

# Laboratorio de Programación - Labo03

## Funciones en C++ - Pasaje por copia y referencia

Algoritmos y Estructuras de Datos I

Departamento de Computación, FCEyN, Universidad de  
Buenos Aires.

## Recap: C++

Hasta ahora hemos visto:

- ▶ función `main` (punto de entrada)
- ▶ Librerías (Bibliotecas): `#include <...>`
- ▶ `cout`: salida por pantalla
- ▶ `cin`: entrada desde teclado
- ▶ Tipos de datos: `int`, `bool`, `char`, `float`
- ▶ Declaración, inicialización y asignación de variables
- ▶ Declaración de nuevas funciones
- ▶ Ejecución secuencial de sentencias (ordenes/instrucciones)
- ▶ Ejecución de formas alternativas de control

# Menú del día

- ▶ Manejo de datos
- ▶ Pasaje por valor y por referencia

## Resumen: Funciones en C++

- ▶ En la programación imperativa, las **funciones** son el mecanismo básico para hacer el código más legible y **reutilizar comportamiento**.

## Resumen: Funciones en C++

- ▶ En la programación imperativa, las **funciones** son el mecanismo básico para hacer el código más legible y **reutilizar comportamiento**.

La definición de una función tiene la siguiente forma:

# Resumen: Funciones en C++

- ▶ En la programación imperativa, las **funciones** son el mecanismo básico para hacer el código más legible y **reutilizar comportamiento**.

La definición de una función tiene la siguiente forma:

```
tipoRetorno nombreFuncion(tipoParam1 nombreParam1, ...) {  
    [cuerpo de la funcion]  
    return valor;  
}
```

# Resumen: Funciones en C++

- ▶ En la programación imperativa, las **funciones** son el mecanismo básico para hacer el código más legible y **reutilizar comportamiento**.

La definición de una función tiene la siguiente forma:

```
tipoRetorno nombreFuncion(tipoParam1 nombreParam1, ...) {  
    [cuerpo de la funcion]  
    return valor;  
}
```

- ▶ Por ejemplo, para obtener el máximo entre dos elementos lo ideal sería poder contar con una función **máximo** en lugar de tener que escribir siempre el mismo código.

# Funciones en C++

Por ejemplo...

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



## Llamando a las funciones

Para llamar a una función, basta simplemente con poner su nombre y pasarle los parámetros correspondientes. En caso de que la función retorne un valor, normalmente lo asignaremos a una variable.

## Llamando a las funciones

Para llamar a una función, basta simplemente con poner su nombre y pasarle los parámetros correspondientes. En caso de que la función retorne un valor, normalmente lo asignaremos a una variable. Por ejemplo:

```
int un_valor = 4;  
int otro_valor = 8;  
int el_maximo = maximo(un_valor, otro_valor);
```

## Llamando a las funciones

Para llamar a una función, basta simplemente con poner su nombre y pasarle los parámetros correspondientes. En caso de que la función retorne un valor, normalmente lo asignaremos a una variable. Por ejemplo:

```
int un_valor = 4;  
int otro_valor = 8;  
int el_maximo = maximo(un_valor, otro_valor);
```

### Cuidado con el orden

Dado que el compilador recorrerá el archivo de arriba hacia abajo, es necesario definir las funciones antes de utilizarlas.

## Llamando a las funciones

Para llamar a una función, basta simplemente con poner su nombre y pasarle los parámetros correspondientes. En caso de que la función retorne un valor, normalmente lo asignaremos a una variable. Por ejemplo:

```
int un_valor = 4;
int otro_valor = 8;
int el_maximo = maximo(un_valor, otro_valor);
```

### Cuidado con el orden

Dado que el compilador recorrerá el archivo de arriba hacia abajo, es necesario definir las funciones antes de utilizarlas.

Opcionalmente, podemos **declarar** la función antes de definir su implementación:

```
int maximo(int a, int b);
```

## Archivos y librerías

- ▶ Para poder compilar y ejecutar nuestro código, escribiremos el mismo en un **archivo de texto plano**. Por convención, en C++ estos archivos llevan la extensión **.cpp**.

