

FUNCIONES

- Hasta ahora, hemos visto y utilizado funciones que estaban predefinidas en el estándar ANSI C, por ejemplo `printf()`, `sqrt()`, `pow()`, `scanf()`, etc. Este tipo de funciones predefinidas son denominadas funciones de biblioteca.

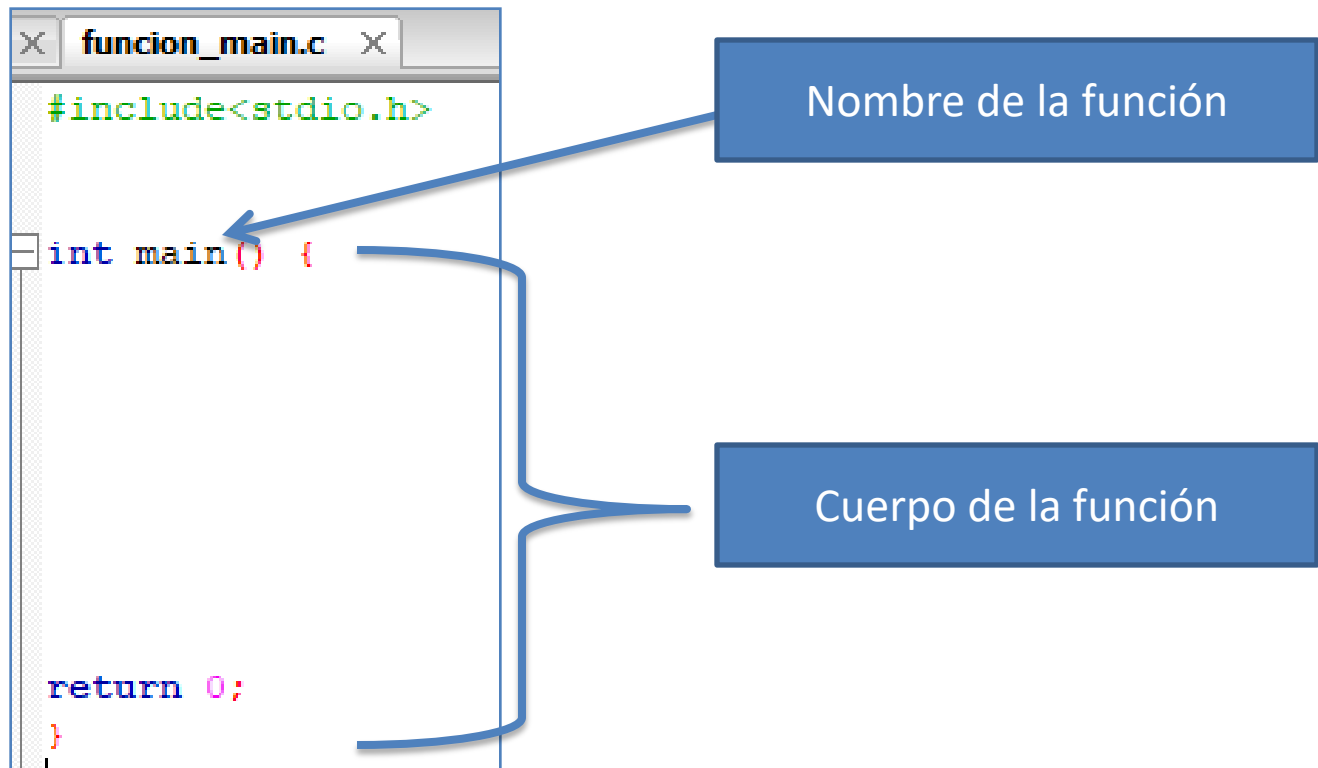
- Una función es un fragmento de código que realiza una tarea bien definida. Sin embargo, el programador puede definir sus propias funciones de acuerdo a sus necesidades.

- El uso de funciones permite dividir un gran programa en pequeños módulos que realizan tareas concretas. Por ejemplo, es probable que dentro de un mismo programa se realicen las mismas tareas varias veces, por lo tanto, construyendo funciones para esas tareas, se facilita el desarrollo, mejorando además la legibilidad del programa.

- La filosofía en la que se basa el lenguaje C es el empleo de funciones. Por esta razón, un programa en C contiene al menos una función, la función `main`. Esta función es particular dado que la ejecución del programa se inicia con las instrucciones contenidas en su interior. Una vez iniciada la ejecución del programa, desde la función `main` se puede llamar a otras funciones y, puede ser también que desde estas funciones a otras.

