

Programación I

Funciones

Funciones

Una función es un conjunto de instrucciones que realizan una tarea específica de manera independiente del resto del programa. Se caracterizan por tener un **nombre**, un **valor devuelto** y **parámetros**.

*

*


Nombre: El nombre de una función debe obedecer las reglas para nombrar variables.

Valor devuelto: En su tarea a resolver, una función puede devolver un valor (**entero**, **float**, **char**, etc) o no devolver nada (**void**).

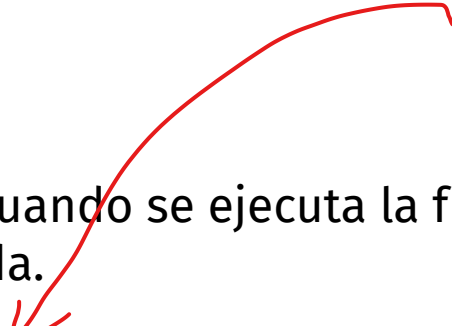
Parámetros: Para poder resolver su tarea, una función puede recibir ninguno o varios parámetros. Estos se guardarán en variables que serán **locales a la función**.

Funciones


```
void saludar(string nombrePersona);
```



■ **Llamado:** Cuando se ejecuta la función y realiza la tarea para la que fue desarrollada.



■ **Declaración:** Se determinan el valor devuelto de la función, el nombre de la misma y los parámetros que recibe. Suelen aparecer en archivos H.



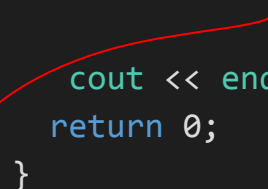
■ **Definición:** Se determinan el valor devuelto de la función, el nombre de la misma, los parámetros que recibe y el código que representa el algoritmo de la función. Suelen aparecer en archivos CPP.

```
void saludar(string nombrePersona){  
    cout << "Hola " << nombrePersona;  
}
```

Llamado a una función

Ejemplo de uso de una función llamada *sumar* desde la función *main*

```
#include <iostream>
using namespace std;
int main(){
    int n1, n2, suma;
    cin >> n1;
    cin >> n2;
    suma = sumar(n1, n2);
    cout << endl << suma;
    return 0;
}
```



Llamado a la función *sumar*

- El objetivo de la función **sumar** es sumar dos números.
- Recibe como dos números enteros que sumará
- Devuelve como valor de retorno un valor **int** con el resultado de la suma.

Declaración

```
tipo nombreFuncion (parámetros);
```

La **declaración** de una función debe determinar el tipo de dato que devuelve primero.

Luego cómo se llamará la función.

Por último, los tipos de datos de los parámetros que recibirá separados por coma. Si no recibe parámetros no se pone nada entre los paréntesis.

Declaración de una función

Un ejemplo de una declaración de función puede ser la siguiente:

```
bool es_primo(int);
```

Declaración de función que recibe un entero y devuelve un valor bool.

```
bool suma(int nro1, int nro2);
```

Declaración de función que recibe dos enteros y devuelve un valor entero. También es válido nombrar la/s variable/s de los parámetros.

```
void mostrar_primos_entre(int, int);
```

Declaración de función que recibe dos enteros y no devuelve nada.

Definición

```
tipo nombreFuncion (parámetros) {  
    // Código de la función  
}
```

La **definición** de una función es similar a la **declaración** pero se nombran las variables en los parámetros y se establece lo que hace la función entre las llaves de la misma.

Definición de una función

```
int sumar(int nro1, int nro2){  
    int resultado;  
    resultado = nro1 + nro2;  
    return resultado;  
}
```

Definición de la función *sumar*

- La función recibe los parámetros enteros y los declara como **nro1** y **nro2**.
- La función declara la variable local **resultado** para realizar su tarea.
- La función **devuelve** el contenido de resultado que contiene la suma de ambos números.

