

Estructura de Datos

1

RECURSIVIDAD

Recursividad

2

Se dice que una función es recursiva cuando se define en función de si misma.

No todas la funciones pueden llamarse a si mismas, sino que deben estar diseñadas especialmente para que sean recursivas, de otro modo podrían conducir a bucles infinitos, o a que el programa termine inadecuadamente.

Tampoco todos los lenguajes de programación permiten usar recursividad.

C permite la recursividad. Cada vez que se llama a una función, **se crea un juego de variables locales, de este modo, si la función hace una llamada a si misma,** se guardan sus variables y parámetros, **usando la pila,** y la nueva instancia de la función trabajará con su propia copia de las variables locales. Cuando esta segunda instancia de la función retorna, recupera las variables y los parámetros de la pila y continua la ejecución en el punto en que había sido llamada.

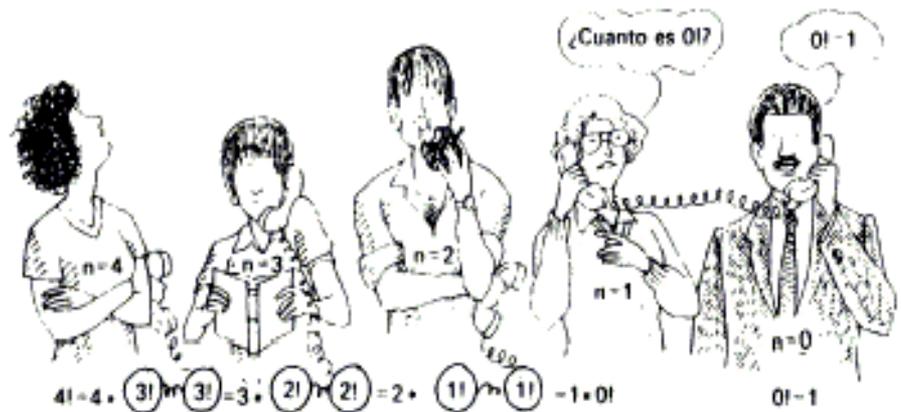
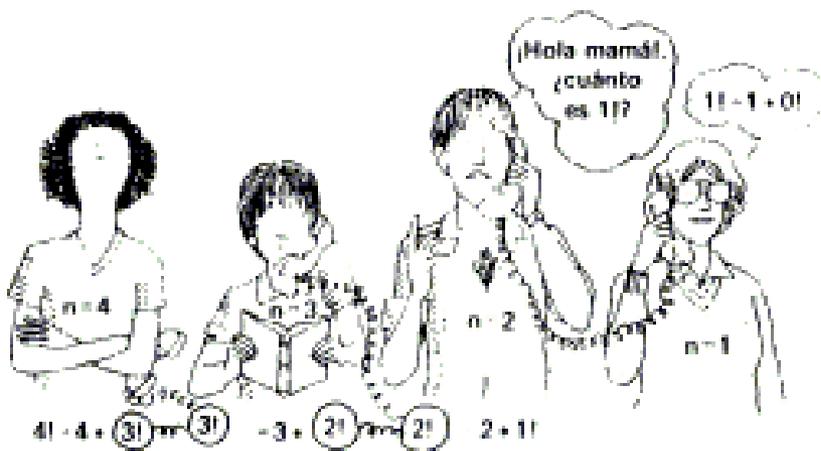
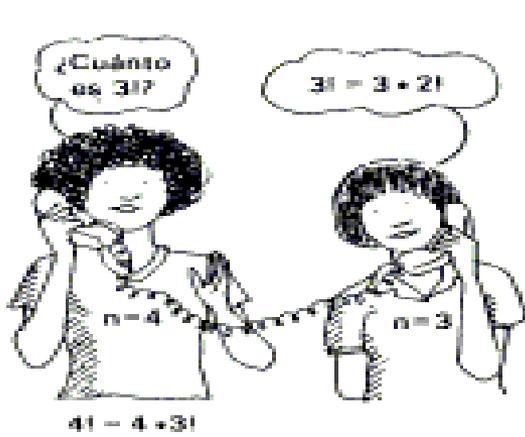
Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.

Caso recursivo:

Una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base.

Los pasos que sigue el caso recursivo son los siguientes:

1. El procedimiento se llama a sí mismo
2. El problema se resuelve, resolviendo el mismo problema pero de tamaño menor
3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará



Ejemplo de Recursividad

6



Función recursiva que calcula n factorial.

```
int Factorial(int n){  
    if (n==0)  
        n=1;  
    else  
        n=n*Factorial(n-1);  
    return(n);  
}
```

Ejemplo de Recursividad

7

- La Matrushka es una artesanía tradicional rusa. Es una muñeca de madera que contiene otra muñeca más pequeña dentro de sí. Esta muñeca, también contiene otra muñeca dentro. Y así, una dentro de otra.



Ejemplo de Recursividad

8

Prodríamos crear una función recursiva para calcular el factorial de un número entero.

El factorial se simboliza como $n!$, se lee como "n factorial", y la definición es:

$$n! = n * (n-1) * (n-2) * \dots * 1$$

Hay algunas limitaciones:

- No es posible calcular el factorial de números negativos, no está definido.
- El factorial de cero es 1.

