



## Computación II - Prácticas

Violeta González Pérez

2022

# Introducción a C++

C++ es un lenguaje de programación de alto nivel (los programas requieren ser compilados) orientado a objetos (manipulación de punteros de memoria).

- Los programas en C++ han de estar escritos en ficheros terminados en **.cpp** o **.cxx**.
- Una vez tenemos un programa escrito, por ejemplo **hola.cpp**, necesitamos compilarlo para generar un ejecutable, **hola**. Para ello, escribimos en la línea de comandos (indicada con el símbolo \$):

```
$ g++ -o hola hola.cpp
```

```
o
```

```
$ g++ hola.cpp -o hola
```

- Podemos ejecutar el programa desde el directorio en el que hemos creado su ejecutable:

```
$ ./hola
```

Los ejecutables pueden recibir cualquier nombre pero suelen recibir el mismo nombre que el programa, sin la terminación, o terminar en **.out** o **.exe**.

# Hola mundo

---

```
#include <iostream>

int main(){
    std::cout << "Hola mundo" << std::endl;
    return 0;
}
```

## Practica

Escribe el código anterior en el fichero 'hola.cpp', compílalo de las dos formas propuestas y ejecútalo ( ./hello). ¿Qué ocurre? ¿Encuentras diferencias entre las dos compilaciones?

## C++ distingue entre mayúsculas y minúsculas

---

```
#include <iostream>

Int main(){
    std::Cout << "Hola mundo" << std::endl;
    return 0;
}
```

### Practica

Compila y corre el programa anterior (lo puedes encontrar en Moodle como 'hola.cpp'), ¿qué ocurre? Lee lo que te indica el compilador, ¿dónde está el problema?

# Todos los programas en C++ tienen un bloque principal

```
int main() {  
    return 0;  
}
```

- El bloque principal se declara de forma parecida a las funciones pero tiene que estar declarado como un entero (`int`) y por lo tanto devuelve al sistema, a través del comando `return`, una variable del mismo tipo.
- Por convenio, el bloque principal retorna 0 cuando todo va bien y `-1` si ha habido problemas.
- En el bloque principal se puede obviar el `'return 0;'`.

## Practica

¿Qué ocurre si declaras tu bloque principal como `'void main() {...}'` y como `'float main() {...}'`?

# Analizando el programa 'Hola mundo'

- Comentarios: `//` en 1 línea o `/*`  
... `*/` en varias.
- `include` para incluir librerías normales de C++ (`<LIBRERIA>`) o programas que están en el ordenador  
(`"RUTA/AL/FICHERO.h"`)
- `iostream` Nos permite utilizar `std::cout` (standard cout) para escribir en pantalla.
- Cada línea termina con `;`.

```
/* Compila: g++ -o hola hola.cpp
   Ejecuta: ./hola
*/

#include <iostream> // Directiva para el preprocesador

int main(){ // Comienzo del bloque principal

    // Escribimos un mensaje en pantalla (std::cout) y
    // terminamos con una nueva línea (std::endl)
    std::cout << "Hola mundo" << std::endl;

    // Devolvemos 0 al sistema si todo ha ido bien y
    // -1 si hay errores, por convenio
    return 0;
}
```

## Practica

Si quitas `"<< std::endl"`, ¿qué diferencia encuentras al ejecutar el programa?

## namespace

Con namespace podemos indicarle al compilador dónde buscar nombres de comandos y así nos ahorramos especificar la librería al llamar al comando. También podemos especificar los nombres de los comandos a utilizar.

```
#include <iostream>

int main(){
    using std:: cout;
    using std:: endl;

    cout << "Hola mundo" << endl;
    return 0;
}
```

```
#include <iostream>

using namespace std;

int main(){
    cout << "Hola mundo" << endl;
    return 0;
}
```

### Practica

Compila y corre estos 2 programas, en los que se utiliza 'using' en distintas posiciones. ¿Encuentras alguna diferencia en la ejecución? ¿Qué sintaxis te parece más compacta?

