RECURSION


RECURSION
RECURSION
RECURSION
RECURSION
RECURSION
RECURSION

Temario

- ✓ Concepto de Recursión
- ✓ Características de algoritmos recursivos
- ✓ Ejemplos

RECURSION

Metodologías para resolver problemas:

1. Diseño Top Down
2. Recursión
3. Abstracción de Datos
4. Diseño orientado a objetos

RECURSION - CONCEPTO

Una solución recursiva resuelve un problema por resolución de instancias más pequeñas del mismo problema.

Suponga que quiere resolver el problema de contar cuantas monedas hay en una pila de monedas

```
cant=0
while (hay monedas en la pila) do
    cuento una más (cant + 1)
    saco una moneda de la pila
```

¿Cómo se piensa una solución recursiva?

RECURSION - CONCEPTO

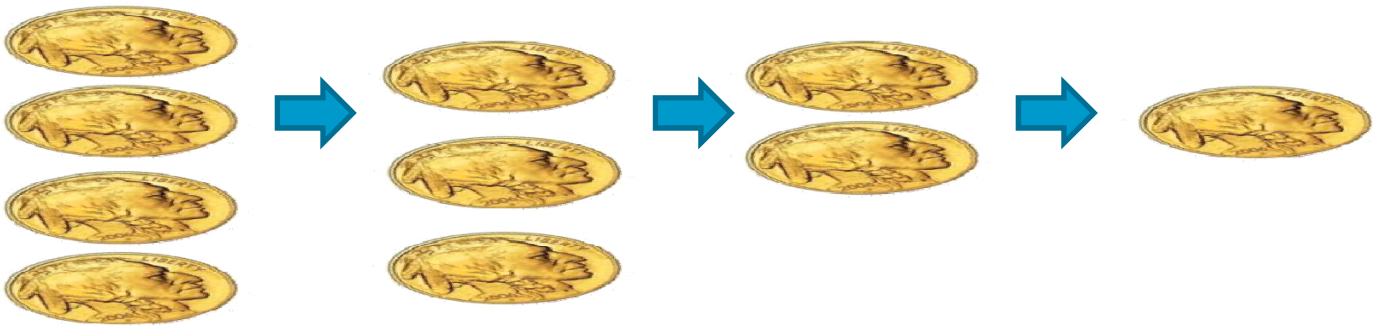
Toda solución recursiva tiene:

- un caso base o degenerado
- el problema es el mismo pero de menor tamaño

contar monedas de la pila

```
cant=0
Contar monedas de la pila
  si (hay monedas en la pila)
      cuento una más (cant + 1)
      saco una moneda de la pila
      contar monedas de la pila
```

RECURSION - CONCEPTO



Observaciones importantes:

→Se llama a sí mismo.

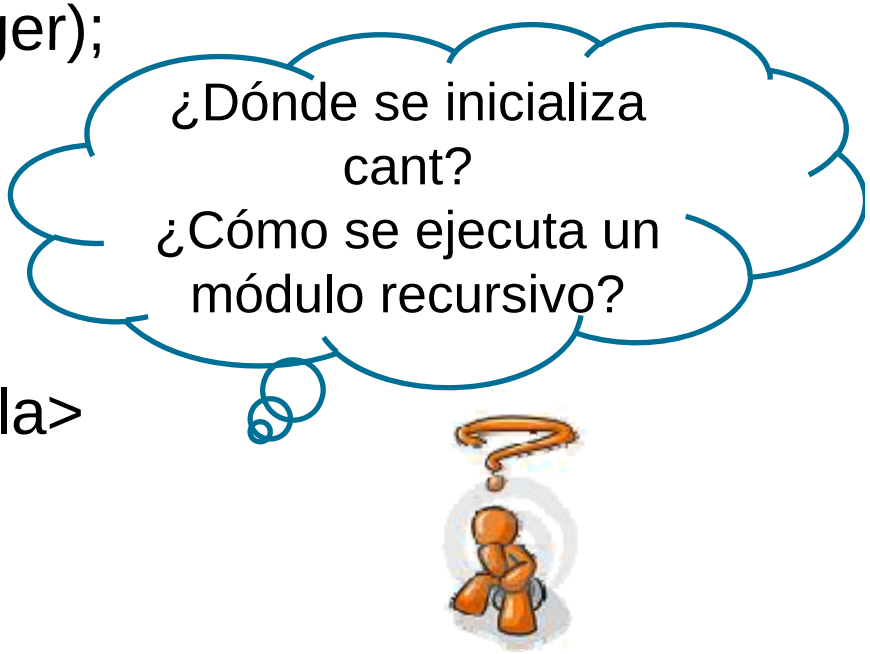
→En cada llamada el tamaño el problema se achica (la pila disminuye).