- Otra particularidad de la función `main` es que se llama directamente desde el sistema operativo y no desde ninguna otra función. De esta manera, un programa en C sólo puede contener una función `main`.

Las funciones son bloques de código con un nombre asociado que realizan una tarea en particular. En C todo se construye con funciones, (**main es la función principal del programa**)



A diagram illustrating the structure of a C program. It shows a code editor window titled "funcion_main.c" containing the following code:

```
#include<stdio.h>

int main() {

    return 0;
}
```

Two blue callout boxes with arrows point to specific parts of the code:

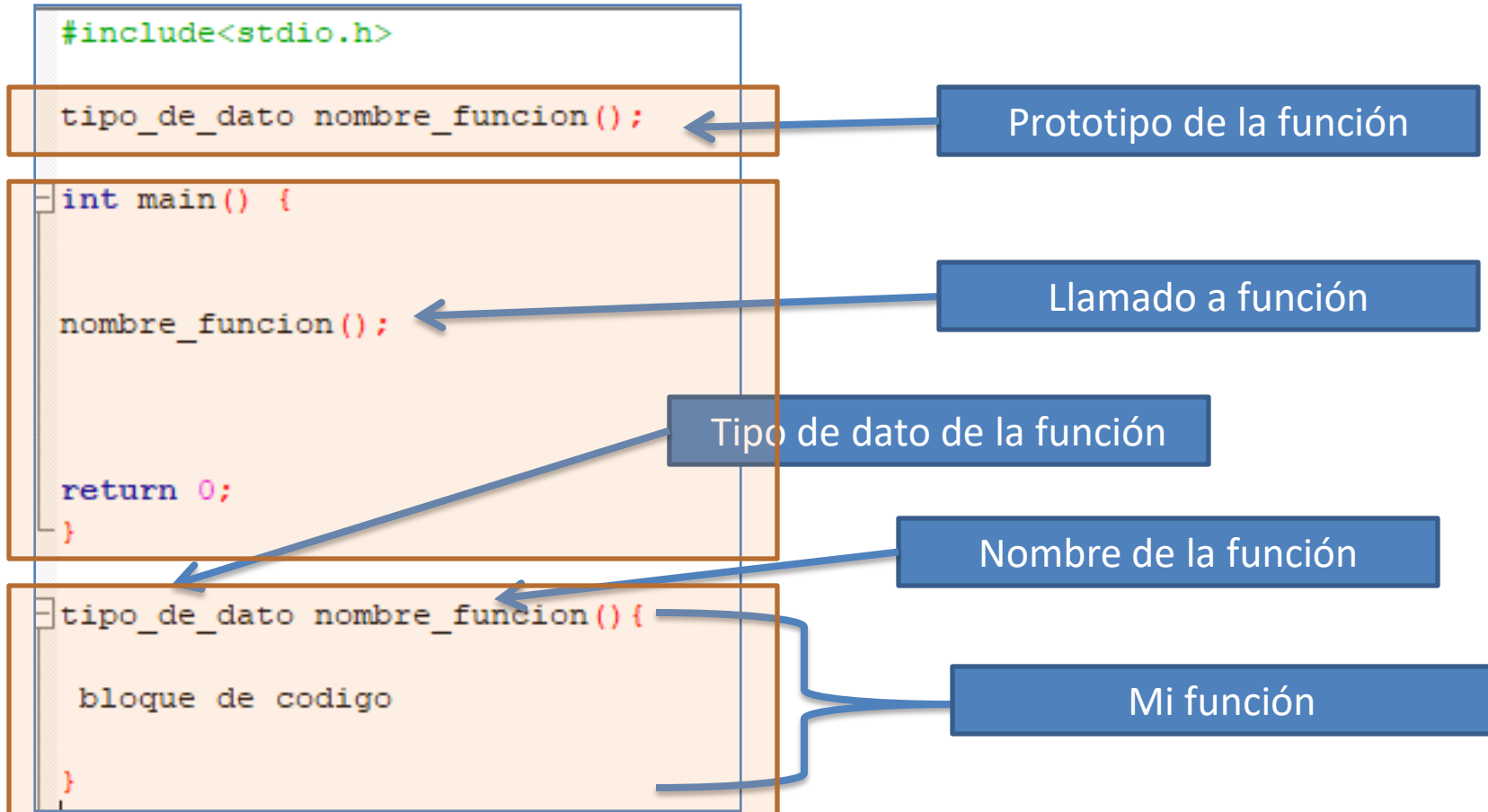
- The first box, labeled "Nombre de la función", has an arrow pointing to the `main()` part of the `int main()` declaration.
- The second box, labeled "Cuerpo de la función", has a bracket pointing to the body of the function, which includes the opening curly brace `{`, the `return 0;` statement, and the closing curly brace `}`.

