

# Recursión

Introducción a la programación II

# Proceso o módulo recursivo

Un módulo recursivo es aquel que se llama a si mismo para resolver un problema particular. Aunque en tiempo y memoria la solución recursiva es menos eficiente, existen soluciones alternativas en la recursividad que hacen a la solución más simple y natural al problema.

Ejemplos de problemas recursivos

## Factorial

$$n! = \begin{cases} 1, & \text{si } n=0 \\ n * (n-1)! & \text{si } n>0 \end{cases}$$

## Fibonacci

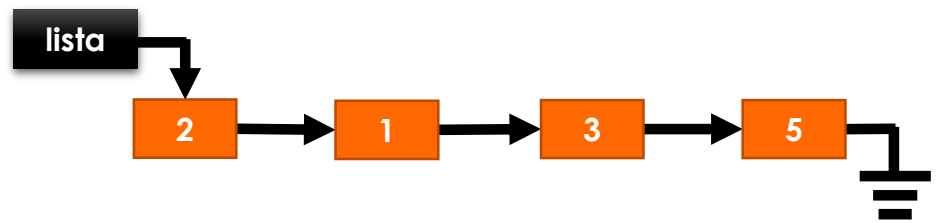
$$F(n) = \begin{cases} 1, & \text{si } n=0, 1 \\ F(n-1) + F(n-2) & \text{si } n>1 \end{cases}$$

# Proceso o módulo recursivo

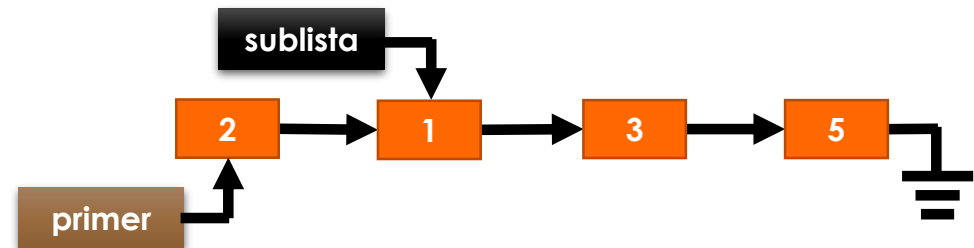
Hay ciertas estructuras de datos que pueden tener un formato recursivo en el almacenamiento de la información.

Ejemplo

Un problema de lista puede llegar a resolverse analizando el puntero al primer nodo y la solución del mismo problema para la sublista restante.



`Suma_de_elementos(lista) => 11`



`primer^.dato + Suma_de_elementos(sublista)`

# Proceso o módulo recursivo

Un problema de arreglos puede resolverse considerando la resolución de a partes.

Arr 

2	1	3	5	6	4
---	---	---	---	---	---

Supongamos que nos piden obtener el menor valor de los elementos de un arreglo. Para ello hacemos un método que le damos como parámetro el arreglo y entre que dos posiciones (inicio y fin) deberá buscar y obtener el menor

**Function Obtener\_menor(Arr:arreglo; inicio, fin:integer):integer;**

Los posibles casos de solución son:

**Caso 1:** si inicio es igual a fin (**Opción de corte**)

Obtener\_menor:=Arr[fin]

**Caso 2:** si inicio es distinto de fin pueden usarse distintas soluciones (**Opción de continuidad**)

Obtener\_menor := Mas\_chico(Arr[inicio], Obtener\_menor(Arr, inicio+1, fin))

Obtener\_menor := Mas\_chico(Obtener\_menor(Arr, inicio, (inicio+fin)/2),  
Obtener\_menor(Arr, (inicio+fin)/2 + 1, fin))

# Proceso o módulo recursivo

Arr 

2	1	3	5	6	4
---	---	---	---	---	---

Supongamos que nos piden obtener la suma de los elementos de un arreglo. Para ello hacemos un método que le damos como parámetro el arreglo y entre que dos posiciones (inicio y fin) deberá calcular la suma

**Function Suma\_de\_elementos(Arr:arreglo; inicio, fin:integer):integer;**

Los posibles casos de solución son:

**Caso 1:** si inicio es igual a fin (**Opción de corte**)

Suma\_de\_elementos:=Arr[fin]

**Caso 2:** si inicio es distinto de fin pueden usarse distintas soluciones (**Opción de continuidad**)

Suma\_de\_elementos := Arr[inicio] + Suma\_de\_elementos(Arr, inicio+1, fin)

Suma\_de\_elementos := Suma\_de\_elementos(Arr, inicio, (inicio+fin)/2) +  
Suma\_de\_elementos(Arr, (inicio+fin)/2 + 1, fin)

# Solución recursiva

Para plantear una solución recursiva es necesario tener en cuenta:

- Una o más opciones de corte para terminar el proceso.
- Una o más opciones de continuidad en la búsqueda, calculo o generación de la solución.
- El pasaje de parámetros (copia o referencia), que condiciona los valores que van a tener en el anidamiento recursivo.
- Analizar que solución es más conveniente: recursiva o iterativa.

