducir tanto números como secuencias de caracteres, todos los datos que se vayan introduciendo por teclado se almacenarán en un string (array de caracteres), y posteriormente se convertirán a números.
4. Declarar una variable de tipo array de double para almacenar los números introducidos por teclado (hasta el máximo permitido) y una cadena de caracteres para almacenar la lectura del teclado, con el tamaño adecuado.
5. Crear un bucle de control principal que lea cadenas por teclado hasta que se teclee **“salir”** o **“SALIR”** o se haya superado el número máximo.
6. Leer la cadena con la función `scanf()` (utilizar el formato %s) o con `gets()`.
7. Utilizar la función `strcmp()` para comparar cadenas.
8. Utilizar la función `atof()` para convertirla a un número.
9. Definir las funciones que calculan la media y la varianza de un array de valores especificado como parámetro. Utilizar los siguientes prototipos:
  - La función *Media* tomará como parámetros el vector con los valores y un entero que indicará el tamaño del vector:  
$$\text{double Media}(\text{double valor}[], \text{int num});$$
  - La función *Varianza* tomará como parámetros el vector con los valores, un entero que indica el tamaño del vector y la media:  
$$\text{double Varianza}(\text{double valor}[], \text{int num}, \text{double media});$$
10. Llamar a dichas funciones desde `main()` con los parámetros adecuados presentando en pantalla la lista de valores, la media, la varianza y la desviación típica.
11. En la siguiente tabla se muestra la descripción de las funciones de librerías que pueden utilizarse para resolver este ejercicio:

Prototipo de la función	Descripción	Cabecera
double atof( const char *string );	Convierte una cadena a un número en coma flotante. Devuelve 0 si la entrada es incorrecta	stdlib.h
int strcmp( const char *string1, const char *string2 );	Compara dos cadenas (devuelve 0 si son iguales)	string.h
double sqrt( double x );	Calcula la raíz cuadrada del número que se le pasa como parámetro	math.h

Para más detalles sobre estas funciones, se puede consultar la página web <http://www.cplusplus.com/ref/>, librerías 'string.h' y 'math.h'.

12. Dados  $n$  números reales  $x_1, x_2, \dots, x_n$ , la media se calcula como:

$$\text{media} = \frac{\sum_{i=1}^n x_i}{n}$$

La varianza es:

$$\text{varianza} = \frac{\sum_{i=1}^n (\text{media} - x_i)^2}{n}$$

Y la desviación típica se puede calcular como la raíz cuadrada de la varianza.

### EJERCICIO 3

**Objetivo:**

Familiarizarse con el tratamiento de matrices y su paso como parámetro de una función.

**Resultado que debe obtenerse:**

Programa que pida las notas parciales de varios alumnos. Al finalizar, el programa debe mostrar las notas de todos los alumnos y la media de cada uno. Asimismo, se debe mostrar la máxima nota parcial introducida y la máxima nota media.

Ejemplo de ejecución:

En **grita** aparecen los datos introducidos por el usuario.

```
Introduce el numero de evaluaciones: 3
Introduce el numero de alumnos: 4
Introduce las notas del alumno 1:
  Nota 1: 5.3
  Nota 2: 6.8
  Nota 3: 7.4
Introduce las notas del alumno 2:
  Nota 1: 4.8
  Nota 2: 3
  Nota 3: 5.5
Introduce las notas del alumno 3:
  Nota 1: 10
  Nota 2: 9.8
  Nota 3: 9.5
Introduce las notas del alumno 4:
  Nota 1: 8.4
  Nota 2: 4.8
  Nota 3: 7.6

Las notas obtenidas son:
Alumno 1: 5.3, 6.8, 7.4   Nota final: 6.5
Alumno 2: 4.8, 3.0, 5.5   Nota final: 4.4
Alumno 3: 10.0, 9.8, 9.5   Nota final: 9.8
Alumno 4: 8.4, 4.8, 7.6   Nota final: 6.9

La maxima nota parcial es: 10.0
La maxima nota final es 9.8
```

**Ficheros a entregar:**

Se entregará el fichero fuente impreso y comentado (*ejercicio3.c*) y el fichero ejecutable con el nombre *ejercicio3.exe* en formato electrónico.