# Archivos y librerías

- ▶ Para poder compilar y ejecutar nuestro código, escribiremos el mismo en un **archivo de texto plano**. Por convención, en C++ estos archivos llevan la extensión **.cpp**.
- ▶ Si nuestros programas crecen en líneas de código, incluir todo el código en un mismo archivo de texto resulta poco práctico.

# Archivos y librerías

- ▶ Para poder compilar y ejecutar nuestro código, escribiremos el mismo en un **archivo de texto plano**. Por convención, en C++ estos archivos llevan la extensión **.cpp**.
- ▶ Si nuestros programas crecen en líneas de código, incluir todo el código en un mismo archivo de texto resulta poco práctico.
- ▶ Además, en C++ se acostumbra a **separar las declaraciones** de funciones (de su implementación) en archivos **.h** (las **implementaciones** irán en un **.cpp** con el **mismo nombre**).

# Archivos y librerías

- ▶ Para poder compilar y ejecutar nuestro código, escribiremos el mismo en un **archivo de texto plano**. Por convención, en C++ estos archivos llevan la extensión **.cpp**.
- ▶ Si nuestros programas crecen en líneas de código, incluir todo el código en un mismo archivo de texto resulta poco práctico.
- ▶ Además, en C++ se acostumbra a **separar las declaraciones** de funciones (de su implementación) en archivos **.h** (las **implementaciones** irán en un **.cpp** con el **mismo nombre**).
- ▶ Podemos entonces separar nuestras funciones en múltiples archivos de texto (.cpp y .h) que conformarán una **librería**. Esto permitirá, además, utilizar la misma función en más de un programa.



# Archivos y librerías

- ▶ Para poder compilar y ejecutar nuestro código, escribiremos el mismo en un **archivo de texto plano**. Por convención, en C++ estos archivos llevan la extensión **.cpp**.
- ▶ Si nuestros programas crecen en líneas de código, incluir todo el código en un mismo archivo de texto resulta poco práctico.
- ▶ Además, en C++ se acostumbra a **separar las declaraciones** de funciones (de su implementación) en archivos **.h** (las **implementaciones** irán en un **.cpp** con el **mismo nombre**).
- ▶ Podemos entonces separar nuestras funciones en múltiples archivos de texto (.cpp y .h) que conformarán una **librería**. Esto permitirá, además, utilizar la misma función en más de un programa.

Veamos un ejemplo...

## Archivos y librerías: Ejemplo

Definiremos nuestra función "maximo" como una librería conformada por los archivos *maximo.h* y *maximo.cpp*, para luego poder usarla en uno de nuestros programas.

## Archivos y librerías: Ejemplo

Definiremos nuestra función "maximo" como una librería conformada por los archivos *maximo.h* y *maximo.cpp*, para luego poder usarla en uno de nuestros programas.

### maximo.h

```
int maximo(int a, int b);
```

## Archivos y librerías: Ejemplo

Definiremos nuestra función "maximo" como una librería conformada por los archivos *maximo.h* y *maximo.cpp*, para luego poder usarla en uno de nuestros programas.

### maximo.h

```
int maximo(int a, int b);
```

### maximo.cpp

```
#include "maximo.h"

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

# Archivos y librerías: Ejemplo

Para usar la librería, basta con incluir el archivo **.h** en cuestión utilizando la instrucción `#include "libreria.h"`.

`usando_maximo.cpp`

```
#include "maximo.h"

int un_valor = 4;
int otro_valor = 8;
int el_maximo = maximo(un_valor, otro_valor);
```

# Funciones en C++: Pasaje por copia vs. por referencia

- ▶ En C++ tenemos dos formas de pasar parámetros a las funciones: Por **referencia** o por **copia**.

# Pasaje de argumentos por copia

- ▶ Hasta ahora los argumentos entre funciones se pasaron siempre **por copia**.
- ▶ ¿Qué pasa cuando ejecuto el siguiente programa?

## Demo #1: Pasaje por copia

---