De modo que una función bien hecha para cálculo de factoriales debería incluir un control para esos casos:

Funcion Factorial(n)

```
if n == 1 entonces return 1
else return n * factorial(n-1)
```

Sucesión Fibonacci

9

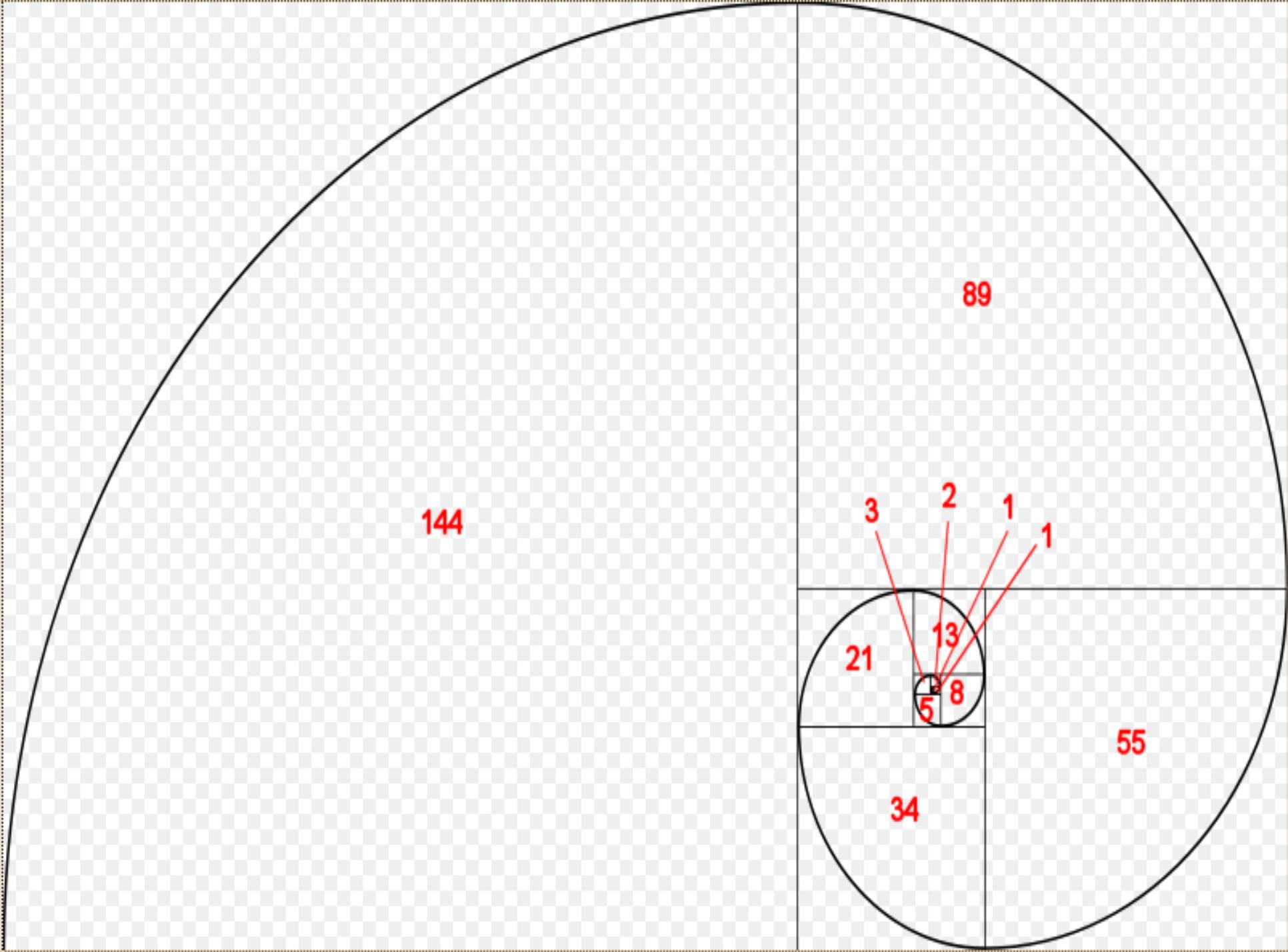
En matemáticas, la **sucesión de Fibonacci** es la siguiente sucesión infinita de números naturales:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144...

El primer elemento es 0, el segundo es 1 y cada elemento restante es la suma de los dos anteriores:

$$f_i = \begin{cases} 0 & \text{si } i = 0 \\ 1 & \text{si } i = 1 \\ f_{(i-2)} + f_{(i-1)} & \text{si } i > 1 \end{cases}$$

A cada elemento de esta sucesión se le llama **número de Fibonacci**



144

89

55

34

3

2

1

1

21

13

8

5

Torres de Hanoi

11

1. Sólo se puede mover un disco cada vez.
2. Un disco de mayor tamaño no puede descansar sobre uno más pequeño que él mismo.
3. Sólo puedes desplazar el disco que se encuentre arriba en cada varilla.



¿Cuándo usar recursividad?

- Para simplificar el código.
- Cuando la estructura de datos es recursiva ejemplo : árboles.

¿Cuándo no usar recursividad?

- Cuando los métodos usen arreglos largos.
- Cuando el método cambia de manera impredecible de campos.
- Cuando las iteraciones sean la mejor opción.