# Variables

---

- En C++ es necesario declarar las variables con su tipo: `TIPO NOMBRE_VARIABLE;`
- Las variables también se pueden inicializar:  
`TIPO NOMBRE_VARIABLE = VALOR_INICIAL;`
- En C++ los nombres de las variables pueden ser alfanuméricos y contener `_`, pero no pueden empezar con un número. Tampoco se pueden utilizar nombres que coincidan con funciones intrínsecas a C++ (por ejemplo `int`).
- Se pueden declarar varias variables en una misma línea:  
`TIPO NOMBRE_VARIABLE1 = VALOR_INICIAL1, NOMBRE_VARIABLE2 = VALOR_INICIAL2;`
- Es recomendable declarar las variables lo más cerca de su primer uso en el programa.
- Se pueden definir variables como constantes que no pueden ser modificadas:  
`const TIPO NOMBRE_VARIABLE = VALOR_INICIAL`



# Declarando variables, los operadores <<, >> y el uso de cin

- Utilizamos el operador << para **escribir** en la terminal (con cout), en un fichero, etc.
- Utilizamos el operador >> para **leer** variables de la terminal (con cin), de un fichero, etc.

Practica con  
multiplicacion.cpp

Completa el programa y ejecútalo.  
¿Tiene sentido el resultado?

```
#include <iostream>
using namespace std;

int main(){
    cout << "Escribe un real:";
    float num1;
    cin >> num1;

    cout << "Escribe otro real:";
    //...
    //Completa el código
    //...

    cout << num1 << "*" << num2 << "=" << num1*num2 << endl;
    return 0;
}
```

# Tipos de variables

## Practica

Comprueba el número de bytes que ocupan las variables `int`, `float` y `double`, utilizando la función `sizeof(TIPO)`.

Nota: El número de bits en cada byte depende del ordenador en el que estés trabajando.

| Key word                          | Size in bytes | Interpretation   | Possible values  |
|-----------------------------------|---------------|--|--|
| bool                              | 1             | boolean  | true and false   |
| unsigned char                     | 1             | Unsigned character   | 0 to 255   |
| char (or signed char)             | 1             | Signed character   | -128 to 127  |
| wchar_t                           | 2             | Wide character (in windows, same as unsigned short)  | 0 to $2^{16}-1$  |
| short (or signed short)           | 2             | Signed integer   | $-2^{15}$ to $2^{15}-1$  |
| unsigned short                    | 2             | Unsigned short integer   | 0 to $2^{16}-1$  |
| int (or signed int)               | 4             | Signed integer   | $-2^{31}$ to $2^{31}-1$  |
| unsigned int                      | 4             | Unsigned integer   | 0 to $2^{32}-1$  |
| Long (or long int or signed long) | 4             | signed long integer  | $-2^{31}$ to $2^{31}-1$  |
| unsigned long                     | 4             | unsigned long integer  | 0 to $2^{32}-1$  |
| float                             | 4             | Signed single precision floating point (23 bits of significand, 8 bits of exponent, and 1 sign bit. )  | $3.4 \cdot 10^{-38}$ to $3.4 \cdot 10^{38}$ (both positive and negative)   |
| long long                         | 8             | Signed long long integer   | $-2^{63}$ to $2^{63}-1$  |
| unsigned long long                | 8             | Unsigned long long integer   | 0 to $2^{64}-1$  |
| double                            | 8             | Signed double precision floating point (52 bits of significand, 11 bits of exponent, and 1 sign bit. ) | $1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$ (both positive and negative) |
| long double                       | 8             | Signed double precision floating point (52 bits of significand, 11 bits of exponent, and 1 sign bit. ) | $1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$ (both positive and negative) |

Crédito: <https://www.go4expert.com/articles/cpp-inbuilt-data-types-t34623/>

## Especificando la precisión de un resultado

Podemos elegir la precisión con la que escribimos una variable, utilizando la librería `iomanip` (input-output manipulation) y especificando la precisión para `cout`, escribiendo el número máximo de decimales a escribir, `n`, en la función `setprecision()`:

```
#include <iomanip>
...
int main(){
    cout << setprecision(n);
...
    return 0;
}
```

### Practica con `multiplicacion.cpp`

Modifica tu programa con las multiplicaciones de reales para que el resultado aparezca con un máximo de 3 decimales. ¿Qué ocurre si pasas enteros?

