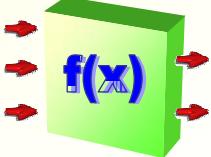




Metodología de la Programación $f(x)$

Tema 2.- Funciones

Las funciones. La función main()
Los parámetros y la devolución
La pila. Variables static
Sobrecarga de funciones
Funciones inline



UGR Universidad de Granada DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons]



Las funciones. La función main() $f(x)$

Introducción

Las funciones ...

- ... permiten generalizar algoritmos.
- ... evitan reescribir algoritmos más de una vez.
- ... permiten hacer abstracción procedural.
- ... permiten construir programas modulares.
- ... permiten escribir programas muy complejos de una forma sencilla.

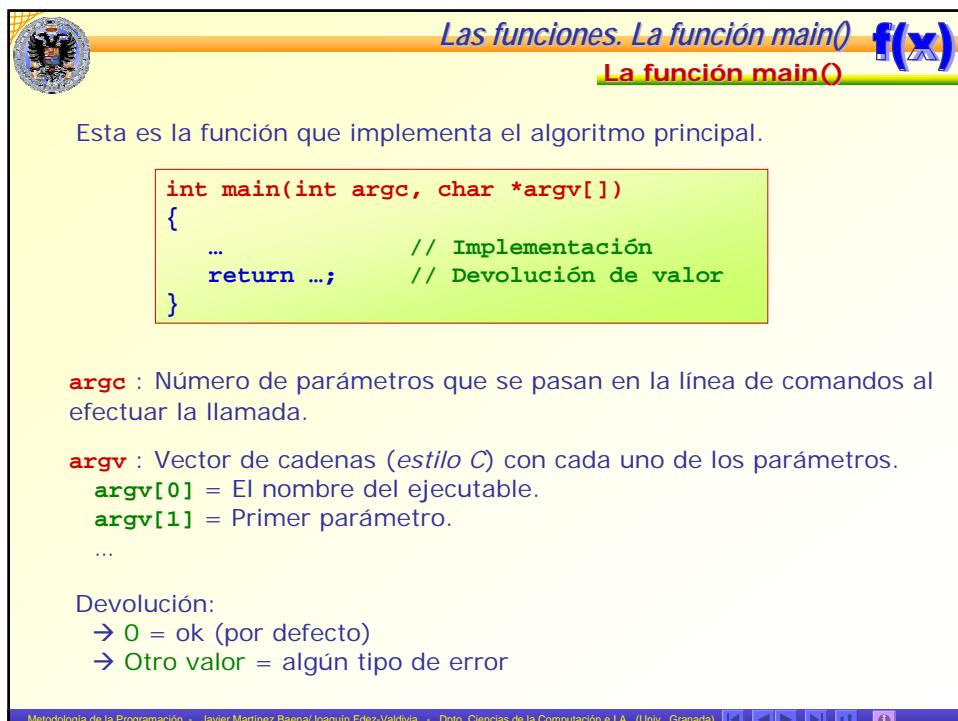
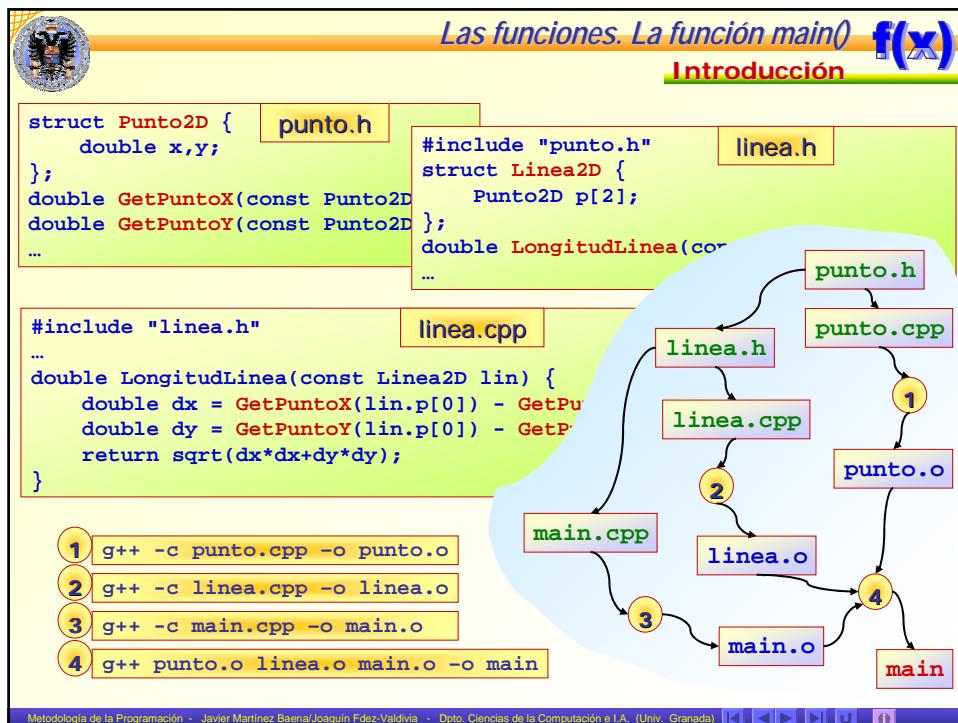
Prototipo o interfaz

```
tipo_devuelto nombre(parámetros_formales)
{
    ... // Implementación de algún algoritmo
    return ...; // Devolución de valor
}
```

