

Algoritmos y Estructuras de Datos

Representación de algoritmos:

Existen varias formas de representar algoritmos. Diagramas de flujo (no recomendado), pseudocódigo, diagramas de Chapin, etc.

Adoptaremos en el curso los diagramas de Chapin.

Lenguajes de Programación:

Una vez desarrollado un algoritmo en Chapin, debemos traducirlo a un lenguaje que sea “comprendido por la computadora”. Hemos adoptado el lenguaje Pascal.

Constantes y variables:

A lo largo del desarrollo de un programa, habrá entidades que no sufren cambios (son las constantes) y otras que pueden ir cambiando su valor (son las variables).

Constantes:

Trabajaremos con constantes numéricas, de carácter y booleanas.

Dentro de las numéricas distinguiremos las que tienen “punto” decimal (**3.1415 -745.2 22.0 0.0 0.0256** etc.) que llamaremos *reales* y las que no tienen punto decimal (**17 -356 12988 0 22 -30000**) que son las *enteras*.

Las constantes tipo carácter son por ejemplo ‘c’, ‘Z’, ‘9’, ‘-’, ...

Las booleanas son dos **true** y **false**

Variables:

Son componentes del programa que pueden ir cambiando su valor o contenido a lo largo del desarrollo del algoritmo. Las variables son identificadas por medio de nombres o *identificadores*. (Pascal permite también identificar constantes por medio de identificadores).

Asignación:

En el desarrollo de nuestros algoritmos nos encontraremos con formas del tipo
variable ← expresión

esto es lo que llamamos *asignación* y significa “asignar a la variable el resultado de la expresión”

¿Cómo construimos una expresión? Por medio de la combinación de constantes, variables, operadores y funciones. También emplearemos paréntesis, cuyo uso ya conocemos

Operadores:

Tenemos operadores *aritméticos* , *relacionales* , *lógicos* , *de caracteres* , *etc.*

Operadores aritméticos:

Son los conocidos + (suma) , - (resta) , * (multiplicación) y / (división)

Para evaluar una expresión aritmética hay reglas en cuanto a la “fuerza” de los operadores + y - tienen igual fuerza , pero menos fuerza que * y / (la conocida regla de que “ + y - separan y * y / juntan”). Pero acá debemos agregar que si hay una sucesión de operadores de igual fuerza, se van evaluando las operaciones de izquierda a derecha. Por ejemplo si tenemos

$3 * xx / y + 12 * z * qual * m / 17 / ra / p / s * ty * w + 11 * a / b / c * nu$

estaríamos representando la siguiente expresión con notación algebraica común

$$\frac{3 \cdot xx}{y} + \frac{12 \cdot z \cdot qual \cdot m \cdot ty \cdot w}{17 \cdot ra \cdot p \cdot s} + \frac{11 \cdot a \cdot nu}{b \cdot c}$$

las funciones tienen mas fuerza que los operadores y las expresiones entre paréntesis deben ser evaluadas prioritariamente, respetando dentro de ellas las prioridades enunciadas.

Operadores de relación:

Se usan para comparar expresiones del mismo tipo, la mayoría de las veces expresiones aritméticas, pero pueden comparar expresiones de tipo carácter, booleanas (raro) (por definición **false** es menor que **true**), o incluso, en Pascal, datos definidos por el programador.

El resultado de estas comparaciones es **false** o **true**.

Los operadores son

< (menor)

<= (menor o igual)

= (igual)

> (mayor)

>= (mayor o igual)

<> (distinto)

supongamos que tenemos las variables **a = -3 b = 15 c = 8 jj = 'pepe' w = 'luisito'**

la expresión	3 * a + b < 16	devolverá el valor	true
la expresión	a + 2 * b > 4 * c	devolverá el valor	false
la expresión	a + b <> 12	devolverá el valor	false
la expresión	c + b <= 36	devolverá el valor	true
la expresión	w >= jj	devolverá el valor	false

Operadores lógicos:

Son **and** , **or** y **not** (y , o , no)

and y **or** vinculan dos expresiones lógicas y podríamos decir que el resultado del **and** sólo es verdadero si las dos expresiones que vincula son verdaderas; el **or** sólo resulta falso si las dos

expresiones son falsas. El **not** sólo se aplica a una expresión lógica, dando como resultado el opuesto a la expresión. Todo esto podemos representarlos así

p	q	p and q	p or q	not q
F	F	F	F	V
F	V	F	V	F
V	F	F	V	
V	V	V	V	

El **or** que hemos definido es el “inclusivo”. Existe otro **or**, el “exclusivo” pero no lo usaremos en el curso.

Se pueden armar expresiones lógicas complejas combinando operadores de relación y operadores lógicos. Para evaluar las mismas también existen reglas y prioridades, pero las explicaremos en cada caso particular si es que se hace necesario usar alguna.

Volvamos a la **sentencia de asignación**: $\text{variable} \leftarrow \text{expresión}$

La expresión debe ser del mismo tipo que la variable, es decir que si la variable es numérica, la expresión debe ser numérica; si la variable es booleana, la expresión debe ser booleana; etc.

Por ejemplo, si **x** e **y** son dos variables numéricas, la expresión booleana **x=y** tiene sentido y devolverá **true** si **x** e **y** son iguales y devolverá **false** si son distintas.

Si **z** es una variable booleana, tiene sentido la asignación $\mathbf{z} \leftarrow \mathbf{x=y}$

Si bien todavía no estamos en condiciones de comprender el siguiente programa, mas adelante pueden probarlo

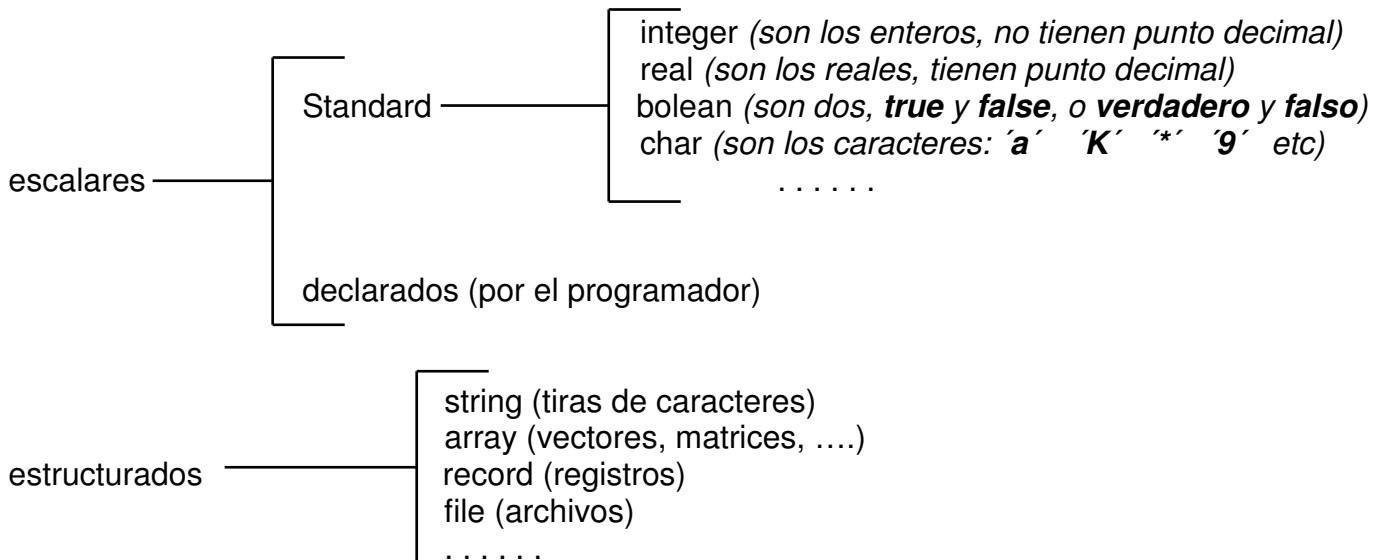
```
program asignboo(input , output);
var z: boolean; x, y: real; rta: char;
begin write('Ingrese dos reales'); readln(x, y);
      z := x=y;
      if z then writeln('La variable z quedo en TRUE')
        else writeln('La variable z quedo en FALSE');
      write('Ingrese nuevamente dos reales'); readln(x, y);
      z := x=y;
      if z then writeln('La variable z quedo en TRUE')
        else writeln('La variable z quedo en FALSE');
      write('Pulse cualquier tecla para terminar . . .'); read(rta)
end.
```

Ingresa la primera vez que pide dos valores, por ejemplo, los valores **3.75** y **3.75** y verán que la respuesta es **La variable z quedo en TRUE**.

Luego, cuando vuelve a pedir otros dos valores, ingresa por ejemplo **9.01** y **2.22** ahora la respuesta será **La variable z quedo en FALSE**.

Tipos de datos:

A lo largo del curso trabajaremos con diferentes tipos de datos los que podemos clasificarlos de la siguiente manera



Nótese que hemos puesto puntos suspensivos ya que la mayoría de los compiladores (éste incluido) admite mas tipos de datos. Por ejemplo, simplemente como comentario, hacemos notar otros tipos de enteros que generalmente son admitidos:

shortint : enteros entre -128 y 127

longint : enteros entre -2147483648 y 2147483647 *(puede llegar a ser muy útil este tipo)*

byte : enteros positivos entre 0 y 255

word : enteros positivos entre 0 y 65535

veamos como ejemplo directamente un programa en Pascal

```

program tipos(output);
var ashor, negashor: shortint;
    ainte, negainte: integer;
    along, negalong: longint;
    abyte, negabyte: byte;
    aword, negaword: word;
    rta:char;
begin ashor:= 123; negashor:= -123;
      ainte:= 12345; negainte:= -12345;
      along:= 1234567890; negalong:= -1234567890;
      abyte:= 234; negabyte:= -1 * abyte;      Pascal no acepta una asignación
      aword:= 65432; negaword:= -1 * aword;     directa fuera del rango
      writeln('Tipo shortint ', ashor, ' ', negashor);
      writeln('Tipo integer ', ainte, ' ', negainte);
      writeln('Tipo longint ', along, ' ', negalong);
      writeln('Tipo byte ', abyte, ' ', negabyte);
      writeln('Tipo word ', aword, ' ', negaword);
      write('Fin, pulse...');
  
```

```

    read(rta)
end.

```

y la salida que genera este programa es

```

Tipo shortint 123      -123
Tipo integer 12345     -12345
Tipo longint 1234567890 -1234567890
Tipo byte 234         22
Tipo word 65432       104
Fin, pulse...

```

este valor (22) y el valor siguiente (104) se deben a que al calcular el producto por -1 nos salimos fuera del rango, hecho que no podía prever el compilador, podemos pensar que los domicilios de las variables "rebalsaron", produciendo un resultado "inesperado". Lo mismo pudo haber ocurrido con los otros tipos.

Escalares:

Son aquellos que no tienen ninguna estructura

Standard: No necesitan ser declarados. Los reconoce Pascal directamente

Declarados: Pascal permite al programador definir tipos especiales de datos.

Son reconocidos únicamente dentro del módulo donde fueron creados

Escalares declarados:

El programador puede "inventar" datos a los efectos de su programa, este nuevo tipo de dato nace y muere en ese módulo. Por ejemplo

```

. . . . .
type diasem = (lun, mar, mie, jue, vie, sab, dom);
var x, y : diasem;
. . . . .

```

a partir de este momento, el programa reconocerá además de los tipos de datos Standard, el tipo **diasem**, un nuevo tipo escalar en el cual la relación de orden la da el orden en que fueron definidos.

vale la relación como en los Standard **vie<dom** , **sab>lun** , etc.

pueden hacerse asignaciones **x := vie** , **y := lun** y por ejemplo

x <= y devolverá como resultado **false**

x > mar devolverá como resultado **true**

Tipo subrango:

Son subconjuntos de los escalares. Se pueden crear subrangos de cualquier escalar, excepto real
 [17 .. 25] ['j' .. 'p'] [mar .. sab]

Estructurados:

Tienen una cierta estructura u orden y los veremos en su momento.

Estructura de un programa Pascal:

Un programa Pascal puede constar de siete partes, de las cuales sólo la primera y la última son obligatorias.

- 1 – Encabezamiento
- 2 – Parte de declaración de rótulos
- 3 – Parte de definición de constantes
- 4 – Parte de definición de tipos
- 5 – Parte de declaración de variables
- 6 – Parte de declaración de procedimientos y funciones
- 7 – Parte de sentencias

Encabezamiento:

Da al programa su nombre y lista los parámetros a través de los cuales el programa se comunicará con su entorno.

Declaración de rótulos:

Se puede marcar cualquier sentencia de un programa anteponiéndole un rótulo seguido del carácter : (*de esta manera podemos referenciarla desde la sentencia **goto**, pero a esta sentencia evitaremos usarla, por lo menos al principio del aprendizaje*). Los rótulos deben ser declarados en esta sección, la que comienza con la palabra **label**.

Definición de constantes:

Sirve para introducir un identificador como sinónimo de una constante. Esta sección comienza con la palabra **const**.

Definición de tipos:

Un tipo de datos puede ser descrito directamente en la declaración de la variable, o referido mediante un *identificador de tipo* en esta sección, la que comienza con la palabra **type**.

Declaración de variables:

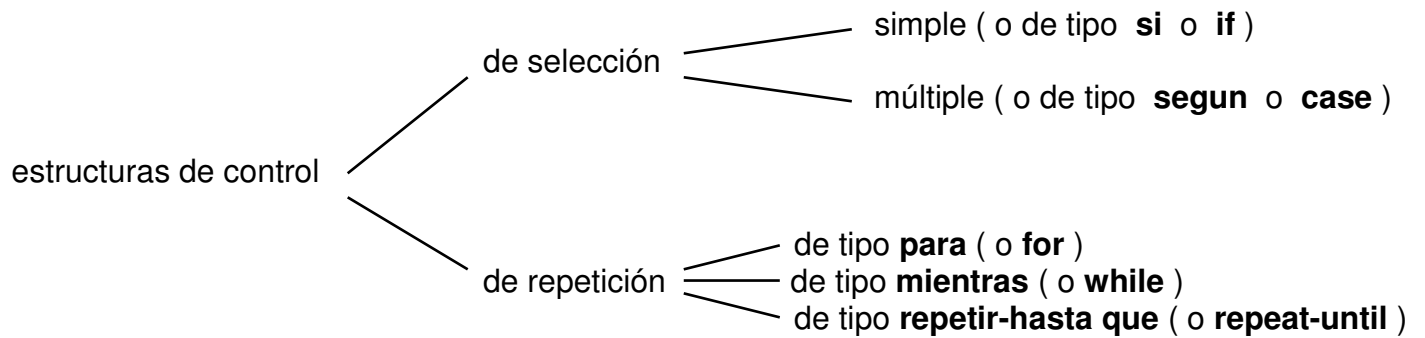
Toda variable que aparezca en el programa, debe ser declarada en esta sección. Esta sección comienza con la palabra **var**.

Declaración de procedimientos y funciones: Todo procedimiento o función debe definirse en esta sección, no comienza con ninguna palabra en especial, sino que el encabezamiento de cada función o procedimiento es lo que indica de que tipo de subprograma se trata.

Sintaxis del Pascal:

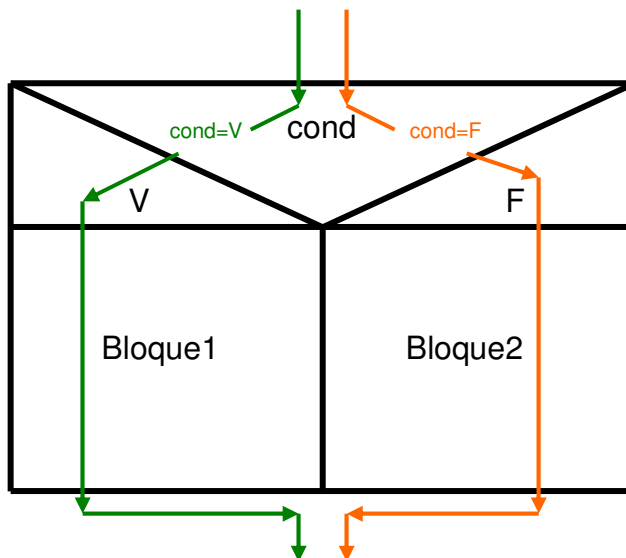
La sintaxis está definida en los **Diagramas Sintácticos** que son entregados al alumno oportunamente.

Nuestros algoritmos serán representados en diagramas de Chapin utilizando una manera gráfica con algunas estructuras que son las que detallamos a continuación

Estructuras de control

Estructura de selección simple: o de tipo **si** o **if**. Permite seleccionar uno de entre dos posibles caminos según sea el valor de una condición.

En Chapin la representamos así



donde:

cond: es una expresión booleana, es decir algo que puede ser *verdadero* o *falso*

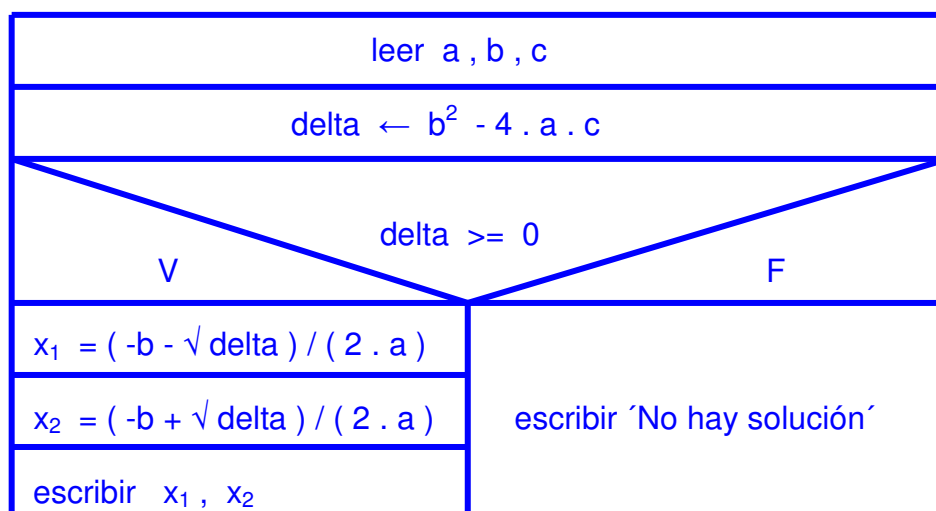
Bloque1: es un conjunto de una o mas acciones

Bloque2: es un conjunto de cero o mas acciones

funciona de la siguiente manera: se ingresa al bloque, se evalúa la condición, si es Verdadera, se ejecuta el Bloque1 y se abandona la estructura, si es Falsa, se ejecuta el Bloque2 y se abandona la estructura.

No hay restricciones para las estructuras que conforman los bloques, pueden contener otros **si** o cualquier otra estructura que definamos de aquí en mas.

Problema: Leer los tres coeficientes **a** (distinto de cero), **b**, **c**, de una ecuación de segundo grado y resolverla en el campo real.



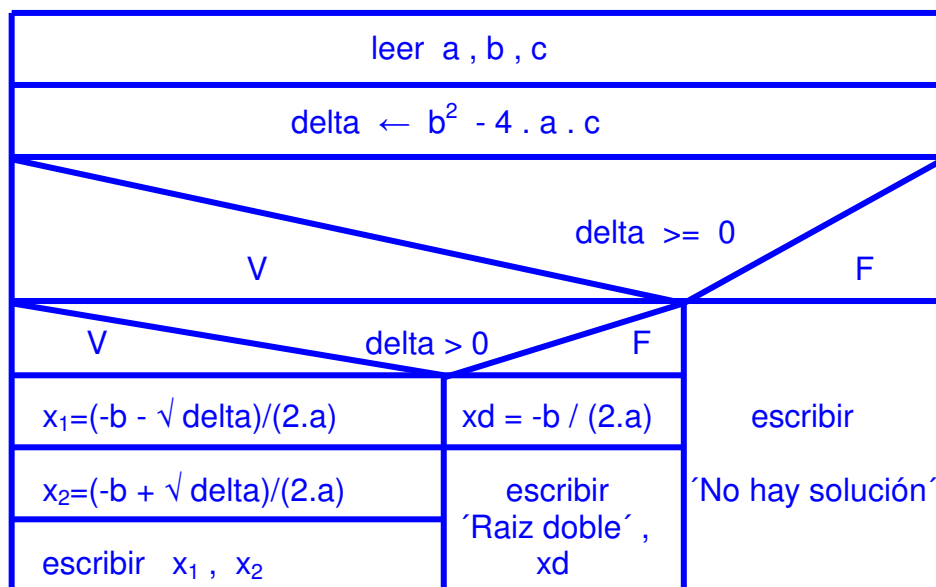
y en Pascal

```

program ecua (input,output) ;
var a , b , c , delta , x1 , x2 : real ;
begin  read ( a , b , c ) ;
      delta := b * b - 4 * a * c ;
      if delta >= 0 then begin x1 := ( -b - sqrt(delta) ) / ( 2 * a ) ;
                             x2 := ( -b + sqrt(delta) ) / ( 2 * a ) ;
                             write ( x1 , x2 )
                           end
      else write ( 'No hay solucion' )
end .

```

supongamos que deseamos distinguir el caso de raíces reales y distintas del de raíz doble. Tenemos que poner una estructura **si** dentro de otra **si**.



y en Pascal

```

program ecua2 (input,output);
var a, b, c, delta, x1, x2, xd : real ;
begin  read( a, b, c);
      delta := b*b - 4*a*c;
      if delta >= 0 then
        if delta > 0 then begin x1 := (-b - sqrt(delta)) / (2*a) ;
                               x2 := (-b + sqrt(delta)) / (2*a) ;
                               write (x1, x2)
                             end
        else begin xd := -b / (2*a);
                  write ( 'Raiz doble' , xd)
                end
      else write ( 'No hay solucion' )
end .

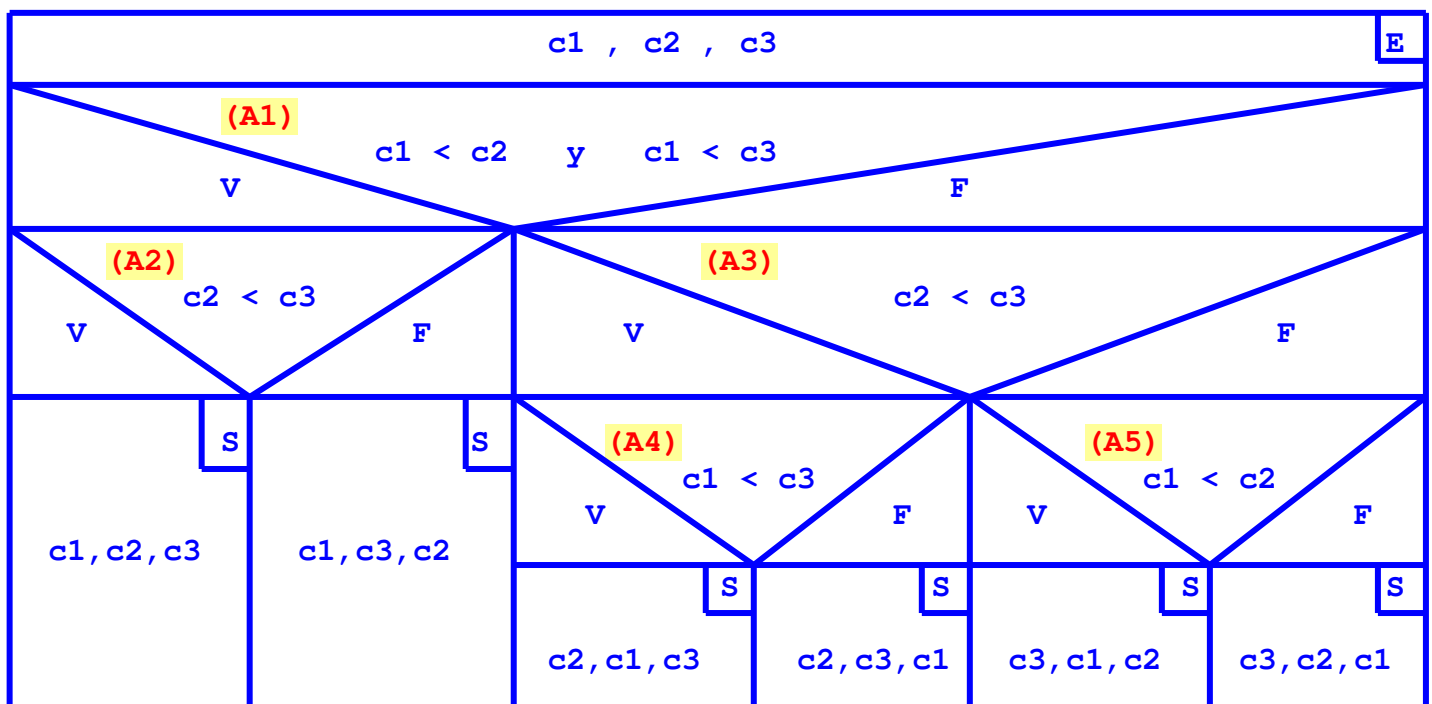
```

A continuación plantearemos un problema y veremos como generalmente, para un mismo problema puede haber varios algoritmos que lo solucionen, y tal vez todos medianamente eficientes (el alumno puede omitir esta porción del apunte entre los símbolos ■)



Problema: Leer tres letras mayúsculas e imprimirlas ordenadas (clasificadas) alfabéticamente.
(llamaremos $c1$, $c2$, $c3$ a las variables que contendrán estas letras mayúsculas)

Método 1 : Primero vemos si la menor es $c1$, si es así vemos entre $c2$ y $c3$ cual es menor e imprimimos en consecuencia si no es así, vemos si la menor es $c2$ y en caso afirmativo comparamos $c1$ con $c3$ e imprimimos en consecuencia, en caso de que también esto sea falso, evidentemente la menor es $c3$, comparamos $c1$ con $c2$ e imprimimos según corresponda.

