Funciones y Parámetros en C++

Contenido

1. Definición y llamada de función

Una **función** en C++ es un bloque de código que realiza una tarea específica.
Consta de:

- **Tipo de retorno**
- **Nombre**
- **Parámetros (opcionales)**
- **Cuerpo**

Importancia

Separar el código en funciones aumenta la **modularidad**, la **reutilización** y la **legibilidad**.

Ejemplo

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

// Función que calcula el cuadrado de un número
int cuadrado(int x) {
    return x * x;
}

int main() {
    int numero = 7;
    cout << "El cuadrado de " << numero << " es " << cuadrado(numero) << endl;
    return 0;
}
```

2. Parámetros de una función

Los parámetros permiten enviar datos a una función. Una función puede tener:

- Ningún parámetro
- Uno o varios
- Parámetros con valor por defecto

Los parámetros hacen las funciones más flexibles y potentes.

Ejemplo

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

// Función que calcula la potencia con exponente opcional
int potencia(int base, int exponente = 2) {
    int resultado = 1;
    for (int i = 0; i < exponente; i++)
        resultado *= base;
    return resultado;
}

int main() {
    cout << potencia(5) << endl;      // Usa exponente por defecto (2)
    cout << potencia(5, 3) << endl;   // Exponente específico
    return 0;
}
```

3. Paso por valor

En el paso por valor, se envía una copia del dato. La variable original no cambia aunque dentro de la función se modifique.

Es útil cuando queremos proteger los datos originales.

Ejemplo

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

void duplicar(int x) {
    x *= 2;
    cout << "[Dentro] x = " << x << endl;
}

int main() {
    int n = 10;
    duplicar(n);
    cout << "[Fuera] n = " << n << endl; // No cambia
    return 0;
}
```

4. Paso por referencia

En el paso por referencia, se envía la dirección de memoria. La función puede modificar directamente la variable original.

Evita copias innecesarias y permite modificar datos.

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

void normalizar(double &x) {
    if (x < 0) x = -x;
}

int main() {
    double valor = -12.7;
    normalizar(valor);
    cout << "Valor normalizado: " << valor << endl; // Cambia
    return 0;
}
```

5. Parámetros tipo arreglos

Los arreglos se pasan por referencia implícita, ya que su nombre representa un puntero al primer elemento.

Modificar el arreglo dentro de la función lo modifica también fuera.

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

void incrementarTodos(int arr[], int tam) {
    for (int i = 0; i < tam; i++)
        arr[i]++;
}

int main() {
    int datos[] = {3, 6, 9, 12};

    incrementarTodos(datos, 4);

    cout << "Arreglo modificado: ";
    for (int x : datos) cout << x << " ";
    return 0;
}
```

6. Parámetros tipo matrices

Una matriz requiere que al menos una dimensión esté fijada para que C++ pueda calcular su desplazamiento en memoria.

Se almacenan en forma contigua y por filas.

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

```cpp
void transponer(int m[][3], int filas) {
    cout << "Matriz transpuesta:" << endl;
    for (int j = 0; j < 3; j++) {
        for (int i = 0; i < filas; i++)
            cout << m[i][j] << " ";
        cout << endl;
    }
}

int main() {
    int matriz[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    transponer(matriz, 2);

    return 0;
}
```

7. Ejercicios Propuestos

1. Crear una función `esPrimo(int n)` que determine si un número es primo. Optimizar usando límite sqrt(n).
2. Crear una función `rotarArreglo(int arr[], int n)` que rote todos los elementos una posición a la derecha.
3. Crear una función que reciba una matriz 3×3 y calcule su determinante.
4. Crear una función que reciba un número por referencia y lo convierta en su valor absoluto.
5. Implementar una función que reciba un arreglo y devuelva el segundo elemento mayor.