Cuerpo

Lo Único necesario para la etapa de compilación es el prototipo

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons]



Las funciones. La función main() **f(x)**
La función main()

```
#include <iostream>                                         saludo.cpp
using namespace std;
int main(int argc, char *argv[])
{
    if (argc!=2) {
        cout << "Dime tu nombre" << endl;
        return 1; // Error de ejecución
    } else {
        cout << "Hola " << argv[1] << endl;
        cout << "Yo me llamo " << argv[0] << endl;
        return 0; // Ok
    }
}
```

prompt> saludo
Dime tu nombre

prompt> saludo Pepe
Hola Pepe
Yo me llamo saludo

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]

Las funciones. La función main() **f(x)**
La función main()

```
#include <iostream>                                         saludo.cpp
using namespace std;
int main(int argc, char *argv[])
{
    if (argc!=2) {
        cout << "Dime tu nombre" << endl;
        return 1; // Error de ejecución
    } else {
        cout << "Hola " << argv[1] << endl;
        cout << "Yo me llamo " << argv[0] << endl;
        return 0; // Ok
    }
}
```

prompt> saludo Pepe
Hola Pepe
Yo me llamo saludo

prompt> echo \$?
0

Consultar el resultado de main:

prompt> saludo
Dime tu nombre

prompt> echo \$?
1

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]



Metodología de la Programación $f(x)$

Tema 4.- Funciones

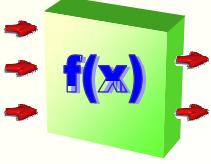
Las funciones. La función main()

Los parámetros y la devolución

La pila. Variables static

Sobrecarga de funciones

Funciones inline



Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)

Universidad de Granada  DECSAI Universidad de Granada



Los parámetros y la devolución $f(x)$

Paso de parámetros

El paso de parámetros se puede hacer ...

$\left. \begin{array}{l} \dots \text{por valor} \\ \dots \text{por referencia} \end{array} \right\}$

Paso por valor

El parámetro formal es una variable local cuyo valor inicial es el del parámetro actual. Son variables distintas.
Cualquier modificación sobre el parámetro formal **no** afecta al parámetro actual.

Paso por referencia (&)

El parámetro formal es una referencia (un segundo nombre) al parámetro actual. Son la misma variable.
Cualquier modificación al parámetro formal **también se hace** sobre el parámetro actual.

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)

 Los parámetros y la devolución **f(x)**
Vectores como parámetros

¿Es un paso por valor o por referencia?

```
void funcion(int v[], int n)
{
    for (int i=0; i<n; i++)
        v[i] = 0;
}
```

```
void funcion(int *v, int n)
{
    for (int i=0; i<n; i++)
        v[i] = 0;
}
```

```
int main(int argc, char *argv[])
{
    int v[10] = {1,2,3,4,5,6,7,8,9,10};
    funcion(v,10);
    for (int i=0; i<10; i++)
        cout << v[i] << endl;
}
```

¿Qué resultado muestra en pantalla?

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]

 Los parámetros y la devolución **f(x)**
Parámetros const

¿Qué ocurre en este caso?

```
void funcion(const int v[], int n)
{
    for (int i=0; i<n; i++)
        v[i] = 0;
}
```

```
struct A {
    double x, y, z;
    string c1, c2, c3;
    int a, b, c;
    ...
    // Ocupa mucha memoria
};
```

¿Qué opción es mejor?

```
void funcion1(A p1)
{
    ... // Paso por valor
}
```

```
void funcion2(A &p1)
{
    ... // Paso por referencia
}
```

```
void funcion3(const A &p1)
{
    ... // ¿Qué sentido tiene?
}
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]

Los parámetros y la devolución f(x)

Valor por defecto



Los parámetros pueden tener un valor por defecto.

<pre>void funcion(int a, int b) { ... }</pre>	<pre>... funcion(3,8); ...</pre>
<pre>void funcion(int a, int b=7) { ... }</pre>	<pre>... funcion(3,8); funcion(2); // b=7 ... </pre>
<pre>void funcion(int a=3, int b=7) { ... }</pre>	<pre>... funcion(9,8); funcion(2); // b=7 funcion(); // a=3, b=7 ... </pre>

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]

Los parámetros y la devolución f(x)

Valor por defecto



```
void funcion(int a=3, int b=7)
{
    ...
}
```

¿Podemos llamar a `funcion()` con un valor para `b` y que `a` tome el valor por defecto?