# Cuidado con mezclar distintos tipos de variables

La mezcla de variables de distintos tipos puede dar lugar a resultados inesperados. En general, si declaras reales con float o double, utiliza un punto o punto y 0, para evitar problemas (por ejemplo double var=2.).

## Practica con div\_int.cpp

Ejecuta el programa para varios números impares. ¿Tiene sentido el resultado? ¿Cómo podrías obtener un resultado decimal?.

```
#include <iostream>
using namespace std;

int main(){
    // Pide que se escriba un número en la terminal
    cout << "Escribe un entero: ";

    // Declaración de variable para almacenar el entero
    int innum;

    // Guarda el número dado
    cin >> innum;

    // Escribe la división entre 2
    cout << innum << "/2=" << innum/2 << endl;

    return 0;
}
```

# Variables que son caracteres, uso de string

Los caracteres individuales pueden declararse como char.

Para nombres de ficheros o para leer líneas enteras de ficheros, es más práctico utilizar la librería string, que permite manipular texto.

## Practica con saludo\_string.cpp

Ejecuta el programa. ¿Qué hace .length()? ¿Qué ocurre si en lugar de la función getline lees con cin?. ¿Qué ocurre si declaras las variables como char?

Nota que lo que pasamos a cout puede ocupar varias líneas.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
    // Declaramos un saludo
    string saludo ("Buenas tardes ");

    // Pide el nombre
    cout << "Escribe tu nombre y apellidos: ";
    string nombre; // ¿Qué pasa si declaras char?
    getline(cin,nombre); //¿Qué pasa si utilizas cin >>?

    cout << saludo << nombre << endl;
    cout << "Tu nombre completo tiene "
         << nombre.length()
         << " caracteres, incluyendo espacios."<<endl;

    return 0;
}
```

# Variables que son vectores

En C++ las variables que son vectores pueden declararse indicando la longitud del vector (esta longitud ha de ser un entero).

- Para inicializar el vector con un mismo valor:

```
TIPO NOMBRE_VARIABLE [LONGITUD_VECTOR] = VALOR_INICIAL;
```

Ejemplo:

```
int veclen=2;  
double nums[veclen]={0.};
```

- Para inicializar el vector con distintos valores:

```
TIPO NOMBRE_VARIABLE [N] = VALOR_INICIAL1, VALOR_INICIAL2, ...,  
VALOR_INICIALN;
```

Ejemplo:

```
int len1=2, len2=3;  
int nums[len1][len2]={{0,1,2},{3,4,5}};
```

# Operadores

---

## Arithmetic Operator

| Operator | Description    |
|----------|----------------|
| *        | multiplication |
| /        | division       |
| %        | modulo         |
| +        | addition       |
| -        | subtraction    |

## Logical Operator

| Operator | Description |
|----------|-------------|
| !        | NOT         |
| &&       | AND         |
|          | OR          |

## Relational Operator

| Oerator | Description           |
|---------|-----------------------|
| <       | less than             |
| >       | greater than          |
| >=      | greater than or equal |
| ==      | equal to              |
| !=      | not equal             |

## Bitwise Operators

| Operator | Description      |
|----------|------------------|
| ~        | One's complement |
| <<       | Left shift       |
| >>       | Right shift      |
| &        | Bitwise AND      |
| ^        | Bitwise XOR      |

# Incrementos y decrementos

---

- Incrementar una variable, VAR, +1, simplemente escribiendo ++VAR o VAR++.
- Reducir una variable en -1: --VAR o VAR--
- ++VAR o --VAR: Incrementa la variable antes de usarla (esto es lo que te hará falta en general).
- VAR++ o VAR--: Primero utiliza la variable, luego la incrementa.

Incrementar o reducir variables es especialmente útil en bucles. En estos casos, solemos referirnos a la variable que controla el bucle como **contador** y por convenio solemos designarlos con las letras (o nombres de variables que comiencen con las letras): i, j, k, ...



# Iteraciones: bucles con for

Comando para salir de un bucle infinito: Ctrl-c

Sintaxis:

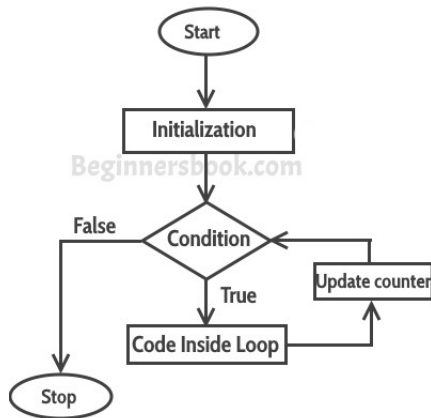
```
for(int COUNTER_VARIABLE=INITIAL_VALUE;  
CONDITION_FOR_COUNTER; COUNTER_CHANGE){  
...Cálculo iterativo... }
```

Ejemplo:

```
for (int i=0; i<nlen; ++i){  
    cout << "vect[" << i << "]= " << vect[i] << endl;  
}
```

## Practica

Escribe un programa en el que declares un vector de longitud 5, inicializado con valores distintos, y que escribas en pantalla el índice y el valor del vector correspondiente.



Crédito: <https://beginnersbook.com/>

## Iteraciones: bucles con while

---

Ejemplo:

```
int i = 0;
while (i<nlen){
    cout << "vect[" << i << "]= " << vect[i] << endl;
    ++i;
}
```

### Practica

Utiliza un bucle con `while` para escribir por pantalla el índice y los valores de un vector de longitud 5, que haya sido declarado e inicializado con valores distintos.

## Iteraciones: bucles con do .. while

En este caso, los comandos dentro del `do{}` se ejecutarán al menos una vez.

Ejemplo:

```
int j = 0;
do{
    cout << "vect[" << j << "]= " << vect[j] << endl;
    ++j;
}while(j<nlen);
```

### Practica

Utiliza un bucle con `while` para escribir por pantalla el índice y los valores de un vector de longitud 5, que haya sido declarado e inicializado con valores distintos.

## Condiciones: if ...else

Ejemplo:

```
if (num>2.){  
    cout << num << " es mayor que 2." << endl;}  
else if (num == 2.){  
    cout << num << " es igual a 2." << endl;}  
else {  
    cout << num << " es menor que 2." << endl;}  
}
```

### Practica

Escribe un programa, siguiendo el ejemplo de arriba, que nos indique si los valores de un vector inicializado por ti y de longitud 5 son menores, mayores o iguales a 2.

# Declarando y llamando funciones

## Practica con area.cpp

Ejecuta el programa. ¿Qué ocurre si incluyes un 'return 0' en la función area()? ¿Por qué?

Genera el programa de la derecha en base a lo que ya tienes. ¿En qué se diferencian los dos programas?

Comprueba en ambos casos si tienes acceso a pi desde main.

```
#include <iostream>
using namespace std;

// Declaramos la función
void area();

int main(){
    area();

    return 0;
}

// Definimos una función
void area(){
    // Definimos pi
    float pi = 22./7.;

    // Pide que se escriba un número en la terminal
    cout << "Escribe el radio:";
    float rad=0.;
    cin >> rad;

    // Escribe el área del círculo
    cout << "Área del círculo =" << pi*rad*rad << endl;
}
```

# Variables globales

---

El uso de variables globales, accesibles a todos los programas está desaconsejado porque puede dar lugar a errores difíciles de resolver.

Las **constantes**, sí que es aconsejable declararlas como constantes globales.

## Practica con `area.cpp`

Define  $\pi$  como una constante global y comprueba que tienes acceso a `pi` desde el programa principal, `main`.

# Funciones con parámetros

Los parámetros de funciones son variables locales. Una función puede tener múltiples parámetros.

## Practica con area.cpp

Modifica tu programa para obtener el programa de la derecha. Compila y ejecútalo.

Modifica tu nuevo programa para que  $\pi$  sea un parámetro en lugar de una constante global.