Nombre de la función

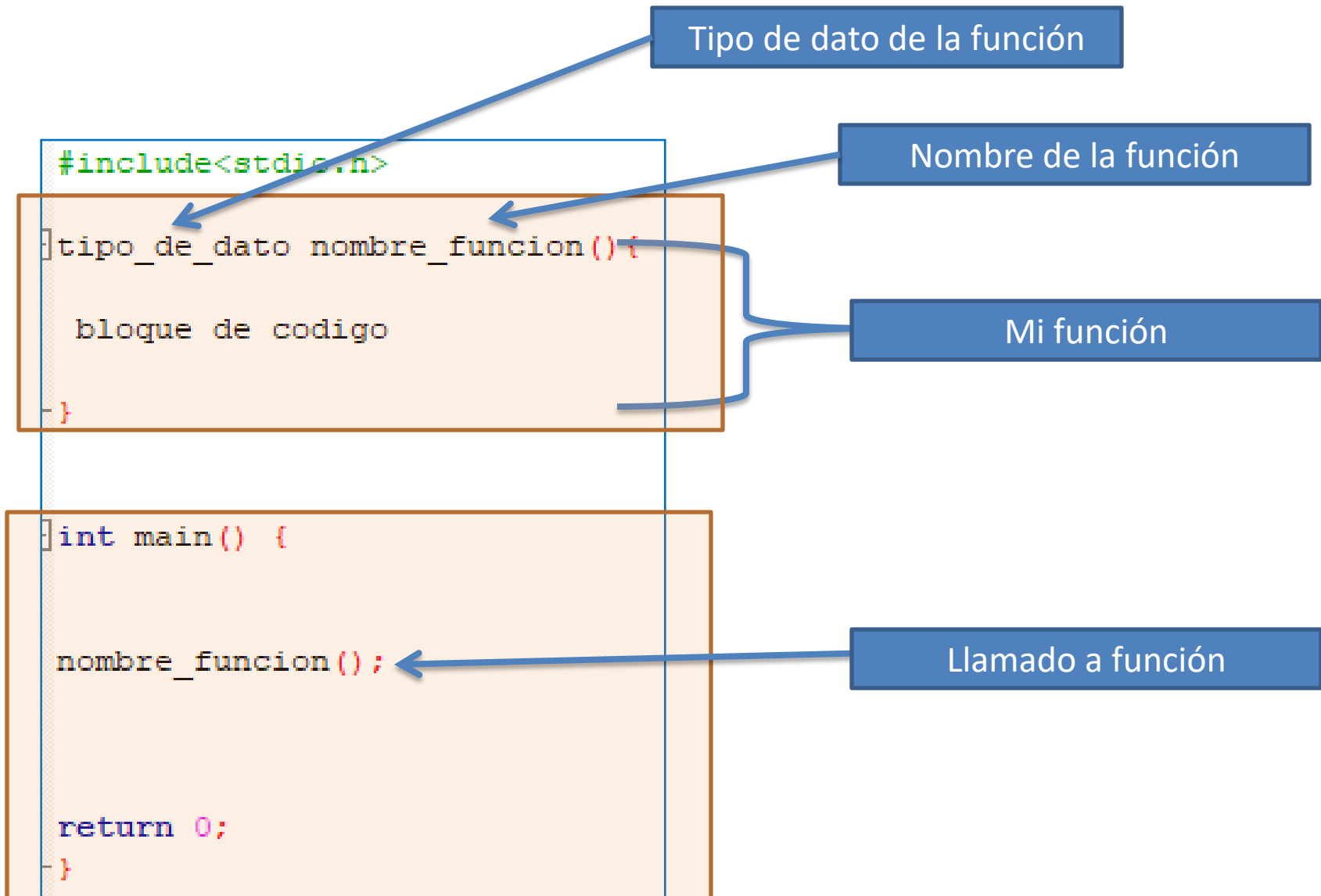
Cuerpo de la función

FORMAS DE ESCRIBIR LAS FUNCIONES EN C

Escribir un programa con funciones – Primera Forma



Escribir un programa con funciones – Segunda forma



- Con el propósito de permitir un manejo eficiente de los datos, las funciones en C no se pueden anidar. En otras palabras, una función no se puede declarar dentro de otra función, por lo que todas las funciones son “externas”, lo que hace que puedan llamarse desde cualquier parte de un programa.

