**Programas Recursivos**

1. **Introducción**

Los programas recursivos son una parte fundamental de la programación que aprovechan la naturaleza auto-referencial de las funciones para resolver problemas de manera elegante y eficiente. En este documento, exploraremos varios ejemplos de programas recursivos, analizando cómo funcionan y por qué la recursividad puede ser preferible a la iteración en ciertos casos.

1. **Ejemplos de Programas Recursivos**

**2.1. Cálculo del Factorial**

El cálculo del factorial es un ejemplo clásico de un problema que se puede resolver de manera recursiva. La función factorial de un número *n*, denotada como *n!*, se define como el producto de todos los enteros positivos menores o iguales a *n*. Aquí está la versión recursiva de la función factorial:

int factorial(int n) {

**if** (n == **0**) {

**return** **1**;

} **else** {

**return** n \* factorial(n - **1**);

}

}

**2.2. Torres de Hanoi**

Las pilas de Hanoi son un sistema recursivo que saca la pila de un lado para llevarlo a otra pila de destino, esto funciona sacando los factores de la pila original para una pila auxiliar y luego a una pila de destino Aquí la representación funcional de las pilas de Hanoi:

void hanoi(int n, char origen, char auxiliar, char destino) {

**if** (n == **1**) {

std::cout << "Mover disco 1 desde " << origen << " hacia " << destino << std::endl;

**return**;

}

hanoi(n - **1**, origen, destino, auxiliar);

std::cout << "Mover disco " << n << " desde " << origen << " hacia " << destino << std::endl;

hanoi(n - **1**, auxiliar, origen, destino);

}

**2.3. Cálculo de la Secuencia de Fibonacci**

El cálculo de la secuencia de Fibonacci es otro ejemplo clásico de un problema que se puede resolver de manera recursiva. La secuencia de Fibonacci es una serie de números en la que cada número es la suma de los dos anteriores. Aquí está la versión recursiva de la función Fibonacci:

int fibonacci(int n) {

**if** (n <= **1**) {

**return** n;

} **else** {

**return** fibonacci(n-**1**) + fibonacci(n-**2**);

}

}

1. **Funcionamiento de los Programas Recursivos**

En un programa recursivo, la función se llama a sí misma repetidamente con argumentos diferentes hasta que se alcanza una condición de terminación, también conocida como caso base. En ese punto, las llamadas recursivas se detienen y las funciones comienzan a devolver valores hacia atrás a través de las llamadas anidadas, eventualmente devolviendo un resultado final.

Por ejemplo, en el caso del cálculo del factorial, la función se llama a sí misma con argumentos decrecientes hasta que alcanza el caso base n=0n=0, momento en el cual devuelve 1. Luego, los valores se multiplican recursivamente a medida que se desenrollan las llamadas recursivas hasta llegar al resultado final.

1. **Ventajas de la Recursividad sobre la Iteración**

La recursividad puede ser preferible a la iteración en ciertos casos por varias razones:

Claridad y Simplicidad: En algunos problemas, la solución recursiva puede ser más clara y concisa que su contraparte iterativa, lo que hace que el código sea más fácil de entender y mantener.

Naturaleza del Problema: Algunos problemas tienen una estructura naturalmente recursiva, lo que hace que la solución recursiva sea más intuitiva y directa.

Facilidad de Implementación: En ciertos escenarios, la solución recursiva puede ser más fácil de implementar y menos propensa a errores que la iterativa.

Sin embargo, es importante tener en cuenta que la recursividad también puede tener algunas desventajas, como un mayor consumo de memoria y la posibilidad de desbordamiento de pila en casos de profundidad recursiva excesiva.

**5. Referencias**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2019). "Introduction to Algorithms." MIT Press.

Sedgewick, R., & Wayne, K. (2021). "Algorithms." Addison-Wesley.