RECURSIÓN

- Introducción
- Funciones recursivas
- Programas recursivos
- Demostración por recursividad
- Eliminación de la recursión

INTRODUCCIÓN

Definiciones recursivas de sumas y productos

- Existe una notación para indicar sumas de muchos términos: ∑ (sigma)
- Se hará una definición *recursiva* de la notación sigma y se aplicará para demostrar resultados que contienen sumas por *inducción matemática*.

- Si se desea obtener …

$$a_m + a_{m+1} + \ldots\ldots + a_n$$

se escribe en la forma …

$$\sum_{i=m}^{n} a_i$$

Por lo tanto …

$$\sum_{i=m}^{n} a_i = a_m + a_{m+1} + \ldots\ldots + a_n$$

donde …

i	es el índice de la suma
m	es la cota inferior
n	es la cota superior

Ejemplo:

Expresar la suma de los números enteros del 3 al 20 empleando la notación sigma

$$\sum_{i=3}^{20} i$$

- Se puede sustituir el + por otros operadores, tales como x, Λ (&) y V
- Para expresar un producto, se emplea el símbolo Π
- Si se desea obtener …

$$a_m \times a_{m+1} \times \ldots\ldots \times a_n$$

se escribe en la forma …

$$\prod_{i=m}^{n} a_i = a_m \times a_{m+1} \times \ldots\ldots \times a_n$$

- Así pues, se puede expresar …

$$\bigvee_{i=m}^{n} P_i = P_m \vee P_{m+1} \vee \ldots\ldots \vee P_n$$

$$\bigwedge_{i=m}^{n} Q_i = Q_m \wedge Q_{m+1} \wedge \ldots\ldots \wedge Q_n$$

- Generalizando …

$$\bigotimes_{i=m}^{n} R_i = R_m \otimes R_{m+1} \otimes \ldots\ldots \otimes R_n$$

PROGRAMAS RECURSIVOS

- Expresar la suma de los números enteros del 3 al 20 empleando la notación sigma

$$\sum_{i=3}^{20} i$$

```
Program Suma;
{Suma de los enteros del 3 al 20}
Var n,suma,i : integer;
Begin
    n:=20; {Inicializa cota sup}
    suma:=0; {Inicializa sumatoria}
    i:=3; {Inicializa cota inf}
    While i<n do
    Begin
        i:=i+1;
        suma:=suma+i;
    End;
End.
```

RECURSIVIDAD EN LOS PROGRAMAS

- Los procedimientos *recursivos* son aquellos que se invocan a sí mismos

Ejemplos: El método de ordenamiento Quick Sort y la Búsqueda Binaria

Datos en el Quick Sort:

Vector	Arreglo de números naturales
INI	Extremo izquierdo del subconjunto de elementos del Vector
FIN	Extremo derecho del subconjunto de elementos del Vector
flag	Bandera para controlar las iteraciones
pos	Posición del elemento pivote

```
void QSORT(int INI, int FIN)
{
  int izq=INI,der=FIN,pos=INI,aux,flag=1;

  while(flag)
  {
    flag=0;

    while(VECTOR[pos]<=VECTOR[der] && pos!=der)
      der--;

    if(pos!=der)
    {
      aux=VECTOR[pos]; VECTOR[pos]=VECTOR[der];
VECTOR[der]=aux;
      pos=der;
      izq++;

      while(VECTOR[pos]>=VECTOR[izq] && pos!=izq)
      izq++;

      if(pos!=izq)
      {
      aux=VECTOR[pos];  VECTOR[pos]=VECTOR[izq];
VECTOR[izq]=aux;
      pos=izq;
      der--;
      flag=1;
       }
    }
  }

  if(pos-1>INI)
    QSORT(INI,pos-1);
  if(FIN>pos+1)
    QSORT(pos+1,FIN);
  return;
}
```

Datos en la Búsqueda Binaria:

Vector	Arreglo de números naturales
Clave	Número que se desea buscar
Primero	Parámetro de inicio del intervalo donde se desea buscar
Ultimo	Parámetro final del intervalo donde se desea buscar
Pos	Indica la celda del arreglo que contiene el número buscado

- El vector está ordenado en forma ascendente, o sea, *Vector[i+1]>Vector[i]*

- El procedimiento *Buscar* recibe los parámetros *(Primero, Ultimo, Clave)* y trabaja con el intervalo comprendido entre *Primero* y *Ultimo*.

```
int BUSCAR(int primero, int ultimo, int clave)
{
int medio,pos=0;
   if(primero>ultimo)
     return(-1);

   medio=(primero+ultimo) / 2;

   if(clave<VECTOR[medio])
     pos=BUSCAR(primero, medio-1, clave);
   else
     if(clave>VECTOR[medio])
       pos=BUSCAR(medio+1, ultimo, clave);
     else
       pos=medio;
   return(pos);
}
```

DEMOSTRACIÓN POR RECURSIVIDAD

- Son *demostraciones inductivas* no necesariamente relacionadas con los números naturales
- Es mas general que la *inducción matemática* de los números naturales
- Se puede aplicar para la evaluación de expresiones lógicas de *cálculo de predicados*, ya que las proposiciones lógicas cumplen con ciertas propiedades
- P. ejem. Puede emplearse para mostrar que todas las expresiones lógicas pueden evaluarse por medio de su tabla de verdad.
- Para ello, se toma cualquier expresión lógica x y se demuestra que tiene un valor de verdad.
- Si x es atómica, su valor de verdad está dado por asignación (caso base)

- Si x es molecular, se divide en sus componentes y si cada uno de ellos tiene un valor de verdad, entonces x también lo tiene.
- De esta forma, el problema se reduce recursivamente a subproblemas (hasta alcanzar el caso base)

CONSIDERACIONES EN LA DEMOSTRACIÓN POR RECURSIVIDAD

- Es crucial que la *recursividad* llegue a su fin
- Se debe introducir la noción de *sucesión descendiente* (sucesión de elementos cada vez más pequeños)
- Si todas las *sucesiones descendientes* es **finita** se denomina *dominio bien fundado*.
- P. ejem. Al utilizar *recursividad* en *programación*, se debe reducir el problema a cada llamada recursiva; en caso contrario, el programa podría no terminar.

EL PRINCIPIO DE DEMOSTRACIONES POR RECURSIVIDAD

- Para demostrar que $\forall x\ P(x)$ es necesario efectuar los pasos:

Dominio bien fundado	Seleccionar un predicado G y demostrar las sucesiones finitas de G
Base inductiva	Si x es mínima, demostrar P(x)
Hipótesis inductiva	Se selecciona un x y se asume que P(x) para todos los *y* que satisfagan *G(x,y)*
Paso inductivo	Se demuestra P(x)
Conclusión	$\forall x\ P(x)$

ELIMINACIÓN DE LA RECURSIÓN

- La *recursividad* es una herramienta poderosa para crear algoritmos claros.
- Sin embargo, puede ocupar demasiada memoria
- Una subrutina, puede tener *variables locales* y *parámetros* y cuando se hace una llamada recursiva, éstos deben conservarse, así como la *dirección de retorno*.
- Para evitar la recursividad, se utilizan las *pilas* (almacenando los valores de las variables locales y los parámetros).
- P. ejem. Considere el siguiente vector a ordenar mediante el algoritmo Quick Sort, donde el número 15 es el pivote:

INI		pos					FIN
0	1	2	3	4	5	6	7
12	8	15	16	44	27	67	35
1er. Grupo		Pivote		2º. Grupo			

Al encontrar la posición final del elemento pivote, el vector original se divide en dos grupos:

1er. Grupo	Desde INI hasta pos-1
2º. Grupo	Desde pos+1 hasta FIN

3
0

PilaMenor

7
1

PilaMayor