- Se puede acceder (llamar) a una determinada función desde cualquier parte de un programa.

- Cuando se llama a una función, se ejecutan las instrucciones que constituyen dicha función.

- Una vez que se ejecutan las instrucciones de la función, se devuelve el control del programa a la siguiente instrucción (si existe) inmediatamente después de la que provocó la llamada a la función.

- Una función en C sólo puede retornar un valor, pero puede ocurrir que no necesitemos que retorne un valor y nuestra función sea de tipo void.

DEFINICIÓN DE UNA FUNCIÓN

- A la implementación de una función se conoce como definición. Es precisamente en la definición de una función donde se especifican las instrucciones que forman parte de la misma y que se utilizan para llevar a cabo la tarea específica de la función.

- La definición de una función consta de dos partes, el encabezado y el cuerpo de la función.

- En el encabezado de la función, al igual que en el prototipo de la misma, se tienen que especificar los parámetros de la función, si los utiliza y el tipo de datos que devuelve, mientras que el cuerpo se compone de las instrucciones necesarias para realizar la tarea para la cual se crea la función.

La estructura general de una función en C es la siguiente:

Funciones que retornan un valor

```
tipo_de_dato_de_retorno nombre_de_la_función (lista_de_parámetros)
{
    cuerpo_de_la_función

return expresión
}
```

- **tipo_de_retorno:** es el tipo del valor devuelto por la función.
- **nombre_de_la_función:** es el nombre o identificador asignado a la función.
- **lista_de_parámetros:** es la lista de declaración de los parámetros que son pasados a la función. Éstos se separan por comas. Debemos tener en cuenta que pueden existir funciones que no utilicen parámetros.
- **cuerpo_de_la_función:** está compuesto por un conjunto de sentencias que llevan a cabo la tarea específica para la cual ha sido creada la función.
- **return expresión:** mediante la palabra reservada return, se devuelve el valor de la función, en este caso representado por expresión.

Funciones void()

```
void nombre_de_la_función (lista_de_parámetros)
{
    cuerpo_de_la_función

}
```

- **tipo de retorno:** No existe, por lo tanto es void.
- **nombre_de_la_función:** es el nombre o identificador asignado a la función.
- **lista_de_parámetros:** es la lista de declaración de los parámetros que son pasados a la función. Éstos se separan por comas. Debemos tener en cuenta que pueden existir funciones que no utilicen parámetros.
- **cuerpo_de_la_función:** está compuesto por un conjunto de sentencias que llevan a cabo la tarea específica para la cual ha sido creada la función.
- No hay return o puede haber un return sin expresión.

EJEMPLOS

Ejemplo 1 – Función void sin parámetros

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

```
void f_saluda() {
```

```
    printf("Holaaaaa!!! c¿mo est¿cn??\n", 162,160);
```

```
}
```

```
int main() {
```

```
    f_saluda();
```

```
    return 0;
```

```
}
```

```
Holaaaaa!!! c¿mo est¿cn??
```

```
Process returned 0 (0x0)   execution time : 0.094 s  
Press any key to continue.
```

Al iniciarse la función main, ejecuta el bloque de instrucciones que encuentra. En este ejemplo, encuentra el llamado a la función f_saluda, entonces entrega el mando a la función que ejecuta todo su bloque de instrucciones, y al terminar devuelve el mando a main, que continúa con el programa. Cabe aclarar que un mensaje **NO** es un retorno de dato.

Ejemplo 2 – Función void sin parámetros – Con 3 llamados

```
#include<stdio.h>

void f_saluda(){

    printf("Holaaaaaa!!! c¿cmo est¿cn??\n", 162,160);

}

int main(){

    f_saluda();
    f_saluda();
    printf("Invocamos una vez m¿cs\n", 160);
    f_saluda();

    return 0;
}
```

```
Holaaaaaa!!! ¿ómo est¿n??
Holaaaaaa!!! ¿ómo est¿n??
Invocamos una vez m¿s
Holaaaaaa!!! ¿ómo est¿n??
```

Podemos llamar a la misma función varias veces dentro del programa.

