PROCEDIMIENTOS Y FUNCIONES EN C++

ÍNDICE

1 Procedimientos y funciones en C++.

1.1 Diferencia entre procedimiento y función.

Un **procedimiento** es una porción de código que realiza una tarea específica, pero no devuelve un valor. Se utiliza para realizar acciones como imprimir en pantalla, modificar variables, etc. No retorna ningún valor al punto de llamada.

Una **función** es similar a un procedimiento, pero devuelve un valor. Se utiliza para realizar cálculos y devolver un resultado. Retorna un valor al punto de llamada.

1.2 Declaración y llamadas a procedimientos en C++.

Declaración:

cpp Copy code

```cpp
// Declaración de procedimiento
void imprimirMensaje();

// Declaración de procedimiento con parámetros
void sumar(int a, int b);
```

Llamada:

cpp Copy code

```cpp
// Llamada a procedimiento
imprimirMensaje();

// Llamada a procedimiento con parámetros
sumar(5, 3);
```

1.3 Declaración y llamadas a funciones en C++.

Funciones:

A diferencia de los procedimientos, las funciones en C++ devuelven un valor. Aquí tienes un ejemplo de una función que calcula el cuadrado de un número:

cpp Copy code

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

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

int main() {
    int numero = 5;
    int resultado = calcularCuadrado(numero);
    cout << "El cuadrado de " << numero << " es: " << resultado << endl;
    return 0;
}
```

En este ejemplo, la función **calcularCuadrado()** toma un parámetro x y devuelve el cuadrado de ese número. Cuando se llama a la función **calcularCuadrado()** con numero como argumento, devuelve el cuadrado de **numero**, que luego se almacena en la variable resultado.

1.4 Ámbito de las Variables en C++:

El ámbito de una variable en C++ se refiere a la parte del código donde esa variable es válida y puede ser utilizada. Hay varios niveles de ámbito, y la visibilidad de una variable depende de dónde se declara.

Ámbito global:

- Las variables globales se declaran fuera de cualquier función o bloque de código.
- Son accesibles desde cualquier parte del programa.
- Tienen una duración que abarca toda la ejecución del programa.

Ámbito de función:

- Las variables locales a una función se declaran dentro de esa función.
- Solo son accesibles dentro de la función en la que se declaran.
- Se crean cuando la función es llamada y se destruyen cuando la función termina su ejecución.

Ámbito de bloque:

- Un bloque de código se define entre llaves {}.
- Las variables declaradas dentro de un bloque son locales a ese bloque.
- Solo son accesibles dentro de ese bloque y de bloques internos.
- Se crean al entrar al bloque y se destruyen al salir del bloque.

Ejemplo:

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

int globalVar = 10; // Variable global

void ejemploFuncion() {
    int localVarFuncion = 20; // Variable local a la función
    cout << "globalVar desde la función: " << globalVar << endl;
    cout << "localVarFuncion: " << localVarFuncion << endl;
}

int main() {
    int localVarMain = 30; // Variable local a la función main
    cout << "globalVar desde main: " << globalVar << endl;
    cout << "localVarMain: " << localVarMain << endl;
    ejemploFuncion();

    {
        int localVarBloque = 40; // Variable local a un bloque
        cout << "localVarBloque: " << localVarBloque << endl;
    } // localVarBloque deja de existir al salir del bloque

    // cout << "localVarBloque fuera del bloque: " << localVarBloque << endl; // Esto

    return 0;
}
```

En este ejemplo:

- **globalVar** es una variable global, accesible desde cualquier parte del programa.
- **localVarMain** es una variable local a la función main(), solo accesible dentro de main().
- **localVarFuncion** es una variable local a la función ejemploFuncion(), solo accesible dentro de ejemploFuncion().
- **localVarBloque** es una variable local a un bloque específico dentro de main(), solo accesible dentro de ese bloque.

1.5 Paso de parámetros.

El paso de parámetros en C++ puede realizarse de varias maneras, siendo las más comunes:

- **Por valor:** Se pasa una copia del valor del parámetro al procedimiento o función. Cualquier modificación realizada en el parámetro dentro del procedimiento o función no afectará al argumento original.

- **Por referencia:** Se pasa una referencia al argumento original. Esto significa que cualquier modificación realizada en el parámetro dentro del procedimiento o función afectará al argumento original.

- **Por puntero:** Se pasa la dirección de memoria del argumento original. Esto permite a la función o procedimiento acceder directamente al valor del argumento original y modificarlo si es necesario.

A continuación, proporcionaré ejemplos para cada uno de estos métodos de paso de parámetros:

Paso de parámetros por valor:

cpp Copy code

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

void duplicarPorValor(int num) {
    num *= 2;
    cout << "Dentro de la función, el valor duplicado es: " << num << endl;
}

int main() {
    int numero = 5;
    cout << "Número original: " << numero << endl;
    duplicarPorValor(numero);
    cout << "Número después de la función: " << numero << endl;
    return 0;
}
```

Este código imprimirá:

mathematica Copy code

```
Número original: 5
Dentro de la función, el valor duplicado es: 10
Número después de la función: 5
```

Observa que el valor de `numero` no cambia fuera de la función, ya que solo se modificó una copia de su valor dentro de la función.

Paso de parámetros por referencia:

cpp Copy code

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

void duplicarPorReferencia(int &num) {
    num *= 2;
    cout << "Dentro de la función, el valor duplicado es: " << num << endl;
}

int main() {
    int numero = 5;
    cout << "Número original: " << numero << endl;
    duplicarPorReferencia(numero);
    cout << "Número después de la función: " << numero << endl;
    return 0;
}
```

Este código imprimirá:

mathematica Copy code

```
Número original: 5
Dentro de la función, el valor duplicado es: 10
Número después de la función: 10
```

En este caso, el valor de `numero` cambia fuera de la función porque se pasó por referencia y se modificó directamente.

Paso de parámetros por puntero:

cpp Copy code

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

void duplicarPorPuntero(int *numPtr) {
    *numPtr *= 2;
    cout << "Dentro de la función, el valor duplicado es: " << *numPtr << endl;
}

int main() {
    int numero = 5;
    cout << "Número original: " << numero << endl;
    duplicarPorPuntero(&numero);
    cout << "Número después de la función: " << numero << endl;
    return 0;
}
```

Este código imprimirá lo mismo que el ejemplo de paso por referencia, ya que `numPtr` apunta directamente a la dirección de memoria de `numero` y permite modificar su valor.

```cpp
        switch (opcion) {
            case 1:
                cout << "¡Hola! ¡Bienvenido al programa!" << endl;
                break;
            case 2:
                cout << "¡Hasta luego! Que tengas un buen día." << endl;
                break;
            case 3:
                cout << "Saliendo del programa..." << endl;
                return 0; // Terminar el programa
            default:
                cout << "Opción inválida. Por favor, seleccione una opción válida." <
        }
    }

    return 0;
}
```