Ámbito de variables

Una variable es local al **ámbito** en el que fue declarada. El ámbito está dado por las llaves. Fuera de ellas la variable no existe.

```
int main(){
    int i=0, numero= 9;
    float val;
    val = mi_funcion(numero);
    return 0;
}
```

Variable	Valor

```
float mi_funcion(int dato){
    float i;
    i = dato / 2;
    return i;
}
```

Variable	Valor

Ámbito de variables

Una variable es local al **ámbito** en el que fue declarada. El ámbito está dado por las llaves. Fuera de ellas la variable no existe.

```
int main(){  
    int i=0, numero= 9;  
    float val;  
    val = mi_funcion(numero);  
    return 0;  
}
```

Variable	Valor
i	0
numero	9
val	4.5

```
float mi_funcion(int dato){  
    float i;  
    i = (float) dato / 2;  
    return i;  
}
```

Variable	Valor
dato	9
i	4.5

Retornar valores

Una función puede retornar un valor o ninguno. Esto se logra mediante la palabra reservada **return**.

Si se encuentra un `return` en una función, la misma finaliza y devuelve el valor indicado a *quien* la llamó.

Ejemplo: *return valor;*

En los casos de funciones que no devuelven nada. El funcionamiento y la lógica son iguales sólo que no hay que indicar ningún valor al llamar a `return`.

Ejemplo: *return;*

Parámetros

Una función puede recibir ninguno, uno o varios parámetros. Los parámetros recibidos deben permitir a la función procesarlos y asistir en la tarea que la función debe realizar.

Un parámetro a una función puede ser recibido por **valor**, **referencia** o **dirección**.

Parámetros por valor

Un parámetro enviado por valor permite que la función reciba una copia del valor de la variable/constante original o bien un valor constante explícito.

Al ser una copia del valor, cualquier cambio realizado al contenido de la variable recibida no afectará al valor de la variable original.

```
int main(){  
    int bart = 1000;  
    mi_funcion(bart);  
    cout << bart; // Mostrará 1000  
}
```

```
void mi_funcion(int bort){  
    cout << bort; // Mostrará 1000  
    bort = 10;  
    cout << bort; // Mostrará 10  
}
```

Parámetros por referencia

Un parámetro enviado por referencia permite que la función reciba la variable original. Es decir, un alias o sinónimo de la variable original.

Al ser una referencia, cualquier cambio realizado al contenido de la variable recibida afectará al valor de la variable original.

```
int main(){
    int bart = 1000;
    mi_funcion(bart);
    cout << bart; // Mostrará 10
}
```

```
void mi_funcion(int &bort){
    cout << bort; // Mostrará 1000
    bort = 10;
    cout << bort; // Mostrará 10
}
```

Funciones

Ventajas de utilizar funciones:

- Dividir un problema grande en pequeños problemas más chicos. Abstrayendo esa tarea del problema general a resolver.
- Dar legibilidad al código, haciendo algoritmos más fáciles de entender y mantener.
- Dar capacidad de reutilización al código. Haciendo funciones que resuelven tareas únicas aumenta la posibilidad de poder utilizarlas nuevamente.
- Evita la repetición de código. Cada vez que necesito realizar la misma tarea puedo llamar nuevamente a la función.
- Modificar el funcionamiento de un programa o encontrar errores es más fácil si tenemos la lógica separada en funciones.

~~Ejemplo~~

Ejercicios

- ✓ - Hacer un programa que determine si un número ingresado es primo de Sophie Germain o no.
- ✓ - Hacer un programa que muestre los números primos entre el 1 y el 1000. Deben aparecer con un asterisco aquellos que sean de Sophie Germain.
- ✓ - Hacer una función llamada incrementarDiaAFecha que reciba un día, un mes y un año. Y agregue un día a la fecha. La función debe alterar uno, dos o los tres parámetros según corresponda.

- Un primo de Sophie Germain es un número primo 'p' tal que $2p + 1$ también es primo. En otras palabras, si duplicas un número primo y le sumas 1, y el resultado es también primo, entonces el número original es un primo de Sophie Germain
- $2 \rightarrow 2 * 2 + 1 = 5$ (5 es primo), por lo que 2 es un primo de Sophie Germain.
- $3 \rightarrow 2 * 3 + 1 = 7$ (7 es primo), por lo que 3 es un primo de Sophie Germain.
- $7 \rightarrow 7 * 2 + 1 = 15$ (15 no es primo), por lo que 15 no es un primo de Sophie Germain.