Ejemplo 3 – Función void sin parámetros - funciones que llaman a otras funciones

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

```
void f_pregunta(){  
    printf("Todo bien??\n");  
}
```

```
void f_saluda(){  
    printf("Holaaaaaa!!!\n");  
    f_pregunta();  
}
```

```
int main(){  
  
    f_saluda();  
    f_saluda();  
    printf("Invocamos una vez más\n", 160);  
    f_saluda();  
  
    return 0;  
}
```

```
Holaaaaaa!!!  
Todo bien??  
Holaaaaaa!!!  
Todo bien??  
Invocamos una vez más  
Holaaaaaa!!!  
Todo bien??
```

En este ejemplo, la función main invoca a f_saluda, y f_saluda invoca a la función f_pregunta. Como se ve en el ejemplo, las definiciones de las funciones son independientes pero pueden ser llamadas por otras funciones y también por main.

Ejemplo 4 – Función void sin parámetros – variables locales

```
#include<stdio.h>

void f_suma(){
    int a, b, suma;

    printf("Ingrese dos números enteros\n", 163);
    scanf("%d%d", &a,&b);
    suma = a + b;
    printf("La suma es %d\n", suma);
}

int main(){

    f_suma();

    return 0;
}
```

Variables locales en f_suma

```
Ingrese dos números enteros
7
4
La suma es 11
```

Dentro de las funciones se pueden declarar todas las variables necesarias, siendo las mismas variables locales a la función. Se crean cuando la función se activa y se destruyen al finalizar la tarea de la función.

Ejemplo 5 – Función void sin parámetros – llamadas en un ciclo

```
#include<stdio.h>

void f_suma(){
    int a, b, suma;

    printf("Ingrese dos números enteros\n");
    scanf("%d%d", &a,&b);
    suma = a + b;
    printf("La suma es %d\n", suma);
}

int main(){
    int i;

    for (i = 1; i <= 3; i++){
        f_suma();
    }

    return 0;
}
```

Variable local en main

```
Ingrese dos números enteros
3
4
La suma es 7
Ingrese dos números enteros
5
6
La suma es 11
Ingrese dos números enteros
7
8
La suma es 15
```

Las funciones pueden ser invocadas dentro de un ciclo. Y ese ciclo puede estar en main u otra función.

Ejemplo 6 – Función void con parámetros

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

```
void f_suma(int a, int b){  
    int suma;  
  
    suma = a + b;  
    printf("La suma es %d\n", suma);  
}
```

Parámetros formales, también las variables de los argumentos son locales en la función

```
int main(){  
    int a, b;  
  
    printf("Ingrese dos números enteros\n", 163);  
    scanf("%d%d", &a, &b);  
    f_suma( a, b );  
  
    return 0;  
}
```

Argumentos actuales o reales

```
Ingrese dos números enteros  
4  
5  
La suma es 9
```

Los parámetros deben coincidir en el orden, cantidad y tipo de datos. Pueden ser pasados en main u otra función

Ejemplo 7 – Función con valor de retorno y sin parámetros

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

Tipo de dato a retornar

```
int f_suma(){  
    int a, b, suma;  
  
    printf("Ingrese dos números enteros\n", 163);  
    scanf("%d%d", &a, &b);  
    suma = a + b;  
  
    return suma;  
}
```

Valor a retornar

```
int main(){  
    int suma;  
  
    suma = f_suma();  
    printf("La suma es %d\n", suma);  
  
    return 0;  
}
```

El dato retornado puede ser guardado en una variable

```
Ingrese dos números enteros  
6  
7  
La suma es 13
```

Ejemplo 8 – Función con valor de retorno y sin parámetros

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

Tipo de dato a retornar

```
int f_suma(){  
    int a, b, suma;
```

```
    printf("Ingrese dos números enteros\n", 163);
```

```
    scanf("%d%d", &a, &b);
```

```
    suma = a + b;
```

```
    return suma;
```

```
}
```

Valor a retornar

El valor retornado también puede ser emitido directamente sin ser guardado.

```
int main(){
```

```
    printf("La suma es %d\n", f_suma());
```

```
    return 0;
```

```
}
```

```
Ingrese dos números enteros
```

```
8
```

```
3
```

```
La suma es 11
```

Ejemplo 9 – Función con valor de retorno y con parámetros

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

```
int f_suma(int a, int b){  
    int suma;  
  
    suma = a + b;  
  
    return suma;  
}
```

Parámetros formales y locales en la función

Valor que se retorna

```
int main(){  
    int a, b, suma;  
  
    printf("Ingrese dos números enteros\n", 163);  
    scanf("%d%d", &a, &b);  
  
    suma = f_suma( a, b );  
  
    printf("La suma es %d\n", suma);  
  
    return 0;  
}
```

Se le puede pasar argumentos a una función con retorno

```
Ingrese dos números enteros  
5  
8  
La suma es 13
```

Variables locales

Cuando declaramos variables dentro de la función main, están únicamente asociadas a esta función, o sea que son variables locales de la función main y no se puede acceder a ellas a través de ninguna otra función.