→Existe un caso distinto que se resuelve de manera particular y directa (sin recursión).

RECURSION - CONCEPTO

Formalicemos un poco mas el algoritmo

```
Procedure cuenta (var cant:integer);
Begin
  if (la pila tiene monedas) then
    begin
      cant:= cant + 1;
      <sacar una moneda de la pila>
      cuenta (cant);
   end;
End;
```



RECURSION - CONCEPTO

Pensemos en una pila que no sabemos cuantos elementos puede tener

¿Qué estructura de datos la puede representar?

Lista

```
Type Pila = ^listap;
     listap = record
              valor: integer;
              sig: ^listap;
      end;
```

¿Cómo debemos construirla si debe funcionar como una pila?

Él último elemento en apilar es el primero en salir

RECURSION - CONCEPTO

Lista

```
Type Pila = ^listap;
       listap = record
                  valor: integer;
                  sig: ^listap;
                end;
```

Agrego por el frente, (puntero al primero)

<Sacar un elemento de la pila>
Saco por el frente el primer elemento

¿Cómo es la implementación de sacar?

RECURSION - CONCEPTO

```
Programa recursivo;
Procedure cuenta (P:Pila; var cant:integer);
Begin
  if (la pila P tiene monedas) then
    begin
       cant:= cant + 1;
       <sacar una moneda de la pila P>
       cuenta (P,cant);
   end;
Var
  cantidad:integer; P:pila
Begin
  cargarPila(P); cantidad:=0;
  cuenta (P, cantidad);
End;
```

```
Programa iteretivo;
Procedure cuenta (P:Pila; var cant:integer);
Begin
  while (la pila P tiene monedas) do
    begin
       cant:= cant + 1;
       <sacar una moneda de la pila P>
    end;
 end;
Var
  cantidad:integer; P:pila
Begin
  cargarPila(P); cantidad:=0;
  cuenta (P, cantidad);
End;
```

RECURSION - CONCEPTO

```
Programa recursivo;
Procedure cuenta (P:Pila; var cant:integer);
Begin
  if (la pila P tiene monedas) then
    begin
      cant:= cant + 1;
      <sacar una moneda de la pila P>
      cuenta (P, cant);
   end;
Var
  cantidad:integer; P:pila
Begin
  cargarPila(P); cantidad:=0;
  cuenta (P, cantidad);
End;
```

```
Programa iteretivo;
Procedure cuenta (P:Pila; var cant:integer);
Begin
  cant:=0;
  while (la pila P tiene monedas) do
    begin
      cant:= cant + 1;
      <sacar una moneda de la pila P>
    end;
 end;
Var
  cantidad:integer; P:pila
Begin
  cargarPila(P);
  cuenta (P, cantidad);
End;
```

RECURSION - CONCEPTO

Suponga que se quiere calcular el factorial de un número

Solución secuencial

```
function factorial (n:integer):integer;
Var
  i,fac:integer;
Begin
 fac:=1;
 for i:= n downto 2 do
   fac:= fac*i;
 factorial:= fac;
End;
```

Solución recursiva

```
function factorial (n:integer):integer;
Var
  i,fac:integer;

Begin
```

Caso base?

$$N! \begin{cases} 1 & \text{si } N<=1 \\ N*(N-1)! & \text{si } N>1 \end{cases}$$

```
End;
```

RECURSION - CONCEPTO

```
Function factorial ( n: integer ) :  integer;
{ Recibe un entero en n y devuelve un valor integer con su
   factorial}
 begin
   if ( n <=1 ) then factorial :=1
                else  factorial := n * factorial ( n - 1);
 end;
```

¿Cómo es la ejecución de una solución recursiva?

