RECURSIÓN

Definición

Un procedimiento o función se dice recursivo si durante su ejecución se invoca directa o indirectamente a sí mismo. Esta invocación depende al menos de una condición que actúa como condición de corte que provoca la finalización de la autoinvocación.

Definición

Repetición por autoreferencia, cuando una función se invoca a sí misma (puede ser en forma directa o indirecta).

Es la forma en la cual se especifica un proceso basado en su propia definición, pero con menor complejidad. De esta forma, en algún momento se llega a un proceso muy simple que puede resolverse facilmente (se llama solución trivial).

Parece un concepto nuevo, pero ya lo hemos visto en varias definiciones matemáticas…

Definiciones por inducción

- *El uno pertenece a los Naturales, y si N pertenece a los Naturales, N+1 también.*

- *Factorial(0) = 1 (solución trivial)*

  *Factorial(N) = N * Factorial(N-1), para todo N > 0.*

- *0 ! = 1*

  *N ! = N * (N - 1) !*

Desarrollo de la función factorial

```
int factorial (int x)
{
   int rta;
   If ( x = = 0)
         rta = 1;
   else
         rta = x * factorial (x-1) ;
   return rta;
}
```

```
z = factorial(3)
{  int rta;
   if (3 = = 0)  rta = 1;
   else
      rta= 3 * factorial(2)
               {  int rta;
                  if (2 = = 0)  rta = 1;
                  else
                     rta= 2 * factorial(1)
                              {  int rta;
                                 if (1 = = 0)  rta = 1;
                                 else
                                    rta= 1 *
```

```
                              return rta;
                  return rta; }
               }
   return rta;
}
```

Cómo funciona esto ??

En el momento de realizar una invocación recursiva, se suspende la ejecución de la función invocante hasta que se termina de resolver la función invocada.

Si bien es la misma función la que se invoca, se genera un nuevo espacio de memoria para la resolución de la misma.

Cada espacio tiene sus propias variables locales y parámetros, con valores propios.

Para poder resolver **factorial(3),** primero se debe resolver factorial(2), pero…

Para poder resolver **factorial(2),** primero se debe resolver factorial(1), pero…

Para poder resolver **factorial(1),** primero se debe resolver factorial(0), y…

factorial(0) es 1

Reglas de la buena recursión

Toda función recursiva debe tener:

1. Al menos una Condición de Corte con su respectiva Solución Trivial.
2. Al menos una Llamada Recursiva (o Entrada en Recursión).
3. En cada Llamada Recursiva, se vuelve a invocar el mismo algoritmo, pero con un caso más simple de resolver. Se produce un acercamiento a la Condición de Corte.
4. Al llegar a la Solución Trivial, queda expresada la solución total (considerando toda la fórmula que quedó pendiente de resolver).

```
int factorial (int x)
{
   int rta;
   If ( x = = 0)
        rta = 1;
   else
        rta = x * factorial (x-1) ;
   return rta;
}
```

Condición de Corte

Solución Trivial

Acercamiento a la Condición de Corte

Llamada Recursiva

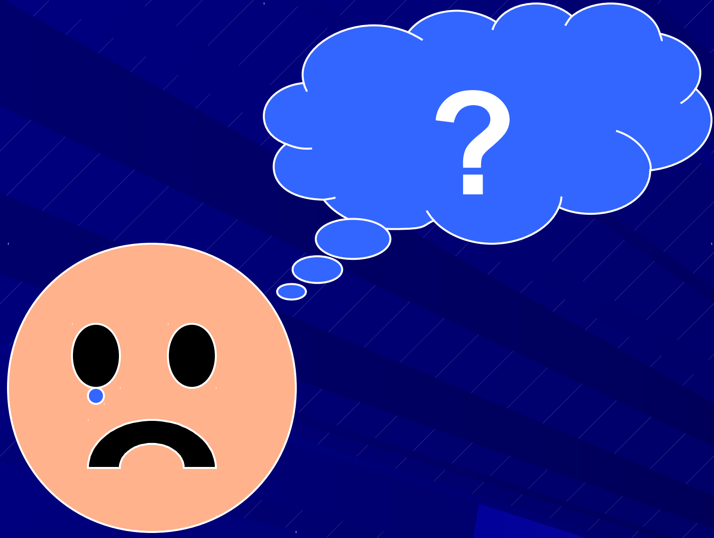
Usar recursión cuando:

1. La estructura de la función es recurrente y el algoritmo resulta más sencillo.
2. La estructura de datos es recursiva.