Al igual que cualquier variable que declaremos dentro de una función, es local a esa función, es decir, su ámbito esta confinado a dicha función.

Esta situación permite que existan variables con el mismo nombre en diferentes funciones y que no mantengan ninguna relación entre sí.

Variables globales

El ámbito de las variables globales se extiende a todo el programa.

En otras palabras, si definimos una variable al principio del programa, cualquier función que forme parte de éste podrá utilizarla simplemente haciendo uso de su nombre.

El uso de variables globales proporciona un mecanismo de intercambio de información entre funciones sin necesidad de utilizar parámetros o argumentos.

Su uso puede llevar a programas de difícil interpretación y complejos de depurar por lo tanto debemos evitar utilizarlas.

Ejemplo 10 – Variables locales

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

```
int f_suma(int a, int b){  
    int suma;  
  
    suma = a + b;  
  
    return suma;  
}
```

a, b y suma son variables locales a la función f_suma

```
int main(){  
    int a, b, suma;  
  
    printf("Ingrese dos números enteros\n", 163);  
    scanf("%d%d", &a, &b);  
  
    suma = f_suma( a, b );  
  
    printf("La suma es %d\n", suma);  
  
    return 0;  
}
```

a, b y suma son variables locales a la función main

a, b y suma de main son variables distintas a a, b y suma de f_suma

PASAJE DE ARGUMENTOS POR VALOR

Ejemplo 11– Variables Locales - Parámetros pasados por valor

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

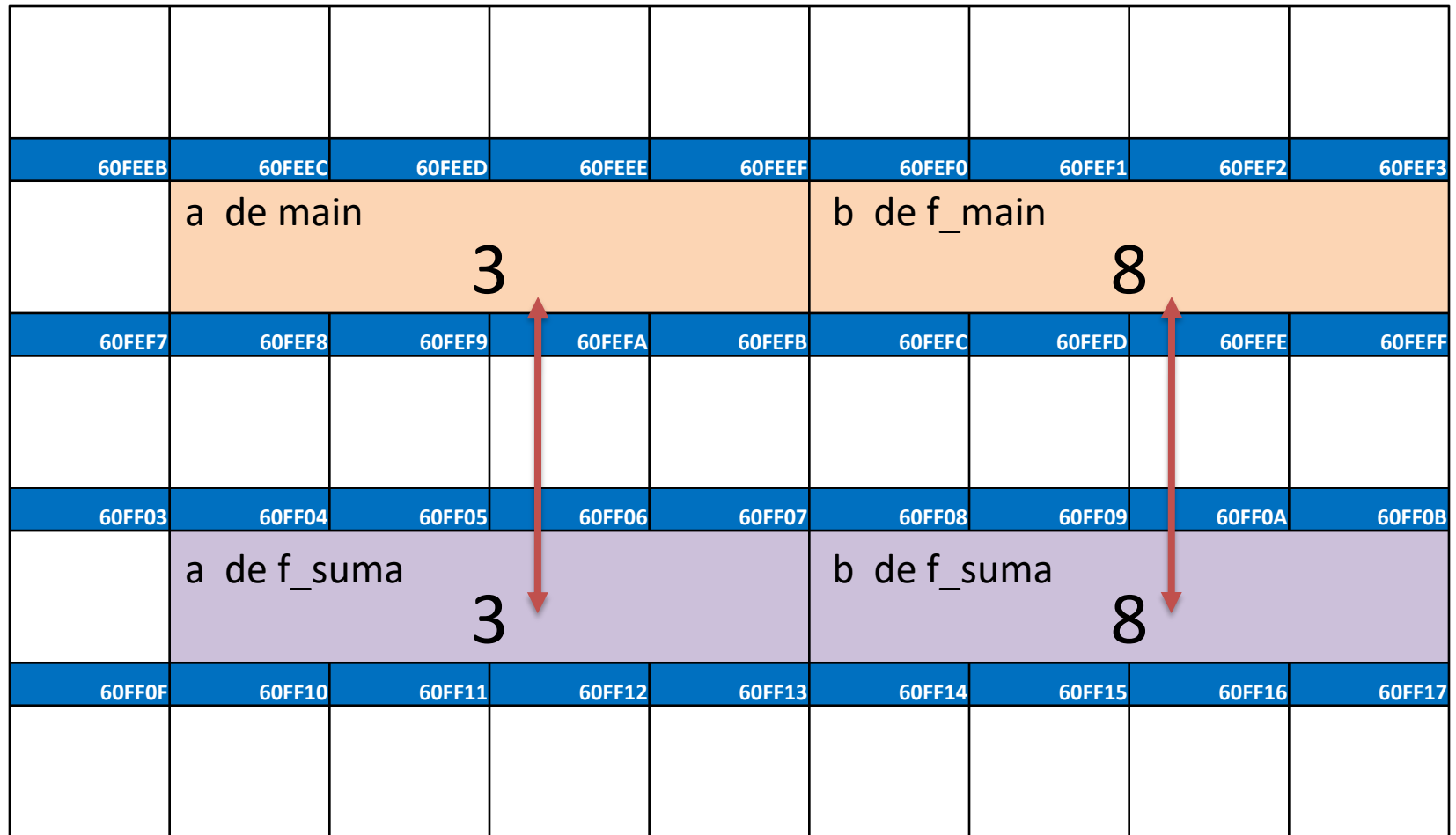
```
void f_suma(int a, int b){  
    int suma;  
  
    suma = a + b;  
    printf("La suma es %d\n", suma);  
}
```