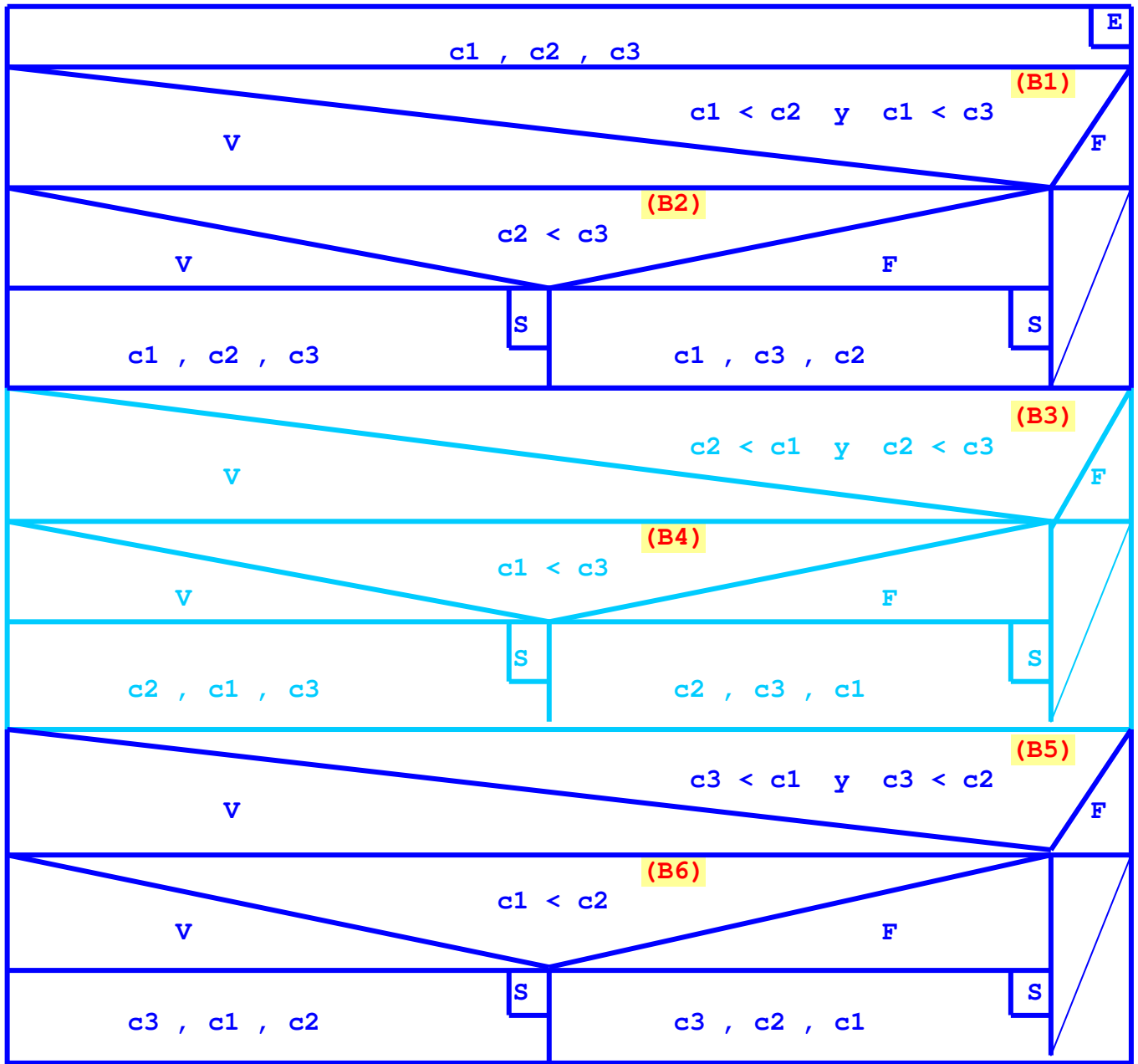


```

program clasifica3(input,output);
var c1,c2,c3:char;
begin
  read(c1,c2,c3);
  if c1<c2 and c1<c3 then if c2<c3 then write(c1,c2,c3)
                        else write(c1,c3,c2)
  else if c2<c3 then if c1<c3 then write(c2,c1,c3)
                    else write(c2,c3,c1)
  else if c1<c2 then write(c3,c1,c2)
                    else write(c3,c2,c1)
end.
  
```

La anterior tal vez sea la forma mas ortodoxa de resolver el problema. Veamos otra

Método 2 : Otra forma sería poner tres “**IF**”, uno a continuación de otro buscando cual es el menor de los tres caracteres (sin poner nada en la bifurcación por **FALSO**), de esta manera uno de los tres tomará la situación y a los otros se los pasará de largo. Después, ya dentro del **IF** correspondiente se buscará cual de los restantes dos caracteres es el menor, y se imprimirá en consecuencia. Tal vez esta otra solución sea mas fácil de comprender. Veámosla



```

program clasifica3bis(input,output);
var c1,c2,c3:char;
begin read(c1,c2,c3);
    if c1<c2 and c1<c3 then if c2<c3 then write(c1,c2,c3)
                                else write(c1,c3,c2);
    if c2<c1 and c2<c3 then if c1<c3 then write(c2,c1,c3)
                                else write(c2,c3,c1);
    if c3<c1 and c3<c2 then if c1<c2 then write(c3,c1,c2)
                                else write(c3,c2,c1)
end.
  
```

Comparemos ambas soluciones. Si bien la segunda es mas clara para leer (tanto el Chapin como el Pascal), es menos eficiente. Veamos por que:

Supongamos que los tres caracteres sean 'D', 'P' y 'H', es decir $c1='D'$, $c2='P'$ y $c3='H'$,

En el primer algoritmo, se ejecutaría el IF (A1), luego el IF (A2), luego un WRITE y se terminaría el algoritmo.

En cambio, en el segundo algoritmo, se ejecutaría el IF (B1), luego el IF (B2), luego un WRITE, luego el IF (B3), luego el IF (B5), y se terminaría el algoritmo. *Hemos realizado en el segundo algoritmo dos IFs de mas.*

Supongamos que los tres caracteres sean 'M', 'B' y 'W', es decir $c1='M'$, $c2='B'$ y $c3='W'$,

En el primer algoritmo, se ejecutaría el IF (A1), luego el IF (A3), luego el IF (A4), luego un WRITE y se terminaría el algoritmo.

En cambio, en el segundo algoritmo, se ejecutaría el IF (B1), luego el IF (B3), luego el IF (B4), luego un WRITE, luego el IF (B5), y se terminaría el algoritmo. *Hemos realizado en el segundo algoritmo un IF de mas.*

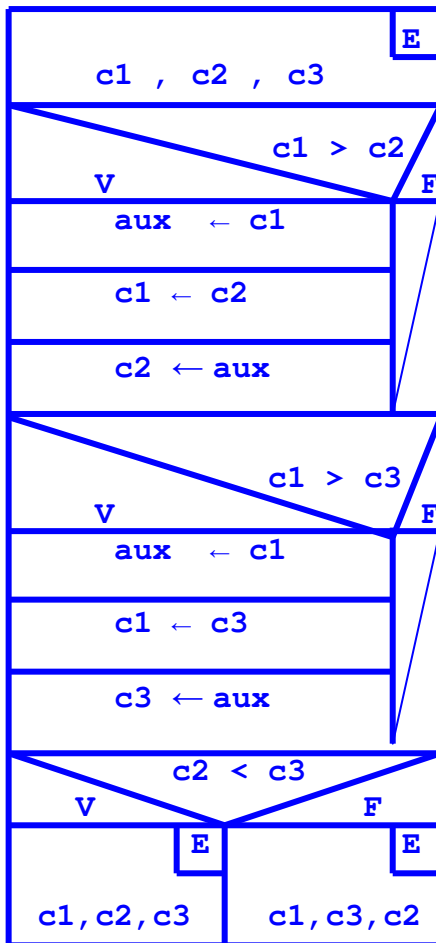
Si el juego de tres caracteres es 'Z', 'F' y 'M', es decir $c1='Z'$, $c2='F'$ y $c3='M'$, no hay diferencia en cuanto a ambos algoritmos (queda para el alumno la verificación).

De todos modos, ya que no conocemos el juego de datos que entrará, concluimos que el primer algoritmo es mas eficiente.

Pero tenemos otra posibilidad de resolver el problema que tal vez no sea la primer forma que nos viene a la mente al intentar el algoritmo, pero que tal vez sea tanto o mas eficiente que la primera

Método 3 : Leamos las tres letras e intercambiemos para dejar la menor en c1. Para esto comparo c1 con c2 y si c1 es mayor que c2 las intercambio mediante un valor auxiliar. En c1 quedó la menor de esas dos. Luego comparo c1 con c3 y hago lo mismo que en el caso anterior. Estoy seguro que en c1 está la letra mas chica de las tres. Luego simplemente comparo c2 con c3 e imprimo en consecuencia.

Pongamos este método en Chapin y Pascal



```

program clasifica3tres(input, output) ;
var c1 , c2 , c3 : char ;
begin if c1>c2 then begin aux := c1 ;
                        c1 := c2 ;
                        c2 := aux
                        end ;
      if c1>c3 then begin aux := c1 ;
                        c1 := c3 ;
                        c3 := aux
                        end
      if c2<c3 then write(c1,c2,c3)
      else write(c1,c3,c2)
end.
  
```

Por si desean probarlos en la computadora hemos juntado los tres métodos en un solo programa y ejecutamos cada uno para las seis posibilidades de orden en que hubiesen podido entrar, y en vez de leer las ternas seis veces, las hemos asignado utilizando un **case** (se verá en el punto siguiente), formando las seis permutaciones posibles. Hemos supuesto que las tres letras a comparar son **P Q R**.

Las seis posibilidades son **PQR** , **PRQ** , **QPR** , **QRP** , **RPQ** y **RQP**

```

program TresOrd3(output) ;
var c1, c2, c3, rta, aux: char; h, m, s, hund: word; indi: integer;
begin
  for indi:=1 to 6 do
    begin
      case indi of
        1: begin c1 := 'P' ; c2 := 'Q' ; c3 := 'R' end ;
        2: begin c1 := 'P' ; c2 := 'R' ; c3 := 'Q' end ;
        3: begin c1 := 'Q' ; c2 := 'P' ; c3 := 'R' end ;
        4: begin c1 := 'Q' ; c2 := 'R' ; c3 := 'P' end ;
        5: begin c1 := 'R' ; c2 := 'P' ; c3 := 'Q' end ;
        6: begin c1 := 'R' ; c2 := 'Q' ; c3 := 'P' end
      end;
      write('Metodo 1 - ',c1,c2,c3,' ');
      if ((c1<c2) and (c1<c3)) then
        if c2<c3 then writeln(c1,c2,c3)
      end;
    end;
  end;
end.
  
```

```

        else writeln(c1,c3,c2)
      else if ((c2<c1) and (c2<c3)) then
        if c1<c3 then writeln(c2,c1,c3)
        else writeln(c2,c3,c1)
        else
          if c1<c2 then writeln(c3,c1,c2)
          else writeln(c3,c2,c1)
      end; writeln;
for indi:=1 to 6 do
begin
  case indi of
    1: begin c1 := 'P' ; c2 := 'Q' ; c3 := 'R' end ;
    2: begin c1 := 'P' ; c2 := 'R' ; c3 := 'Q' end ;
    3: begin c1 := 'Q' ; c2 := 'P' ; c3 := 'R' end ;
    4: begin c1 := 'Q' ; c2 := 'R' ; c3 := 'P' end ;
    5: begin c1 := 'R' ; c2 := 'P' ; c3 := 'Q' end ;
    6: begin c1 := 'R' ; c2 := 'Q' ; c3 := 'P' end
  end;
  write('Metodo 2 - ',c1,c2,c3,' ');
  if ((c1<c2) and (c1<c3)) then if c2<c3 then writeln(c1,c2,c3)
                                else writeln(c1,c3,c2);
  if ((c2<c1) and (c2<c3)) then if c1<c3 then writeln(c2,c1,c3)
                                else writeln(c2,c3,c1);
  if ((c3<c1) and (c3<c2)) then if c1<c2 then writeln(c3,c1,c2)
                                else writeln(c3,c2,c1);

  end; writeln;
for indi:=1 to 6 do
begin
  case indi of
    1: begin c1 := 'P' ; c2 := 'Q' ; c3 := 'R' end ;
    2: begin c1 := 'P' ; c2 := 'R' ; c3 := 'Q' end ;
    3: begin c1 := 'Q' ; c2 := 'P' ; c3 := 'R' end ;
    4: begin c1 := 'Q' ; c2 := 'R' ; c3 := 'P' end ;
    5: begin c1 := 'R' ; c2 := 'P' ; c3 := 'Q' end ;
    6: begin c1 := 'R' ; c2 := 'Q' ; c3 := 'P' end
  end;
  write('Metodo 3 - ',c1,c2,c3,' ');
  if c2<c1 then begin aux:=c1 ; c1:=c2 ; c2:=aux end;
  if c3<c1 then begin aux:=c1 ; c1:=c3 ; c3:=aux end;
  if c2<c3 then writeln(c1,c2,c3)
                else writeln(c1,c3,c2)

  end;
  write('Pulse una tecla . . .'); read(rta)
end.

```



Estructura de Selección Múltiple: (o de tipo CASE, o SEGÚN SEA) Permiten seleccionar uno de entre varios caminos

Esta estructura es la que mas difiere con las implementaciones en un Lenguaje u otro.

Se usa principalmente cuando debe armarse un menú de opciones de la forma:

¿ Qué tarea desea hacer ?

1 – Altas

2 – Bajas

3 – Modificaciones

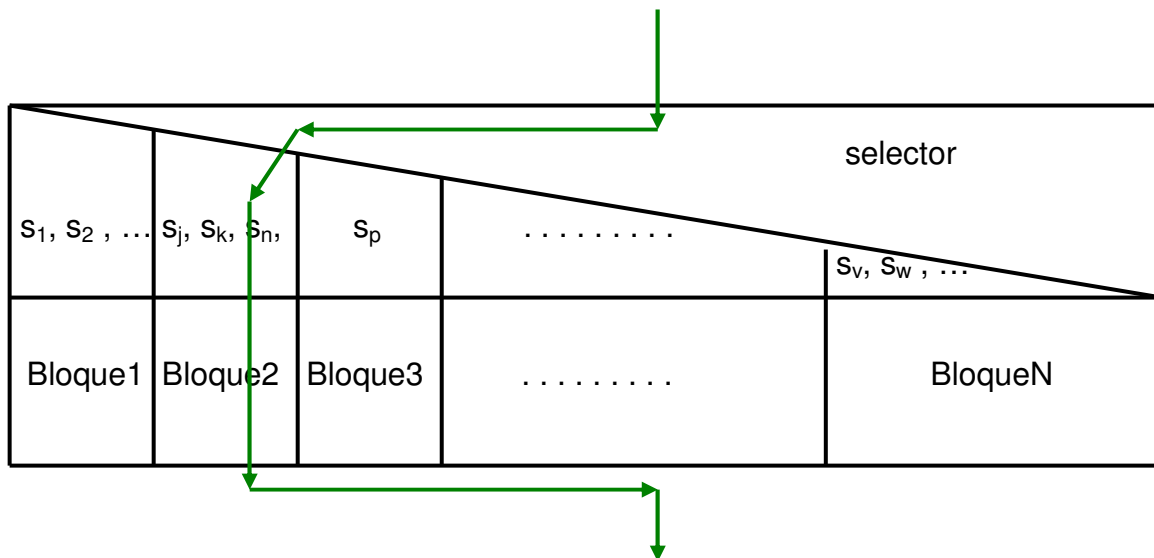
4 – Listados

5 – Terminar

Elija su opción:

La simbolizaremos así

(las flechas en verde son para el ejemplo posterior)



donde:

selector: es una variable simple de cualquier tipo, excepto *real* (la mayoría de las veces de tipo integer)

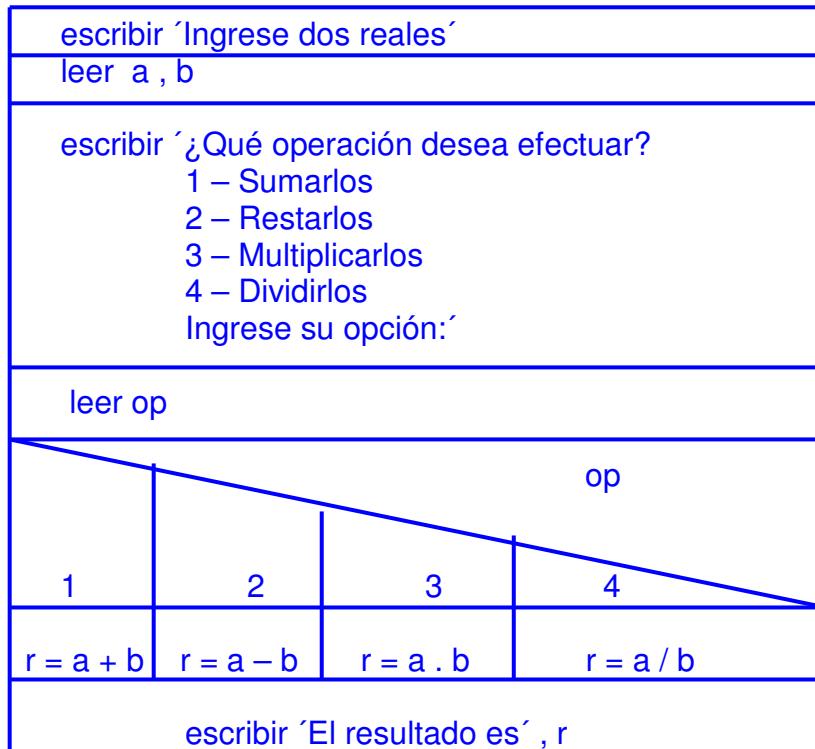
S_1, S_2, \dots, S_w son constantes, del mismo tipo que el selector

funciona de la siguiente manera: se ingresa a la estructura, se evalúa el selector, se busca ese valor dentro de las constantes S_i y se ejecuta el bloque de la columna donde aparezca.

No hay restricciones para las estructuras que conforman los bloques, pueden contener otros **según** o **si** o cualquier otra estructura que definamos de aquí en mas.

ejemplifiquemos: supongamos que ingresamos a la estructura, evaluamos el selector y su valor es S_k , ejecutamos el Bloque2 y salimos de la estructura.

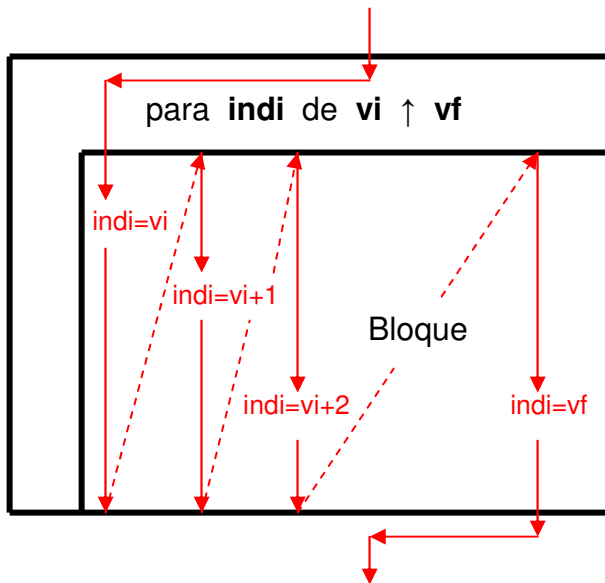
Problema: Leer dos números reales (el segundo distinto de cero) y efectuar una operación (suma, resta, multiplicación o división) según un menú de opciones.



```

program operación (input , output) ;
var a , b , r : real ; op : integer ;
begin write ('Ingrese dos reales') ;
      read ( a , b ) ;
      writeln ;
      writeln ('¿Que operación desea efectuar?') ;
      writeln ('1 – Sumarlos') ;
      writeln ('2 – Restarlos') ;
      writeln ('3 – Multiplicarlos') ;
      writeln ('4 – Dividirlos') ;
      write ('Ingrese su opción:') ;
      read ( op ) ;
      case op of 1 : r := a + b ;
                 2 : r := a - b ;
                 3 : r := a * b ;
                 4 : r := a / b
      end ;
      write ('El resultado es' , r )
end.
  
```

Estructura de repetición de tipo *para*: o de tipo **for**. Permite repetir un bloque de sentencias un número conocido de veces



donde:

indi: es una variable simple de cualquier tipo (excepto *real*)

vi y **vf** son expresiones del mismo tipo que **indi**

Bloque: es un conjunto de una o mas acciones

funciona de la siguiente manera: (lo explicaremos para el caso en que **indi**, **vi**, **vf** sean tipo integer)
Se ingresa a la estructura, se recorre el Bloque por primera vez con un valor de **indi** igual a **vi**. Se salta nuevamente a la cabecera y se recorre nuevamente el bloque con **indi=vi+1**, y así sucesivamente

hasta recorrerlo por última vez con **indi=vf**.

No hay restricciones para las estructuras que conforman el bloque, pueden contener otros **para** o **según** o **si** o cualquier otra estructura que definamos de aquí en mas.

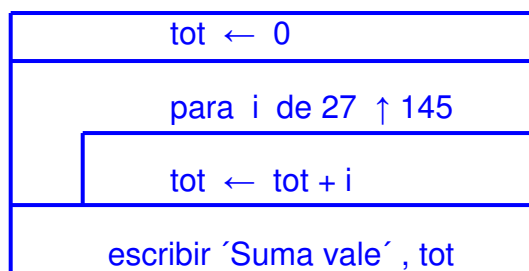
Tenemos que hacer varias observaciones:

- si **indi**, **vi**, **vf** fuesen char en vez de **indi=vi+1** tendríamos que decir **indi=el sucesor de vi**, etc.
- si al intentar ingresar a la estructura fuese **vi>vf**, se salta la estructura y el bloque no se ejecuta nunca
- Pascal no tiene incremento, es decir que el valor del índice siempre será el siguiente (si se desea un escalón hay que programarlo dentro del bloque)

Pascal tiene el **para descendente** y en Chapin lo indicaremos con ↓ en vez de ↑. El índice, en vez de ir creciendo va decreciendo y en cuanto a las observaciones, son análogas.