RECURSION - EJECUCION

Hay que comprender la manera en que se resuelven las llamadas a los módulos (sean éstos recursivos o no).

Al ejecutarse un programa existe una zona de memoria denominada pila donde se almacena información por cada modulo activo, donde activo quiere decir que ha sido invocado y aún no ha retornado al programa principal.

Esta información consiste en:

- Los parámetros de la función o procedure.
- Las variables locales del módulo
- La “dirección” de retorno, es decir el punto del programa que debe pasar a ejecutarse cuando el Módulo retorne.

RECURSION - EJECUCION

Cuando un módulo es invocado toda esa información es apilada en la pila, al finalizar la ejecución del módulo esa información es retirada de la pila y en caso de una función sustituida por el valor de retorno, pasándose el control a la dirección indicada.

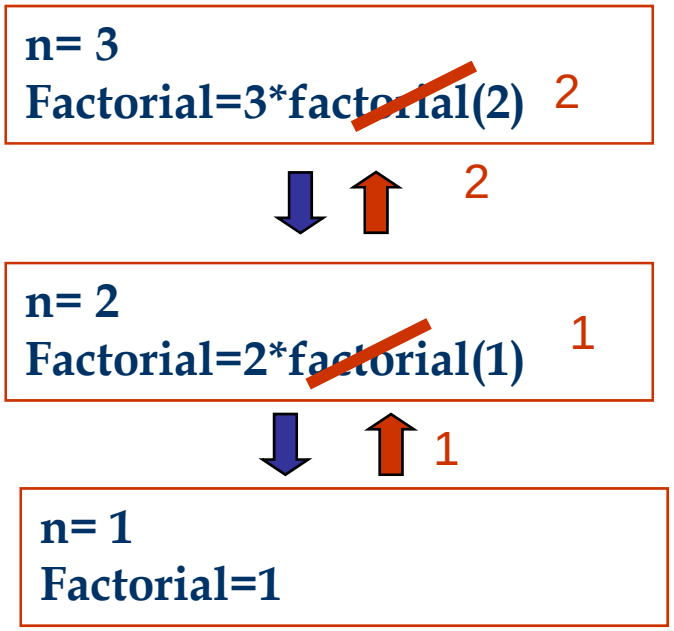
Veamos la ejecución de la función factorial.

RECURSION - CONCEPTO

Supongamos que esta función es llamada desde un programa principal con y := factorial(3) ; Sigamos la ejecución de la función:

```
Function factorial ( n: integer ) :
   integer;

begin

   if ( n <=1 ) then factorial :=1

  else  factorial := n * factorial ( n -
   1);

end;
```

La función devuelve 6



RECURSION - CONCEPTO

- Cada vez que se invoca el llamado recursivo vuelve a cargarse en memoria módulo recursivo a ejecutar.
- No es más eficiente que una solución secuencial
- En muchos casos se obtienen soluciones mas fáciles de leer que las soluciones secuenciales

Cómo se se calcula la potencia de un número de manera recursiva?

RECURSION - EJEMPLO

```
Function Pot ( x:integer ; n:integer ) :integer;
begin

        Pot  { 1              si n=0
             { x* pot(x,n-1)  si n>0

end;
```

RECURSION - EJEMPLO

```
Function pot ( x, n: integer ) :  integer;
begin
    if ( n =0 ) then pot:=1
    else if ( n =1 ) then pot :=x
            else pot := x * pot (x, n - 1);
end;
```

¿QUE PASA CON EL PASAJE DE PARAMETROS ?