### **Orientaciones:**

Se sugieren las siguientes tareas para realizar el programa anterior:

1. *Crear un proyecto en Dev C++ según los pasos descritos en prácticas anteriores. El proyecto deberá llamarse ejercicio3.*
2. *Crear un archivo de código C dentro del proyecto que implemente la función **main()**.*
3. *Se asume que como máximo pueden introducirse 10 alumnos y como máximo habrá 4 evaluaciones. Las notas de los alumnos se guardarán en una matriz, cuya última columna deberá almacenar la nota media de cada alumno. Por lo tanto, en main, se definirá una matriz de tamaño 10x5.*
4. *Se debe definir una función llamada **IntroducirNotas** que tomará como parámetro la matriz, el número de alumnos y el número de evaluaciones. Utilizar el siguiente prototipo:*

*void IntroducirNotas(float matriz[10][5], int alum, int eval)*

*Esta función pedirá al usuario que introduzca las notas parciales y las guardará en la matriz notas.*

5. *Se debe definir una función llamada **CalculaMedia** que tomará como parámetro la matriz, el número de alumnos y el número de evaluaciones. Utilizar el siguiente prototipo:*

*void CalculaMedia(float matriz[10][5], int alum, int eval)*

*Esta función irá calculando la nota media de cada alumno y guardándola en la última columna de la matriz.*

6. *Se debe definir una función llamada **MostrarNotas** que tomará como parámetro la matriz, el número de alumnos y el número de evaluaciones. Utilizar el siguiente prototipo:*

*void MostrarNotas(float matriz[10][5], int alum, int eval)*

*Esta función mostrará las notas parciales y finales de los alumnos, tal y como aparece en el ejemplo de ejecución.*

7. *Se debe definir una función llamada **CalculaMaxParcial** que tomará como parámetro la matriz, el número de alumnos y el número de evaluaciones. Utilizar el siguiente prototipo:*

*float CalculaMaxParcial(float matriz[10][5], int alum, int eval)*

*Esta función calculará la máxima nota parcial que ha introducido el usuario y la devolverá como resultado.*

8. *Se debe definir una función llamada **CalculaMaxMedia** que tomará como parámetro la matriz, el número de alumnos y el número de evaluaciones. Utilizar el siguiente prototipo:*

*float CalculaMaxMedia(float matriz[10][5], int alum, int eval)*

*Esta función calculará la máxima nota final y la devolverá como resultado.*

## EJERCICIO 4

### **Objetivo:**

Estudiar algoritmos de tratamiento de cadenas de caracteres y el uso de funciones para estructurar un programa.

### **Resultado que debe obtenerse:**

Un programa que lea por teclado dos cadenas de caracteres e imprima la longitud de cada una, la concatenación de la primera a continuación de la segunda e indique si la segunda es una subcadena de la primera, y en caso afirmativo a partir de qué carácter.

A continuación se muestran dos ejemplos de la ejecución del programa (en negrita aparecen los datos tecleados por el usuario):

### Ejemplo 1:

```
Introduce la primera cadena:  
Fundamentos de Informatica  
Introduce la segunda cadena:  
Informatica  
  
Longitud de la cadena 1: 26  
Longitud de la cadena 2: 11  
  
Concatenacion de las cadenas:  
Fundamentos de InformaticaInformatica  
  
La cadena 2 es subcadena de la 1, desde el caracter 16
```

### Ejemplo 2:

```
Introduce la primera cadena:  
Teoria de Circuitos  
Introduce la segunda cadena:  
Informatica  
  
Longitud de la cadena 1: 19  
Longitud de la cadena 2: 11  
  
Concatenacion de las cadenas:  
Teoria de CircuitosInformatica  
  
La cadena 2 no es subcadena de la 1
```

### **Ficheros a entregar:**

Se entregará el fichero fuente impreso y comentado (*ejercicio4.c*) y el fichero ejecutable con el nombre *ejercicio4.exe* en formato electrónico.