Ejemplo:

$$n! = \begin{cases} 1, & \text{si } n=0 \\ n * (n-1)! & \text{si } n>0 \end{cases}$$

```
Function Factorial(n:integer):integer;  
Begin  
    if n=0 then //OPCION DE CORTE  
        Factorial:=1  
    else //OPCION DE CONTINUIDAD  
        Factorial:=n * Factorial(n-1);  
End;
```

Nivel recursivo	Código	Contenido
0	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else <b>Factorial:=n * Factorial(n-1);</b> End;	n = 4 Factorial(4) = 4 * <b>Factorial(3)</b>
1	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else <b>Factorial:=n * Factorial(n-1);</b> End;	n = 3 Factorial(3) = 3 * <b>Factorial(2)</b>
2	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else <b>Factorial:=n * Factorial(n-1);</b> End;	n = 2 Factorial(2) = 2 * <b>Factorial(1)</b>

Nivel recursivo	Código	Contenido
3	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else <b>Factorial:=n * Factorial(n-1);</b> End;	n = 1 Factorial(1) = 1 * <b>Factorial(0)</b>
4	Function Factorial(n:integer):integer; Begin if n=0 then <b>Factorial:=1</b> else Factorial:=n * Factorial(n-1); End;	n = 0 Factorial(0) = <b>1</b>

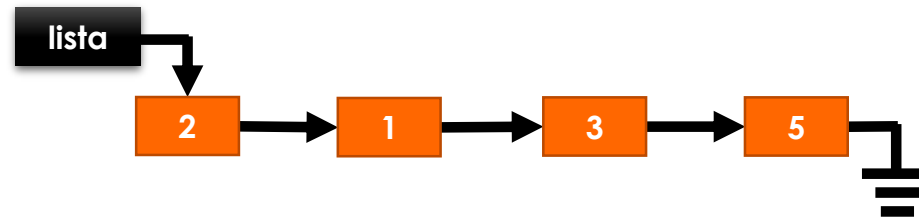
Nivel recursivo	Código	Contenido
3	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else Factorial:=n * Factorial(n-1); End;	n = 1 Factorial(1) = 1 * 1
2	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else Factorial:=n * Factorial(n-1); End;	n = 2 Factorial(2) = 2 * 1

Nivel recursivo	Código	Contenido
1	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else Factorial:=n * Factorial(n-1); End;	n = 3 Factorial(3) = 3 * 2
0	Function Factorial(n:integer):integer; Begin if n=0 then Factorial:=1 else Factorial:=n * Factorial(n-1); End;	n = 4 Factorial(4) = 4 * 6



# Solución recursiva

Ejemplo: imprimir una lista simple en orden inverso



Por consola

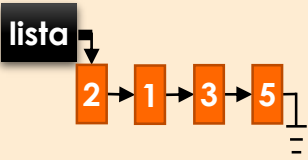

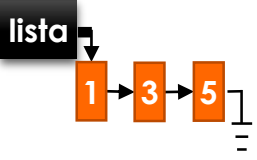

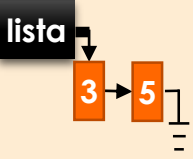
...5 3 1 2...


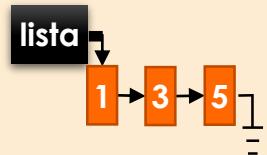
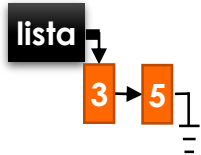
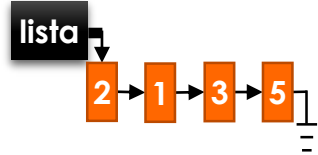
```
Procedure ImprimirListaInverso(lista:Plista);  
Begin
```

```
    //LA OPCION DE CORTE ES SI LA LISTA ES NIL  
    if lista<>nil then begin //OPCION DE CONTINUIDAD  
        ImprimirListaInverso(lista^.ste);  
        //avanzo y luego imprimo en el regreso de la cadena recursiva  
        write(lista^.dato);
```

```
    end;
```

```
End;
```

Nivel recursivo	Código	Contenido Consola	Nivel recursivo	Código	Contenido Consola
0	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin if lista&lt;&gt;nil then begin <b>ImprimirListaInverso(lista^.ste);</b> write(lista^.dato); end; End; </pre>	<p>...</p> 	3	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin if lista&lt;&gt;nil then begin <b>ImprimirListaInverso(lista^.ste);</b> write(lista^.dato); end; End; </pre>	<p>...</p> 
1	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin if lista&lt;&gt;nil then begin <b>ImprimirListaInverso(lista^.ste);</b> write(lista^.dato); end; End; </pre>	<p>...</p> 	4	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin <b>if lista&lt;&gt;nil then begin</b> ImprimirListaInverso(lista^.ste); write(lista^.dato); <b>end;</b> End; </pre>	<p>...</p> 
2	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin if lista&lt;&gt;nil then begin <b>ImprimirListaInverso(lista^.ste);</b> write(lista^.dato); end; End; </pre>	<p>...</p> 			

Nivel recursivo	Código	Contenido Consola	Nivel recursivo	Código	Contenido Consola
3	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin   if lista&lt;&gt;nil then begin     ImprimirListaInverso(lista^.ste);     write(lista^.dato);   end; End; </pre>	<p>...5</p> 	1	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin   if lista&lt;&gt;nil then begin     ImprimirListaInverso(lista^.ste);     write(lista^.dato);   end; End; </pre>	<p>...5 3 1</p> 
2	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin   if lista&lt;&gt;nil then begin     ImprimirListaInverso(lista^.ste);     write(lista^.dato);   end; End; </pre>	<p>...5 3</p> 	0	<pre> Procedure ImprimirListaInverso(lista:Plista); Begin   if lista&lt;&gt;nil then begin     ImprimirListaInverso(lista^.ste);     write(lista^.dato);   end; End; </pre>	<p>...5 3 1 2</p> 

# Solución recursiva

**Observación:** a diferencia de los procedimientos recursivos, las funciones recursivas siempre deben retornar un resultado.

```
Function Factorial(n:integer):integer;  
Begin
```

```
    if n=0 then
```

```
        Factorial:=1
```

```
    else
```

```
        Factorial:=n * Factorial(n-1);
```

```
End;
```

```
Procedure ImprimirListaInverso(lista:Plista);
```

```
Begin
```

```
    if lista<>nil then begin
```

```
        ImprimirListaInverso(lista^.ste);
```

```
        write(lista^.dato);
```

```
    end;
```

```
End;
```

## Solución recursiva

**Observaciones:**

- La solución recursiva es más lenta
- Puede ocupar más espacio en memoria.
- Puede ser más natural o no a un problema particular.

```
Procedure ImprimirArreglo(Arr:Arreglo;n:integer);  
Begin
```

```

if n<=MAX then begin
    write(Arr[n]);
    ImprimirArreglo(Arr,n+1);
end;

```

**End;**

```

Procedure ImprimirArreglo(Arr:Arreglo);
var n:integer;
Begin

```

```
for n:=1 to MAX do
    write(Arr[n]);
```

**End;**

Nivel recursivo	Memoria	2	1	3	5	6	4
1	Procedure ImprimirArreglo(Arr:Arreglo;n:integer); Begin if n<=MAX then begin write(Arr[n]); ImprimirArreglo(Arr,n+1); end; End;						
...	...						
6	Procedure ImprimirArreglo(Arr:Arreglo;n:integer); Begin if n<=MAX then begin write(Arr[n]); ImprimirArreglo(Arr,n+1); end; End;						
...	...						
1	Procedure ImprimirArreglo(Arr:Arreglo); var n:integer; Begin for n:=1 to MAX do write(Arr[n]); End;						

# Tipos de recursividad

Hay dos tipos de recursividad:

- Directa: un módulo se llama a si mismo más de una vez.
- Indirecta: un módulo llama a otro, y que a su vez llama al primero.

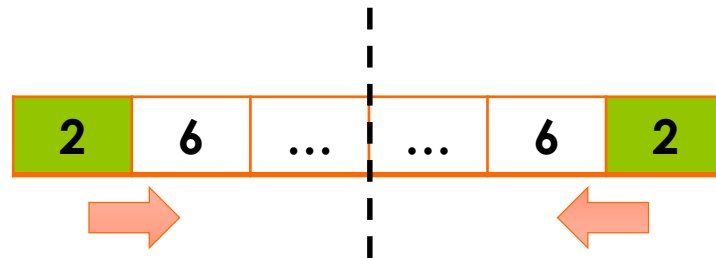
Ejemplo:

```
Function A(...):integer;  
Begin  
    ...  
    resultadoparcialA:=B(...)  
    ...  
End;  
  
Function B(...):integer;  
Begin  
    ...  
    resultadoparcialB:=A(...)  
    ...  
End;
```

# Ejemplos de recursividad

Capicúa:

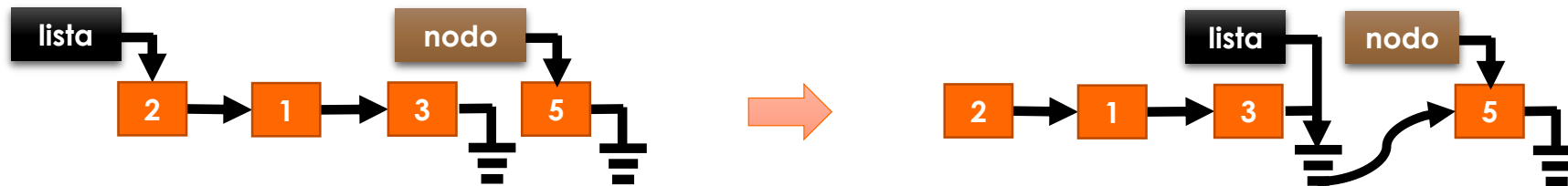
```
Function arreglo_capicua(arr:Arreglo;ini,fin:integer):booe  
Begin  
    if ini>=fin then//CORTE  
        arreglo_capicua:=true  
    else if (arr[ini]<>arr[fin]) then//CORTE  
        arreglo_capicua:=false  
    else //CONTINUIDAD, entra en este caso cuando son iguales  
        arreglo_capicua:=arreglo_capicua(arr,ini+1,fin-1);  
End;
```



# Ejemplos de recursividad

Agregar al final de la lista simple:

```
Procedure agregar_final_LS(var lista:Plista;nodo:Plista);  
Begin  
    if lista<>nil then//CONTINUIDAD  
        agregar_final_LS(lista^.ste,nodo)  
    else//CORTE  
        lista:=nodo;  
End;
```





# Ejemplos de recursividad

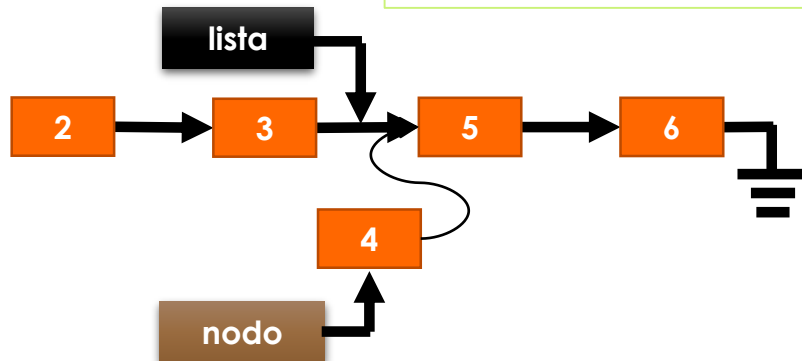
```
Procedure insertarordnodolista(var lista:punterolista;nodo:punterolista);  
Begin
```

```
//Caso 1 y 4          Caso 2 y 3  
if (lista=NIL)      or  (lista^.dato>=nodo^.dato) then begin//CORTE  
    nodo^.ste:=lista;  
    lista:=nodo;
```

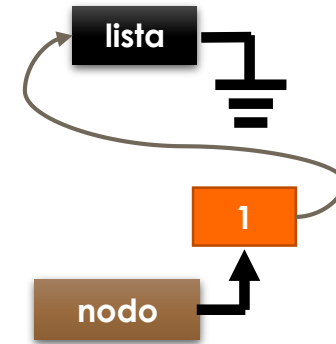
```
end  
Else //CONTINUIDAD la llamada recursiva reemplaza la búsqueda  
    insertarordnodolista(lista^.ste,nodo);
```

```
End;
```

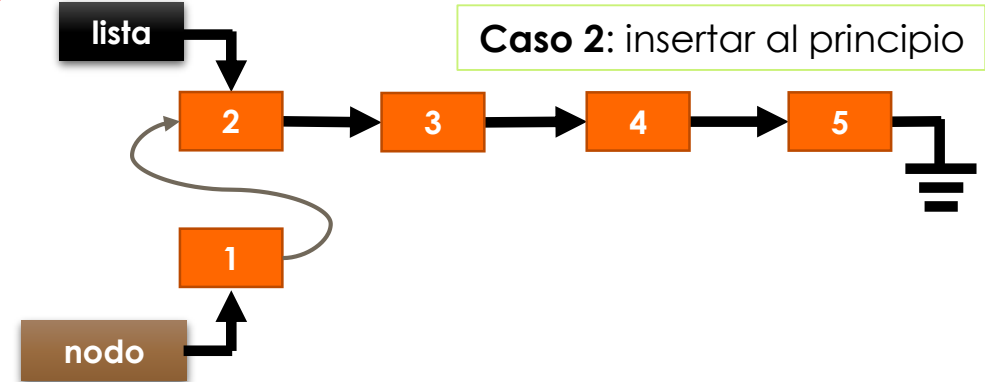
Caso 3: insertar al medio



Caso 1: lista vacia



Caso 2: insertar al principio



Caso 4: insertar al final