```
#include <iostream>
using namespace std;

const double pi = 22./7.;

double area_circle(double rad){
    double area = 0.;
    area = pi*rad*rad;
    return area;
}

int main(){
    // Pide un radio en la terminal
    cout << "Introduce un radio:";
    double rad=0.;
    cin >> rad;

    // LLama a la función que calcula el área
    double area = 0.;
    area = area_circle(rad);

    // Escribe el resultado
    cout << "Área del círculo = " << area << endl;

    return 0;
}
```

# Funciones con parámetros opcionales y valores por defecto

Los parámetros de funciones pueden ser opcionales. Este tipo de parámetros debe aparecer al final de la lista de parámetros y se definen con valores por defecto. Al llamar a la función no es necesario incluir estos parámetros opcionales.

## Practica con matrix.h c++ inverse matrix

Adapta tu programa a lo que aparece en la derecha.  
Modifica el código para llamar a la función definiendo un valor para

pi.



Universidad Autónoma  
de Madrid

```
#include <iostream>
using namespace std;

double area_circle(double rad, double pi=3.){
    double area = 0.;
    area = pi*rad*rad;
    return area;
}

int main(){
    // Pide un radio en la terminal
    cout <<"Introduce un radio: ";
    double rad = 0.;
    cin >> rad;

    // Llama a la función que calcula el área
    double area = 0.;
    area = area_circle(rad);

    // Escribe el resultado
    cout << "Área del círculo = " << area << endl;

    return 0;
}
```



# Vectores como argumentos de funciones

Los vectores pueden pasarse como argumento de las funciones, sin especificar su longitud:

TIPO VECTOR[.].

Existen librerías específicas para facilitar los cálculos con vectores (vector) y matrices.

## Practica con fvect.cpp

Ejecuta el programa.

Modifica el código añadiendo una función para un vector de caracteres.

```
#include <iostream>
using namespace std;

void elementos(const double vect[], const int len){
    // Escribe los elementos del vector
    for (int i=0; i<len; ++i){
        cout << "Elemento" << i << " = "
             << vect[i] << endl;
    }
}

int main(){
    // Declara un vector
    int len = 4;
    double vect[len] = {24.3, -56.2, 304, 0.1};

    /* Utiliza el vector como argumento
       para la función */
    elementos(vect, len);

    return 0;
}
```

# Funciones como argumentos

```
#include <iostream>
using namespace std;

double doble(double num){
    return num*2.;
}

double mitad(double num){
    return num/2.;
}

void resultado(const double num,
               double (*f)(double)){
    cout << "Resultado=" << f(num) << endl;
}

int main(){
    // Pide un número
    cout << "Real: ";
    double num = 0.;
    cin >> num;

    // Pide la operación
    cout << "Doble (1) o Mitad (0):";
    int op;
    cin >> op;

    // Realiza el cálculo pedido
    if (op == 0) {resultado(num,&mitad);}
    else if (op == 1) {resultado(num,&doble);}
    else{
        cout << "Introduce 1 o 0.";
        return 1;
    }
    return 0;
}
```

Se pueden pasar funciones como argumentos de funciones.

La función que tiene otra función como argumento estará declarada siguiendo este ejemplo:

```
double func (double (*f)(int));
```

En el programa principal, la función se pasa como **argumento por referencia**:

```
ff (arg, &func);
```

## Practica con farg.cpp

Utiliza la librería **cmath**, que da acceso a operaciones matemáticas comunes, y modifica el programa farg.cpp, para que el resultado sea la función sin del doble o la mitad del número dado.

## Funciones con varias salidas: argumentos por referencia

Las funciones sólo pueden dar un valor utilizando `return`. De modo, que necesitamos utilizar **argumentos por referencia** si queremos que una función de más de un valor de salida.

El signo **&** (et) nos permite declarar funciones con argumentos que pueden ser modificados y por lo tanto ser tanto una salida de la función. Por ejemplo:

```
void geo_circulo(const double rad, double& area, double& perimetro){  
    ...  
}
```

### Practica con `area.cpp`

Modifica el programa con el que calculabas el área de un círculo para que además calcule el perímetro del mismo, siguiendo el ejemplo de sintaxis de arriba, con argumentos por referencia. Define  $\pi$  utilizando `cmath`, como:

```
const double pi = atan(1)*4.;
```

## Práctica I

Realiza una función que obtenga el máximo, el mínimo y los índices correspondientes de los elementos de un vector.

## Práctica II

Programa la función,  $F$ , de funciones, que a cada función real le hace corresponder otra función real con la siguiente definición:

$$F(g)(x) = \begin{cases} 2.5, & g(x) < 1 \\ 2.0, & 1 \leq g(x) \leq 2 \\ 1.5, & 2 \leq g(x) \leq 3 \\ 1.0, & 3 < g(x) \end{cases}$$

, para funciones  $g(x)$  definidas en el intervalo  $x \in [0, 1]$ . Prueba la función anterior en un punto  $x \in [0, 1]$  con los siguientes ejemplos desde un programa principal:  $\sin(x)$ ,  $\exp(5x)$ ,  $\ln(0.01 + x)$ .

# Manipulando ficheros

---

La librería `fstream` permite generar y leer ficheros. Los ficheros hay que

1. abrirllos, `open()`,
  2. comprobar que están correctamente abiertos y
  3. cerrarlos, `close()`.
- Si sólo se va a **escribir** en un fichero, utiliza la librería `ofstream`.
  - Si sólo se va a **leer** un fichero, utiliza la librería `ifstream`.

# Generando ficheros

Si sólo queremos escribir en el fichero, podemos utilizar `ofstream`.

## Practica con `hola_fichero.cpp`

¿Qué hace la cláusula `if`? y ¿la partícula `\n`?

Utiliza el comando `more` para explorar el contenido del fichero que has generado, `mifichero.txt`. Modifica el código para definir el nombre del fichero como una variable tipo `string` (utiliza la librería `string`) y escribe en la terminal una frase que muestre el nombre del fichero.

```
#include <iostream> // para salida entrada por terminal
#include <fstream> // para salida entrada por ficheros.
using namespace std;

int main(){
    // Sólo vamos a escribir en el fichero
    ofstream ff ("mifichero.txt");

    if (ff.is_open()){
        // Escribimos sólo si está abierto correctamente
        ff << "Hola fichero \n" << endl;
        // Cerramos el fichero
        ff.close();
    }
    else cout << "No se ha podido abrir el fichero" << endl;

    return 0;
}
```

# Escribiendo variables en un fichero

## Practica con var\_fichero.cpp

Modifica el código para que:

1. Los números aparezcan en líneas distintas.
2. Define un vector en tu programa principal y escribe el vector en tu fichero de salida, como una columna.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(){
    // Pedimos 2 números enteros
    cout << "Escribe 2 enteros separados por un espacio: ";
    int num1=0, num2=0;
    cin >> num1 >> num2;

    // Definimos el fichero de salida y lo abrimos
    string infile = "enteros.txt";
    ofstream ff(infile);
    if (ff.is_open()){
        /* Normalmente se añade un encabezamiento
           indicando que contiene cada columna del fichero */
        ff << "# Enteros" << endl;

        // Escribimos los números en el fichero
        ff << num1;
        ff << num2;

        cout << "Fichero generado: " << infile << endl;
        ff.close();
    }
    else cout << "No se ha podido abrir el fichero "
              << infile << endl;

    return 0;
}
```

# Escribiendo una matriz en un fichero

## Practica con matriz2fichero.cpp

Primero completa el código y después modifícalo para que la separación entre elementos de una fila sea un tabulador utilizando `\t`.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(){
    // Define una matriz 2x3

    // Definimos el fichero de salida y lo abrimos
    string infile = "matriz.txt";
    ofstream ff(infile);
    if (ff.is_open()){
        ff << "# Matriz" << endl;

        // Escribe los elementos de las filas separados por un espacio
        // y cada fila en una nueva línea

        cout << "Fichero generado: " << infile << endl;
        ff.close();
    }
    else cout << "No se ha podido abrir el fichero "
               << infile << endl;

    return 0;
}
```



# Leyendo ficheros

## Practica con leer\_fichero.cpp

Compila y corre el programa. ¿Tiene sentido el resultado?

Modifica el programa para que lea el fichero que has generado con una matriz, *matriz.txt*.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(){
    // Definimos el nombre del fichero a leer
    string infile = "enteros.txt";
    ifstream ffin(infile);
    if (ffin.is_open()){
        // Leemos la línea de encabezado
        string header;
        getline(ffin,header);
        cout << "Encabezado: " << header << endl;

        // Leemos los enteros
        int num=0, ii=1;
        while (ffin >> num){
            ++ii;
            cout << "Número en línea " << ii << " = " << num << endl;
        }

        ffin.close();
    }
    else cout << "No se ha podido abrir el fichero "
              << infile << endl;

    return 0;
}
```

## Prácticas III y IV

### Práctica III

Realiza un programa que obtenga los siguientes sumatorio y producto:

$$\sum_{i=0}^8 x_i^2 ; \prod_{i=0}^8 e^{x_i}$$

, donde  $x_i$  son nueve números que leerá de un fichero.

### Práctica IV

Realiza un programa que lea números reales de un fichero y guarde en otro fichero la raíz cuadrada de aquellos que sean mayores que 1.

## Manipulando matrices con `matrix.h`

Escribir y leer matrices en C++ requiere de varios bucles y lo mismo ocurre para operar con ellas. Para facilitar la manipulación de matrices, vamos a utilizar la librería `matrix.h`.

Para utilizar esta librería, necesitáis:

1. Tener el fichero `matrix.h` en vuestro ordenador o PC virtual. Podéis descargar el fichero desde Moodle.
2. Incluir la librería en vuestro código. Para ello, si tenéis el fichero `matrix.h` en el mismo directorio que vuestro programa necesitáis incluir en vuestro programa las siguientes líneas (**en este orden exactamente**):

```
#include <string.h>
#include "matrix.h"
using namespace math;
```

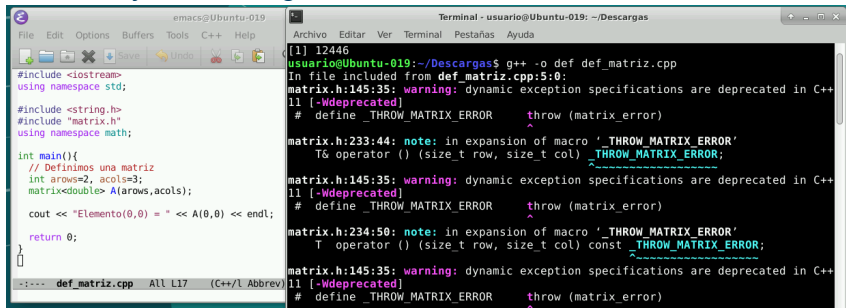
**En el caso de que el fichero `matrix.h` no esté en el mismo directorio que vuestro programa principal, basta con que especifiquéis la ruta al fichero:**

```
#include "/RUTA/AL/FICHERO/matrix.h"
```

# Compilando programas con `matrix.h`

Debido a que esta versión de la librería `matrix.h` no es la última, hay que tener en cuenta que

- Al compilar van a aparecer muchas advertencias. Veréis que, a pesar de las advertencias, el ejecutable se genera.



The image shows two windows from a Linux desktop. The left window is Emacs, titled 'emacs@Ubuntu-019', showing a C++ file named 'def\_matriz.cpp'. The code includes `<iostream>`, `<string>`, and `"matrix.h"`, and uses the `std` namespace. It defines a `main` function that creates a 2x3 matrix of doubles and prints the element at (0,0). The right window is a terminal titled 'Terminal - usuario@Ubuntu-019: ~/Descargas'. It shows the command `g++ -o def def_matriz.cpp` being executed. The output shows several warnings from `matrix.h` regarding deprecated dynamic exception specifications and deprecated macros like `_THROW_MATRIX_ERROR`. The warnings are highlighted in pink and green in the terminal output.

```
emacs@Ubuntu-019
File Edit Options Buffers Tools C++ Help
[Icons] Save Undo [Icons]
#include <iostream>
using namespace std;

#include <string>
#include "matrix.h"
using namespace math;

int main(){
    // Definimos una matriz
    int rows=2, acols=3;
    matrix<double> A(rows,acols);

    cout << "Elemento(0,0) = " << A(0,0) << endl;

    return 0;
}

-:--- def_matriz.cpp All L17 (C++/l Abbrev)

Terminal - usuario@Ubuntu-019: ~/Descargas
Archivo Editar Ver Terminal Pestañas Ayuda
[1] 12446
usuario@Ubuntu-019:~/Descargas$ g++ -o def def_matriz.cpp
In file included from def_matriz.cpp:5:0:
matrix.h:145:35: warning: dynamic exception specifications are deprecated in C++
11 [-Wdeprecated]
# define _THROW_MATRIX_ERROR throw (matrix_error)
matrix.h:233:44: note: in expansion of macro '_THROW_MATRIX_ERROR'
T& operator () (size_t row, size_t col) _THROW_MATRIX_ERROR;
matrix.h:145:35: warning: dynamic exception specifications are deprecated in C++
11 [-Wdeprecated]
# define _THROW_MATRIX_ERROR throw (matrix_error)
matrix.h:234:50: note: in expansion of macro '_THROW_MATRIX_ERROR'
T operator () (size_t row, size_t col) const _THROW_MATRIX_ERROR;
matrix.h:145:35: warning: dynamic exception specifications are deprecated in C++
11 [-Wdeprecated]
# define _THROW_MATRIX_ERROR throw (matrix_error)
```