**Tareas:**

1. Crear un proyecto en Dev C++ según los pasos descritos en prácticas anteriores. El proyecto deberá llamarse *ejercicio4*. Crear en el proyecto un fichero fuente con el nombre *ejercicio4.c*
2. Definir las siguientes funciones:
  - Una función llamada *LongCad* para calcular la longitud de una cadena, con el siguiente prototipo:  

```
int LongCad(char cad[]);
```
  - Una función llamada *Concatenar* que concatena dos cadenas en una tercera, con el siguiente prototipo:  

```
void Concatenar(char a[], char b[], char resultado[]);
```
  - Una función llamada *Buscar* que busque la cadena *a* en la cadena *b* y devuelva como resultado un 0 si *a* no es subcadena de *b*, y el carácter donde empieza la cadena *a* dentro de la cadena *b*, si *a* es subcadena de *b*. Usar el siguiente prototipo:  

```
int Buscar(char a[], char b[]);
```

### **MUY IMPORTANTE**

1. La práctica se considerará aprobada únicamente si funcionan correctamente los programas correspondientes a los ejercicios propuestos.
2. Las fechas de entrega son:
  - Grupo 1: Lunes 14 de enero de 2008.
  - Grupo 2: Martes 15 de enero de 2008.
  - Grupo 3: Miércoles 16 de enero de 2008.
  - Grupo 4: Jueves 17 de enero de 2008.No se admitirán prácticas fuera de estos plazos.
3. El profesor podrá convocar a cualquier alumno para responder preguntas relacionadas con la práctica entregada.
4. Todos los ficheros fuente se deben entregar impresos, comentados y con un estilo de programación correcto.
5. Además, se debe entregar un disquete con los ficheros fuente y los ficheros ejecutables correspondientes.



## ANEXO. Librerías de funciones.

En C, existen una serie de funciones ya programadas que realizan operaciones o cálculos que se usan comúnmente y que nosotros podemos utilizar en nuestros programas. Estas funciones están contenidas en varios archivos (denominados librerías, que tienen extensión *.h*) que acompañan a cada compilador de C.

Durante el proceso de convertir un programa fuente de C en programa objeto ejecutable, el programa fuente compilado se enlaza con las librerías para producir el programa ejecutable final. De este modo, el programa final se construirá a partir de varios archivos, aunque el fichero fuente original esté contenido en un único archivo. Para que funcione correctamente, se debe añadir al principio del programa con la instrucción *#include*, para que el compilador conozca donde están definidas las funciones que hemos utilizado en nuestro programa y que no hemos programado nosotros.

La mayoría de compiladores de C incluyen varias librerías, agrupadas en varias categorías básicas. A continuación, se muestran las funciones más utilizadas, junto con una breve descripción y la librería en la que se encuentran:

### **FUNCIONES MATEMÁTICAS:**

FUNCIÓN	TIPO DEVUELTO	DESCRIPCIÓN	CABECERA
abs(i)	int	Devuelve el valor absoluto de i.	stdlib.h
acos(d)	double	Devuelve el arcocoseno de d	math.h
asin(d)	double	Devuelve el arcoseno de d	math.h
atan(d)	double	Devuelve el arcotangente de d	math.h
cos(d)	double	Devuelve el coseno de d	math.h
exp(d)	double	Calcula e <sup>d</sup> .	math.h
log(d)	double	Devuelve el logaritmo natural de d	math.h
log10(d)	double	Devuelve el logaritmo decimal de d	math.h
pow(d1, d2)	double	Devuelve d1 elevado a d2.	math.h
sin(d)	double	Devuelve el seno de d	math.h
sqrt(d)	double	Devuelve la raíz cuadrada de d	math.h
tan(d)	double	Devuelve la tangente de d	math.h

### **FUNCIONES PARA TRABAJAR CON CADENAS DE CARACTERES:**

FUNCIÓN	TIPO DEVUELTO	DESCRIPCIÓN	CABECERA
atof(s)	double	Convierte la cadena s a un número en formato double	stdlib.h
atoi(s)	int	Convierte la cadena s a un número en formato entero	stdlib.h
strcmp(s1, s2)	int	Compara dos cadenas de caracteres. Devuelve 0 si son iguales.	string.h
strcpy(s1, s2)	char*	Copia la cadena de caracteres s2 en la cadena s1.	string.h
strlen(s)	int	Devuelve el número de caracteres de la cadena s	string.h