```
void funcion(int a=3, int b)
{
    ...
}
```

¿Es correcta esta función?
Los parámetros con valor por defecto siempre van al final.

```
void funcion(int a, int b=2, float c=4.3, char *d=0)
{
    ...
}
```

¿Qué llamadas son correctas?

<code>funcion(2);</code>	<code>funcion(2,4);</code>
<code>funcion(2,3,"hola");</code>	<code>funcion(2,4,7.8);</code>
<code>funcion(2,3.5,"abc");</code>	<code>funcion("xyz");</code>

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]

 Los parámetros y la devolución **f(x)**
Devolución por referencia

Una función puede devolver una referencia a un dato

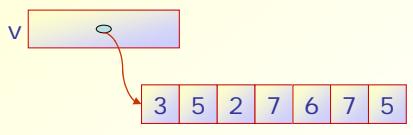
```
int v[7] = {3,5,2,7,6,7,5};
```

Considerar las siguientes implementaciones de **funcion()**

```
int funcion1(int v[], int p)
{
    return v[p];
}
```

```
int & funcion2(int v[], int p)
{
    return v[p];
}
```

¿Qué devuelven las siguientes llamadas?

v 

funcion1(v,3);
Devuelve el valor 7

funcion2(v,3);
Devuelve una referencia a v[3]

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 

 Los parámetros y la devolución **f(x)**
Devolución por referencia

¿Qué ocurre en el siguiente programa?

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

int & funcion()
{
    int x;
    x = 3;
    return x;
}

int main(int argc, char *argv[])
{
    int y=2;
    y = funcion();
    cout << y << endl;
}
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 

The title slide features a yellow header bar with the text "Metodología de la Programación f(x)" and the UGR logo. Below the header is a green bar with the title "Tema 4.- Funciones". The main content area is white with a vertical yellow sidebar on the left. The sidebar contains five blue links: "Las funciones. La función main()", "Los parámetros y la devolución", "La pila. Variables static", "Sobrecarga de funciones", and "Funciones inline". To the right of the sidebar is a green 3D cube labeled "f(x)" with four red arrows pointing towards it. At the bottom of the slide are logos for the University of Granada (UGR) and the Department of Computer Sciences and I.A. (DECSAI), along with navigation icons.

This slide focuses on the stack (Pila). It shows a diagram of memory regions with labels: "Código" (Code), "Instrucciones" (Instructions), "Memoria estática" (Static Memory), "Constantes, Globales, static" (Constants, Globals, static), "Montón (heap) o memoria dinámica" (Heap or dynamic memory), and "Se reserva y libera en tiempo de ejecución a petición del programa" (Allocated and deallocated at runtime by request of the program). A bracket on the left indicates that the first three items are fixed at compilation time. A bracket on the right indicates that the last two items are managed at runtime. The bottom part of the diagram is labeled "Pila (stack)". Below the diagram, a statement says "Desde una función tenemos acceso" followed by a brace that spans two items: "a las variables globales del programa" (to global variables of the program) and "a las variables locales de la función" (to local variables of the function). Another statement says "En la pila almacenamos:" followed by a brace that spans two items: "Las variables locales de las funciones" (local variables of functions) and "Los parámetros formales" (formal parameters).



La pila. Variables static f(x) ¿Cómo funciona la pila?

Las variables locales y parámetros formales son creados al comienzo de la ejecución de una función. Se destruyen al finalizar la ejecución de la misma.

¿Qué ocurre con la memoria en este programa?

```
int z; // Var. global
void funcion(int p1, char p2)
{
    double x; // Var. local a funcion()
    int *loc2; // Var. local a funcion()
    ...
}
int main(int argc, char *argv[])
{
    int x; // Var. local a main()
    char y; // Var. local a main()
    funcion(x,y);
    ...
}
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]



La pila. Variables static f(x) Variables static

Las variables **static** son locales a las funciones (desde el punto de vista del ámbito).

No se crean ni se destruyen al llamar a las funciones. Mantienen su valor entre llamadas.

```
void funcion()
{
    static int a=3;
    ...
}
```

Es obligatorio asignarles un valor en su declaración.

La primera vez que se llama a la función se inicializan.
En sucesivas llamadas conservan el último valor que hubiesen tomado en la última ejecución de la función.