Parámetros o argumentos formales, también locales en la función f_suma

```
int main(){  
    int a, b;  
  
    printf("Ingrese dos números enteros\n", 162);  
    scanf("%d%d", &a, &b);  
    f_suma( a, b );  
  
    return 0;  
}
```

Parámetros o argumentos actuales también locales en función main

Qué sucede con los parámetros o argumentos en memoria?



Cuando se pasan argumentos actuales, los valores de los mismos se copian en los argumentos formales y la función trabaja con la copia sin alterar los valores originales. A Este procedimiento se le llama **pasaje por valor** (valor de las variables)

PASAJE DE ARGUMENTOS POR DIRECCIÓN

Ejemplo 12- Intercambio del valor de dos variables

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

```
void f_intercambio(int a, int b){  
    int aux;  
  
    aux = a;  
    a = b;  
    b = aux;  
}
```

```
int main(){  
    int a, b, suma;  
  
    printf("Ingrese dos números enteros\n", 163);  
    scanf("%d%d", &a, &b);  
  
    printf("Antes de la función a:%d y b:%d\n", 162, a, b);  
  
    f_intercambio( a, b );  
  
    printf("Después de la función a:%d y b:%d\n", 130, 162, a, b);  
  
    return 0;  
}
```

Ingrese dos números enteros

9

5

Antes de la función a:9 y b:5

Después de la función a:9 y b:5



En la ejecución se observa que después de llamar a la función, las variables no han intercambiado su valor.

Ejemplo 13 - Intercambio del valor de dos variables

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

```
void f_intercambio(int * a, int * b)
```

```
int aux;
```

```
aux = *a;
```

```
*a = *b;
```

```
*b = aux;
```

```
}
```

```
int main() {
```

```
int a, b, suma;
```

```
printf("Ingrese dos números enteros\n", 163);
```

```
scanf("%d%d", &a, &b);
```

```
printf("Antes de la función a:%d y b:%d\n", 162, a, b);
```

```
f_intercambio(&a, &b);
```

```
printf("Después de la función a:%d y b:%d\n", 130, 162, a, b);
```

```
return 0;
```

```
}
```