```
1  #include <iostream>
2  using namespace std;
3
4  void cambiarValor(int x) {
5      x = 15;
6  }
7  int main() {
8      int y = 10;
9      cambiarValor(y);
10     cout << y << endl; // que valor se imprime?
11     return 0;
12 }
```

---

# Pasaje de argumentos en C++

## **Pasaje por valor** (o por copia)

- ▶ Coloca en la posición de memoria del argumento de entrada el **valor** de la expresión usada en la invocación.
- ▶ Si la función modifica el valor, no se cambian las variables en el llamador.



# Pasaje de argumentos en C++

## **Pasaje por valor** (o por copia)

- ▶ Coloca en la posición de memoria del argumento de entrada el **valor** de la expresión usada en la invocación.
- ▶ Si la función modifica el valor, no se cambian las variables en el llamador.
- ▶ **Declaración** de la función: `int f(int b);`
- ▶ **Invocación** de la función: `f(x)`, o bien `f(x+5)` o bien `f(5)`.
- ▶ Es el modo *por defecto* de pasaje de argumentos en C++.

# Pasaje de argumentos por referencia

**Pasaje por referencia**

# Pasaje de argumentos por referencia

## **Pasaje por referencia**

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.

# Pasaje de argumentos por referencia

## **Pasaje por referencia**

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.
- ▶ La función puede leer esa posición de memoria pero también puede escribirla.

# Pasaje de argumentos por referencia

## **Pasaje por referencia**

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.
- ▶ La función puede leer esa posición de memoria pero también puede escribirla.
- ▶ Todas las asignaciones hechas dentro del cuerpo de la función cambian el contenido de la memoria **del llamador**.

# Pasaje de argumentos por referencia

## Pasaje por referencia

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.
- ▶ La función puede leer esa posición de memoria pero también puede escribirla.
- ▶ Todas las asignaciones hechas dentro del cuerpo de la función cambian el contenido de la memoria **del llamador**.
- ▶ La expresión con la que se realiza la invocación debe ser necesariamente una *variable*.

# Pasaje de argumentos por referencia

## Pasaje por referencia

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.
- ▶ La función puede leer esa posición de memoria pero también puede escribirla.
- ▶ Todas las asignaciones hechas dentro del cuerpo de la función cambian el contenido de la memoria **del llamador**.
- ▶ La expresión con la que se realiza la invocación debe ser necesariamente una *variable*.
- ▶ **Declaración** de la función: `int f(int &b);`

# Pasaje de argumentos por referencia

## Pasaje por referencia

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.
- ▶ La función puede leer esa posición de memoria pero también puede escribirla.
- ▶ Todas las asignaciones hechas dentro del cuerpo de la función cambian el contenido de la memoria **del llamador**.
- ▶ La expresión con la que se realiza la invocación debe ser necesariamente una *variable*.
- ▶ **Declaración** de la función: `int f(int &b);`
- ▶ **Invocación** de la función: `f(x)`, **pero no** `f(x+5)` ni `f(5)`.



# Pasaje de argumentos por referencia

- Modificamos el pasaje de parámetros con & para indicar que es una referencia y no una copia:

## Demo #2: Pasaje por referencia

---

```
1  #include <iostream>
2  using namespace std;
3
4  void cambiarValor(int &x) { // referencia a y
5      x = 15;
6  }
7
8  int main() {
9      int y = 10;
10     cambiarValor(y);
11     cout << y << endl; // que pasa ahora?
12     return 0;
13 }
```

---

# Ejemplos de pasaje de argumentos en C++

Pasaje por referencia vs. Pasaje por copia

```
void A_por_ref(int &i) {  
    i = i-1;  
}
```

# Ejemplos de pasaje de argumentos en C++

Pasaje por referencia vs. Pasaje por copia

```
void A_por_ref(int &i) {  
    i = i-1;  
}
```

```
void A_por_copia(int i) {  
    i = i-1;  
}
```

# Ejemplos de pasaje de argumentos en C++

Pasaje por referencia vs. Pasaje por copia

```
void A_por_ref(int &i) {  
    i = i-1;  
}
```

```
void A_por_copia(int i) {  
    i = i-1;  
}
```

```
void C() {  
    int j = 6;  
  
    A_por_ref(j);  
  
    A_por_copia(j);  
}
```

# Ejemplos de pasaje de argumentos en C++

## Pasaje por referencia vs. Pasaje por copia

```
void A_por_ref(int &i) {  
    i = i-1;  
}  
  
void A_por_copia(int i) {  
    i = i-1;  
}  
  
void C() {  
    int j = 6;  
    // En este momento tenemos j == 6;  
    A_por_ref(j);  
  
    A_por_copia(j);  
}
```

# Ejemplos de pasaje de argumentos en C++

## Pasaje por referencia vs. Pasaje por copia

```
void A_por_ref(int &i) {  
    i = i-1;  
}  
  
void A_por_copia(int i) {  
    i = i-1;  
}  
  
void C() {  
    int j = 6;  
    // En este momento tenemos  $j == 6$ ;  
    A_por_ref(j);  
    // Ahora tenemos  $j == 5$ ;  
    A_por_copia(j);  
}
```

# Ejemplos de pasaje de argumentos en C++

## Pasaje por referencia vs. Pasaje por copia

```
void A_por_ref(int &i) {  
    i = i-1;  
}  
  
void A_por_copia(int i) {  
    i = i-1;  
}  
  
void C() {  
    int j = 6;  
    // En este momento tenemos j == 6;  
    A_por_ref(j);  
    // Ahora tenemos j == 5;  
    A_por_copia(j);  
    // Seguimos teniendo j == 5;  
}
```

# Pasaje de parámetros por referencia

- ▶ Del mismo modo que podemos indicar que un entero se pasa por referencia con `int &a`, podemos hacer lo mismo con:
  - ▶ `float &f` (pasar por referencia un float)
  - ▶ `bool &b` (pasar por referencia un bool)
  - ▶ `char &c` (pasar por referencia un char)



# Ejemplos de pasaje de argumentos en C++

## Pasaje por referencia const

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.
- ▶ La función puede leer esa posición de memoria pero **no puede modificarla**.
- ▶ Hay un error de compilación si se intenta modificar la variable.
- ▶ **Declaración** de la función: `int f(const int &b);`
- ▶ **Invocación** de la función: `f(x)`, y también `f(x+5)`.

## Parámetros in, out, inout

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser in, out o inout.
  - ▶ in: parámetros de entrada
  - ▶ out: parámetros de salida
  - ▶ inout: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?

# Parámetros in, out, inout

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser in, out o inout.
  - ▶ in: parámetros de entrada
  - ▶ out: parámetros de salida
  - ▶ inout: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?
  - ▶ Para un parámetro in:

## Parámetros in, out, inout

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser in, out o inout.
  - ▶ in: parámetros de entrada
  - ▶ out: parámetros de salida
  - ▶ inout: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?
  - ▶ Para un parámetro in: un argumento que se pase por **copia**.

## Parámetros in, out, inout

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser in, out o inout.
  - ▶ in: parámetros de entrada
  - ▶ out: parámetros de salida
  - ▶ inout: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?
  - ▶ Para un parámetro in: un argumento que se pase por **copia**.
  - ▶ Para un parámetro inout:

# Parámetros in, out, inout

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser in, out o inout.
  - ▶ in: parámetros de entrada
  - ▶ out: parámetros de salida
  - ▶ inout: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?
  - ▶ Para un parámetro in: un argumento que se pase por **copia**.
  - ▶ Para un parámetro inout: un argumento que se pase por **referencia**.

## Parámetros in, out, inout

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser in, out o inout.
  - ▶ in: parámetros de entrada
  - ▶ out: parámetros de salida
  - ▶ inout: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?
  - ▶ Para un parámetro in: un argumento que se pase por **copia**.
  - ▶ Para un parámetro inout: un argumento que se pase por **referencia**.
  - ▶ Para un parámetro out:

# Parámetros in, out, inout

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser in, out o inout.
  - ▶ in: parámetros de entrada
  - ▶ out: parámetros de salida
  - ▶ inout: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?
  - ▶ Para un parámetro in: un argumento que se pase por **copia**.
  - ▶ Para un parámetro inout: un argumento que se pase por **referencia**.
  - ▶ Para un parámetro out:
    - ▶ un argumento que se pase por **referencia**, o
    - ▶ el valor de retorno de la función



## Resumen: Pasaje por copia vs. pasaje por referencia

Pasaje por copia	Pasaje por referencia
Por defecto. No es necesario anotar el parámetro	Hay que anotar el parámetro usando &
Crea una copia del dato	Crea un alias al dato
Los cambios son locales a la función	Los cambios modifican el dato original
in, out	inout, out

# Bibliografía

- ▶ B. Stroustrup. The C++ Programming Language.
  - ▶ 12.2 Argument Passing
- ▶ Deitel & Deitel. Como programar C++.