Problema: Calcular la suma de los números enteros entre 27 y 145 (ambos incluidos).

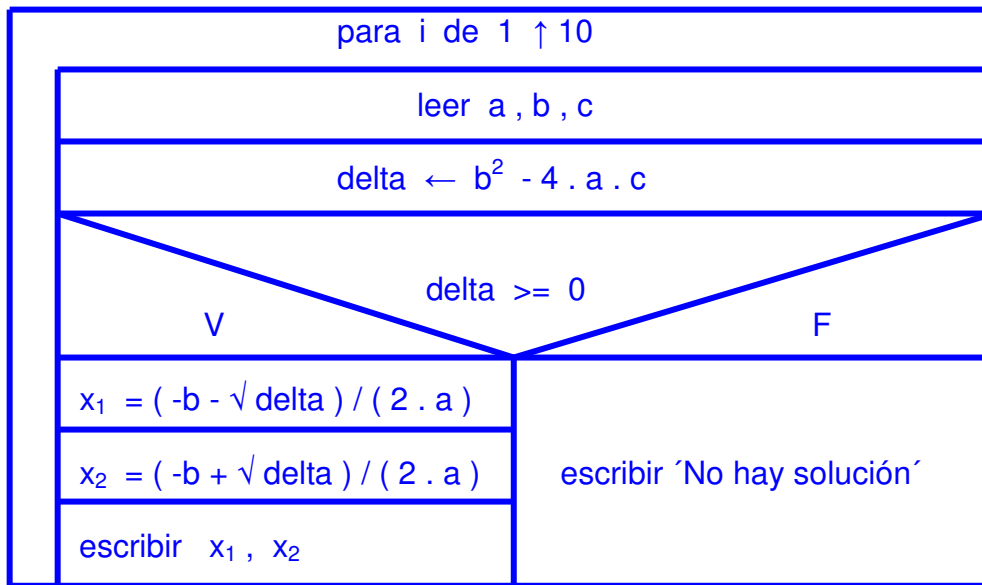
(aclaramos que se puede calcular mediante una simple fórmula, pero vamos a usar un **para**)



```
program unasuma (input , output) ;
var tot , i : integer ;
begin tot := 0 ;
      for i := 27 to 145 do tot := tot + i ;
      write ('Suma vale' , tot)
end.
```

hacemos notar que dentro del Bloque hemos usado el índice del **para**. Veamos un ejemplo donde no se usa, simplemente funciona como un contador.

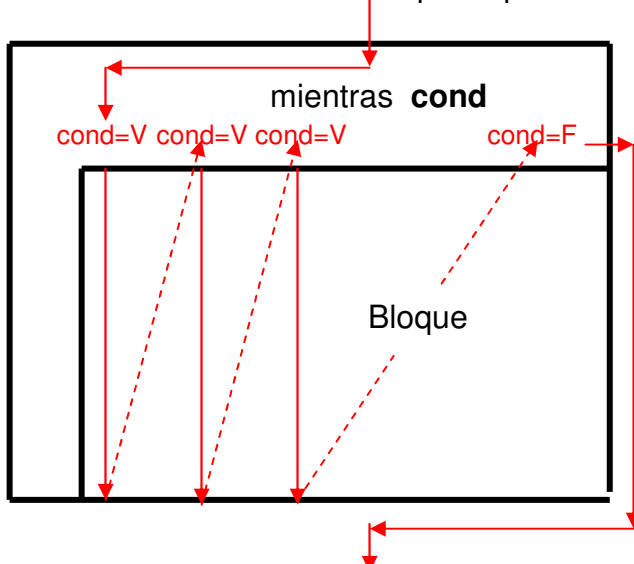
Problema: Leer los tres coeficientes **a** (distinto de cero), **b**, **c**, de diez ecuaciones de segundo grado e ir resolviéndolas en el campo real a medida que se lee cada terna.



```

program ecuas (input,output) ;
var a , b , c , delta , x1 , x2 : real ;
    i : integer ;
begin  for i:=1 to 10 do begin
        readln(a,b,c);
        delta := b*b - 4*a*c;
        if delta >= 0 then
            begin x1 := (-b - sqrt(delta))/(2*a);
                  x2 := (-b + sqrt(delta))/(2*a);
                  writeln(x1,x2)
            end
        else
            write ( 'No hay solucion' )
        end
    end
end .
  
```

Estructura de repetición de tipo mientras: o de tipo **while**. Permite repetir un bloque de sentencias un cierto número de veces lo que depende del resultado de una condición.



donde:

cond: es una expresión booleana

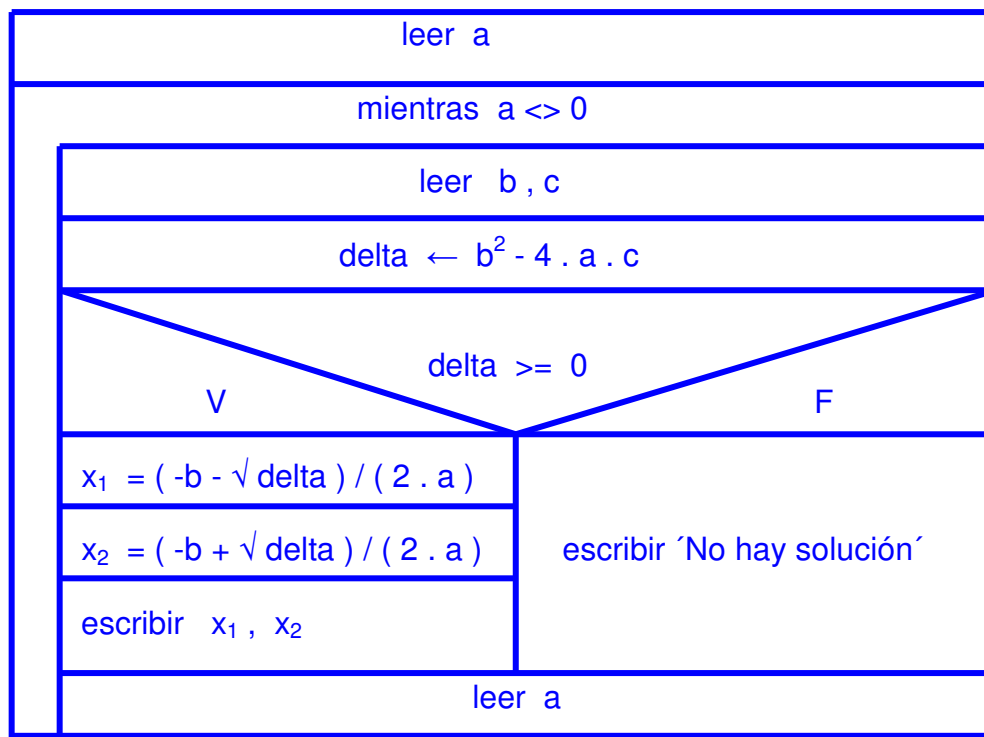
Bloque: es un conjunto de una o mas acciones

funciona de la siguiente manera: ingresamos a la estructura, evaluamos la condición, si es verdadera se recorre el Bloque; se salta a la cabecera, se vuelve a recorrer el Bloque; y así sucesivamente, mientras la condición sea verdadera. Cuando al volver a la cabecera, la condición es falsa, se abandona la

estructura sin recorrer el Bloque. Si al intentar ingresar por primera vez a la estructura, la condición es falsa, se sale de la misma sin ejecutar el bloque. Conclusión: en el **mientras** podría ocurrir que el bloque no se ejecute nunca.

Problema: Leer los tres coeficientes **a** (distinto de cero), **b**, **c**, de varias ecuaciones de segundo grado (no se sabe cuantas) e ir resolviéndolas en el campo real a medida que se lee cada terna.

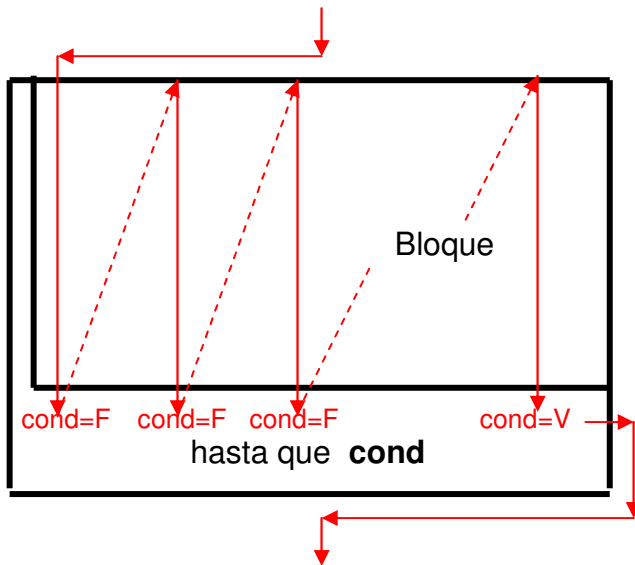
Acá se nos presenta un problema muy común en algoritmia, tenemos que reiterar un cierto conjunto de acciones pero no se sabe cuantas veces, no vamos a usar un **para** ya que este lo usaremos cuando al ingresar a la estructura se conoce la cantidad de veces que se debe repetir el Bloque. Usaremos un **mientras**, pero ahora la cuestión es como indicamos el fin de la repetición. Esto se consigue mediante un hecho (una lectura, un cálculo,...) que no pueda ser interpretado como un dato, sino como una señal de fin de entrada de los mismos. En este ejemplo será ingresando un valor de **cero** para **a**



```

program ecuas2 (input,output) ;
var a , b , c , delta , x1 , x2 : real ;
begin read(a) ; while a<>0 do begin
    readln(b,c) ;
    delta := b*b - 4*a*c;
    if delta >= 0 then
        begin x1 := (-b - sqrt(delta))/(2*a);
              x2 := (-b + sqrt(delta))/(2*a);
              writeln(x1,x2)
        end
    else
        writeln ( 'No hay solucion' )
    end
    read(a)
end
end .
  
```

Estructura de repetición de tipo repetir: o de tipo **repeat**. Permite repetir un bloque de sentencias un cierto número de veces que depende del resultado de una condición.



donde:

cond: es una expresión booleana

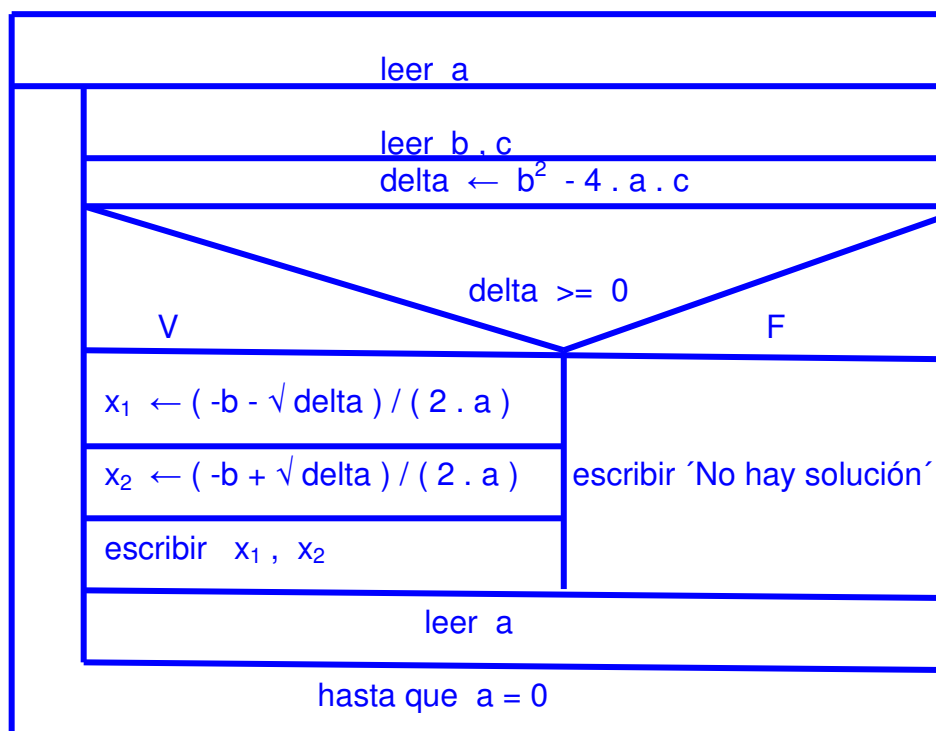
funciona de la siguiente manera: ingresamos a la estructura, recorremos el Bloque (siempre recorremos el Bloque por lo menos una primera vez); evaluamos la condición, si es falsa, saltamos al comienzo del Bloque y volvemos a recorrerlo; y así sucesivamente hasta que la condición sea verdadera. Cuando la condición es verdadera, se abandona la estructura.

Esta estructura también la podemos usar cuando al ingresar a la misma, no se sabe la cantidad de veces que debe repetirse el Bloque. Presta una utilidad muy similar al **mientras**, tan es así que hay lenguajes que no la tienen implementada. Sin embargo, conceptualmente es opuesta al **mientras**, ya que en el **mientras**, se ejecuta el Bloque mientras la condición es verdadera, y se abandona la estructura en el momento que la condición se hace falsa. En el **repetir**, en cambio, el Bloque se ejecuta en tanto y en cuanto la condición sea falsa, y se abandona la estructura cuando la condición se hace verdadera.

Otra diferencia, es que en el **repetir** como acabamos de ver, el Bloque se ejecuta por lo menos una vez, en cambio en el **mientras**, podría no ejecutarse nunca.

Problema: Leer los tres coeficientes **a** (distinto de cero), **b**, **c**, de varias ecuaciones de segundo grado (no se sabe cuantas, pero por lo menos una) e ir resolviéndolas en el campo real a medida que se lee cada terna.

Nuevamente usaremos como valor de corte **a = 0**.



```

program ecuas3 (input,output);
var  a, b, c, x1, x2, delta : real ;
begin  read( a ) ;
      repeat read( b, c ) ;
            delta := b*b - 4*a*c ;
            if delta >= 0 then begin x1 := (-b - sqrt(delta))/(2*a);
                                   x2 := (-b + sqrt(delta))/(2*a);
                                   write(x1,x2)
                                end
            else write ('No hay solución');
            read( a )
        until a = 0
end.

```

Vale la pena hacer notar que hemos resuelto el mismo problema con un **mientras** y con un **repetir**, pero la condición fue exactamente una contraria a la otra.

Validación de una entrada:

Supongamos que queremos leer un entero que debe ser mayor o igual que 7 y menor o igual que 15. Lo haremos con un **mientras** y con un **repetir**

con while

```

.
.
write('Ingrese un entero entre 7 y 15');
read( n );
while n < 7 or n > 15 do
begin write('Error, ingrese de nuevo');
      read( n )
end ;
.
.

```

con repeat

```

.
.
.
repeat write('Ingrese un entero entre 7 y 15');
      read( n )
until n >= 7 and n <= 15;
.
.
.

```

Ambas funcionan bien, pero con el **while** resulta mas amigable.
 Notar nuevamente que las condiciones son opuestas.

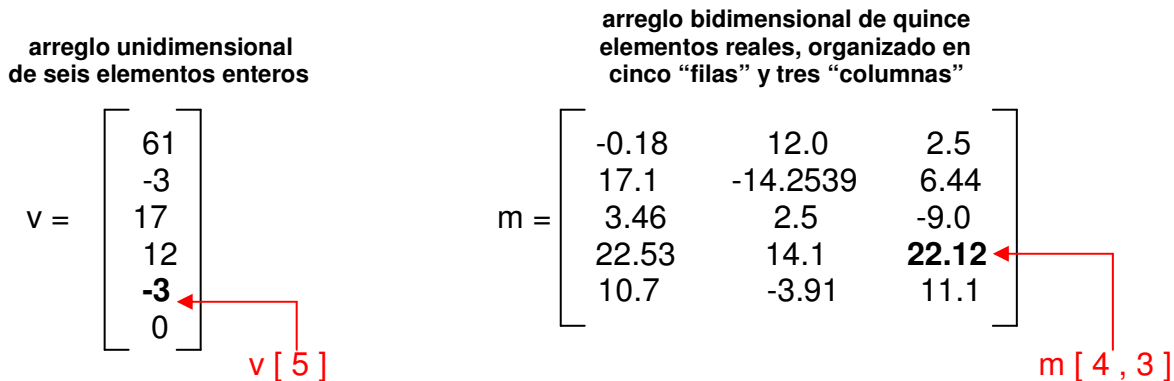
Tipo Array (Arreglos):

Es un tipo de dato estructurado. Permite agrupar en una entidad varios datos. Tiene las siguientes características:

- Tienen tamaño fijo
- Todos sus elementos son del mismo tipo
- El tipo de sus elementos puede ser cualquiera (real, string, otro array, record....)
- Están ubicados en la memoria principal

Un tipo de arreglo son nuestros conocidos, vectores y matrices

Los elementos del arreglo son referenciados a través de índices de manera unívoca



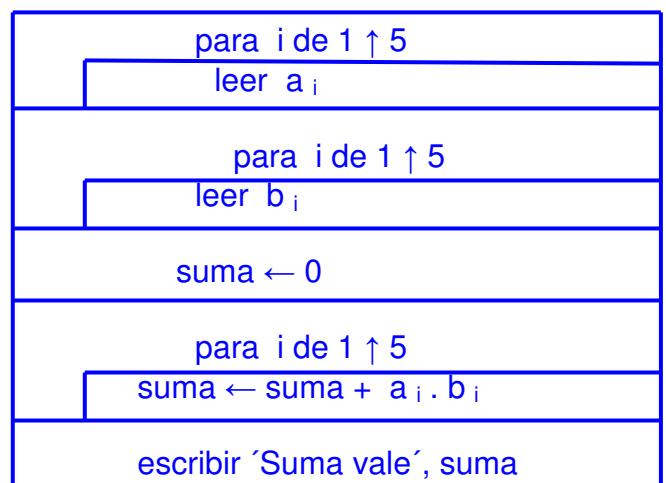
hemos adoptado esa forma de visualizarlos, para seguir lo que se realiza en Algebra, pero bien pudimos haberlos dibujado de otra, por ejemplo a **v** horizontal.

Los arreglos de mas de dos dimensiones, resultan algo mas difícil de visualizar, pero eso se va adquiriendo con la práctica.

Problema: Leer dos arreglos unidimensionales **a** y **b** de 5 elementos reales cada uno. Calcular la suma del producto de sus elementos de igual posición e informarlo.

(acá deberíamos aclarar en el enunciado si se lee primero **a** y luego **b**, o si leeremos **a**₁ y **b**₁, luego **a**₂ y **b**₂, etc. Supongamos que leemos primero **a** y luego **b**)

```
program dosarre(input,output);
type v5=array[1..5]of real;
var suma:real; a,b:v5; i:integer;
begin for i:=1 to 5 do read(a[i]);
      for i:=1 to 5 do read(b[i]);
      suma:=0;
      for i:=1 to 5 do
        suma:=suma+a[i]*b[i];
      write('Suma vale',suma)
end.
```



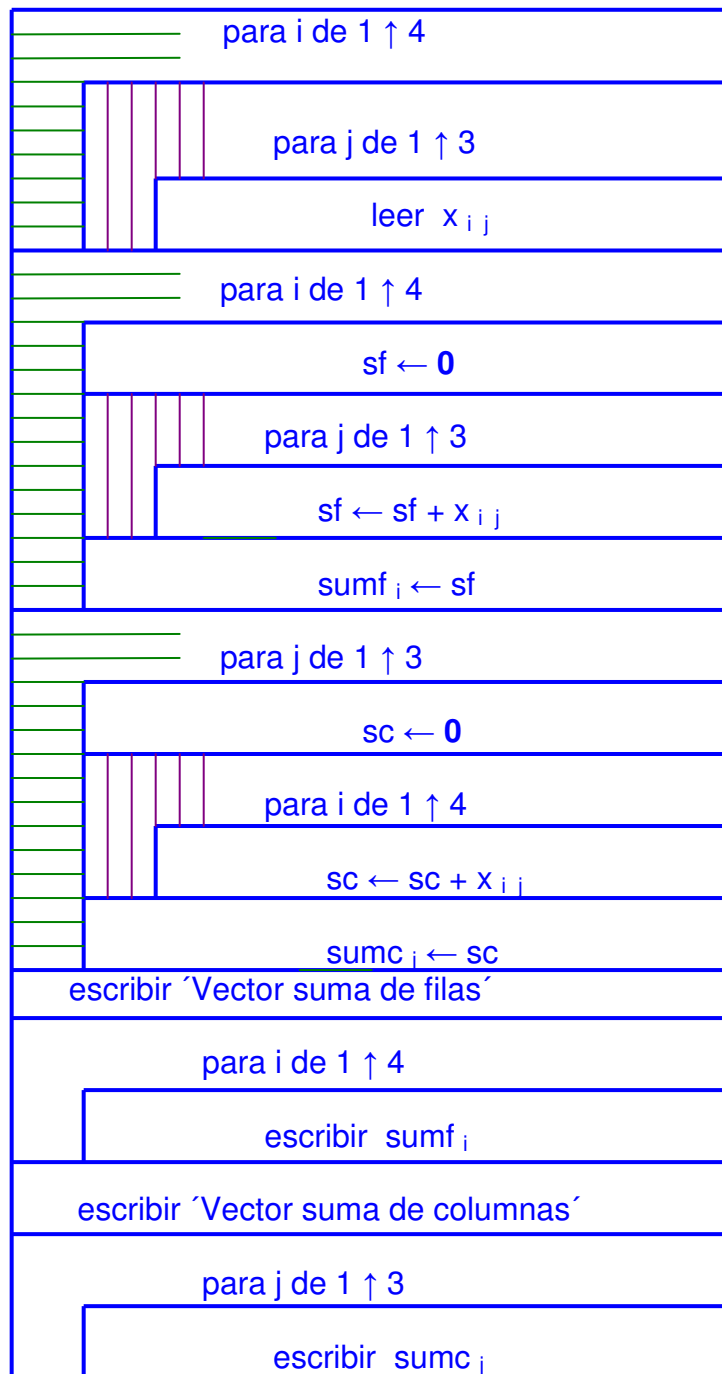
pudimos haber evitado la declaración de tipo **type** simplemente declarando los arreglos en la sección **var**.

```
program dosarre(input,output);
var suma:real; a,b:array[1..5]of real; i:integer;
. . . . .
```


Problema: Leer por filas un arreglo entero bi-dimensional de cuatro filas y tres columnas y armar un arreglo unidimensional con la suma de los elementos de cada fila y otro con la suma de los elementos de cada columna. Mostrar los resultados
(agregamos un ejemplo numérico para ayudar a comprender el enunciado)

$$\mathbf{x} = \begin{bmatrix} -3 & 12 & 0 \\ 9 & 0 & 17 \\ 12 & 9 & -2 \\ -2 & 6 & 9 \end{bmatrix} \quad \mathbf{sumf} = \begin{bmatrix} 9 \\ 26 \\ 19 \\ 13 \end{bmatrix}$$

$$\mathbf{sumc} = [16 \quad 27 \quad 24]$$



```

program sumaele(input,output);
var i, j, sf, sc: integer; sumf: array[1..4] of integer;
    x: array[1..4,1..3] of integer; sumc: array[1..3] of integer;

```

```

begin for i:= 1 to 4 do for j:= 1 to 3 do read(x[i, j]);
    for i:= 1 to 4 do begin sf:=0;
        for j:= 1 to 3 do sf:= sf + x[i, j];
        sumf[i]:= sf
    end;
    for j:= 1 to 3 do begin sc:=0;
        for i:= 1 to 4 do sc:= sc + x[i, j];
        sumc[j]:= sc
    end;
    write('Vector suma de filas');
    for i:= 1 to 4 do write(sumf[i]);
    write('Vector suma de columnas');
    for j:= 1 to 3 do write(sumc[j])
end.

```

Los índices de los arreglos son tipo subrango. Una matriz puede tener un índice subrango de un tipo, y otro índice subrango de otro tipo.

Problema: Una escuela tiene siete grados (numerados del 1 al 7), de cada uno de los cuales hay cuatro divisiones ('d', 'e', 'f', 'g') y se desea analizar para los grados 3º, 4º, 5º, 6º y 7º las cantidades de alumnos que concurren a cada uno de esos 20 grados. Leer por grado la cantidad de alumnos de cada uno de los veinte grados e informar cual es el grado y división que tiene mas alumnos (se supone que es uno solo).

Supongamos que los datos entran así:

	div. d	div. e	div. f	div. g
grado 3	35	32	29	30
grado 4	31	28	33	36
grado 5	27	39	25	32
grado 6	30	31	33	34
grado 7	29	35	35	28

y la respuesta del algoritmo debería ser

Con 39 alumnos 5e es el mayor

```

program escuela (input,output);
var escu: array[3..7,'d'..'g'] of integer;
    i, gradomax, maxalu: integer; j, divmax: char;
begin for i:=3 to 7 do for j:= 'd' to 'g' do read(escu[i, j]);
    gradomax:=3; divmax:='d'; maxalu:=escu[3,'d'];
    for i:=3 to 7 do for j:= 'd' to 'g' do
        if escu[i, j]>maxalu then begin gradomax:=i;
            divmax:=j;
            maxalu:=escu[i, j]
        end;
    write('Con',maxalu,' alumnos ',gradomax,divmax,' es el mayor')
end.

```

Problema: Un sanatorio desea analizar la cantidad de casos de gripe atendidos determinados días según la temperatura mínima del día.

A tal efecto desea totalizar los casos atendidos en los 3 últimos años (sólo interesa analizar temperaturas mínimas del día entre -5 y 12 grados). (*18 temperaturas mínimas distintas*) Construir un algoritmo para leer y totalizar esos datos y luego informarlos. Finalmente, ingresando una temperatura mínima (entre -5 y 12 grados), informar la cantidad total de casos atendidos en días con esa temperatura mínima.

```

program sanatorio (input,output);
var casos: array[-5..12] of integer; te, i, cantcasos: integer;
begin
  for i:= -5 to 12 do casos[i]:= 0;
  write('Ingrese mínima del día, fuera de rango termina');
  read(te);
  while te>=-5 and te<=12 do
    begin write(' y ahora la cantidad de casos'), readln(cantcasos);
          casos[te]:= casos[te] + cantcasos;
          write('Ingrese mínima del día, fuera de rango termina');
          read(te)
    end;
  writeln;
  writeln('Cantidad de casos totalizados por temperatura mínima');
  writeln('-----');
  for i:=-5 to 12 do writeln( i, casos[i] );
  write('Ingrese una temperatura dentro del rango'); read(te);
  write('La cantidad total de casos para esa mínima es'; casos[te])
end.

```

CLASIFICACION (ORDENAMIENTO), BUSQUEDA E INTERCALACION

Trabajando con arreglos unidimensionales, surgen inmediatamente tres tareas típicas: **clasificación (ordenamiento)**, **búsqueda** e **intercalación**.

Clasificación (Ordenamiento): Dado un arreglo, clasificar (reordenar) sus elementos de modo que queden en forma creciente (o decreciente).

Búsqueda: Dado un arreglo, determinar si un cierto dato se encuentra o no dentro de él.

Intercalación: Dados dos arreglos que contengan elementos del mismo tipo, ambos clasificados con el mismo criterio, no necesariamente ambos con la misma cantidad de elementos. Armar un tercer arreglo que contenga los elementos de ambos y que esté clasificado con el mismo criterio.

Observación: Si bien el resultado sería el pedido, en intercalación no consideraremos correcto armar un arreglo con los elementos de ambos y luego clasificarlo, ya que esto implicaría un número mucho mayor de operaciones.

Clasificación:

Si bien hay capítulos de libros dedicados a este tema, sólo veremos dos métodos: **por intercambio** (burbuja) y **por selección del mínimo**. (ejemplificaremos para clasificar de menor a mayor)

Por intercambio: Consiste en comparar el último elemento del arreglo con el penúltimo, si están en el orden buscado, se los deja así, si no, se los *intercambia*. Luego se compara el penúltimo con el antepenúltimo, si están en el orden buscado, se los deja así, si no, se los *intercambia*. Y así sucesivamente hasta repetir lo mismo, comparando el primer elemento con el segundo. De esta manera, nos aseguramos que en la primer posición quedó el menor o el mas *liviano*, elemento que se comportó como una *burbuja*.

Repetimos lo mismo, siempre comenzando desde el *fondo* del arreglo, pero esta vez llegamos hasta la segunda posición del arreglo. Estamos seguros que los dos mas pequeños quedaron en las dos primeras posiciones y en orden.

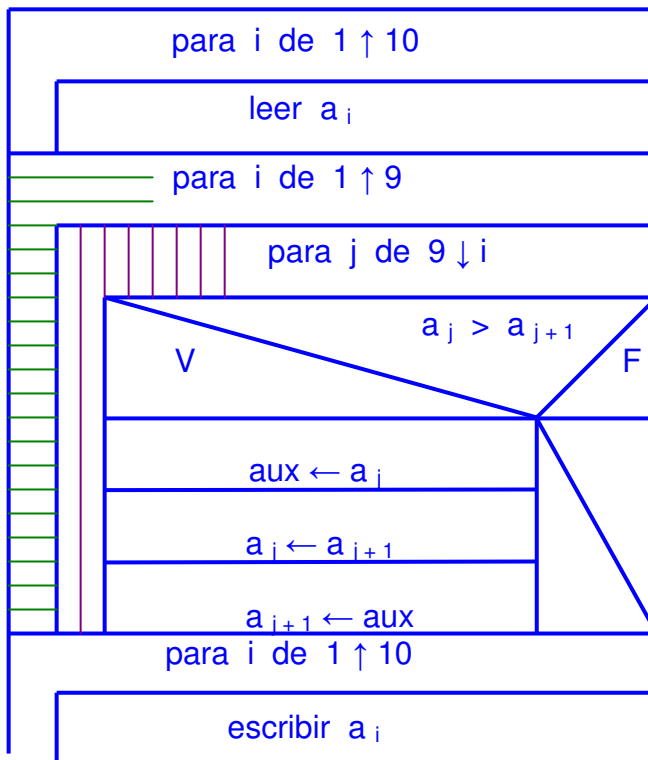
Repetimos siempre lo mismo hasta la penúltima posición con lo que garantizamos que hasta la penúltima posición quedaron en orden, y por lo tanto la última también.

Veamos un ejemplo numérico. Tratemos de clasificar el arreglo (8 2 0 5 1 4)

8	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0
2	2	2	2	0	8	8	8	8	1	1	1	1	1	1	1
0	0	0	0	2	2	2	2	1	8	8	8	2	2	2	2
5	5	1	1	1	1	1	1	2	2	2	2	8	8	4	4
1	1	5	5	5	5	4	4	4	4	4	4	4	4	8	5
4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	8

1ra. recorrida 2da. recorrida 3ra. recorrida 4ta. recorrida 5ta. recorrida

Problema: Leer un arreglo de 10 elementos reales, clasificarlo por el método de intercambio. Imprimirlo clasificado.



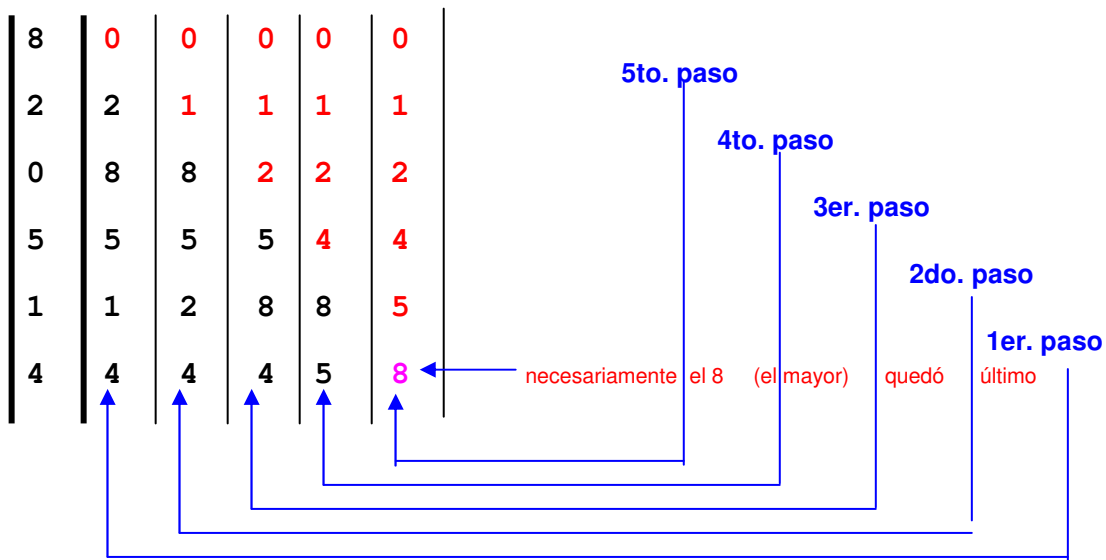
podría incorporarse una verificación tal que si antes de completar el para de mas afuera nos damos cuenta que el arreglo YA quedó clasificado, abandonemos el proceso.

y en Pascal

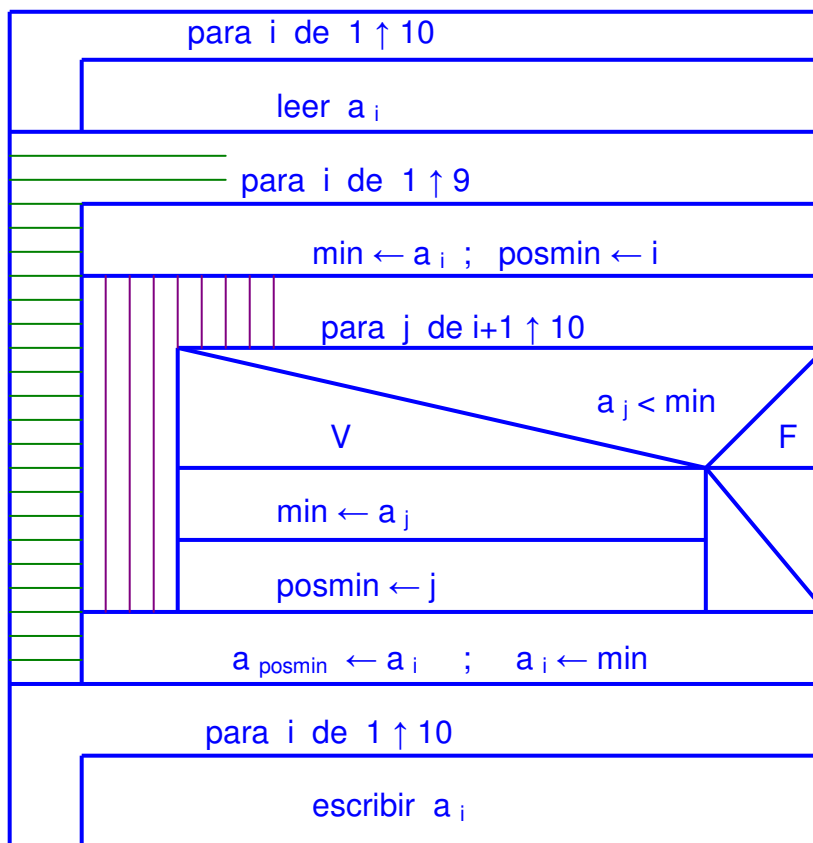
```
program burbuja (input,output);
var i,j:integer; aux:real; a:array[1..10]of real;
begin for i:= 1 to 10 do read(a[i]);
      for i:= 1 to 9 do for j:= 9 downto i do
        if a[j] > a[j+1] then begin aux:= a[j];
                                a[j]:=a[j+1];
                                a[j+1]:=aux
                                end;
      for i:=1 to 10 do write(a[i])
end.
```

Por selección del mínimo: En un primer paso, se busca el mínimo de todos los elementos del arreglo (memorizando valor y posición) y se lo intercambia con el que está en la primera posición. Luego se vuelve a buscar el mínimo, pero buscando desde la segunda posición en adelante y se lo intercambia con el segundo. Así sucesivamente hasta llegar al penúltimo elemento.

Ejemplifiquemos sobre el mismo arreglo anterior



Problema: Leer un arreglo de 10 elementos reales, clasificarlo por el método de selección del mínimo. Imprimirlo clasificado.



```

program porelmin(input,output);
var a : array[1..10] of real; i, j, posmin : integer; min : real;
begin for i:= 1 to 10 do read(a[i]);
      for i:= 1 to 9 do
        begin min:= a[i]; posmin:= i;

```

```

for j:= i+1 to 10 do if a[j]<min then begin min:= a[j];
                                posmin:= j
                                end;
a[posmin]:= a[i];
a[i]:= min
end;
for i:= 1 to 10 do write([i])
end.

```

Búsqueda:

Consiste en lo siguiente, dado un arreglo averiguar si dentro del arreglo está, o no un determinado elemento (que podemos llamar **k**).

Existen muchos métodos de búsqueda, cada uno con diferentes variaciones. Veremos sólo dos, **búsqueda secuencial** y **búsqueda binaria** (o dicotómica). El uso de uno u otro método, depende de que el arreglo esté clasificado o no.

La búsqueda binaria o dicotómica sirve si el arreglo está clasificado. Si no tenemos garantía de que el arreglo esté clasificado, no hay que usarla.

La búsqueda secuencial sirve en ambos casos, pero en arreglos clasificados, de muchos elementos, si bien funciona, es ineficiente y no deberíamos usarla.

Podemos hacer el siguiente cuadro.

	Arreglo NO clasificado	Arreglo clasificado
Secuencial	USAR	NO USAR (ineficiente)
Binaria	NO USAR (en gral. falla)	USAR

También en cada uno de los métodos podemos distinguir dos posibilidades. Que, en caso de estar el elemento buscado tenga la posibilidad de estar mas de una vez y debamos decidir si nos conformamos con encontrar una ocurrencia, o queramos encontrar todas.

Vale la pena hacer notar que en un arreglo clasificado, si puede haber elementos repetidos, los repetidos deben estar todos en posiciones adyacentes.

Búsqueda secuencial: Consiste en recorrer uno a uno los elementos del arreglo (no clasificado) y compararlos con **k** hasta encontrarlo o no

Para ejemplificar, supongamos que tenemos el arreglo **a** de 7 elementos enteros y deseamos saber si un cierto entero **k** está o no dentro del arreglo, y si está, en que posición está su primer ocurrencia. Supongamos

a = $\begin{bmatrix} 16 \\ -39 \\ 0 \\ 25 \\ 5 \\ 25 \\ 17 \end{bmatrix}$

y supongamos que **k** sea -4
la respuesta del algoritmo debería ser **no está**

si suponemos que **k** es 25
la respuesta del algoritmo debería ser **está en la posición 4**

Si nos interesa conocer todas las ocurrencias.

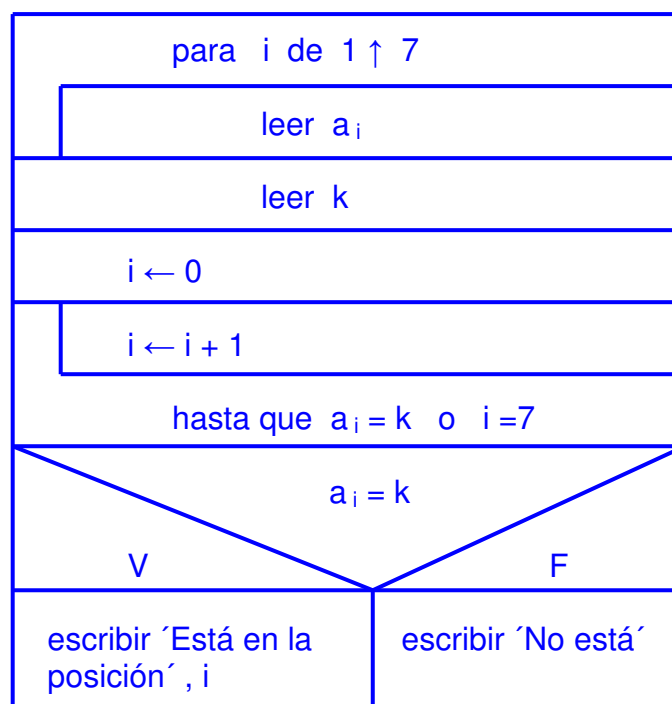
$$a = \begin{bmatrix} 25 \\ -39 \\ 22 \\ 25 \\ 5 \\ 22 \\ 17 \end{bmatrix}$$
 y supongamos que k sea -4
 la respuesta del algoritmo debería ser **no está**

si suponemos que k es 22
 la respuesta del algoritmo debería ser **está en la posición 3**
está en la posición 6

Veamos los algoritmos

Para el caso que interese una sola ocurrencia

Problema: Leer un arreglo de 7 elementos enteros. Leer luego un entero. Informar si este entero está o no. Si está, informar la posición de su primera ocurrencia.



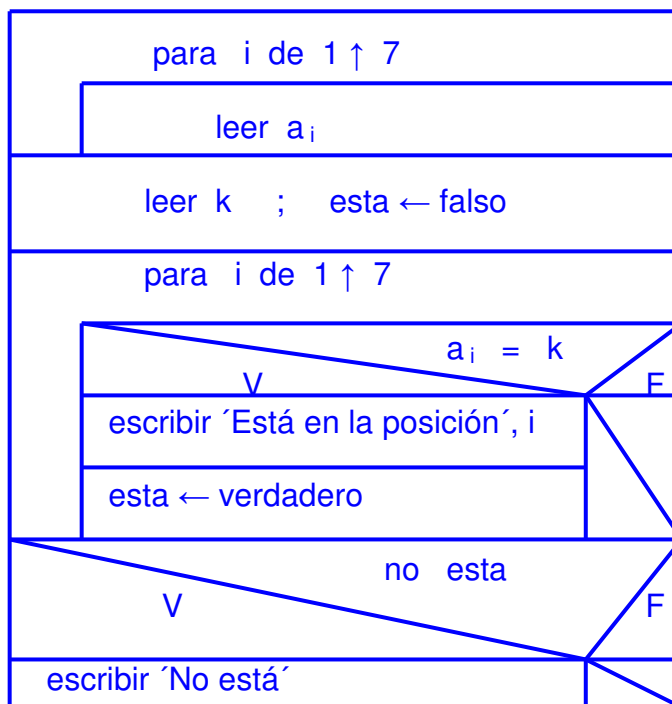
y el programa en Pascal

```

program secuel(input,output);
var a: array[1..7] of integer; i, k: integer;
begin for i:= 1 to 7 do read(a[i]); read(k);
      i:=0;
      repeat i:=i+1 until a[i]=k or i=7 ;
      if a[i]=k then write('Está en la posición',i)
        else write('No está')
end.
  
```


Para el caso que interesen todas las ocurrencias

Problema: Leer un arreglo de 7 elementos enteros. Leer luego un entero. Informar si ese entero está o no. Si está, informar la posición de todas sus ocurrencias.



```

program secue2(input,output);
var a: array[1..7] of integer; i, k: integer; esta: boolean;
begin for i:= 1 to 7 do read(a[i]); read(k); esta:= false;
      for i:= 1 to 7 do if a[i]=k then
                           begin write('Está en la posición',i);
                                esta:=true
                           end;
      if not esta then write('No está')
end.
  
```

Búsqueda dicotómica: Consiste en tener dos indicadores, uno de la posición de menor orden (**min**) y otro de la posición de mayor orden (**max**). Al principio uno valdrá **1** y el otro **N**, suponiendo que la cantidad de elementos sea **N**. (N=11 en los dos ejemplos que siguen).

Probamos si el elemento que está en el “medio” del arreglo es igual al buscado. Si es el buscado, lo hemos encontrado y finaliza el algoritmo (*suponemos que nos conformamos con una sola ocurrencia*). Si no es el buscado, nos fijamos si es mayor o menor al buscado, y como el arreglo está clasificado, si el elemento del “medio” es menor, si el buscado está, está en la parte de “abajo”, y nos quedamos con esa parte bajando **min** hasta **medio**, pero como ya sabíamos que el del “medio” no era, lo bajamos una posición mas, es decir hacemos

min ← **medio** + 1

Análogo razonamiento si el elemento del “medio” es mayor.

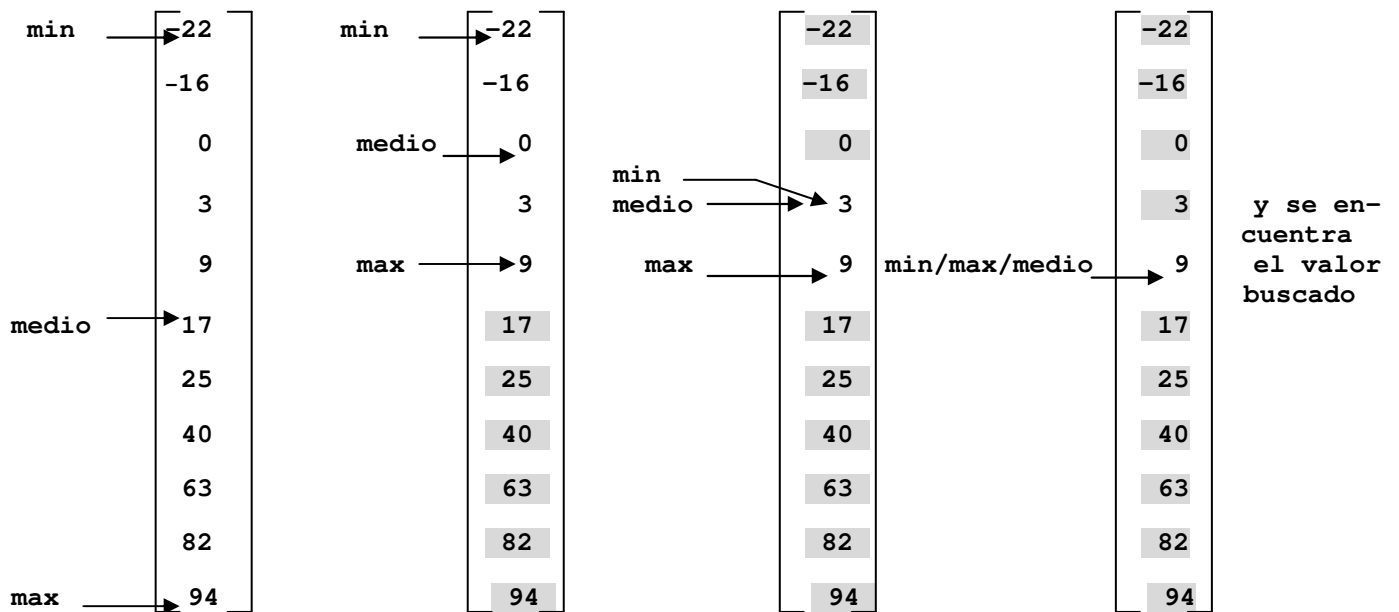
En nuestro algoritmo si **max** y **min** son de distinta paridad, su promedio daría “coma cinco” lo que está prohibido como índice, entonces tomamos la división entera, es decir hacemos

$$\text{medio} \leftarrow (\text{min} + \text{max}) \text{ div } 2$$

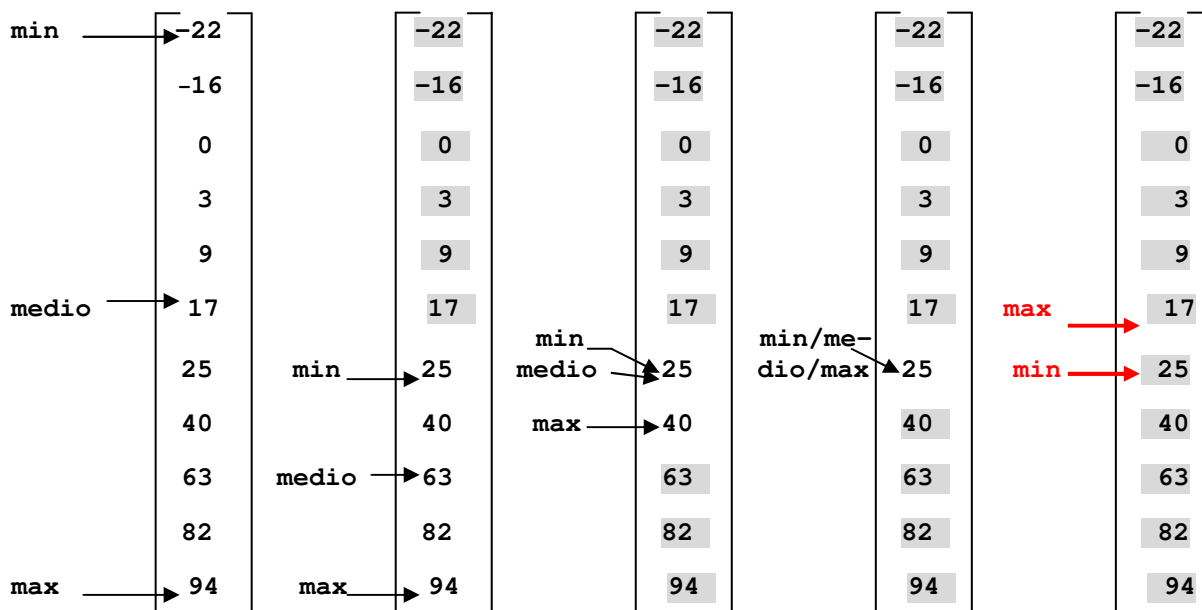
donde con **div** simbolizamos la división entera

Vamos a ejemplificar para el caso de búsqueda exitosa (*el elemento buscado se encuentra en el arreglo*). Supongamos que el elemento que busco es **9** ($k = 9$)

Los elementos sombreados es porque están descartados



Vamos a ejemplificar para el caso de búsqueda fallida (*el elemento buscado NO se encuentra en el arreglo*). Supongamos que el elemento que busco es **23** ($k = 23$)

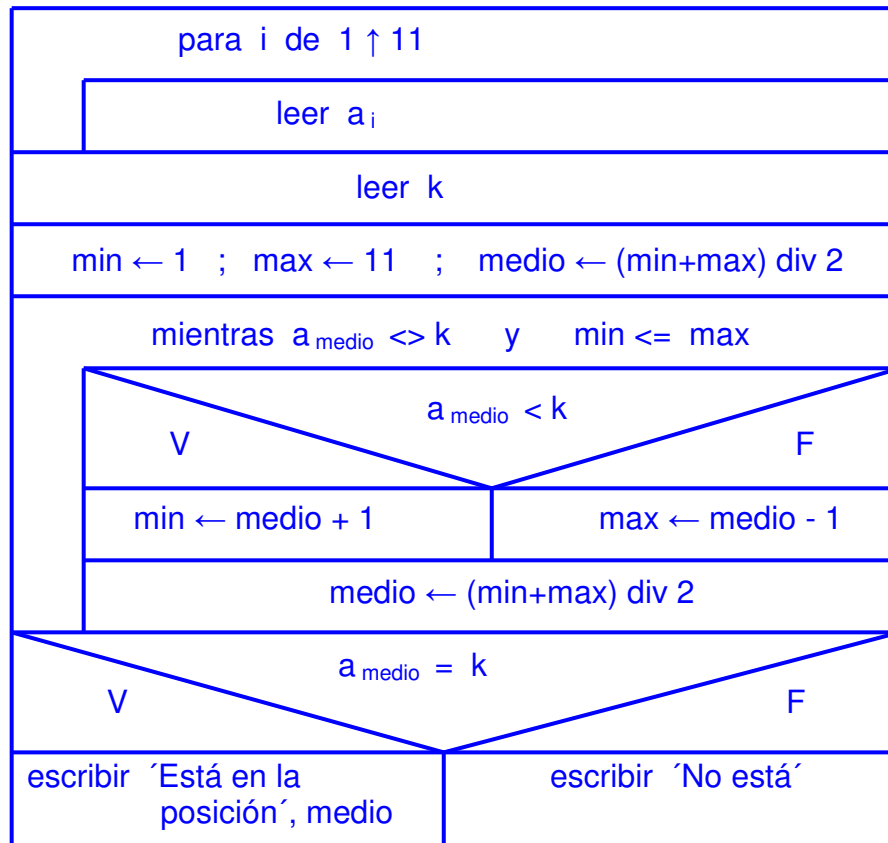


y al quedar el máximo por encima del mínimo ($\text{max} < \text{min}$), se deduce que el elemento buscado no está

No vamos a tratar el caso de búsqueda de todas las ocurrencias, simplemente damos una idea.

Si un arreglo clasificado tiene elementos repetidos, estarán agrupados en posiciones adyacentes. Aplicando el algoritmo de búsqueda dicotómica, que es mucho mas rápido que el secuencial, encontraremos uno, pero no sabemos si es el “primero”, el “último”, o uno cualquiera del grupo, hacemos lo siguiente: vamos “hacia arriba” hasta llegar al primer elemento del grupo igual a la clave y luego “bajamos” listando las posiciones mientras los elementos sean iguales al buscado.

Problema: Leer un arreglo de 11 elementos enteros, se sabe que está clasificado en forma creciente. Leer luego un entero. Informar si ese entero está o no. Si está, informar su posición. Interesa ubicar una sola ocurrencia.



```

program dicoto(input,output);
var min, max, k, i, medio: integer; a: array[1..11] of integer;
begin for i:= 1 to 11 do read(a[i]); read(k);
      min:=1; max:=11; medio:= (min+max) div 2;
      while a[medio]<>k and min<=max do
        begin if a[medio]<k then min:= medio+1
              else max:= medio-1;
              medio:= (min+max) div 2
        end;
      if a[medio]=k then write('Está en la posición',medio)
        else write('No está')
end.

```

Intercalación:

Tenemos dos arreglos, ambos del mismo tipo de elementos. Ambos clasificados con el mismo criterio (ambos en forma creciente o ambos en forma decreciente).

No necesariamente ambos con la misma cantidad de elementos. Llamemos **a** y **b** a ambos arreglos.

Sea **N** la cantidad de elementos de **a**, y sea **M** la cantidad de elementos de **b**. Queremos armar un nuevo arreglo **c** que contenga todos los elementos de **a** y todos los elementos de **b** (es decir que tenga **N+M** elementos) y que esté también clasificado con el mismo criterio.

Veamos un ejemplo

a ($N = 3$)

$$\begin{bmatrix} 9 \\ 35 \\ 47 \end{bmatrix}$$

b ($M = 7$)

$$\begin{bmatrix} 7 \\ 20 \\ 25 \\ 43 \\ 50 \\ 73 \\ 92 \end{bmatrix}$$

c ($7 + 3 = 10$ elementos)

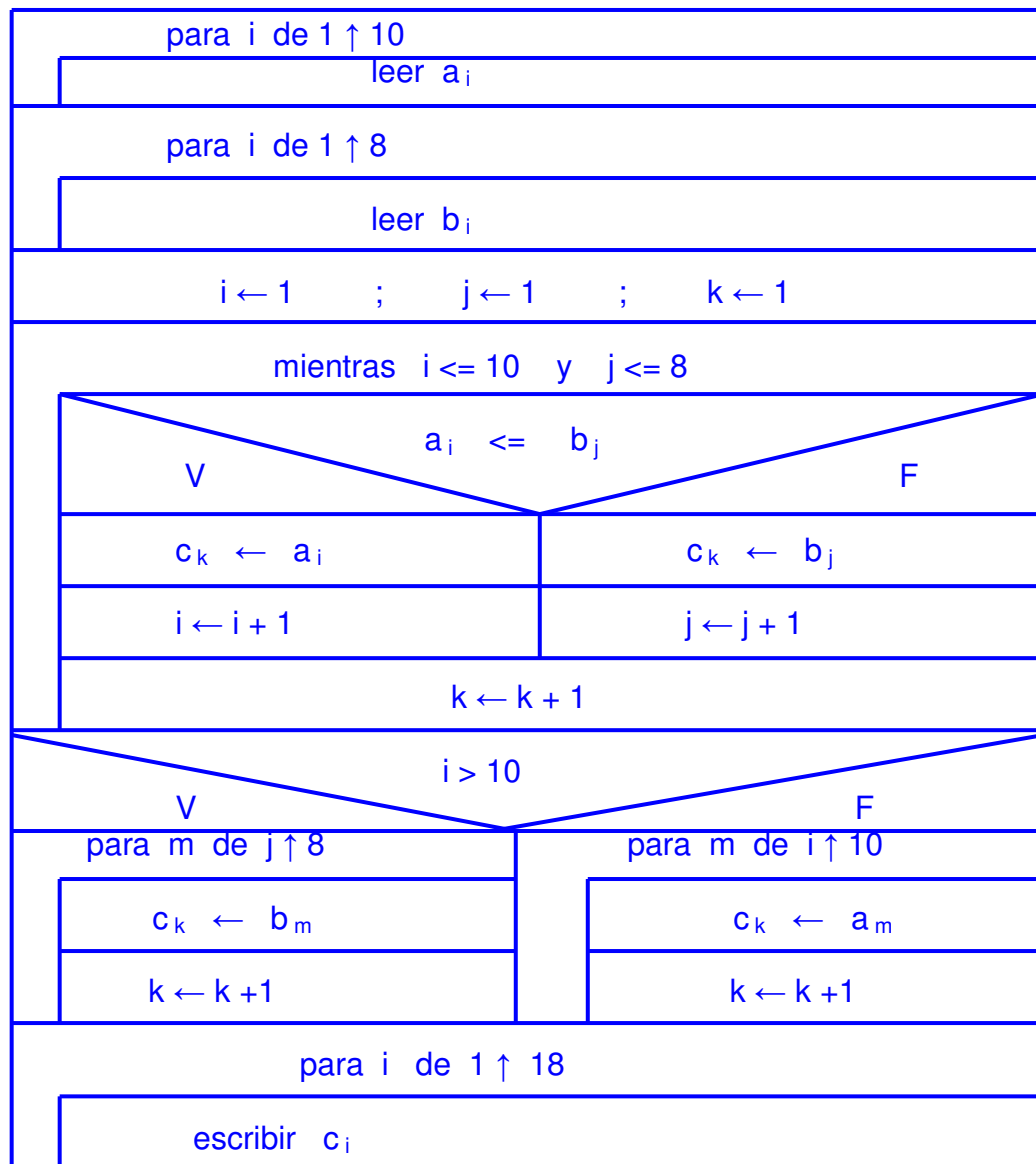
$$\begin{bmatrix} 7 \\ 9 \\ 20 \\ 25 \\ 35 \\ 43 \\ 47 \\ 50 \\ 73 \\ 92 \end{bmatrix}$$

(sería muy ineficiente armar un nuevo arreglo que contenga al principio los elementos de **a** y al final los de **b** y luego clasificarlo).

Sea **i** el índice con que recorremos **a**; **j** el índice con que recorremos **b** y **k** el de **c**.

La idea es recorrer desde el principio los arreglos **a** y **b**, comenzamos con los índices **i**, **j**, **k** en 1 comparamos **a_i** con **b_j** y al elemento menor lo copiamos en **c_k**, incrementamos en 1 a **k** y al índice del arreglo del que hayamos copiado, así hasta agotar uno de los dos arreglos, luego ponemos los elementos que quedaban sin colocar del otro, "al final" de **c**.

Problema: Leer un arreglo de 10 elementos reales (se sabe que ingresa clasificado en forma creciente), leer luego un arreglo de 8 elementos reales (se sabe que ingresa clasificado en forma creciente). Formar un arreglo que contenga todos los elementos del primero y todos los elementos del segundo y que también esté clasificado en forma creciente. Escribirlo



```

program intercala (input, output);
var a: array[1..10] of real; b: array[1..8] of real;
    c: array[1..18] of real; i, j, k, m: integer;
begin for i:=1 to 10 do read(a[i]); for i:=1 to 8 do read(b[i]);
      i:= 1; j:= 1; k:= 1;
      while i<=10 and j<=8 do begin if a[i]<=b[j]then begin c[k]:=a[i];
                                                                    i:=i+1
                                                                    end
                                                                    else begin c[k]:=b[j];
                                                                    j:=j+1
                                                                    end;
                                                                    k:=k+1
                                                                    end;
      if i>10 then for m:=j to 8 do begin c[k]:=b[m]; k:=k+1
                                                                    end
                                                                    else for m:=i to 10 do begin c[k]:=a[m]; k:=k+1
                                                                    end;
      for i:= 1 to 18 do write(c[i])
end.

```

Eficiencia del algoritmo:

¿Por que decimos que el algoritmo que acabamos de presentar es mas eficiente que copiar **a** en **c** y a partir de ahí copiar **b** en **c**, para luego clasificar **c**?

En nuestro algoritmo no tenemos ningún anidamiento de estructuras de repetición. En cambio en los métodos de clasificación tenemos anidadas dos estructuras de repetición.

Supongamos que tenemos dos **para** anidados. El de mas afuera que se repita 200 veces y el interior 400 veces

```
para i de 1 a 200
  para j de 1 a 400
    sentencia ← esta sentencia se ejecutará 80.000 veces
  finpara
finpara
```

Si queremos clasificar por ejemplo por intercambio un arreglo de 200 elementos tendremos dos *para* anidados y si bien el *para* interior se ejecuta cada vez una vez menos, en promedio, las sentencias interiores se ejecutarán unas 20.000 (200x100 aproximadamente) veces, mientras que con el algoritmo de intercalación no tenemos anidamiento y es mucho mas rápido.

Vamos a hacer un ejemplo para ordenar las filas de una matriz (*sin desarmar las filas*) de manera que queden ordenadas según los elementos de una determinada columna, por ejemplo la primera.

Habíamos dicho que los elementos de un array podían ser de cualquier tipo, por ejemplo otro array. A la matriz (arreglo bidimensional de 5 x 4)

$$m = \begin{bmatrix} 12 & 17 & -6 & 9 \\ -1 & 4 & 6 & 8 \\ 3 & 14 & 100 & 40 \\ -9 & 46 & 77 & 11 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

la podemos también tratar como un arreglo de 5 elementos donde cada elemento es un arreglo de 4 elementos

$$m = \begin{bmatrix} (12 & 17 & -6 & 9) \\ (-1 & 4 & 6 & 8) \\ (3 & 14 & 100 & 40) \\ (-9 & 46 & 77 & 11) \\ (5 & 6 & 7 & 8) \end{bmatrix}$$

con $m[2]$ hago referencia a toda la fila

y este elemento puede ser referenciado como $m[4][3]$

Esto, en ciertos problemas puede facilitar el tratamiento.

Supongamos que queremos reordenar las filas de esa matriz de manera que los elementos de una columna (supongamos la primera) queden en orden creciente.

Aclaremos que la clasificación la efectuamos por el método de intercambio (o burbuja) que ya habíamos visto.

```

program VecDeVec(input,output);
type v4=array[1..4] of integer;
var i, j: integer; m:array[1..5] of v4; aux: v4; rta:char;
begin writeln('Ingrese la matriz por filas');
  for i:=1 to 5 do
    for j:=1 to 4 do read(m[i][j]);
  for i:=1 to 4 do
    for j:=4 downto i do if m[j][1]>m[j+1][1] then
      en estas tres sentencias          begin aux:= m[j];
      intercambio toda la fila  j          m[j]:=m[j+1];
      con toda la fila  j+1              m[j+1]:=aux
                                          end;
  writeln('La misma, ordenada por los elementos de la 1a. fila');
  for i:=1 to 5 do begin for j:=1 to 4 do write(m[i][j]:5);
                        writeln
                      end;
  write('Pulse...'); read(rta)
end.

```

se puede probar este programa y si se ingresa

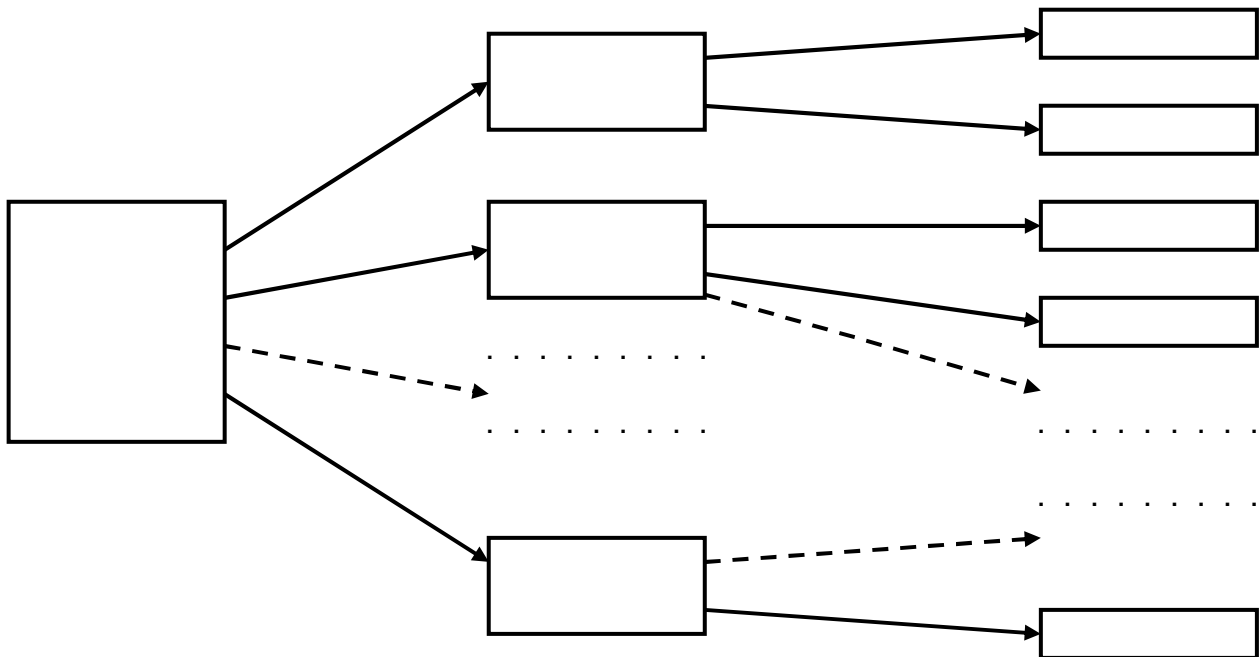
12	17	-6	9
-1	4	6	8
3	14	100	40
-9	46	77	11
5	6	7	8

la salida será

-9	46	77	11
-1	4	6	8
3	14	100	40
5	6	7	8
12	17	-6	9

SUBALGORITMOS:

Cuando tenemos que diseñar un algoritmo de relativa importancia en cuanto a su complejidad, es conveniente dividirlo en partes, o módulos, o subalgoritmos. Si a su vez, a algunos de esos módulos los volvemos a dividir para reducir su complejidad, tenemos un esquema como el siguiente



Hemos hecho lo que llamamos **refinamientos sucesivos** o también análisis **top-down**. (existe otra forma de componer algoritmos y es la *bottom-up* pero no la vamos a utilizar en este curso).

Los subalgoritmos (o subprogramas) son los que permiten aplicar esta idea de los refinamientos sucesivos.

Los subalgoritmos tienen otra importante utilidad y es que pueden desarrollarse con parámetros genéricos y luego utilizarse varias veces con diferentes juegos de datos. Los parámetros de los subalgoritmos pueden ser de cualquier tipo.

Presentaremos dos tipos de subalgoritmos: las **funciones** y los **procedimientos**.

Funciones:

Tienen un uso similar a las funciones internas, pero acá, los cálculos los define el programador. Tienen todas las características de los programas, con la diferencia que no pueden ejecutarse “sueltas”, sino que se ejecutan cuando son utilizadas (o llamadas) por el programa principal u otro subprograma. Esta utilización puede ser a la derecha en una sentencia de asignación, en una condición (de un if, un while o un repeat), en un for o incluso en una expresión en un write.

Las funciones (como las funciones internas) devuelven un único valor a través de su nombre, no devuelven resultados a través de sus parámetros.

(al final del capítulo haremos un comentario sobre el verdadero funcionamiento de las funciones)

Veamos un ejemplo:

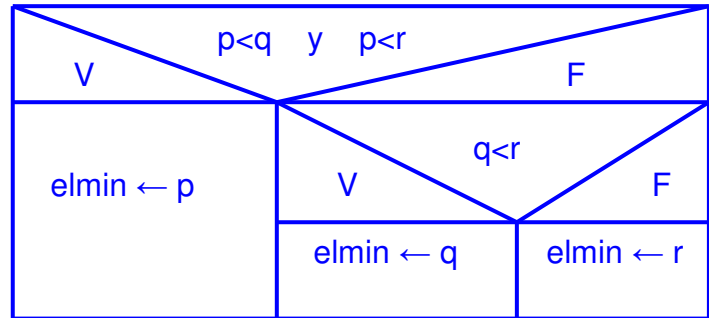
Problema: Leer 3 números enteros e informar el valor del mínimo

programa principal

leer (a , b , c)
menor ← elmin (a,b,c) (*)
escribir ('El menor vale' , menor)

podimos haber eliminado la variable **menor** y también la asignación (*) y en la escritura poner **escribir ('El menor vale' , elmin(a,b,c))**

funcion elmin (p , q , r : enteros) : entera



```

program funcio(input,output);
var a , b , c , menor : integer ;
function elmin( p , q , r : integer ) : integer ;
    begin if p<q and p<r then elmin:= p
           else if q<r then elmin:= q
                else elmin:= r
    end;
begin read ( a , b , c ); ← acá empieza a ejecutarse el programa, lo anterior es declaratorio
      menor:= elmin( a , b , c );
      write ( 'El menor vale' , menor)
end.
  
```

Los **parámetros** p, q, r que utilizamos **en la definición** de la función se llaman **parámetros formales** (o virtuales), ya que sólo sirven para definir el tipo de proceso que sufrirá cada uno. Los **parámetros** a, b, c que utilizamos **en la llamada** a la función se llaman **parámetros actuales** (o verdaderos), ya que es con esos valores con los que trabajará la función.

También dijimos que los parámetros podían ser de cualquier tipo, y puede haber cualquier cantidad de parámetros los cuales a su vez pueden ser de diferente tipo. Veamos, por ejemplo una función que calcule el determinante de una matriz cuadrada de 3x3.

```

.....
function sarrus( m : array [1..3,1..3] of real ) : real ;
var suma , resta : real ;
begin
    suma :=m[1,1]*m[2,2]*m[3,3]+m[1,2]*m[2,3]*m[3,1]+m[2,1]*m[3,2]*m[1,3];
    resta:=m[3,1]*m[2,2]*m[1,3]+m[2,1]*m[1,2]*m[3,3]+m[3,2]*m[2,3]*m[1,1];
    sarrus:=suma-resta
end ;
.....
  
```

Las variables definidas en la function se llaman variables locales (a la propia function) y sólo son reconocidas dentro de ella pero desconocidas fuera.

Vamos a hacer otro problema, donde construiremos la función para calcular por Sarrus un determinante de 3x3 pero sus argumentos serán tres vectores y la usaremos para resolver un sistema no homogéneo de 3x3.

Y al sistema

en vez de verlo así

$$\begin{aligned} a_{11} \cdot x_1 + a_{12} \cdot x_2 + a_{13} \cdot x_3 &= b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + a_{23} \cdot x_3 &= b_2 \\ a_{31} \cdot x_1 + a_{32} \cdot x_2 + a_{33} \cdot x_3 &= b_3 \end{aligned}$$

lo pensaremos así

$$\begin{aligned} p_1 \cdot x_1 + q_1 \cdot x_2 + r_1 \cdot x_3 &= b_1 \\ p_2 \cdot x_1 + q_2 \cdot x_2 + r_2 \cdot x_3 &= b_2 \\ p_3 \cdot x_1 + q_3 \cdot x_2 + r_3 \cdot x_3 &= b_3 \end{aligned}$$

Problema: Leer por filas la matriz de los coeficientes de un sistema de 3x3, a coeficientes reales. Leer luego el vector de los términos independientes. Calcular la solución e informarla (supondremos que tiene solución única)

```
program TresPorTres(input,output);
type v3 = array[1..3] of real;
var p, q, r, b, x: v3; i: integer; detcoef: real;
function sarrus2 (f, g, h:v3): real;
var suma , resta: real;
begin suma := f(1)*g(2)*h(3) + f(2)*g(3)*h(1) + f(3)*g(1)*h(2);
      resta:= f(3)*g(2)*h(1) + f(2)*g(1)*h(3) + f(1)*g(3)*h(2);
      sarrus2 = suma - resta
end;
begin
  for i:=1 to 3 do read (p[i], q[i], r[i]);
  for i:=1 to 3 do read (b[i]);
  detcoef:= sarrus2(p,q,r);
  x[1]:= sarrus2(b,q,r) / detcoef;
  x[2]:= sarrus2(p,b,r) / detcoef;
  x[3]:= sarrus2(p,q,b) / detcoef;
  write ('Las raíces son ');
  for i:= 1 to 3 do write(x[i])
end.
```

en la función pienso
el determinante así:

$$\begin{vmatrix} f_1 & g_1 & h_1 \\ f_2 & g_2 & h_2 \\ f_3 & g_3 & h_3 \end{vmatrix}$$

← acá empieza a ejecutarse el programa, lo anterior es declaratorio

Las funciones pueden ser recursivas, es decir que pueden invocarse a si mismas. Un ejemplo clásico es la función **factorial**. (podemos pensar $n! = n \times (n-1)!$) El siguiente ejemplo muestra por pantalla el factorial de enteros entre 1 y 10

```
program factori(input,output);
var i:integer; z:char;
function facto(n:integer):integer;
begin if n<=0 then facto:=1
      else facto:=n*facto(n-1)
end;
begin for i:=1 to 10 do writeln(facto(i):10);
      read(z)
end.
```

El resultado de la ejecución de este programa es

```
1
2
6
24
120
720
¿ que pasó con los tres últimos números ?
el factorial crece muy rápidamente
el factorial de 8 daría 40.320 que excede
el valor permitido para los enteros
```

5040
 -25216
 -30336
 24330

la capacidad "rebalsó" y dio un valor inesperado
 la solución hubiese sido poner
 function facto(n:integer):longint;

Antes de terminar con el tema reiteramos un concepto importante: **Las funciones devuelven un solo dato a través de su nombre.** (hay una aclaración al final del capítulo)

Esto para diferenciarlas del siguiente tema:

Procedimientos:

Los procedimientos tienen un funcionamiento parecido a las funciones, pero no devuelven nada a través de su nombre. Pueden devolver valores (ninguno, uno o varios) a través de sus parámetros. Por lo tanto habrá parámetros **de entrada**, **de salida**, o incluso **de entrada/salida** al procedimiento.

A los parámetros que sólo sirven para entrar datos desde el programa (*sería mas correcto decir "desde el módulo invocante"*) al procedimiento (*o sea los de entrada*) los llamaremos **parámetros por valor**, a los parámetros que sirven para devolver datos desde el procedimiento al programa (*o sea los de salida, o los de entrada/salida*) los llamaremos **parámetros por referencia**.

Veamos su forma de trabajar:

Los parámetros por valor, son copiados dentro del procedimiento y el procedimiento trabaja sobre esa copia, por lo tanto no se devuelve nada a través de ellos.

Cuando los parámetros son por referencia, no son copiados al procedimiento, sino que el procedimiento trabaja sobre el parámetro actual (es decir sobre las variables del programa) pudiendo por lo tanto salir datos del procedimiento hacia fuera.

En este punto, debemos hacer una aclaración. En muchos lenguajes, como por ejemplo Pascal, el tipo de parámetro (por valor o por referencia) debe ser declarado en el propio procedimiento. Por lo tanto todas las veces que sea invocado el procedimiento, cada parámetro trabajará de la misma forma, siempre por valor o siempre por referencia.

El Fortran por ejemplo permite no definir en el procedimiento la forma de trabajo de cada parámetro, sino que lo deja para el momento de la llamada. Es decir que dentro de un mismo programa, en Fortran, para un procedimiento en particular podría ocurrir que haya parámetros que en una llamada trabajen por valor y en otra llamada por referencia.

Los parámetros formales, es decir los que aparecen en la declaración del subprograma, deben ser nombres de variables (no pueden ser constantes ni expresiones), y estas variables pueden ser de cualquier tipo (entero, real, arreglo, registro, ...).

Los parámetros actuales, es decir, los que aparecen en la "llamada" al subprograma, pueden ser constantes, variables o expresiones si la llamada es por valor. Si la llamada es por referencia, sólo pueden ser variables. En las funciones, los parámetros formales son todos por valor.

Debe haber correspondencia entre los parámetros actuales y los formales, tanto en cantidad como en el orden y como en el tipo.

Problema: Leer los coeficientes **a** y **b** y resolver la ecuación $a.x + b = 0$ (a distinto de cero)

```

program unaprueba(input,output);
var a , b , x : real ;

```

```

procedure ecua( p , q : real ; var r : real )
begin r := -q / p
end ;
begin read ( a , b );
      ecua ( a , b , x );
      write ( 'La raíz es' , x )
end.

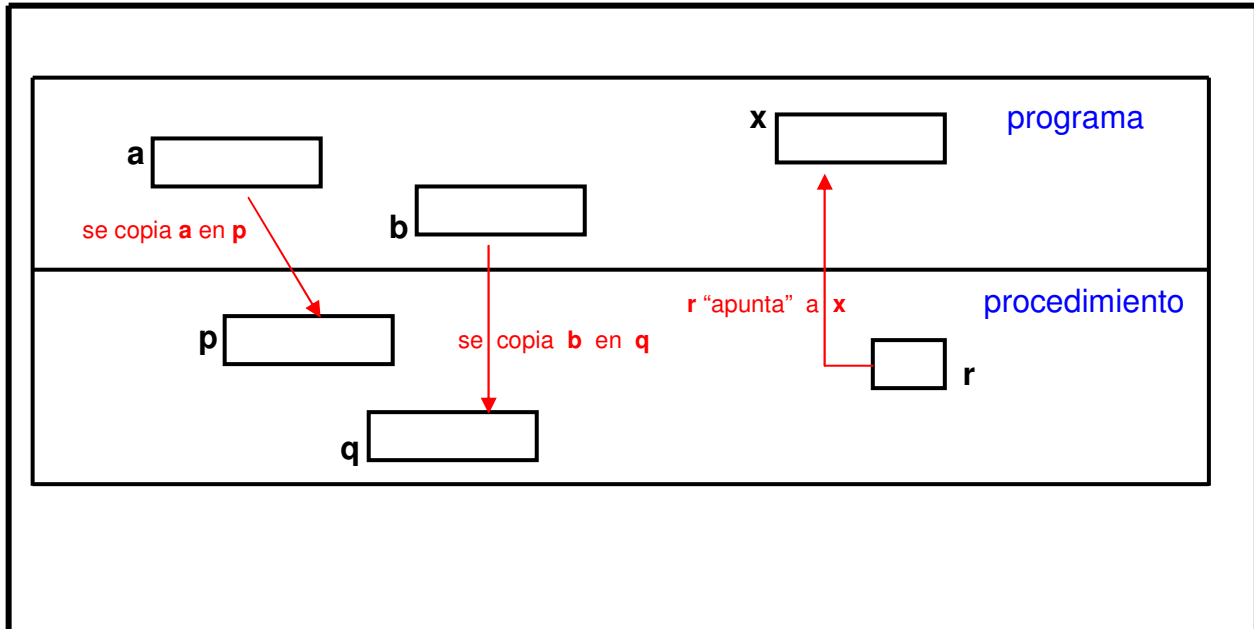
```

el parámetro **r**, al tener antepuesta la palabra **var**, trabajará por referencia, los otros dos, al no tenerla, serán por valor

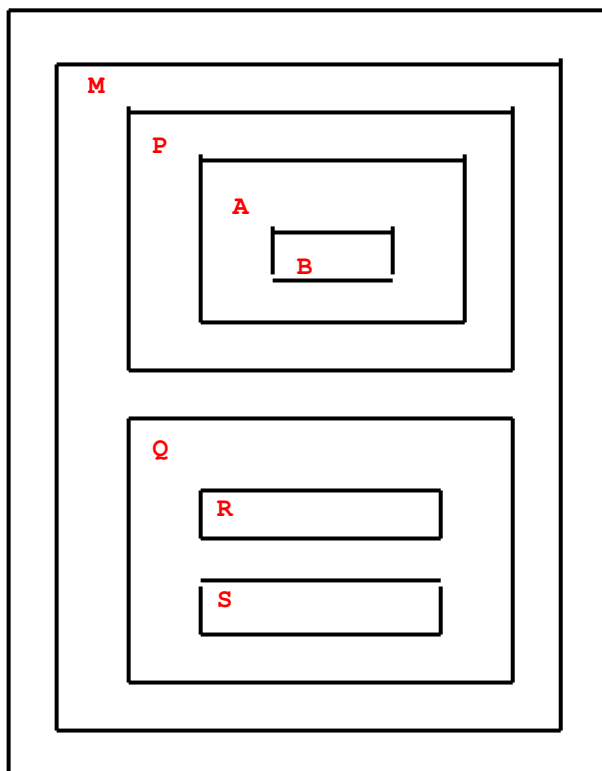
← *acá empieza a ejecutarse el programa, lo anterior es declaratorio*

Veamos ahora como se trabaja en la memoria

memoria



Cualquier función o procedimiento puede llamar a otra función y/o procedimiento, y así sucesivamente, es decir que puede ocurrir una sucesión o cadena de llamadas entre funciones y procedimientos.



los objetos
en el bloque

son accesibles
en los bloques

M
P
A
B
Q
R
S

M , P , A , B , Q , R , S
P , A , B
A , B
B
Q , R , S
R
S

Veamos otro ejemplo para indicar la diferencia entre parámetros por valor y por referencia. En el programa principal, se inicializan dos variables enteras con el mismo valor **5**, ambas se corresponden como parámetros actuales, con parámetros formales que dentro del procedimiento son incrementados ambos en **3**, la única diferencia es que uno de los parámetros es por valor y el otro por referencia.

```
program prueba( output );
var a , b : integer ;
procedure cambia ( x : integer ; var y : integer);
    begin x := x + 3 ;
          y := y + 3 ;
          writeln ( x , y)
    end ;
begin a := 5 ;
      b := 5 ;
      writeln ( a , b );
      cambia ( a , b ) ;
      write ( a , b )
end .
```

Este programa producirá la siguiente salida

5	5
8	8
5	8

No necesariamente los procedimientos deben tener parámetros, mas aún, a veces pueden no devolver nada al programa (o devolver a través de variables globales). Veamos un ejemplo de procedimiento sin parámetros.

Problema: Leer un vector entero de 20 elementos. Buscar el mínimo elemento (suponer que es único), su posición, e informar ambos valores

```
program minimo1( input , output );
var a: array[1..20] of integer ; i: integer;
procedure buscamín1;
    var min , posmin: integer ;           min y posmin son variables locales, sólo se "ven" desde
    begin min := a[1] ; posmin := 1;      el procedimiento (se desconocen fuera del mismo)
      for i:= 2 to 20 do if a[i] < min then begin min := a[i] ;
                                           posmin := i
                                           end ;
      write('Mínimo vale',min,' en posición',posmin)
    end ;
begin for i:= 1 to 20 do read ( a[i] ) ;
      buscamín1
end.
```

Observaciones: El procedimiento trabajó sobre la variable global **a**, también usó la variable global **i**. El procedimiento no necesitó devolver lo calculado, ya que lo informó directamente por pantalla. En este ejemplo el uso de procedimiento simplemente permitió modularizar la solución.

Veamos otro ejemplo donde se hará necesario el uso de parámetros.

Problema: Leer un vector de 20 elementos enteros, buscar el mínimo elemento, su posición e informarlos. Leer luego otro vector de 20 elementos enteros, buscar el mínimo elemento, su

posición e informarlos. Finalmente leer otro vector de 20 elementos enteros, buscar el mínimo elemento, su posición e informarlos. Por último, informar el promedio de los tres mínimos.

```

program minimo2( input , output ) ;
var a: array[1..20] of integer ; min1, min2, min3, i: integer; p: real;
procedure buscamin2(var min: integer);
  var posmin: integer ;
  begin min := a[1] ; posmin := 1;
        for i:= 2 to 20 do if a[i] < min then begin min := a[i] ;
                                                posmin := i
                                                end ;
        write('Minimo vale',min,' en posicion',posmin)
  end ;
begin for i:= 1 to 20 do read ( a[i] ) ; buscamin2(min1);
      for i:= 1 to 20 do read ( a[i] ) ; buscamin2(min2);
      for i:= 1 to 20 do read ( a[i] ) ; buscamin2(min3);
      p := (min1+min2+min3)/3;
      write(' y el promedio de los minimos vale ',p)
end.

```

Veamos otro problema, supongamos que en diferentes partes de un programa, quiero clasificar tres arreglos, de diferente nombre y diferente cantidad de elementos.

Podemos hacer un procedimiento para clasificar un arreglo (por burbuja por ejemplo), pero los parámetros formales y actuales deben corresponderse en tipo, es decir que si un parámetro formal es un arreglo de 30 elementos enteros, el actual DEBE ser de 30 elementos enteros, no puede ser, por ejemplo, ni de 29 ni de 31.

La solución es hacer un procedimiento que tenga como parámetros el arreglo y también la cantidad de elementos del arreglo, y a todos los arreglos darles el tamaño del mayor. Veamos

Problema: Leer un vector a de 20 elementos enteros, clasificarlo en forma creciente e imprimirlo clasificado. Leer luego un vector b de 28 elementos enteros, clasificarlo en forma creciente e imprimirlo clasificado. Finalmente, leer un vector c de 15 elementos enteros, clasificarlo en forma creciente e imprimirlo clasificado.

```

program clasificavarios( input , output ) ;
type v28 = array [ 1 .. 28 ] of integer ;
var a ,b ,c :v28 ; i :integer ;
procedure orden ( n :integer ; var x :v28) ;
  var aux, j :integer ;
  begin for i:= 1 to n-1 do
        for j:= n-1 downto i do
          if x[j] > x[j+1] then begin aux := x[j] ;
                                   x[j] := x[j+1] ;
                                   x[j+1] := aux
          end
        end ;
  end ;
begin for i:= 1 to 20 do read(a[i]) ; orden ( 20 , a ) ;
      for i:= 1 to 20 do writeln(a[i]) ;
      for i:= 1 to 28 do read(b[i]) ; orden ( 28 , b ) ;
      for i:= 1 to 28 do writeln(b[i]) ;
      for i:= 1 to 15 do read(c[i]) ; orden ( 15 , c ) ;
      for i:= 1 to 15 do writeln(c[i])
end.

```

Noten que en la declaración del procedure hemos antepuesto la palabra **var** a **x** para que sea parámetro por referencia y devuelva el arreglo clasificado al programa.

Ahora vamos a hacer un comentario sobre la verdadera forma de trabajo de las funciones:

Si bien quienes escriben este apunte estiman que las funciones deben tener un funcionamiento similar al de las de biblioteca, con la diferencia que son escritas por el programador, en realidad también pueden tener parámetros por referencia, con lo que no sólo devuelven un valor a través de su nombre, sino también valores a través de sus parámetros si son declarados por referencia.

Como ejemplo, veamos el siguiente programa en el que a la variable k le asignamos el valor de la función (15 en el ejemplo) pero a j le asignamos antes del uso de la función el valor 9, pero cuando escribimos por segunda vez j, ya está modificado y vale 10, en cambio i no se modifica.

```
program refenfun(input, output);
var i, j, k: integer;
function refe(x:integer; var y:integer): integer;
var t: integer;
begin t:= 0;
      x:=x+1;
      y:=y+1;
      t:=x+y;
      refe:=t
end;
begin i:=4; j:=9;
      writeln(i:4, j:4);
      k:=refe(i, j);
      writeln(i:4, j:4, k:4);
      readln
end.
```

La salida será

```
4      9
4      10    15
```

Pero no tenemos que perder de vista que el objetivo es, entre otras cosas, aprender algoritmia y no Pascal a fondo.

A lo largo de la profesión se encontrarán con lenguajes que si, deberán conocerlos en profundidad, incluso algunos que no han aparecido todavía. En ese caso, deberán investigar con mayor detalle.

Y para practicar un poco mas con funciones, otro comentario:

Las funciones pueden devolver datos de diferentes tipos, pero no de cualquier tipo. En general pueden devolver datos de tipo integer (en todas sus variantes), real, char, boolean (en todas sus variantes), string y pointer.

Pero también pueden devolver datos de tipo subrango o declarados por el programador.

Veamos el siguiente ejemplo donde le hacemos devolver un tipo **declarado**.

```
program fundecla(input, output);
type juan=(lun, martes, mierc, jueves);
var i: integer; pepe: juan;
function decla(z: integer):juan;
begin case z of
      1: decla:= lun;
      2: decla:= martes;
      3: decla:= mierc;
      4: decla:= jueves
end;
end;
```

```
begin for i:=1 to 4 do begin pepe:= decla(i);  
    case pepe of  
        lun:    write('Hola');  
        martes: write(' que');  
        mierc:  write(' tal');  
        jueves: write(' amigos')  
    end  
end;  
  
    readln  
end.
```

La salida será

Hola que tal amigos

Registros:

Los arreglos permiten agrupar datos, pero todos del mismo tipo. Cuando tenemos necesidad de agrupar datos de diferente tipo (*no está prohibido que sean todos del mismo tipo*) hacemos uso de los **record** (*registros*). Por ejemplo supongamos que deseamos agrupar datos de los alumnos de un curso (*numero, nombre, notas enteras de 4 parciales, promedio*), podemos hacer lo siguiente:

```

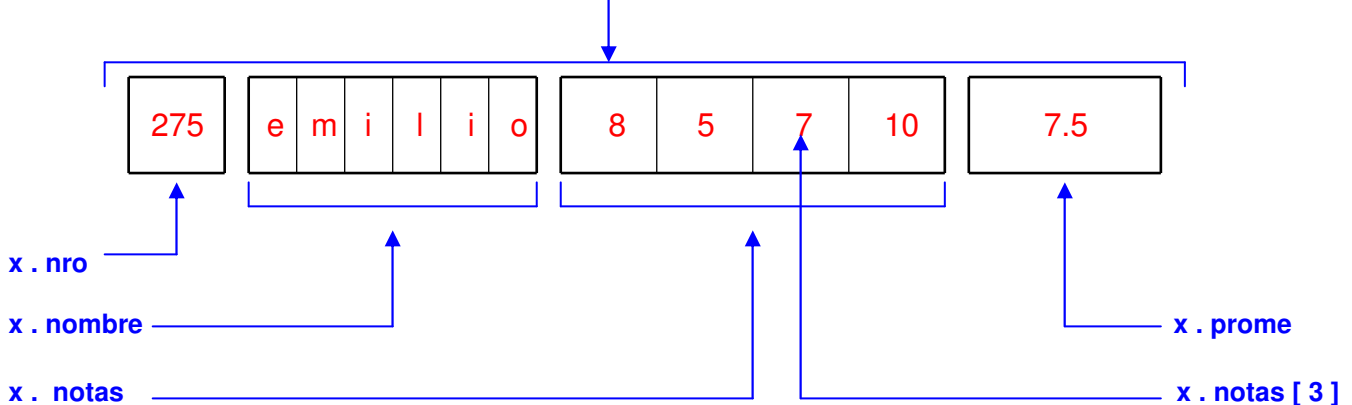
. . . . .
type alu = record nro: integer;
                  nombre: string[6];
                  notas: array[1..4] of integer;
                  prome: real
                end;
. . . . .
var x , y , aux : alu ;
. . . . .

```

numero, nombre, notas enteras de 4 parciales, promedio se denominan **campos** (*field*) del registro y pueden ser de cualquier tipo, incluso **record**. Los campos pueden ser leídos individualmente.

Supongamos en un momento del programa, el registro **x** contiene los siguientes datos

si en alguna parte del programa hago referencia a **x**, estoy refiriendo a todo el registro

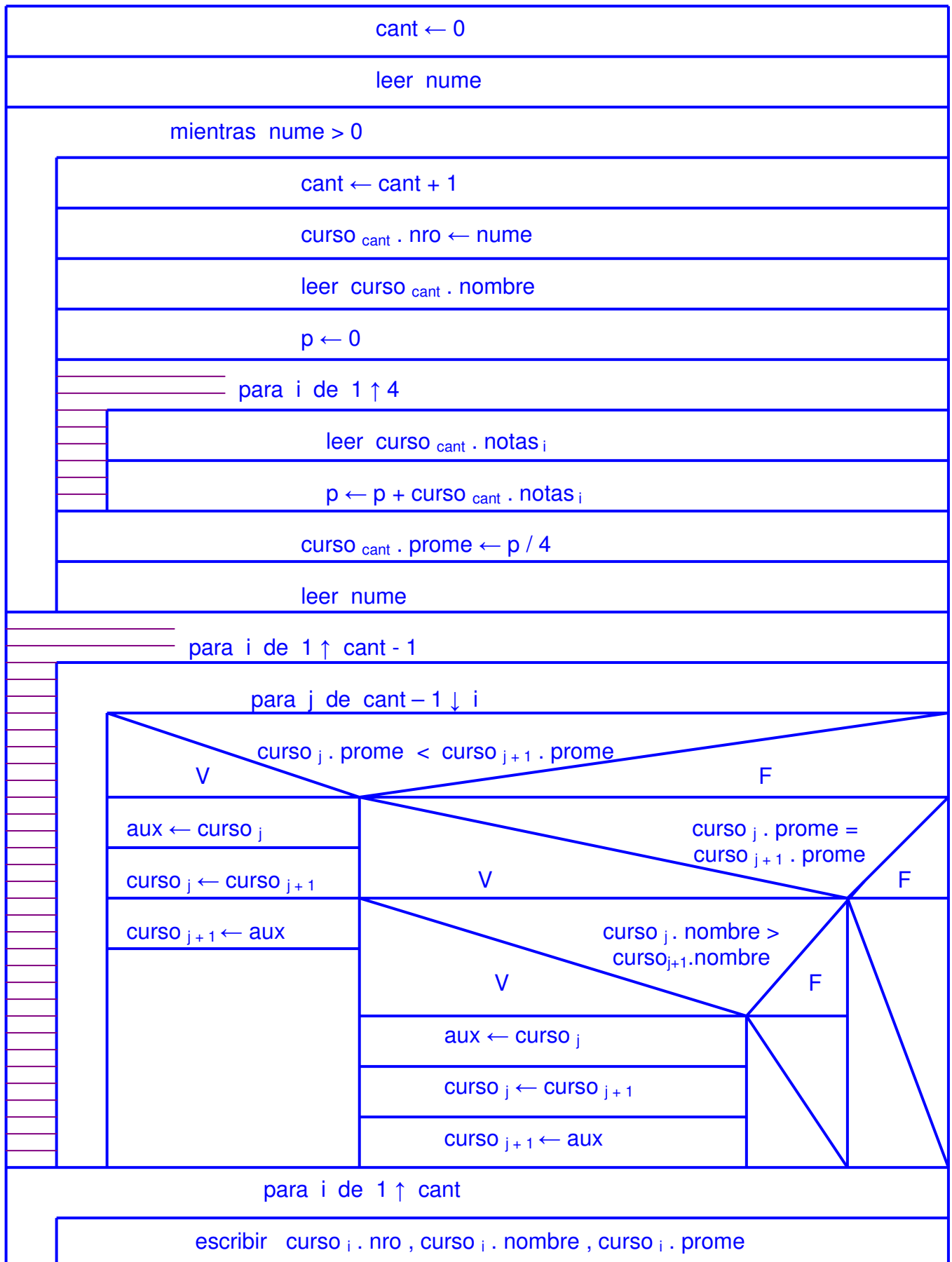


Cualquiera de los datos ejemplificados puede utilizarse, por ejemplo, en una sentencia de asignación

```
aux := x
```

Problema: Leer los datos (*numero, nombre, notas enteras de 4 parciales*) de un cierto número de alumnos (no se sabe cuantos, pero no son mas de 100) de un curso. Efectuar un listado

clasificado por promedio decreciente. En caso de alumnos con igual promedio, deben aparecer en orden alfabético.



```

program uncurso (input,output);
type alu = record nro:integer;
                  nombre:string[6];
                  notas:array[1..4]of integer;
                  prome:real
                end;
var curso:array[1..100] of alu;aux:alu; i,j,cant,nume: integer; p: real;
begin cant:=0; read(nume);
  while nume>0 do
    begin cant:=cant+1; curso[cant].nro:=nume;
      read(curso[cant].nombre); p:=0;
      for i:=1 to 4 do begin read (curso[cant].notas[i]);
        p:= p + curso[cant].notas[i]
      end;
      curso[cant].prome:= p/4; read(nume)
    end;
  for i:= 1 to cant-1 do
    for j:= cant-1 downto i do
      if curso[j].prome < curso[j+1].prome
        then begin aux:= curso[j]; curso[j]:= curso[j+1];
          curso[j+1]:= aux
        end
      else if curso[j].prome = curso[j+1].prome
        then if curso[j].nombre > curso[j+1].nombre
          then begin aux:= curso[j];
            curso[j]:= curso[j+1];
            curso[j+1]:= aux
          end;
    end;
  for i:= 1 to cant do
    writeln(curso[i].nro, curso[i].nombre curso[i].prome)
  end.

```

Aclaremos que ésta no es la manera mas habitual de trabajar en casos como el anterior ya que:

- Los arreglos se trabajan en la memoria principal y luego de terminado el programa, sus datos se pierden
- Los tamaños de los arreglos son fijos, por lo tanto debemos conocer el mismo de antemano, o por lo menos la cantidad máxima de elementos a contener

Para casos similares lo mas práctico es trabajar con archivos.

A partir de la página siguiente entramos al tema Archivos. Se han incorporado muchos temas. No significa que el alumno deba conocer en detalle, por ejemplo, altas, bajas, modificaciones, etc. (temas que con seguridad verán con mas precisión y detalle mas adelante en otras materias). El único objeto de haberlos incluido es para que tengan una buena cantidad de ejercicios de diferente caracter, lo que seguramente les permitirá comprender con mayor detalle el tema de tratamiento de archivos en Pascal.

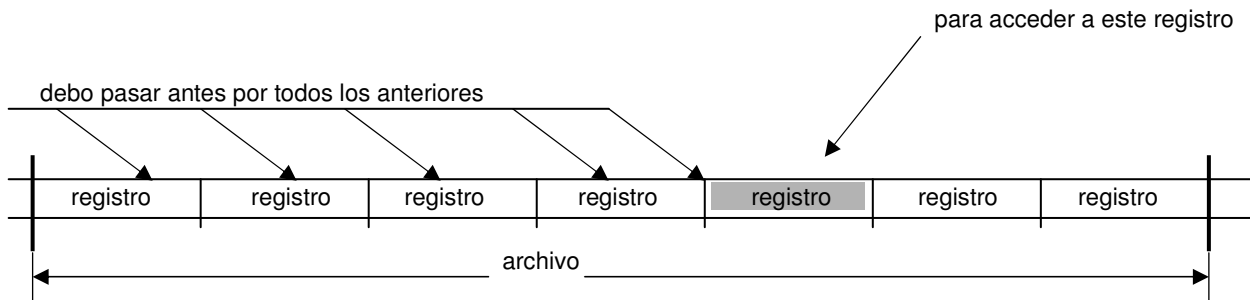
Archivos:

El archivo es una estructura de datos que permite conservarlos después de la finalización del programa ya que no se guardan en la memoria principal, sino en la auxiliar.

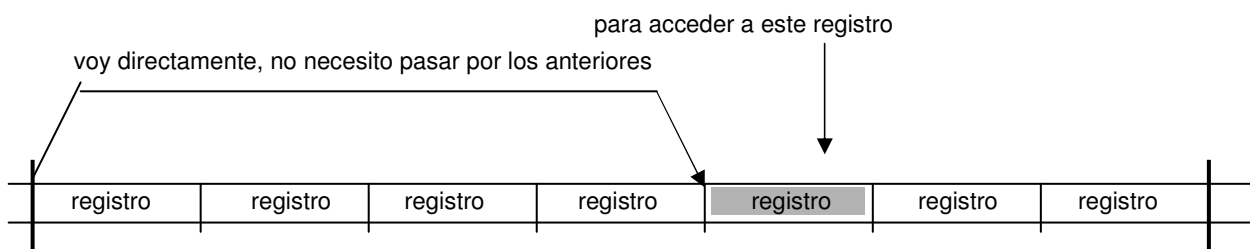
Hay varias formas de armar archivos: archivos de registros, archivos de arreglos, archivos de caracteres, archivos de reales, archivos de enteros, archivos de matrices, etc.

Otra clasificación de archivos es: *secuenciales*, *directos*, *indexados*. (dibujos para archivos de registros)

En los archivos **secuenciales**, para acceder a un registro, se debe pasar por los anteriores, no se puede acceder directamente. Estos archivos pueden implementarse en cualquier tipo de soporte, cintas, discos, etc.



En los archivos **directos** se puede acceder a cualquier registro directamente, sin necesidad de pasar por los anteriores. Evidentemente, para conseguir esto, cada registro debe tener una clave o un campo que le permita al programa, mediante algún cálculo, ir directamente a la posición de memoria donde está almacenado el registro. No se pueden implementar en cintas.



No se describen los **indexados** ya que serán ampliamente tratados en otras asignaturas.

En este apunte de apoyo, veremos principalmente archivos orientando la explicación a como se los trata en Pascal.

Veamos algunas definiciones y sentencias Pascal.

En nuestros programas nos referiremos a los archivos mediante un nombre simbólico, es decir similar a como referenciamos una variable.

Supongamos que nuestro archivo es `c : \AyED \ Anio2011 \ comercio.dat` y que ya existe, Vamos a referirnos a él mediante un nombre simbólico, por ejemplo **unarchi**, para asociarlos tenemos la sentencia **assign**. En nuestro programa tenemos que tener algo así:

```
program unaprueba(input, output);
type producto=record nro: integer;
                     nombre: string[10];
                     precio: real
                     end;
almacen = file of producto;      estamos suponiendo que comercio.dat
                                es un archivo de registros
```

```
var unarchi: almacen; ...
begin assign(unarchi,'c : \ AyED \ Anio2011 \ comercio.dat'); ...
```

Ahora bien, una variante para el **assign** sería que en vez de explicitar el nombre completo del archivo utilicemos una variable de tipo string y que mediante una sentencia read o readln introduzcamos el nombre del archivo, lo cual le da mas ductilidad al programa. Por ejemplo

```
program otraprueba(input, output);
type producto=record nro: integer;
                     nombre: string[10];
                     precio: real
                     end;
    almacen = file of producto;
var unarchi: almacen; elarch: string[20]; ...
begin readln(elarch); assign(unarchi,elarch); ...
```

El archivo puede estar ya con datos cargados (por ejemplo por otro programa), puede estar vacío, o puede no existir.

Pero ¿Cómo hacemos para crear un archivo?. Hay muchas maneras. Veremos una que nos va a permitir crear un archivo vacío. Desde la pantalla de Pascal (no desde el programa), en el menú de barras de la parte superior, hacemos “click” en **file** (o *archivo*), se descuelga un popup, “click” en **save** (o *guardar*) y en la ventana que aparece al costado escribimos el nombre del archivo.

Ahora bien, en el programa, para poder comenzar a trabajar con el archivo (leer registros, grabar registros), tenemos que “abrirlo”, esto lo podemos lograr con la sentencia **reset**. Esta sentencia abre el archivo y queda posicionado para leer o grabar en el primer registro. Tendríamos entonces

```
program unaprueba(input, output);
type producto=record nro: integer;
                     nombre: string[10];
                     precio: real
                     end;
    almacen = file of producto;
var unarchi: almacen; ...
begin assign(unarchi,'c : \ AyE \ Anio2011 \ comercio.dat');
    reset(unarchi); ...
```

si dentro del programa, con el archivo ya abierto, después de haber trabajado en él, volvemos a poner **reset**, quedaremos nuevamente preparados para leer o grabar en el primer registro.

Existe otra manera de crear (y abrir) archivos, con la sentencia **rewrite**, pero puede llegar a ser peligrosa ya que borra los datos existentes si el archivo ya existía y tenía datos cargados. De todos modos, explicaremos su funcionamiento:

Si el archivo no existe, **rewrite** lo crea, pero si ya existe, lo abre y se pierden los datos que contenía.

Primero deben poner el **assign** y luego el **rewrite** o el **reset**.

Al terminar de trabajar con el archivo, siempre debemos “cerrarlo”, lo que logramos con **close** en el programa anterior deberíamos poner la sentencia **close(unarchi)**


Existe una función que nos devuelve el tamaño del archivo en cantidad de registros, es **filesize**. Si para este archivo (supongamos que lo referenciamos como *zarch*),

registro	registro	registro	registro	registro	registro	registro
----------	----------	----------	----------	----------	----------	----------

tenemos en el programa **filesize(zarch)**, nos devolverá el entero **7**. Podríamos tener, por ejemplo una sentencia **n := filesize(zarch)** (estamos suponiendo que *n* es de tipo *integer*)

En Pascal los registros tienen asignados números enteros consecutivos a partir de **0**.

registro 0	registro 1	registro 2	registro 3	registro 4	registro 5	registro 6
------------	------------	------------	------------	------------	------------	------------

señalador o puntero  (en este caso apuntando al registro 4)

Podemos pensar que los archivos tienen un “señalador” o “puntero” que siempre está apuntando al próximo registro a ser procesado.

La función **filepos** devuelve el número del registro apuntado, para el dibujo anterior, **filepos(zarch)** devolverá el entero **4**.

La instrucción **seek** posiciona el puntero en el registro a ser procesado. Por ejemplo **seek(zarch, 5)** hará que el puntero “apunte” al registro 5.

En vez de poner un número entero, podemos poner una variable de tipo integer. Por ejemplo **seek(zarch, k)** (donde suponemos que es *k* una variable entera que tiene un cierto valor).

Veamos ahora dos instrucciones que se usan para leer información de un archivo o grabar información en un archivo. Son, respectivamente **read** y **write**.

(ejemplificamos para archivo de registros)

read(variableTipoArchivo, variableTipoRegistro) lee el registro apuntado y lo asigna a la variable **variableTipoRegistro**, **y pasa a apuntar al registro siguiente**.

write(variTipoArchivo, variTipoRegistro) graba el registro contenido en la variable **variTipoRegistro** en la posición apuntada **y pasa a apuntar al registro siguiente**.

notemos que leemos o grabamos **registros completos**

Reiteramos: después del **read** y/o del **write** el puntero pasa a apuntar al registro siguiente.

El tema de lo que grabamos o leemos de archivos es sencillo si lo analizan con el siguiente criterio:

Supongamos que en el programa tenemos `var a:file of t`,

- Si `t` es de tipo entero, tendré un archivo de enteros y cada vez que haga `write(a,v)`, `v` tendrá que ser tipo entero y grabaré en mi archivo un nuevo entero (y avanzaré al entero siguiente). Cada vez que haga `read(a,v)`, `v` tendrá que ser tipo entero y leeré de mi archivo un entero (y avanzaré al entero siguiente).
- Si `t` es de tipo real, tendré un archivo de reales y cada vez que haga `write(a,v)`, `v` tendrá que ser tipo real y grabaré en mi archivo un nuevo real (y avanzaré al real siguiente). Cada vez que haga `read(a,v)`, `v` tendrá que ser tipo real y leeré de mi archivo un real (y avanzaré al real siguiente).
- Si `t` es de tipo registro, tendré un archivo de registros y cada vez que haga `write(a,v)`, `v` tendrá que ser tipo registro y grabaré en mi archivo un nuevo registro (y avanzaré al registro siguiente). Cada vez que haga `read(a,v)`, `v` tendrá que ser tipo registro y leeré de mi archivo un registro (y avanzaré al registro siguiente).
- Si `t` es de tipo vector, tendré un archivo de vectores y cada vez que haga `write(a,v)`, `v` tendrá que ser tipo vector y grabaré en mi archivo un nuevo vector (y avanzaré al vector siguiente). Cada vez que haga `read(a,v)`, `v` tendrá que ser tipo vector y leeré de mi archivo un vector (y avanzaré al vector siguiente).
- Si `t` es de tipo matriz, tendré un archivo de matrices y cada vez que haga `write(a,v)`, `v` tendrá que ser tipo matriz y grabaré en mi archivo una nueva matriz (y avanzaré a la matriz siguiente). Cada vez que haga `read(a,v)`, `v` tendrá que ser tipo matriz y leeré de mi archivo una matriz (y avanzaré a la matriz siguiente).

Finalmente veamos la función `eof` (end of file). Esta función devuelve `true` después de haberse procesado el último registro del archivo (es decir si el puntero apunta al final del archivo), caso contrario, devuelve `false`.

Los siguientes tres programas son nada mas que para resaltar que lo que se “escribe” en un archivo perdura aunque el programa termine. Además, para variar un poco, no vamos a trabajar con archivo de registros, sino de reales.

Supongamos que en Pascal ejecutamos el siguiente programa:

```
program pantalla(input,output);
var x: real;
begin
  write('Ingrese un real'); readln(x); write(x);
  write('Ingrese un real'); readln(x); write(x);
  write('Ingrese un real'); readln(x); write(x);
  write('Ingrese un real'); readln(x); write(x);
```

```
write('Ingrese un real'); readln(x); write(x)
end.
```

nos pedirá cinco veces que ingresemos por teclado un real y lo “grabará” en la pantalla.

Supongamos que ingresamos los reales **1.75 -2.0 1.0 3.1 y -1.11**

La pantalla se mostrará mas o menos así:

```
Ingrese un real 1.75
1.7500000000E+00
Ingrese un real -2.0
-2.0000000000E+00
Ingrese un real 0.0
1.0000000000E+00
Ingrese un real 3.1
3.1000000000E+00
Ingrese un real -1.11
-1.1100000000E+00
```

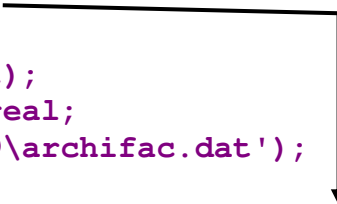
Hemos puesto en “**verde claro**” los carteles y en “**verde oscuro**” lo que ingresamos por teclado y lo que está en negro representa lo que muestra el programa por pantalla.

Cuando el programa termina y comenzamos cualquier otro trabajo, la pantalla se borra y los datos que había en ella se pierden.

Supongamos que no queremos perder los datos leídos. Vamos a guardarlos en un archivo al que llamaremos **archifac.dat** al que lo pondremos en una carpeta cualquiera, por ejemplo **c:\com99**. Para hacer referencia al archivo en el programa lo llamaremos **prueba**.

Simplemente cambiamos las **write** esas

```
program archifacl(input,output);
var prueba: file of real; x: real;
begin assign(prueba, 'c:\com99\archifac.dat');
      rewrite(prueba);
      write('Ingrese un real'); readln(x); write(prueba,x);
      write('Ingrese un real'); readln(x); write(prueba,x);
      write('Ingrese un real'); readln(x); write(prueba,x);
      write('Ingrese un real'); readln(x); write(prueba,x);
      write('Ingrese un real'); readln(x); write(prueba,x); close(prueba)
end.
```



Ahora, (suponiendo que ingresamos los mismos cinco valores) el archivo **c:\com99\ archifac.dat** quedó con esos cinco valores ya mencionados grabados, los cuales se conservarán.

1.75	-2.0	1.0	3.1	-1.11
------	------	-----	-----	-------

Si por ejemplo queremos usar ese archivo en otro momento, podemos ver, por ejemplo, el siguiente programa que lee uno a uno los valores guardados (los cinco ya mencionados), los muestra por pantalla, los va sumando y finalmente muestra el valor de esa suma.

```
program archifac2(input,output);
var prueba: file of real; pum, tota: real; rta:char;
begin assign(pepe, 'c:\com99\archifac.dat');
  reset(pepe); tota:= 0.0; notar que pusimos reset y no rewrite
  while not eof(pepe) do begin read(pepe, pum);
                                writeln('Leo el valor ', pum:10:3);
                                tota:= tota + pum
                                end;
  close(pepe);
  write('Los valores suman ', tota:10:3, ' pulse una tecla . . .');
  read(rta)
end.
```

y verificarán que fueron guardados y leídos correctamente.

A la variable con que leemos del archivo la hemos llamado voluntariamente **pum** para resaltar que no es necesario usar los mismos nombres para leer que para escribir (basta con que se respete el tipo). También hemos referido al archivo con otro nombre simbólico (**pepe**).

Pasemos a ver unos cuantos ejemplos

Problema: Se supone que desde el menú de barras de Pascal fue creado el archivo `c:\archi04.dat` y que está vacío. Leer de teclado los datos de un cierto número de alumnos (nombre:string[6], nota:entero), no se sabe cuantos alumnos son, e ir guardándolos en dicho archivo. A medida que se van leyendo los alumnos, ir asignándole números enteros consecutivos a partir de 1.

Listarlos e informar cuantos alumnos fueron grabados en el archivo.

```

program file01(input, output);
type alumno=record nro: integer;
                   nombre: string[6];
                   nota: integer
                   end;
  curso=file of alumno;
var com22: curso; aux: alumno; canti: integer; brenom: string[6];
begin assign(com22, 'c:\archi04.dat'); reset(com22) (*);
  canti:=0; readln(brenom);
  while brenom<>'*****' do begin canti:=canti+1;
                                aux.nro:=canti;
                                aux.nombre:=brenom;
                                readln(aux.nota);
                                write(com22, aux);
                                read(brenom)
                                end;
  reset(com22);
  writeln('Nro.      Nombre      Nota');
  writeln('----      -')
  while not eof(com22) do
    begin read(com22, aux);
          write(aux.nro:4, '      ', aux.nombre, aux.nota:7)
    end;
  close(com22);
  writeln('Fin de la grabacion, se grabaron ',canti,' alumnos')
end.

```

Si el archivo no hubiese existido, en vez de (*) hubiéramos tenido que poner `rewrite(com22)`

Supongamos que nos piden que, ingresando un Nro. de alumno informemos Nombre y Nota

```

program file02(input, output);
type alumno=record nro: integer;
                   nombre: string[6];
                   nota: integer
                   end;
  curso=file of alumno;
var com22: curso; aux: alumno; n, t: integer;
begin assign(com22, 'c:\archi04.dat'); reset(com22); t:= filesize(com22);
  write('Ingrese Nro. alumno a buscar sus datos'); readln(n);
  while n<1 or n>t do begin write('Incorrecto, ingrese de nuevo');
                          readln(n)
                          end;
  seek(com22,n-1); read(com22, aux); write(aux.nombre, aux.nota);
  close(com22)
end.

```

No siempre hay un dato en el registro que actúe como clave de manera que permita ubicar directamente el registro con un **seek**.

Supongamos que queremos crear el archivo 'c:\com44.dat' pero que los números de alumno, si bien son únicos para cada alumno, no tengan ningún orden.

Supongamos que sean enteros positivos. Un programa para crear el archivo, leyendo los datos de los alumnos, listarlo e informar cuantos alumnos se grabaron sería:

```
program file03(input, output);
type alumno=record nro: integer;
                   nombre: string[6];
                   nota: integer
                   end;
curso=file of alumno;
var comis: curso; aux: alumno; n: integer;
begin assign(comis, 'c:\com44.dat'); rewrite(comis);
  read(n);
  while n>0 do begin aux.nro:= n;           para no complicar el algoritmo
                                     read(aux.nombre); estamos suponiendo que no se
                                     readln(aux.nota); ingresan dos alumnos con un
                                     write(comis, aux); mismo número
                                     read(n)
                   end;
  reset(comis);
  writeln('Nro.      Nombre      Nota');
  writeln('----      -')
  while not eof(comis) do
    begin read(comis, aux);
          write(aux.nro:4, ' ', aux.nombre, aux.nota:7)
    end;
  close(comis);
  writeln('Fin grabacion, se grabaron ', filesize(comis), ' alumnos')
end.
```

Supongamos que nos piden que, ingresando un Nro. de alumno informemos Nombre y Nota. Ahora no podemos ubicar el alumno con un **seek** ya que no hay ninguna relación que vincule al número del alumno con el número del registro. No tenemos mas remedio que recorrer el archivo desde el principio hasta encontrar (o no) al alumno. Y si no lo encontramos, informarlo.

```
program file04(input, output);
type alumno=record nro: integer;
                   nombre: string[6];
                   nota: integer
                   end;
curso=file of alumno;
var comis: curso; aux: alumno; n: integer;
begin assign(comis, 'c:\com44.dat'); reset(comis);
  read(n); read(comis, aux);
  while n<>aux.nro and not eof(comis) do read(comis, aux);
  if n=aux.nro then write(aux.nro:4, ' ', aux.nombre, aux.nota:7)
    else write('Alumno no encontrado');
  close(comis)
end.
```

Mas de una vez tendremos que trabajar con algún archivo que no sabemos si existe o no, pero si existiera, no queremos perder los datos que contiene. No podemos usar **rewrite**.

Si usamos **reset**, en el momento de la ejecución, el Sistema Operativo me informará de un error de entrada/salida si no existiera. Para evitar esto, debo desactivar esta posibilidad a través de la instrucción **{I-}**. De esta manera podré seguir con el programa. Después de una instrucción de E/S, el Sist. Operat. devuelve un parámetro **IORESULT** que indica el tipo de error que encontró, si vale cero es que no hubo error, si hubo error, entre otros valores, si devuelve 2, significa que el archivo no fue encontrado. Es peligroso seguir con la verificación desactivada, debo poner enseguida **{I+}**.