Ingrese dos números enteros

9

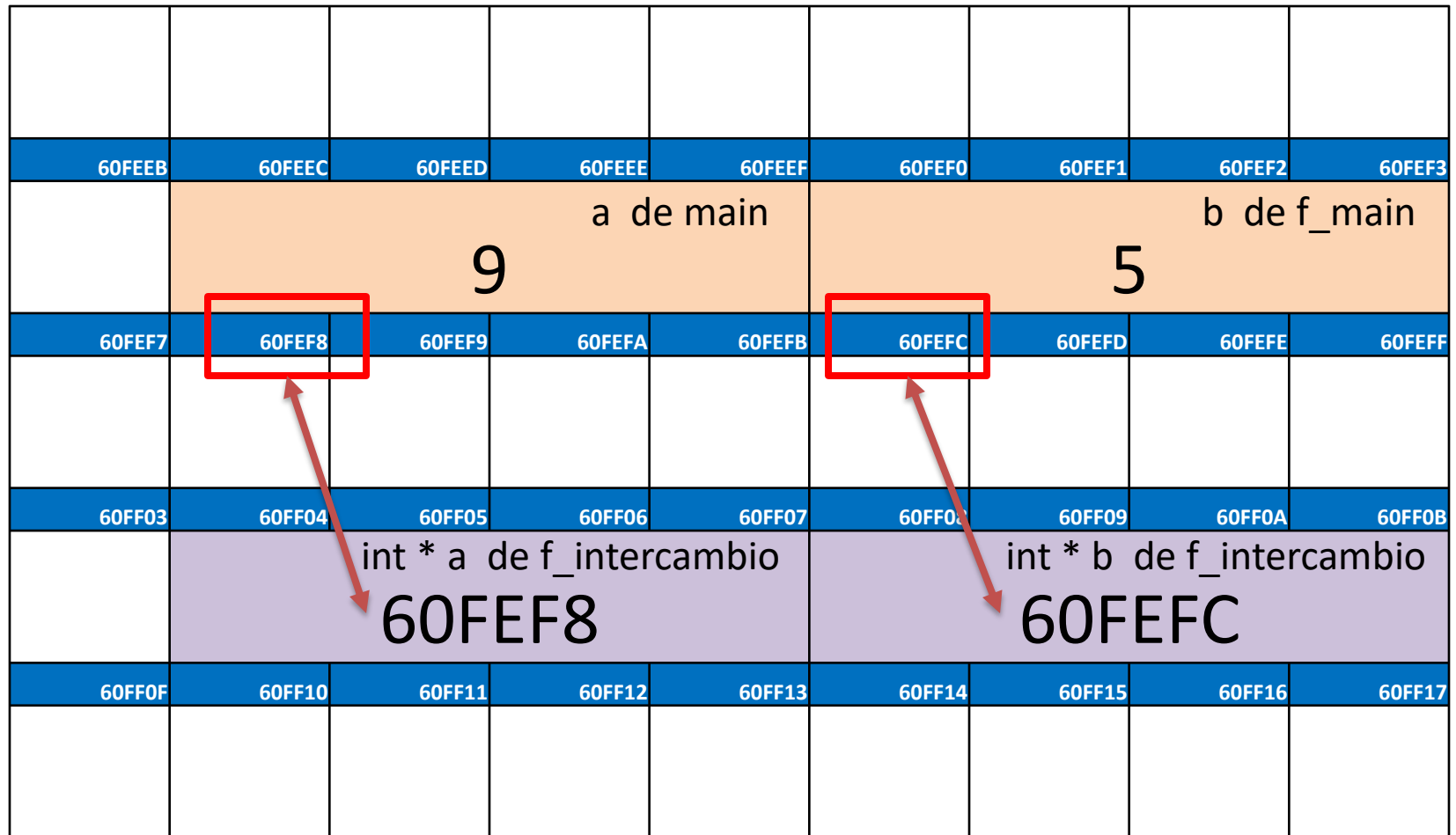
5

Antes de la función a:9 y b:5

Después de la función a:5 y b:9

Si pasamos los argumentos por dirección logramos modificar el valor original de la variable.

Qué sucede con los parámetros o argumentos en memoria?



Cuando se pasan argumentos actuales por dirección, esta se copia en el argumento formal que deben ser variables punteros (porque sólo las variables punteros pueden guardar direcciones). Entonces la función trabaja directamente en la dirección de la variable original. A este procedimiento se le llama **pasaje por dirección** (dirección de las variables)

Uso de parámetros por valor	Uso de parámetros por dirección
Cuando <u>no</u> necesitemos cambiar el valor original de la variable.	Cuando necesitemos modificar el valor de la variable original.
Puede ser que necesitemos pasar parámetros por valor y dirección a una misma función, está permitido, sólo hay que respetar el orden, cantidad y tipo de datos en los argumentos.	
El paso de parámetros por valor o por dirección es independiente del tipo de función que se utilice, es decir función con retorno de valor o void.	