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]

 **La pila. Variables static f(x)**

Variables static

¿Qué resultado muestra?

```
void funcion() {
    static int x=2;
    cout << x << endl;
    x++;
}
main() {
    funcion();
    funcion();
    funcion();
}
```

¿Son correctas las siguientes funciones?

```
int & funcion()
{
    int x=7;
    return x;
}
```

```
int & funcion()
{
    static int x=7;
    return x;
}
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 

 **Metodología de la Programación f(x)**

Tema 4.- Funciones

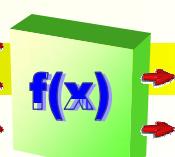
Las funciones. La función main()

Los parámetros y la devolución

La pila. Variables static

Sobrecarga de funciones

Funciones inline



Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 

 **Sobrecarga de funciones $f(x)$**
¿Qué es?

Normalmente las funciones se diferencian, entre otras cosas, por su nombre.

Es posible tener *dos o más funciones con un mismo nombre siempre y cuando sea posible distinguirlas mediante el tipo y número de sus parámetros formales*.

```

void funcion(int x)
{
    ...
}

void funcion(char *c)
{
    ...
}

void funcion(int x, double y)
{
    ...
}

main()
{
    char *c;
    funcion(3);
    funcion(4,9.3);
    funcion(c);
}

```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 

 **Sobrecarga de funciones $f(x)$**
Ejemplo

Permiten simplificar un poco la interfaz de las funciones.

```

struct Punto {
    double x, y;
}

struct Rectangulo {
    Punto esq_sup_izq;
    Punto esq_inf_der;
}

struct Circulo {
    Punto centro;
    double radio;
}

void Dibujar(Punto p) {
    ...
}

void Dibujar(Rectangulo r) {
    ...
}

void Dibujar(Circulo c) {
    ...
}

main() {
    Punto p;
    Rectangulo rec;
    Circulo cir;
    ...
    Dibujar(rec);
    Dibujar(p);
    Dibujar(cir);
}

```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 



Sobrecarga de funciones **f(x)**

Ejemplo

¿Qué resultado producen las siguientes ejecuciones?

```
void f(double x) {
    cout << "double" << x << endl;
}
main() {
    f(3.4);
    f(3);      // Se hace conversión implícita
}
```



```
void f(double x) {
    cout << "double" << x << endl;
}
void f(int x) {
    cout << "int" << x << endl;
}
main() {
    f(3.4);
    f(3);
}
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]



Sobrecarga de funciones **f(x)**

Ejemplo

¿Qué resultado producen las siguientes ejecuciones?

```
void f(int x) {
    cout << "int" << x << endl;
}
main() {
    f(3.4);    // Se hace conversión implícita (warning)
    f(3);
}
```



```
int f(void) {
    return 20;
}
char f(void) {
    return 'c';
}
main() {
    int x=f();
    char c=f();
}
```

Error: No es posible distinguir 2 funciones mirando el tipo devuelto

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]



Sobrecarga de funciones $f(x)$

Ejemplo

¿Qué resultado produce la siguiente ejecución?

```
void f(int a, int b) {  
    ...  
}  
void f(double a, double b) {  
    ...  
}  
main() {  
    f(2,4);  
    f(3.7,9.2);  
    f(2,9.6);      // Ambiguo  
    f(7.1,8);       // Ambiguo  
    f(7.1,static_cast<double>(8));  
    f(7.1,8.0);  
}
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]



Sobrecarga de funciones $f(x)$

Ejemplo

Necesitamos una función para listar el contenido de un vector de 100 números enteros.

Debemos poder hacer llamadas del tipo:

```
Listado(v,20);    // Lista los 20 primeros elementos  
Listado(v);        // Lista el vector completo
```

Considerar hacerlo usando: Valores por defecto.
Sobrecarga de funciones.

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons]

Tema 4.- Funciones

Las funciones. La función main()
Los parámetros y la devolución
La pila. Variables static
Sobrecarga de funciones
Funciones inline

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons]

Funciones inline f(x)

¿Qué son?

Son **similares a las macros** (aunque con comprobación de tipos).
Su ejecución suele ser más **rápida**.
El código generado puede ocupar **más espacio**.
inline sólo es una sugerencia al compilador.
Su **implementación completa** es necesaria para poder compilar (*no basta con el prototipo*).
La implementación suele ir en los ficheros de cabecera (.h).
Suelen ser **funciones pequeñas** o que son llamadas con mucha frecuencia.

```
inline funcion()
{
    ...
}
main() {
    funcion();
}
```

¿Qué diferencias hay en la ejecución de **funcion()** siendo y no siendo **inline**?

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons]