TEMA 1

Recursividad

Recursividad

T
E
M
A

1

CONTENIDO DEL TEMA

1.- Introducción.
2.- Verificación de funciones y procedimientos recursivos
3.- Escritura de programas recursivos
4.- Ejemplos.
5.- ¿Recursión o iteración?
6.- Depuración
7.- Ejemplos
8.- Asignación estática y dinámica de memoria.

Introducción

- Definición de Recursividad: Técnica de programación muy potente que puede ser usada en lugar de la iteración.
- Ambito de Aplicación:
  - General
  - Problemas cuya solución se puede hallar solucionando el mismo problema pero con un caso de menor tamaño.
- Razones de uso:
  - Problemas "casi" irresolubles con las estructuras iterativas.
  - Soluciones elegantes.
  - Soluciones más simples.
- Condición necesaria: ASIGNACIÓN DINÁMICA DE MEMORIA

Introducción

- ¿En qué consiste la recursividad?
  - En el cuerpo de sentencias del subalgoritmo se invoca al propio subalgoritmo para resolver "una versión más pequeña" del problema original.
  - Habrá un caso (o varios) tan simple que pueda resolverse directamente sin necesidad de hacer otra llamada recursiva.
- Aspecto de un subalgoritmo recursivo.

```
ALGORITMO Recursivo(...)
INICIO
   ...
   Recursivo(...);
   ...
FIN
```

Introducción

- Ejemplo: Factorial de un natural.

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

Introducción

- Ejemplo: Factorial de un natural.

```
ALGORITMO N Factorial(E n:N)
VAR
  N fact
INICIO
  SI n == 0 ENTONCES fact = 1
  SINO fact = n*Factorial(n-1)
  FINSI
  DEVOLVER fact
FIN
```

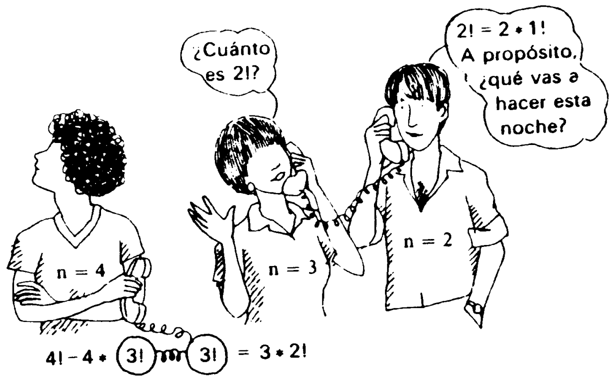
Introducción

- ¿Cómo funciona la recursividad?

4!=4*3!

Introducción

- **3!=3*2!**

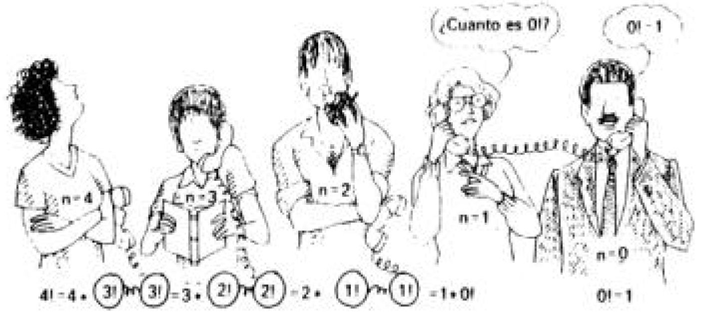


Introducción

•2!=2*1!

Introducción

- 1!=1*0!=1*1



Introducción

Verificación de funciones y procedimientos recursivos

Método de las tres preguntas

- La pregunta Caso-Base: ¿Existe una salida no recursiva o caso base del subalgoritmo? Además, ¿el subalgoritmo funciona correctamente para ella?
- La pregunta Más-pequeño: ¿Cada llamada recursiva se refiere a un caso más pequeño del problema original?
- La pregunta Caso-General: ¿es correcta la solución en aquellos casos no base?

Escritura de programas recursivos

- 1.-Obtención de una definición exacta del problema
- 2.-Determinar el tamaño del problema completo que hay que resolver $\Longrightarrow$ Parámetros en la llamada inicial
- 3.-Resolver el(los) casos bases o triviales (no recursivos).
- 4.-Resolver el caso general en términos de un caso más pequeño (llamada recursiva).

Distintos parámetros

Ejemplos

- Combinaciones:¿cuántas combinaciones de cierto tamaño pueden hacerse de un grupo total de elementos?
  - C:número total de combinaciones
  - Grupo:tamaño total del grupo del que elegir
  - Miembros:tamaño de cada subgrupo
  - Grupo>=Miembros

$$C(Grupo,Miembros) \begin{cases} -Grupo & \text{si } Miembros=1 \\ -1 & \text{si } Miembros=Grupo \\ -C(Grupo-1,Miembros-1)+C(Grupo-1,Miembros) & \text{si } Grupo>Miembros>1 \end{cases}$$

Ejemplos

• FUNCIÓN COMBINACIONES

- Definición:Calcular cuantas combinaciones de tamaño Miembros pueden hacerse del tamaño total Grupo
- Tamaño :Número de procesos dado en la llamada original
- Casos-base:1)Miembros==1 ⇨ Combinaciones=Grupo
  2)Miembros==Grupo ⇨ Combinaciones=1
- Caso General:Grupo>Miembros>1 ⇩

Combinaciones = Combinaciones(Grupo-1 , Miembros -1) +
Combinaciones(Grupo-1, Miembros)

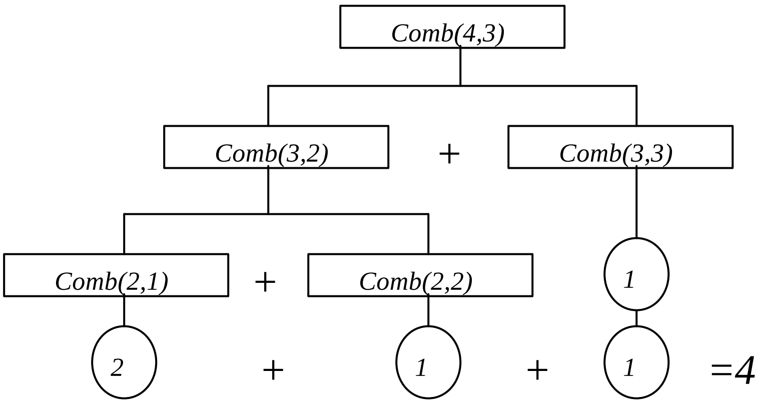
Ejemplos

```
ALGORITMO N Comb(E N Grupo, Miembros)
VAR
   N cmb
INICIO
   SI Miembros == 1 ENTONCES
      cmb =  Grupo (*Caso Base 1*)
   SINOSI Miembros == Grupo ENTONCES
      cmb = 1     (*Caso Base 2*)
   SINO      (*Caso General*)
      cmb = Comb(Grupo-1,Miembros-1) +
            Comb(Grupo-1,Miembros)
   FINSI
   DEVOLVER cmb
FIN
```

Llamada : `Escribir(“Número de combinaciones=“, Comb(20,5))`

Ejemplos

- Seguimiento de Comb(4,3)



Ejemplos

FUNCIÓN FIBONACCI

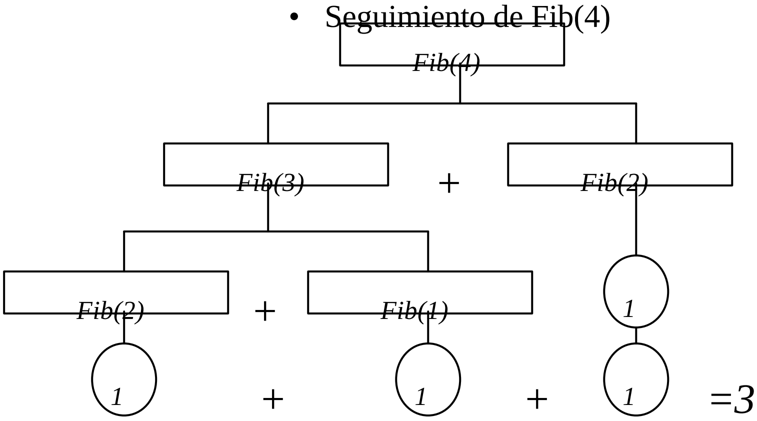
- Definiciones:Calcular el valor de la función de Fibonacci para un número **n** dado.
- Tamaño:Número n de la llamada original
- Casos-base:n<=2 ⇨ fib = 1
- Caso General:n>2 ⇨ fib(n) = fib(n-1) + fib(n-2)

Ejemplos

```
ALGORITMO N Fib(E N n)
VAR
    N fb
INICIO
    SI  (n <= 2) ENTONCES
        fb = 1
    SINO
        fb = Fib(n-1) + Fib(n-2)
    FINSI
    DEVOLVER fb
FIN
```

Ejemplos

- Seguimiento de Fib(4)



Ejemplos

- Imprimir el equivalente binario de un número decimal

N	**N MOD 2**	**N DIV 2**
23	1	11
11	1	5
5	1	2
2	0	1
1	1	0
0		

Ejemplos

$$\text{Bin de N=} \begin{cases} \text{N} \quad \text{Si N<2} \\ \\ \text{Binaria de (N DIV 2)||(N MOD 2)} \end{cases}$$

con || la concatenación

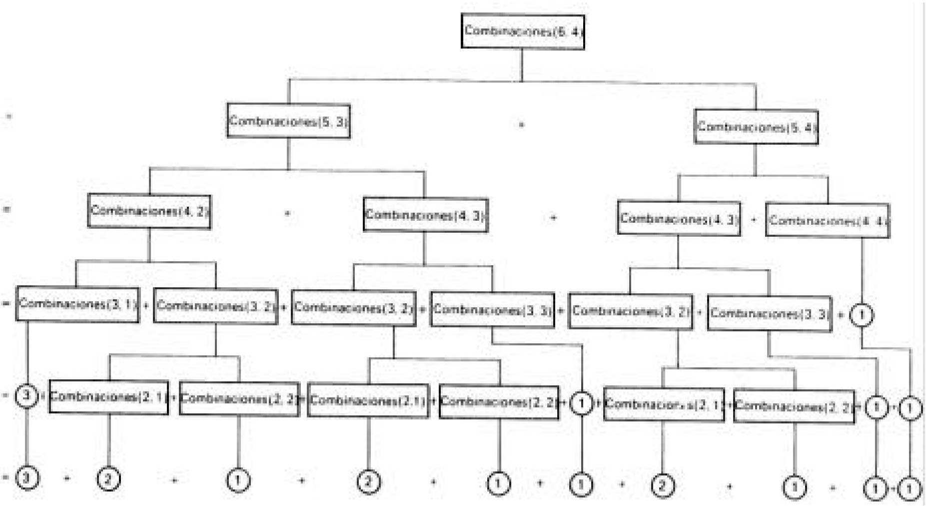
- Ventaja: no requiere arrays

Ejemplos

```
ALGORITMO DecimalAbinario(E N num)
INICIO
  SI num >= 2 ENTONCES
    DecimalABinario(num DIV 2)
    Escribir(num MOD 2)
  SINO
    Escribir (num)
  FINSI
FIN
```

¿Recursión o iteración?

- Ventajas de la Recursión ya conocidas
  - Soluciones simples, claras.
  - Soluciones elegantes.
  - Soluciones a problemas complejos.
- Desventajas de la Recursión: INEFICIENCIA
  - Sobrecarga asociada con las llamadas a subalgoritmos
    - Una simple llamada puede generar un gran numero de llamadas recursivas. (Fact(n) genera n llamadas recursivas)
    - ¿La claridad compensa la sobrecarga?
    - El valor de la recursividad reside en el hecho de que se puede usar para resolver problemas sin fácil solución iterativa.
  - La ineficiencia inherente de algunos algoritmos recursivos.



¿Recursión o iteración ?

- A veces, podemos encontrar una solución iterativa simple, que haga que el algoritmo sea más eficiente.

```
ALGORITMO N Fib(E N n)
VAR
   N r = 1, r1 = 1, r2 = 1, i
INICIO
   PARA i = 3 HASTA n HACER
      r = r1 + r2
      r2 = r1
      r1 = r
   FINPARA
   DEVOLVER r
FIN
```

¿Recursión o iteración?

LA RECURSIVIDAD SE DEBE USAR CUANDO SEA REALMENTE NECESARIA, ES DECIR, CUANDO NO EXISTA UNA SOLUCIÓN ITERATIVA SIMPLE.

Depuración

ERRORES COMUNES

– Tendencia a usar estructuras iterativas en lugar de estructuras selectivas. El algoritmo no se detiene.

Comprobar el uso de SI o CASO

– Ausencia de ramas donde el algoritmo trate el caso-base.

– Solución al problema incorrecta

Seguir el método de las 3 preguntas

Ejemplos

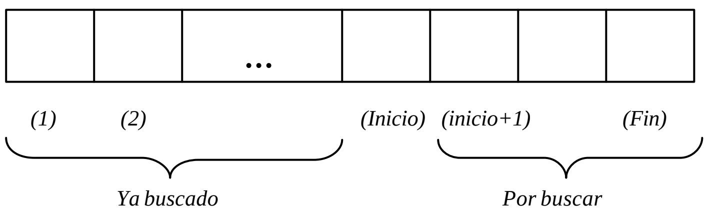
BUSQUEDA EN UN ARRAY

Función ValorEnLista:Buscar el valor Val en un arrayLista:TLista

Solución recursiva

DEVOLVER (Val en 1ª posición)OR(Val en resto del ARRAY)

Para buscar en el resto del ARRAY, uso la misma función ValorEnLista



Ejemplos

Algoritmo B `ValorEnLista(E Tlista l; Tvalor val; E Z ini,fin)`

- Invocación:
  `SI ValorEnLista(l, val,1,MaxLista) ENTONCES....`
- Casos Base:
  `l[Inicio] == val` ⇨ `Verdadero`
  `ini == fin y l[ini] <> val` ⇨ `Falso`
- Caso General:buscar en el resto del ARRAY
  `ValorEnLista(l, val, ini+1, fin)`

Ejemplos

```
ALGORITMO B ValorEnLista(E Tlista l; E Tvalor val; E Z ini,fin)
(*Busca recursiva en lista de Val dentro del rango del indice
   del ARRAY*)
VAR
   B enc
INICIO
   SI Lista[Inicio] == val ENTONCES
      enc = Verdadero
   SINOSI ini == fin ENTONCES
      enc = Falso
   SINO
      enc = ValorEnLista(l, val, ini+1, fin)
   FINSI
   DEVOLVER enc
FIN
```

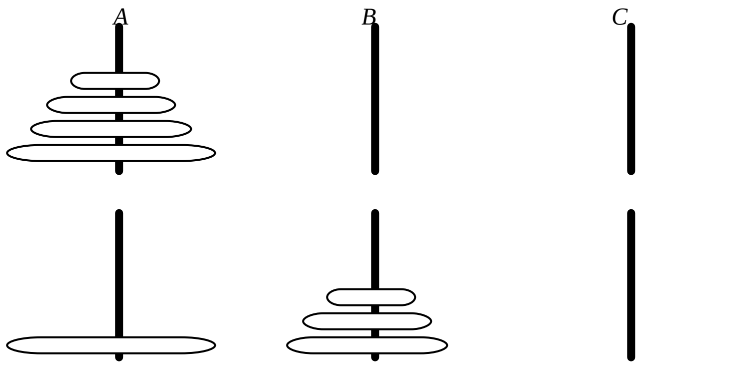

Ejemplos

Torres de Hanoi

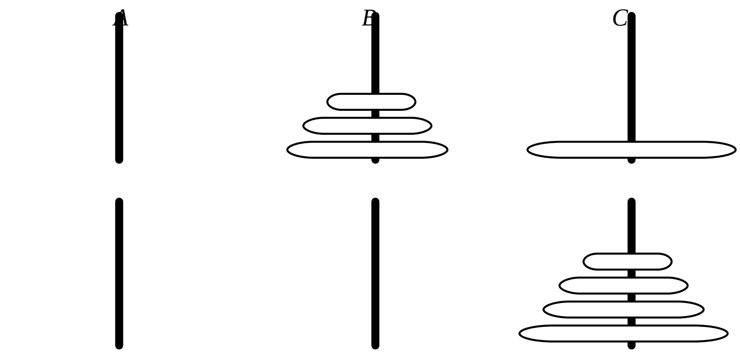
- Se tienen 3 palos de madera, que llamaremos palo izquierdo, central y derecho. El palo izquierdo tiene ensartados un montón de discos concéntricos de tamaño decreciente, de manera que el disco mayor está abajo y el menor arriba.
- El problema consiste en mover los discos del palo izquierdo al derecho respetando las siguientes reglas:
  - - Sólo se puede mover un disco cada vez.
  - - No se puede poner un disco encima de otro más pequeño.
  - - Después de un movimiento todos los discos han de estar en alguno de los tres palos.

Leer por teclado un valor N, e imprimir la secuencia de pasos para resolver el problema.

Ejemplos



Ejemplos



Ejemplos

- Solución recursiva a las Torres de Hanoi
  - Si n=1 mueva el disco de A a C y pare
  - Mueva los n-1 discos superiores de A a B, con C auxiliar
  - Mueva los discos restantes de A a C
  - Mueva los n-1 discos de B a C, usando A como auxiliar

Ejemplos

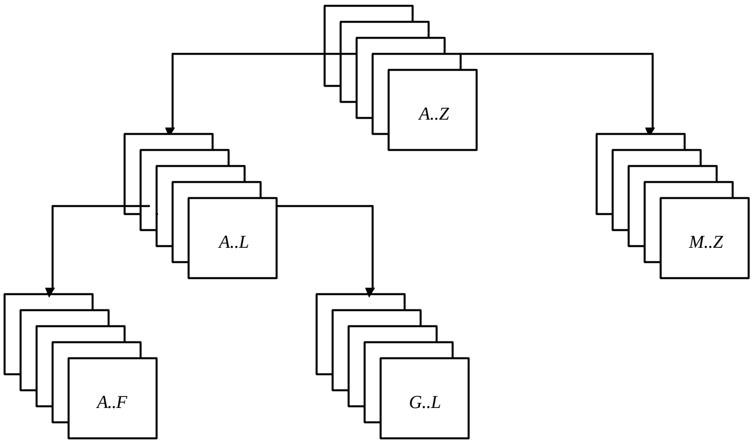
Planteamos un procedimiento recursivo con cuatro parámetros:

- El número de discos a mover.
- El palo origen desde donde moverlos.
- El palo destino hacia el que moverlos.
- El palo auxiliar.

```
ALGORITMO Mueve(E N n; E Tpalos origen,auxiliar,destino)
INICIO
   SI n == 1 ENTONCES
         Mueve un disco del palo origen al destino
   SINO
         Mueve(n-1,origen,destino,auxiliar)
         Mueve un disco del palo origen al destino
         Mueve(n-1,auxiliar,origen,destino)
   FINSI
FIN
```

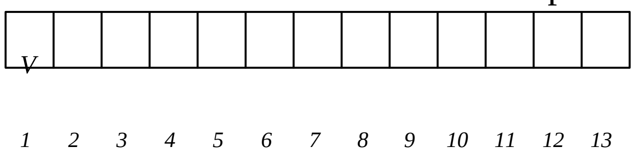
Ejemplos

ORDENACIÓN RÁPIDA (QUICKSORT)



Ejemplos

- Solución recursiva a la ordenación rápida.



- Qué información es necesaria para abastecer a OrdRápida?
  - Nombre del array
  - su tamaño (primer y último índice)

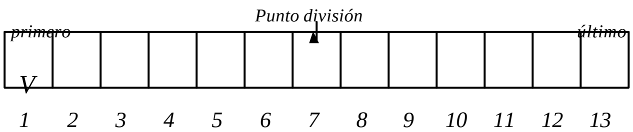
Ejemplos

- El algoritmo básico OrdRápida es:

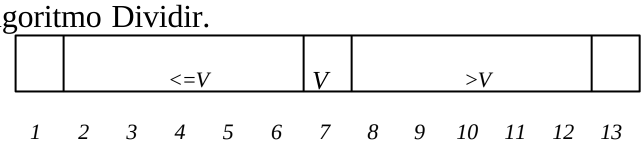
```
SI NOT terminado ENTONCES
  Dividir el array por un valor V (Pivote)
  OrdRápida los elementos menores ó iguales
  que V
  OrdRápida los elementos mayores que V
```

- **<u>Algoritmo</u>** `OrdRápida(E/S Tarray Datos; E Primero,Ultimo:`**`N`**`)`
- La llamada sería `OrdRápida (Datos,1,n)`

Ejemplos



- Usamos el valor de Datos[1] como pivote.
- Subalgoritmo Dividir.



Ejemplos

```
OrdRápida(Datos, Primero, PuntoDivisión-1)
OrdRápida(Datos,Puntodivisión+1,Ultimo)
```

- ¿Cual es el caso base?
  - Si el segmento de array tiene menos de dos elementos:SI Primero<Ultimo

```
ALGORITMO OrdRápida(E/S Tarray Datos; E N Primero,Ultimo)
VAR
  N PuntoDivision
INICIO
  SI Primero<Ultimo ENTONCES
        Dividir(Datos,Primero,Ultimo,PuntoDivision)
        OrdRápida(Datos,Primero,PuntoDivision)
        OrdRápida(Datos,PuntoDivision+1,Ultimo)
  FINSI
FIN
```

Ejemplos

a) Inicialización V=Datos[1] =9

9	*20*	*6*	*10*	*14*	*8*	*60*	*11*
	Izq						*Dcha*

b) Mover Izq a la derecha hasta que Datos[Izq]>V

9	*20*	*6*	*10*	*14*	*8*	*60*	*11*
	Izq						*Dcha*

c) Mover Dcha a la izquierda hasta que Datos[Dcha]<=V

9	*20*	*6*	*10*	*14*	*8*	*60*	*11*
	Izq				*Dcha*		

Ejemplos

d) Intercambiar Datos[Izq] y Datos[Dcha], y mover Izq y Dcha

9	*8*	*6*	*10*	*14*	*20*	*60*	*11*

Izq *Dcha*

e) Mover Izq hasta que Datos[Izq]>V o Dcha<Izq
Mover Dcha hasta que Datos[Dcha]<=V o Dcha<Izq

9	*8*	*6*	*10*	*14*	*20*	*60*	*11*

Dcha *Izq*

f) Izq>Dcha, por tanto no ocurre ningún intercambio dentro del bucle . Intercambiamos Datos[1] con Datos[Dcha]

6	*8*	*9*	*10*	*14*	*20*	*60*	*11*

PuntoDivisión

```
ALGORITMO Dividir(E/S TArray Datos; E Z prim, ult;S Z Pdivision)
VAR Z izq, dcha, v
INICIO
       V = Datos[prim]
       izq = prim + 1
       dcha= Ultimo
       REPETIR
              MIENTRAS(izq < dcha) Y (Datos[izq] <= v) HACER
                     izq = izq + 1
              FINMIENTRAS
              SI (izq == dcha) Y (Datos[Iiq]<= v) ENTONCES
                     izq= izq + 1
              FINSI
              MIENTRAS (izq <= dcha) Y (Datos[dcha] > v) HACER
                     dcha = dcha-1
              FINMIENTRAS
              SI izq < dcha ENTONCES
                     Intercambiar(Datos[izq],Datos[dcha])
                     izq = izq+1
                     dcha = dcha-1
              FINSI
       HASTA izq > dcha
       Intercambiar(Datos[prim],Datos[dcha])
       Pdivision = dcha
FIN
```

Asignación estática y dinámica de memoria

Registro de Activación.

Dirección de Retorno.

Pila (Stack).

Vinculación.

Asignación estática y dinámica de memoria

- Partimos del siguiente ejemplo

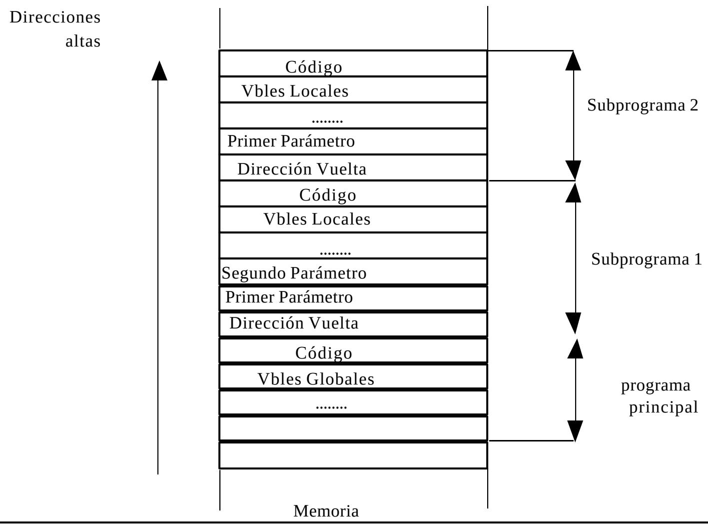
```
ALGORITMO uno(x,y:R)
VAR
   N z
INICIO
   ...
   ...
FIN
```

Asignación estática y dinámica de memoria

Asignación estática

- Se reserva espacio en memoria a partir de una posición **FIJA,** tanto para el código como para los parámetros formales y variables locales de cada subprograma.
- En este caso: x <----> 0100
  y <----> 0101
  z <----> 0111
- La zona reservada para variables locales y parámetros formales usualmente preceden al código del subprograma

Asignación estática y dinámica de memoria



Asignación estática y dinámica de memoria

PROBLEMA

Vinculación de variables en tiempo de compilación

¿almacenamiento de las distintas llamadas recursivas?

Pérdida de los valores de las variables

Asignación estática y dinámica de memoria

Asignación dinámica

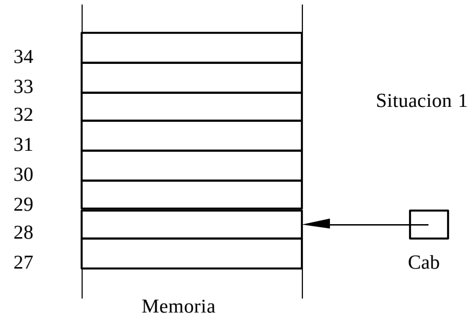
- Asignación de cada variable, parámetro relativa a una posición (CAB)
- En este caso: x <----> 1

  y <----> 2

  z <----> 3
- Dirección de retorno <----> 0

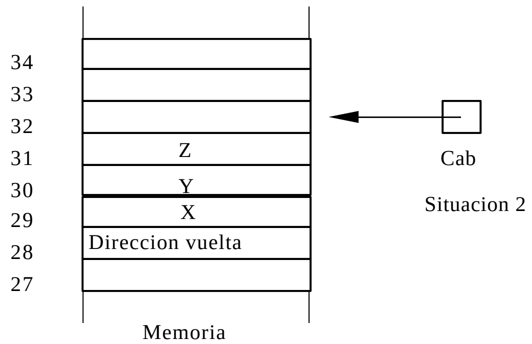
Asignación estática y dinámica de memoria

Tiempo de ejecución:Se reserva espacio para las variables y parámetros a partir de la situación actual de CAB



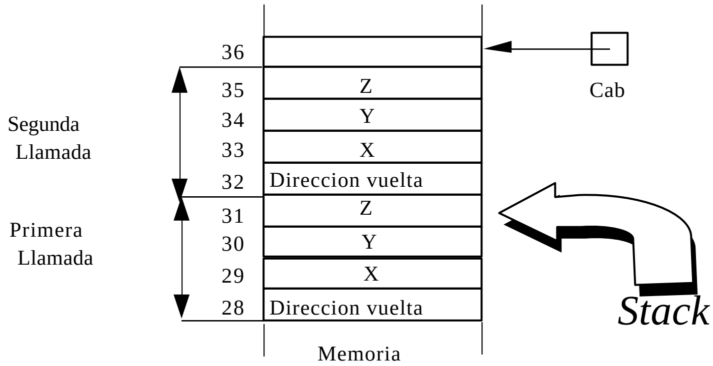
Asignación estática y dinámica de memoria

Llamada al subalgoritmo



Asignación estática y dinámica de memoria

Llamada recursiva al algoritmo



Asignación estática y dinámica de memoria

Ejemplo con la función factorial

```
ALGORITMO N Factorial(E n:N)
VAR
  N fact
INICIO
  SI n == 0 ENTONCES fact = 1
  SINO fact = n * Factorial(n-1)
  FINSI
  DEVOLVER fact
FIN
```

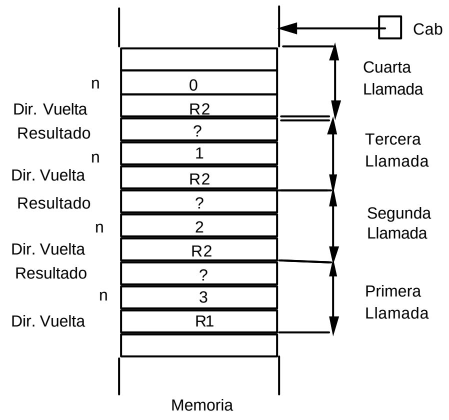
R2

R1

La invocación inicial es: `Resultado:=Factorial(3)`

Asignación estática y dinámica de memoria

Estado de la pila



Asignación estática y dinámica de memoria

Observaciones

- Invocación del subalgoritmo a sí mismo.
- Cada llamada al subalgoritmo se realiza con un valor de parámetro que hace el problema "de menor tamaño".
- La llamada al subalgoritmo se realiza siempre en una sentencia de selección.
- En dicha sentencia de selección, al menos debe de haber un caso donde se actúa de forma diferente (no recursiva).Este es el caso base.
- Ocultación de los detalles de gestión de la memoria en las llamadas recursivas **(Pila interna).**

Bibliografía

- **Pascal**. Dale/Orshalick. Ed McGraw Hill 1986.
- **Pascal y Estructuras de Datos.** Nell Dale, Susan C. Lilly. McGraw Hill. 1989.
- **Fundamentos de programación**. Joyanes Aguilar. McGraw Hill. 1988
- **Introduction to programming with modula-2**. Saim Ural/Suzan Ural. Wiley. 1987.
- **Estructuras de datos en Pascal**. Aaron Tenenbaum. Prentice Hall. 1983