Definición de Lista

estructura de datos recursiva

El NULL es una Lista y además, un Nodo seguido de una Lista también es Lista.



Ejemplo: recorrer una lista vinculada

Se trata de una estructura de datos recursiva. Considerar la estructura que define al nodo.

```
void recorrer( Nodo * lista)
{
    if (lista != NULL)
    {
        printf("%d", lista->dato);
        recorrer(lista->siguiente);
    }
}
```

Condición de Corte
Corta cuando es falso

Acercamiento a la Condición de Corte

Llamada Recursiva

Solución Trivial: ausencia de ELSE. No hace nada al llegar a NULL

Qué hace el siguiente código?

```
void recorrer( Nodo * lista)
{
   if (lista != NULL)
   {
   recorrer(lista->siguiente);
   printf("%d", lista->dato);
   }
}
```

Tener en cuenta que para recorrer la lista NO hace falta usar while ni for. Ni ningún otra estructura de repetición. En vez de eso, se tiene que:

1. La recursión se encarga de repetir el algoritmo (printf en este caso).
2. La Llamada Recursiva con el acercamiento a la condición de corte hace que se avance en la lista.
3. La condición de corte finaliza el recorrido de la lista, justo cuando la lista se termina.
4. La solución trivial resuelve el algoritmo para la lista más simple, o sea la lista nula. Y la solución es no hacer nada, en este caso no imprimir.

Cómo se piensa en forma recursiva ?

1. **Encontrar la solución trivial.**
2. **Armar la expresión de la solución total del problema, contando con la solución al problema para una etapa “una vez mas cercana” a la solución trivial.**

Las reglas de la buena recursión completan todo lo que falta para que esta forma simple de razonamiento dé resultado.

Ejemplo: sumar el contenido de una lista de enteros.

Para solucionar este problema planteo el siguiente algoritmo:

"La suma de una lista vacía es 0"

"La suma de los números de una lista será igual a la suma entre el primero y el resultado de la suma de la sublista siguiente."

En este caso particular del ejemplo de la lista, se separa el primer elemento y se *supone conocida la solución de los restantes*. También se le llama "separarlo en *cabeza y cola*".

```
int suma(Nodo * lista)
{
	int rta;
	if(lista == NULL)
	   //"La suma de una lista vacía es 0"
	   rta = 0;
	else
	   rta = lista->dato + suma(lista->siguiente);
	   /*"La suma de los números de una lista será igual a la
	suma entre el primero y el resultado de la suma de la
	sublista siguiente."*/
	return rta;
}
```

cola

Ejemplo: determinar si un arreglo es capicúa.

```
int capicua(int a[], int i, int j)
{     int rta;
      if(i<j)  //condición de corte
         if(a[i] == a[j]) //condición de corte
            rta = capicua(a, i+1, j -1); //recursión
         else
            rta = 0; //solución trivial
      else
         rta = 1; //solución trivial
      return rta;
}
```

Relación entre Recursión e Iteración

- *Tanto la iteración como la recursión:*
  - *se basan en una estructura de control: la iteración utiliza una estructura de repetición y la recursión una estructura de selección.*
  - *incluyen repeticion: la iteración utiliza una estructura de repetición de forma explícita; mientras que la recursión consigue la repetición mediante llamadas de función repetidas.*
  - *incluyen una prueba de terminación: la iteración termina cuando falla la condición de continuación del ciclo; la recursión termina cuando se reconoce el caso base.*
  - *pueden ocurrir de forma infinita: la iteración se puede quedar en ciclo infinito si la condición del ciclo nunca se hace falsa; la recursión se hará infinita si el paso de recursión no reduce el problema de tal forma que converja al caso base.*
- *La recursión invoca de manera repetida, y por lo tanto, puede sobrecargar las llamadas de función. Esto puede resultar costoso tanto en tiempo de procesador como en espacio de memoria.*

Recursión e Iteración

- Si un lenguaje de programación como C provee tanto recursión como iteración, cuál de los dos métodos debería ser usado para resolver un problema dado?
- En una solución recursiva el computador debe controlar y resolver cada una de las invocaciones manteniendo información acerca del resultado las mismas.
- Esto puede ser costoso en tiempo y espacio.
- En general, se recomienda el uso de una solución recursiva sólo en caso que:
  - la solución no puede ser fácilmente expresable iterativamente (sucede frecuentemente).
  - la eficiencia de la solución recursiva es satisfactoria.