Recursividad

Recursividad es el proceso por el que algo se define en términos de sí mismo. Cuando se aplica a lenguajes de programación, recursividad significa que una función se puede llamar a sí misma. Por tanto, una función se dice que es recursiva si en una sentencia del cuerpo de la función se llama a sí misma. No todos los lenguajes de ordenador soportan funciones recursivas. En este programa se muestra un ejemplo sencillo de recursividad:

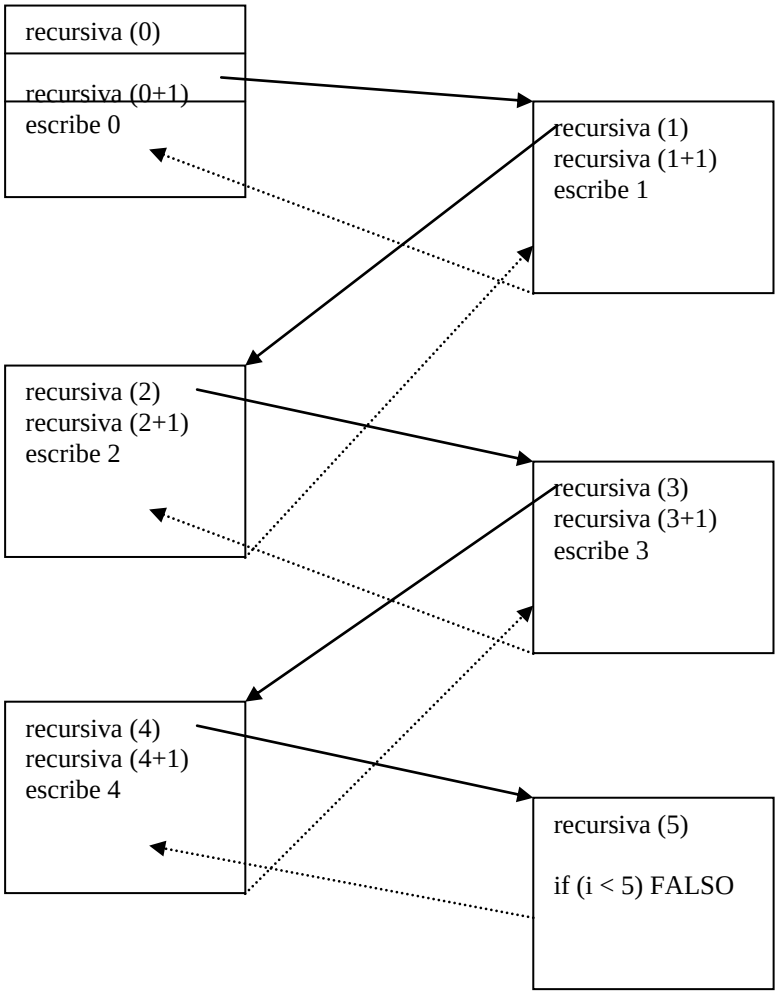
```
/* * recursivad
  */
public class progrecur
{

        static void frecursiva (int i )                 // función recursiva prototipo
        {
                if (i < 5)                              // condición de "finalización"
                {
                        frecursiva (i+1);               // llamada a la propia función desde la misma: recursividad
                         System.out.println ( i );
                }
        } // fin de la función recursiva

        public static void main(String[] args)
        {

          frecursiva (0);                       // llamada desde la función principal a la función recursiva

         } // fin de la función principal

}
```

Este programa imprime 4 3 2 1 0 en la pantalla. Veamos por qué:

Primero se llama a la función recursiva ( ) con 0. Esta es la primera activación de recursiva( ). Dado que 0 es menor que 5, entonces recursiva ( ) se llama a sí misma con el valor de i (en este caso 0) más 1. Esta es la segunda activación de recursiva ( ), e i vale 1. Esto hace que se llama otra vez a recursiva ( ) utilizando el valor 2. Este proceso se repite hasta que se llama a frecursiva () con e valor 5. Esto hace que se vuelva de recursiva ( ). Puesto que vuelve al punto de su llamada, ejecutará la sentencia println ( ) en su activación previa, imprime 4 y regresa. Entonces, ésta vuelve al punto de su llamada en la activación anterior, que hace que se muestre 3. El proceso continúa hasta que se vuelve de todas las llamadas y el programa termina.

Es importante comprender que no existen múltiples copias de una función recursiva. Solamente existe una copia. Cuando se llama a una función, se asigna un espacio en la pila para sus parámetros y datos locales. Por eso, cuando se llama recursivamente a una función, la función se empieza a ejecutar con un nuevo conjunto de parámetros y variables locales, pero el código que constituye la función permanece idéntico.

Si considera el programa anterior, observará que la recursividad es básicamente un nuevo tipo de mecanismo de control de programas. Ese es el motivo por el que toda función recursiva que se escriba tendrá una sentencia if que controle si la función se llamará otra vez a sí misma o regresará. Sin una sentencia así, una función recursiva se ejecutará alocadamente, utilizando toda la memoria asignada a la pila y haciendo que él programa falle.



Por ejemplo: en el siguiente código la función se llamará a si misma repetidamente hasta que falle el programa, ya que no hay una condición que evite la llamada recursiva.

```
void f ( )
{
        int i;
        System.out.println (" en f()  \n");
                                /* llamar a f() 10 veces  */
        for (i=0; i < 10; i++)
                f( );
}
```

La recursividad normalmente se usa de forma comedida. Sin embargo, puede ser muy útil para simplificar ciertos algoritmos. Por ejemplo, es difícil implementar el algoritmo de ordenación Quicksort sin utilizar la recursividad.

El ejemplo clásico de función recursiva es el cálculo del factorial de un número entero. Para realizar una función que calcule el factorial, en principio no hace falta necesariamente una función recursiva. Pero si consideramos que el factorial de un número se puede calcular como :

factorial (n) = n * factorial (n-1)

La función recursiva es mucho más intuitiva:

```
static int factorial (int n)
{
int respuesta;
        if (n == 1)
                respuesta =1;
        else
                respuesta = n * factorial  ( n – 1 );
return respuesta;
}
```

Se pueden distinguir dos tipos de recursividad:

- Recursividad directa: es aquella en la que la función realiza una llamada a si misma desde un punto concreto entre sus llaves de inicio y fin.
- Recursividad indirecta: es aquella en la que la función realiza una invocación a otra función que es quien llama a la primera.

En general, para resolver un caso recursivo generalmente debemos encontrar:

1. Una formula o proceso que reduzca la complejidad y que nos vaya acercando a la solución.
2. Un caso base que haga que nuestra recursividad no sea infinita.

Por ejemplo:

1.- $base^{exp} = base * base^{exp-1}$

2.- $base^{0} = 1$

CASI SIEMPRE, debemos poner un condicional simple/doble que nos salte al caso base para que la recursividad no sea infinita.