- La documentación que podéis encontrar en la página de la librería `matrix.h` es algo distinta a lo que veremos aquí.

# Definiendo una matriz con matrix.h

## Practica con def\_matrix.cpp

¿Qué valores da por defecto la librería matrix.h a los elementos de una matriz sin especificar?

Modifica el código para tener una matriz 3\*3 y utiliza la función Unit() para inicializar tu matriz:

A.Unit()

¿Qué tipo de matriz obtienes? Y ¿con la función Null(), A.Null()?

```
#include <iostream>
using namespace std;

#include <string.h>
#include "matrix.h"
using namespace math;

int main(){
    // Definimos una matriz
    int arows=2, acols=3;
    matrix<double> A(arows,acols);

    cout << "Elemento(0,0) = " << A(0,0) << endl;

    return 0;
}
```

# Leyendo una matriz definida con matrix.h

## Practica con leer\_matrix.cpp

1. Modifica el código para ver por pantalla el número de fila y columnas y la matriz A.
2. Resta a todos los elementos de la matriz 2.
3. Escribe por pantalla la traspuesta de la nueva matriz, utilizando el operador  $\sim$  (AltGr-4),  $\sim A$ .
4. Escribe la nueva matriz traspuesta en un fichero de salida 'matrizT.dat'.

```
#include <fstream>
#include <string>
#include <iostream>
using namespace std;

#include <string.h>
#include "matrix.h"
using namespace math;

int main(){
    string infile = "matriz.dat";
    ifstream ff(infile);
    if (ff.is_open()){
        //Leemos el número de filas y columnas del fichero
        int arows=0, acols=0;
        ff >> arows >> acols;

        // Definimos y leemos la matriz
        matrix<int> A(arows,acols);
        ff >> A;

        ff.close();
    }else cout<<"No se ha podido abrir el fichero "<<infile<<endl;

    return 0;
}
```

# Una matriz definida con `matrix.h` como argumento

## Practica con `arg_matrix.cpp`

1. ¿Qué hace la función `elementosM`? ¿Qué indica el símbolo `&`?
2. En un fichero a parte, escribe una función que compruebe si una matriz es cuadrada. Te puedes ayudar de las funciones `RowNo()` y `ColNo()`.
3. Desde el programa principal, comprueba que la matriz `A` es cuadrada y que su determinante, `A.Det()`, es distinto de 0.
4. Si puedes, define `B`, como la inversa de `A`, `!A`. ¿Tiene sentido el resultado de multiplicarlas?

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

#include <string.h> // Línea 1/3 para incluir matrix.h
#include "matrix.h" // Línea 2/3 para incluir matrix.h
using namespace math; // Línea 3/3 para incluir matrix.h

void elementosM(matrix<double>& M){
    int nrows=0,ncols=0;
    nrows = M.RowNo(); // Función de matrix.h para contar filas
    ncols = M.ColNo(); // Función de matrix.h para contar columnas

    double ii = 0.;
    for (int i=0; i<nrows; ++i){
        for (int j=0; j<ncols; ++j){
            M(i,j) = ii;
            ++ii;
        }
    }
}

int main(){
    // Declaramos una matriz cuadrada
    int n=3;
    matrix<double> A(n,n);
    cout << "Matriz: \n" << A << endl;
    // Llamamos a la función
    elementosM(A);
    cout << "Matriz: \n" << A << endl;

    return 0;
}
```