3.RECURSIVIDAD

ÍNDICE

1 Recursividad C++.

La **recursividad es un concepto importante en la programación** y se refiere a la capacidad de una función para llamarse a sí misma. Esto puede ser útil cuando se necesita resolver un problema que se puede dividir en casos más simples y similares al problema original. Aquí tienes un ejemplo en C++:

1.1 Características de una función recursiva.

- **Llamada a sí misma:** La característica principal de una función recursiva es que se llama a sí misma dentro de su definición. Esto permite que la función se repita y resuelva el problema de forma iterativa.

- **Caso base:** Toda función recursiva debe tener al menos un caso base. El caso base es una condición que detiene la recursión, evitando que la función continúe llamándose a sí misma indefinidamente. Sin un caso base, la función entraría en un bucle infinito.

- **División del problema en subproblemas más pequeños:** Las funciones recursivas dividen el problema original en subproblemas más pequeños y similares. Estos subproblemas deben ser lo suficientemente simples para que eventualmente se alcance el caso base.

1.2 Otras características de la recursividad.

- **Proceso de convergencia:** En cada llamada recursiva, el problema se mueve hacia el caso base. Esto significa que en cada paso de la recursión, el problema se vuelve más simple o más pequeño, acercándose eventualmente al caso base.

- **Uso eficiente de la pila de llamadas:** Cada vez que se llama a una función, se crea un nuevo marco de pila en la memoria conocido como "pila de llamadas". En el caso de la recursión, cada llamada a la función se apila en la pila de llamadas. Cuando se alcanza el caso base, las llamadas recursivas comienzan a desapilarse y a resolver los problemas de forma ascendente.

- **Costo en memoria y tiempo:** Aunque la recursión puede ser una forma elegante de resolver problemas, también puede tener un costo en términos de uso de memoria y tiempo de ejecución. Debido a la creación de múltiples marcos de pila, las funciones

recursivas pueden consumir más memoria que las versiones iterativas del mismo problema. Además, en algunos casos, la recursión puede ser menos eficiente en términos de tiempo de ejecución debido a la sobrecarga asociada con la gestión de la pila de llamadas.

- **Facilidad de comprensión y mantenimiento del código:** En muchos casos, el uso de la recursión puede simplificar la implementación y comprensión del código, especialmente para problemas que naturalmente se prestan a una solución recursiva. Sin embargo, en otros casos, una solución iterativa puede ser más clara y eficiente.

2 Ejemplos de recursividad en C++.

Este código calcula el **factorial** de un número de forma recursiva. La función factorial() se llama a sí misma para calcular el factorial de n-1 y así sucesivamente hasta que n sea 0 o 1 (casos base), momento en el que la recursión se detiene.

```cpp
#include <iostream>

// Función recursiva para calcular el factorial de un número
int factorial(int n) {
    if (n == 0 || n == 1) { // Caso base: el factorial de 0 y 1 es 1
        return 1;
    } else {
        // Llamada recursiva para calcular el factorial de n-1
        return n * factorial(n - 1);
    }
}

// Función principal
int main() {
    int numero;
    std::cout << "Introduce un número para calcular su factorial: ";
    std::cin >> numero;

    // Llamada a la función factorial
    int resultado = factorial(numero);

    std::cout << "El factorial de " << numero << " es: " << resultado << std::endl;

    return 0;
}
```

Aquí hay otro ejemplo que calcula la suma de los primeros n números naturales:

En este caso, la función **sumaNaturales()** se llama a sí misma para calcular la suma de los primeros n-1 números naturales y así sucesivamente hasta que n sea 0 (caso base), momento en el que la recursión se detiene.

```cpp
#include <iostream>

// Función recursiva para calcular la suma de los primeros n números naturales
int sumaNaturales(int n) {
    if (n == 0) { // Caso base: la suma de 0 elementos es 0
        return 0;
    } else {
        // Llamada recursiva para calcular la suma de n-1 números naturales
        return n + sumaNaturales(n - 1);
    }
}

// Función principal
int main() {
    int numero;
    std::cout << "Introduce un número para calcular la suma de los primeros n números
    std::cin >> numero;

    // Llamada a la función sumaNaturales
    int resultado = sumaNaturales(numero);

    std::cout << "La suma de los primeros " << numero << " números naturales es: " <<

    return 0;
}
```