RECURSION - ¿Imprimen lo mismo?

¿Para n = 4 como imprime?

```
Procedure Imprimir (n : integer );

begin
     if n < > 0 then  begin
        n:= n - 1;
        Imprimir ( n );
        write (n);
    end;
end;
```

0 1 2 3

```
Procedure Imprimir (VAR n : integer )
    ;

begin
     if n < > 0 then  begin
        n:= n - 1;
        Imprimir ( n );
        write (n);
    end;
end;
```

0 0 0 0

RECURSION - Ejemplo

Cómo imprimimos los elementos de una lista de enteros?

```
Program iterativo;
Type
  lista = ^nodo;
  nodo = record
    dato:integer;
    sig:lista;
  end;
```

```
Var
  l:lista;
Begin
  l:= nil;
  cargarLista(L); //No se implementa
  imprimir(L);
End.
```

RECURSION - Ejemplo

```
Procedure imprimir (Pri:lista);
Begin
  while (pri<> nil) do
   begin
     write (pri^.dato);
     pri:= pri^.sig;
   end;
End;
```

```
Procedure imprimir (Pri:lista);
Begin
  if (pri<> nil) then
   begin
     write (pri^.dato);
     pri:= pri^.sig;
     imprimir(pri);
   end;
End;
```

RECURSION - Ejemplo

Tomemos en cuenta el algoritmo de búsqueda binaria o dicotómica.

Calcula el medio

while (no se junten los punteros de inicio y de fin)

    if (no es el elemento buscado) then

      tomo la mitad del arreglo que es adecuada

¿Como lo pensamos recursivo?

RECURSION - Ejemplo

n=20 pri = 1 ult= 8 medio= 4

5	17	22	45	108	158	196	223

Como 45 > 20, tengo que realizar la búsqueda en la primera mitad del vecto

n=20 pri = 1 ult= 3 medio= 2

5	17	22	45	108	158	196	223

RECURSION - Ejemplo

Como 17 < 20, tengo que realizar la búsqueda en la segunda mitad del vect

n=20 pri = 3 ult= 3 medio= 3

5	17	22	45	108	158	196	223

Como 22 > 20, tengo que realizar la búsqueda en la primera mitad del vector. Al cruzarse los índices y no ser el número buscado devuelve false.

RECURSION - Ejemplo

La conclusión es que siempre se hacen los mismos pasos:

- Calcular primero, último y medio
- Comparar si es igual elemento del medio con el elemento buscado
- En caso de no ser el número buscado, actualizar índices
- Seguir buscando, pero en un vector más chico.

RECURSION

```
procedure Binaria_Recursiva1 ( a:elemento; v : Vector; n : Indice; var
inf:Indice;  var sup: Indice; var ok:Boolean);
 var central:Indice;
  begin
     central:= (inf + sup) div 2;
     if (a=v[central]) then ok:= TRUE
                        else if (inf<=sup) then
                               begin
                                        if (a<v[central])then sup:=central - 1
                                 else inf:=central +1;
                                 binaria_recursiva1(a,v,n,inf,sup,ok)
                              end
                            else ok:= FALSE;
  end;
```

RECURSION - Ejercicio

Implementar de manera recursiva una función que dada una lista de enteros retorne la suma de sus elementos

RECURSION - Conclusiones

1. Algunos problemas pueden resolverse con la misma simplicidad de manera recursiva que iterativa.
2. Las funciones y procedimientos no recursivos pueden ejecutarse más rápidamente y utilizar la memoria de manera más eficiente que los recursivos (se verá más adelante).
3. Usada de manera apropiada, la recursión es una herramienta muy poderosa para la resolución de problemas para los cuales no es sencillo encontrar una solución no recursiva.
4. La recursión también es apropiada cuando los datos involucrados en el problema están organizados en estructuras que pueden definirse recursivamente como listas, árboles, etc.