Supongamos que se da esta situación (de no saber si existe o no) con un archivo **c:\opera.dat**. Mi programa debería comenzar mas o menos así:

```
program file00(input, output);
type alumno=record nro: integer;
                   nombre: string[6];
                   nota: integer
                   end;
curso=file of alumno;
var comis: curso; . . . .
begin assign(comis, 'c:\opera.dat'); {I-}; reset(comis);
      if ioresult=2 then rewrite(comis); {I+}; . . . .
```

Sigamos practicando con archivos

Problema: Se supone que el archivo *c : \ ejem.dat* tiene una cierta cantidad (no se sabe cuantos) de registros cargados. Se supone además que el archivo *c : \ alrreves.dat* fue creado desde el menú de barras de Pascal y que está vacío. Se pide:

- 1) Listar el archivo *ejem.dat*
- 2) Grabar en *alrreves.dat* los registros de *ejem.dat* pero en el orden inverso
- 3) Listar *alrreves.dat*

```
program davuelta(input,output);
type libro=record nroint,codi,stock:integer;
                 titulo,autor:string[20];
                 precio:real
                 end;
negocio=file of libro;
var libros, dadovue : negocio ; aux:libro ; rta:char ;
    n , i , k : integer ;
begin assign(libros,'c:\ejem.dat');
      assign(dadovue,'c:\alrreves.dat');
      reset(libros); reset(dadovue);
      writeln('Nro Codigo Titulo Stock');
      writeln('--- -----');
      while not eof(libros) do
        begin read(libros,aux) ;
              writeln(aux.nroint:3,aux.codi:8,' ',aux.titulo,aux.stock:5)
        end;
      writeln('Fin del primer listado');
      n:=filesize(libros);
      for i:=1 to n do
        begin k:= n-i;
```

```

        seek(libros,k);
        read(libros,aux);
        write(dadovue,aux);
    end;
    reset(dadovue);
    writeln('Nro  Codigo  Titulo                      Stock');
    writeln('---  -----  -----  -----');
    while not eof(dadovue) do
        begin read(dadovue,aux);
                writeln(aux.nroint:3,aux.codi:8,'  ',aux.titulo,aux.stock:5)
        end;
    writeln('Fin del segundo listado');
    close(libros); close(dadovue);
    writeln('Fin del programa, ingrese cualquier letra');
    readln(rta)
end.

```

Varias veces ponemos al final del programa la instrucción `readln(rta)` nada mas que para que la pantalla “desaparezca” recién después de pulsar una tecla (para poder verificar los resultados). A veces, en vez de leer un char haremos leer un entero. También podemos poner readln.

Problema: Se supone que desde el menú de barras de Pascal fue creado el archivo `c:\libreria.dat` y que está vacío. Leer de teclado los datos de una cierta cantidad de libros (código, título, autor, precio, cantidad en stock) . A medida que se van leyendo los libros, ir agregándolos al archivo asignándole números enteros consecutivos a partir de 1. Luego de cargados listará todos los libros.

Se desea luego realizar la venta de cierta cantidad de ejemplares de un libro, para lo cual deberá verificarse que haya suficiente cantidad en stock. El programa deberá indicar si se puede o no efectuar la venta, informará el precio y actualizará el stock.

Finalmente se repetirá el listado.

```

program libreria(input,output);
type libro=record nroint,codi,stock:integer;
                  titulo,autor:string[20];
                  precio:real
                end;
    negocio=file of libro;
var
libros:negocio;aux:libro;unprecio:real;n,uncodi,unnro,unacanti:integer;
begin assign(libros , 'c:\libreria.dat');
    reset(libros); n:=0;
    write('Ingrese codigo(cero termina):');readln(uncodi);
    while uncodi>0 do
        begin n:=n+1; aux.nroint:=n; aux.codi:=uncodi;
                write('Ingrese titulo:');readln(aux.titulo);
                write('Ingrese autor :');readln(aux.autor);
                write('Ingrese precio:');readln(aux.precio);
                write('Ingrese stock :');readln(aux.stock);
                write(libros,aux);
                write('Ingrese codigo (cero termina):');readln(uncodi)
        end;
    reset(libros);
    writeln('Nro  Codigo  Titulo                      Stock');
    writeln('---  -----  -----  -----');

```

```

while not eof(libros) do
  begin read(libros,aux) ;
        writeln(aux.nroint:3,aux.codi:8,'  ',aux.titulo,aux.stock:5)
  end;
write('Ingrese nro. interno del libro a vender (entre 1 y ',n:3,') :');
readln(unnro); unnro:=unnro-1;           (por la modalidad que elegimos para asignar el código)
write('Ingrese cantidad de ejemplares a vender:'); el libro de código "unnro" ocupará
readln(unacanti);                        el registro unnro - 1)
seek(libros,unnro) ; read(libros,aux);    (me posiciono en el registro y lo leo)(pero, OJO, porque
if aux.stock>=unacanti then              despues de leer ya quedé apuntando al registro siguiente)
  begin unprecio:=unacanti*aux.precio;
        aux.stock:=aux.stock-unacanti;
        write('Venta aceptada,importe:',unprecio:10:2);
        writeln(' quedan en stock:',aux.stock:3);
        seek(libros,unnro);write(libros,aux)      (el "seek" es para volver a ubicarme
  end                                           en el registro, ya que el "read" me había sacado)
  else writeln('Venta rechazada por falta de stock');
reset(libros);
writeln('Nro  Codigo  Titulo                      Stock');
writeln('---  -----  -----  -----');
while not eof(libros) do
  begin read(libros,aux) ;
        writeln(aux.nroint:3,aux.codi:8,'  ',aux.titulo,aux.stock:5)
  end;
close(libros);
writeln('Fin')
end.

```

Trabajando con archivos, permanentemente estaremos haciendo **Altas**, **Bajas** y **Modificaciones (ABM)**.

Lo que va a ocurrir también muchas veces es que no sea tan fácil acceder directamente a un registro a través de una clave como en los casos anteriores.

Veamos un poco mas detalladamente en que consisten las tareas antes mencionadas, ejemplificando para archivos de registros.

Altas: Consiste en agregar un nuevo registro al final del archivo (si se desea agregar varios, se pone todo dentro de un **while**). Si, como en varios ejemplos anteriores un dato clave era asignado por el programa, prácticamente sin verificar nada, agregamos el registro y el dato clave del nuevo registro es asignado por el programa.

Muchas veces esto no es así, los registros en general tienen un campo identificador que suele ser único y que es propuesto por el usuario pero que no permite ubicar el registro. En este caso el programa de altas debe verificar que no exista ya un registro con ese campo identificador, en cuyo caso se informa y se rechaza el alta.

Bajas: Consiste en "eliminar" un registro del archivo (si se desea eliminar varios, se pone todo dentro de un **while**). La eliminación puede ser física o lógica, como se explicará mas adelante. Acá, lo que debemos hacer es buscar el registro dentro del archivo (generalmente a través de su campo identificador), Si no se encuentra hay que avisar; si se encuentra, antes de eliminarlo, hay que mostrar los datos característicos del registro, pedir confirmación, y recién ahí, eliminarlo.

Modificaciones: Consiste en modificar algunos campos de un registro (si se desea modificar varios registros, se pone todo dentro de un **while**). También acá hay que ubicar el registro dentro del archivo, cambiar los campos, y volver a grabarlo. Si no se encontró, hay que avisar.

Algunos comentarios sobre BAJAS en un archivo:

Cuando se intenta dar de baja a un registro de un archivo, hay dos maneras de hacerlo:

Baja física: consiste en hacer “desaparecer” el registro, corriendo todos los que están “a su derecha” un registro “a la izquierda”, con lo cual el registro “borrado” desaparece y el archivo quedará con un tamaño un registro menor al que tenía. **Los datos del registro dado de baja se perdieron.**

Baja lógica: consiste en incorporar a los registros un campo (booleano en los ejemplos, también podría ser char, integer, etc.) que indique si el registro está dado de baja o no. En el ejemplo, al campo en cuestión lo hemos llamado **debaja**.

Supongamos que queremos dar de baja al registro con nroint = 3 (el registro 2)

Baja lógica

antes.de la baja							EOF
0 false	1 false	2 false	3 false	4 false	5 false	6 false	
reg 0	reg 1	reg 2	reg 3	reg 4	reg 5	reg 6	
después.de la baja							
0 false	1 false	2 true	3 false	4 false	5 false	6 false	
reg 0	reg 1	reg 2	reg 3	reg 4	reg 5	reg 6	

En rojo se indican los valores del número de registro
y en azul el valor de **debaja** (el campo que usamos para indicar si está dado de baja o no)
3 true sombreado indica que el registro tiene baja lógica

Baja física (no es necesario utilizar el campo **debaja**)

							EOF
reg 0	reg 1	reg 2	reg 3	reg 4	reg 5	reg 6	

						EOF
reg 0	reg 1	reg 2	reg 3	reg 4	reg 5	

Este tipo de baja tiene varios inconvenientes:

- Se perdieron los datos del registro dado de baja mientras que con la baja lógica, mediante un simple módulo muy similar al de modificaciones, se puede recuperar.
- Si hubiera un campo clave para ubicarlo directamente, éste deja de servir, por lo que habría que redefinir los campos clave y listarlos (imaginar archivos con miles de registros) y aún así correríamos el riesgo de cometer errores.

A continuación vamos a ver un programa que permita hacer alta, bajas modificaciones y listados. Valen las siguientes aclaraciones:

- Los registros tienen un campo que permite ubicarlos directamente (**nroint**)
- Se pueden hacer cualquiera de las tareas enunciadas mediante un menú de opciones
- El módulo de altas permite hacer varias altas
- El módulo de bajas permite hacer varias bajas
- El módulo de modificaciones permite hacer modificaciones a un solo registro
- El módulo de listados sólo permite hacer un listado

Problema: Para el archivo de la librería de uno de los ejemplos anteriores (es decir que el archivo ya existe), a cuyos registros se agregó un campo booleano (*debaja*), confeccionar un programa Pascal que pueda efectuar Altas, Bajas, Modificaciones (sólo modificaremos precio y stock) y Listados. Las Bajas serán lógicas, es decir no eliminarán el registro, sino que colocarán en true el campo *debaja*. Las Altas deberán poner el mismo campo en false. En los listados, no se listan los libros que tengan dicho campo en true.

```

program completo(input,output);
type libro=record nroint,codi,stock:integer;
                  titulo,autor:string[20];
                  debaja:boolean;
                  precio:real
                end;
negocio=file of libro;
var libros:negocio ; aux:libro ; rta:char ;
    n , uncodi , uninte , op : integer ;
begin assign(libros,'c:\ejem.dat');reset(libros);
  writeln('Menu');writeln('----');writeln('1-Altas');writeln('2-Bajas');
  writeln('3-Modificaciones');writeln('4-Listados');writeln('5-Salir');
  write('Elija su opcion:');readln(op);
  while op<5 do
    begin
      case op of
        1: begin writeln('Modulo de altas');writeln('-----');
              n:=filesize(libros); seek(libros,n); me voy al final del archivo para agregarlos
              write('Ingrese codigo(cero termina):');readln(uncodi);
              while uncodi>0 do
                begin aux.nroint:=n+1 ; aux.codi:=uncodi;
                  write('Ingrese titulo:') ; readln(aux.titulo);
                  write('Ingrese autor :') ; readln(aux.autor);
                  write('Ingrese precio:') ; readln(aux.precio);
                  write('Ingrese stock :') ; readln(aux.stock);
                  aux.debaja:=false ; write(libros,aux); n:=n+1;
                  write('Ingrese codigo (cero termina):');readln(uncodi)
                end ;
              writeln('Fin del modulo de altas')
            end;
        2: begin writeln('Modulo de bajas');writeln('-----');
              write('Ingrese Nro. interno(cero termina):');readln(uninte);
              while uninte>0 do
                begin uninte:=uninte-1 ; seek(libros,uninte) ;
                  read(libros,aux) ;
                  writeln('Codigo:',aux.codi) ;

```



```

        writeln('Titulo:',aux.titulo) ;
        write('Lo da de baja ? (S/N):');readln(rta);
        if rta='S' then begin aux.debaja:=true ;
                           seek(libros,uninte);
                           write(libros,aux)
                           end;
        write('Ingrese Nro. interno(cero termina):');
        readln(uninte);
    end ;
    writeln('Fin del modulo de bajas')
end;
3: begin writeln('Modulo de modificaciones');
    writeln('-----');
    write('Ingrese nro. interno del libro a modificar:');
    readln(uninte); uninte:= uninte - 1;
    seek(libros,uninte);read(libros,aux);
    writeln('El libro es');writeln(aux.nroint, ' ',aux.titulo);
    write('Ingrese nuevo precio:');readln(aux.precio);
    write('y nueva cantidad en stock:');readln(aux.stock);
    seek(libros,uninte);write(libros,aux);
    writeln('Fin de la modificacion')
end;
4: begin writeln('Modulo de listados');
    writeln('-----');
    reset(libros);
    writeln('Nro  Codigo  Titulo                               Stock');
    writeln('---  -----  -----  -----');
    while not eof(libros) do
        begin read(libros,aux) ;
            if not aux.debaja then
                writeln(aux.nroint,aux.codi, ' ',aux.titulo,aux.stock)
            end;
        writeln('Fin del listado')
    end;
end;
writeln('Menu');writeln('----');
writeln('1-Altas');writeln('2-Bajas');
writeln('3-Modificaciones');writeln('4-Listados');
writeln('5-Salir');
write('Elija su opcion:');readln(op)
end;
close(libros);
writeln('Fin del programa, ingrese cualquier letra');readln(rta)
end.

```

En el ejemplo anterior, algunas cosas se hacían mas sencillas por el hecho que era fácil ubicar directamente cualquier registro mediante un simple cálculo (en los ejemplos, restándole 1 al número del artículo).

Muchas veces esto no es posible. Veamos un ejemplo sencillo.

Supongamos que tenemos un negocio que tiene un archivo **c:\ferrete.dat** donde guardan los datos de los artículos que venden.

Los datos de los artículos son:

- número del artículo (entero positivo) (no puede haber números de artículos repetidos)
- nombre del artículo (cadena de 6 caracteres)
- precio (real)

Al registro le vamos a agregar un campo booleano para indicar si está dado de baja o no.

A fin de simplificar la comprensión vamos a hacer tres programas separados (uno de altas, otro de bajas y otro de modificaciones).

Para simplificar mas la cosa, vamos a hacer una sola alta, una sola baja y una sola modificación.

Si quieren hacer varias, se hace lo de siempre, se pone todo dentro de un **mientras**.

Los registros están grabados en cualquier orden y los números de artículo no tienen por que ser correlativos.

Vamos a suponer que el archivo ya tiene registros cargados, es decir que puede tener el siguiente aspecto

146	Pinza	75.80	f	33	Tenaza	70.30	t	44	Clavos	8.00	f	...	31	Pincel	15.95	f
-----	-------	-------	---	----	--------	-------	---	----	--------	------	---	-----	----	--------	-------	---

con **f** estamos indicando "false" es decir que el artículo NO está dado de baja y con **t** indicamos "true", o sea que ese artículo fue dado de baja

Vamos a escribir el programa de **altas**.

Vamos a pedir antes que nada el número del artículo al que quiero dar de alta para verificar que no exista en el archivo, ya que si existe, no podemos permitir un alta con el mismo número.

También vamos a adoptar el criterio que aunque exista, pero esté dado de baja, tampoco lo vamos a permitir (podríamos haber adoptado un criterio opuesto)

Para ver si ya existe el número de artículo no podemos hacer otra cosa que recorrer desde el principio todo el archivo

```

program altas (input,output);
type arti=record nro:integer;
                  descri:string[6];
                  precio:real;
                  debaja:boolean;
end;
var a:file of arti; r:arti; n:integer; esta:boolean;
begin assign(a, 'c:\ ferrete.dat'); reset(a);
      write('Ingrese numero de articulo a dar de alta'); readln(n);
      esta:=false;
      while not eof(a) and not esta do begin read(a,r);
                                                if r.nro=n then esta:=true
                                                end;
      if esta then writeln('Numero ya existe, alta rechazada')
      else begin r.nro:=n; r.debaja:=false;
                write('Ingrese nombre del articulo');
                readln(r.descri);
                write('y su precio');
                readln(r.precio);
                write(a,r)
                end;
      close(a)
end.

```

notemos que si el número no existía, estamos en el eof, y el write(a,r) va a grabar el registro al final del archivo, el cual pasa a tener un registro mas.

Vamos a escribir el programa de **bajas**.

Vamos a pedir el número del artículo a dar de baja y recorreremos el archivo desde el principio. Si encontramos el artículo, ponemos el campo **debaja** en **false**.

```
program bajas (input,output);
type arti=record nro:integer;
                  descri:string[6];
                  precio:real;
                  debaja:boolean
                end;
var a:file of arti; r:arti; n:integer;
begin assign(a, 'c:\ ferrete.dat'); reset(a);
      write('Ingrese numero de articulo a dar de baja'); readln(n);
      read(a,r);
      while not eof(a) and r.nro<>n do read(a,r);
      if r.nro<>n then writeln('No existe el articulo')
      else begin r.debaja:=true;
                seek(a,filepos(a)-1);
                write(a,r);
                write('El articulo ',r.descri,' fue eliminado')
              end;
      close(a)
end.
```

para cambiar un poco no usamos la variable booleana esta
vuelvo un registro atrás ya que había avanzado un registro

Por último, vamos a escribir el programa de **modificaciones**.

Vamos a permitir modificar el campo **descri** y el campo **precio**. No es recomendable cambiar el campo que funciona como clave única de identificación, en nuestro caso **nro** (aunque también podríamos hacerlo). Tampoco permitimos modificar el campo **debaja**, ya que para ello tenemos el módulo descrito anteriormente.

El algoritmo es bastante similar al anterior (bajas).

```
program modifica (input,output);
type arti=record nro:integer;
                  descri:string[6];
                  precio:real;
                  debaja:boolean
                end;
var a:file of arti; r:arti; n:integer;
begin assign(a, 'c:\ ferrete.dat'); reset(a);
      write('Ingrese numero de articulo a modificar'); readln(n);
      repeat read(a,r) until r.nro=n or eof(a);
      if r.nro<>n then writeln('No existe el articulo')
      else begin write('Articulo ',r.descri);
                  writeln(' ingrese nuevos datos');
                  write('Descripcion: '); readln(r.descri);
                  write('Precio      : '); readln(r.precio);
                  seek(a,filepos(a)-1);
                  write(a,r)
                end;
      close(a)
end.
```

ahora lo recorri con repeat-until

Un módulo para listar el archivo sería prácticamente igual al de la librería.

A continuación vamos a ver un ejemplo para confirmar que con cada **read** y **write**, el puntero avanza una posición.

La idea es grabar 4 reales (10.0 20.0 30.0 40.0) y luego poner en su lugar la mitad de cada uno, “tapando” al valor original.

Primero vamos a cometer una omisión, leeremos y grabaremos “olvidándonos” que cada vez que lee pasa al siguiente, veremos entonces que:

lee el 10.0, lo divide por 2 y graba tapando al 20.0 (dejando el 10.0 igual) pero ya apunta al 30.0
lee el 30.0, lo divide por 2 y graba tapando al 40.0 dejando el 30.0 igual

Después vamos a grabar 5 en vez de 4 (10.0 20.0 30.0 40.0 50.0) y con el mismo “olvido”
lee el 10.0, lo divide por 2 y graba tapando al 20.0 (dejando el 10.0 igual) pero ya apunta al 30.0
lee el 30.0, lo divide por 2 y graba tapando al 40.0 (dejando el 30.0 igual) pero ya apunta al 50.0
lee el 50.0 y pasa al EOF, por lo tanto el *write* siguiente agrega un real mas (25.0). Quedaron 6 reales en el archivo.

Finalmente pondremos la solución correcta

```
program verskip(input,output);
var pruskip: file of real ; i, fsz, fps:integer; x:real; rta:char ;
begin assign(pruskip,'c:\diviprog\algori~1\ejempl~1\verskip.dat');
  rewrite(pruskip); writeln('Voy a grabar 4 reales');
  for i:=1 to 4 do begin x:= 10.0 * i;          grabo los reales
                        write(pruskip,x)      10.0  20.0  30.0  40.0
                        end;
  fsz:= filesize(pruskip);
  writeln('Se grabaron', fsz:3, ' reales a saber');
  reset(pruskip);
  while not eof(pruskip) do begin read(pruskip,x);
                                writeln(x:8:2)      los listo
                                end;

  reset(pruskip);
  writeln('Ahora leo, divido por 2 y grabo');
  while not eof(pruskip) do begin read(pruskip,x);  leo, pero paso al sig.
                                x:=x/2;             grabo en el siguiente
                                write(pruskip,x)     la mitad del leído
                                end;

  fsz:= filesize(pruskip);
  writeln('Sigue habiendo', fsz:3, ' reales a saber');
  reset(pruskip);
  while not eof(pruskip) do begin read(pruskip,x);
                                writeln(x:8:2)      los listo
                                end;

  rewrite(pruskip);          hago rewrite para borrar lo que se había grabado
  writeln('Ahora borro todo y grabo 5 en vez de 4');
  for i:=1 to 5 do begin x:= 10.0 * i;
                        write(pruskip,x)           todo lo mismo pero
                        end;                        grabo 5 reales
  fsz:= filesize(pruskip);
  writeln('Se grabaron', fsz:3, ' reales a saber');
  reset(pruskip);
```

```

while not eof(pruskip) do begin read(pruskip,x);
                                writeln(x:8:2)    los listo
                                end;

reset(pruskip);
writeln('Ahora leo, divido por 2 y grabo');
while not eof(pruskip) do begin read(pruskip,x); al leer el ultimo real
                                x:=x/2;           paso al EOF y grabo
                                write(pruskip,x)   agregando un registro
                                                mas
                                end;

fsz:= filesize(pruskip);
writeln('Pero ahora quedaron', fsz:3, ' reales a saber');
reset(pruskip);
while not eof(pruskip) do begin read(pruskip,x);
                                writeln(x:8:2)    los listo
                                end;

writeln('Si quiero que cada real quede dividido por 2, hago');
rewrite(pruskip); writeln('Voy a grabar 4 reales');
for i:=1 to 4 do begin x:= 10.0 * i; genero de nuevo el archivo original con
                                write(pruskip,x)    10.0    20.0    30.0    40.0
                                end;

fsz:= filesize(pruskip);
writeln('Se grabaron', fsz:3, ' reales a saber');
reset(pruskip);
while not eof(pruskip) do begin read(pruskip,x);
                                writeln(x:8:2)    los listo
                                end;

reset(pruskip);
writeln('Ahora leo, divido por 2 y grabo');
while not eof(pruskip) do begin
                                fps:= filepos(pruskip); memorizo donde leo
                                read(pruskip,x);
                                x:=x/2;
                                seek(pruskip,fps);      y vuelvo para grabar
                                write(pruskip,x)        en el mismo lugar
                                end;

writeln('Y lo que quedo grabado es');
reset(pruskip);
while not eof(pruskip) do begin read(pruskip,x);
                                writeln(x:8:2)
                                end;

write('Pulse...'); read(rta)    para que no se borre la pantalla hasta pulsar
end.

```

La salida de este programa es

```

Voy a grabar 4 reales
Se grabaron 4 reales a saber
10.00
20.00
30.00
40.00
Ahora leo, divido por 2 y grabo
Sigue habiendo 4 reales a saber
10.00

```

```

    5.00
    30.00
    15.00
Ahora borro todo y grabo 5 en vez de 4
Se grabaron 5 reales a saber
    10.00
    20.00
    30.00
    40.00
    50.00
Ahora leo, divido por 2 y grabo
Pero ahora quedaron 6 reales a saber
    10.00
    5.00
    30.00
    15.00
    50.00
    25.00

Si quiero que cada real quede dividido por 2, hago
Voy a grabar 4 reales
Se grabaron 4 reales a saber
    10.00
    20.00
    30.00
    40.00
Ahora leo, divido por 2 y grabo
Y lo que quedo grabado es
    5.00
    10.00
    15.00
    20.00
Pulse...
```

Vamos a hacer otro ejemplo:

Un grupo de amigos decide organizar un Quini4 (similar al Quini6 pero las apuestas son de cuatro números) no puede haber números repetidos en las apuestas. Hacer un programa Pascal para:

- Leer todas las boletas (número de boleta, nombre del apostador, los cuatro números de su apuesta) e ir grabando cada boleta en **c:\apueq4.dat**
- Informar cuantas boletas se grabaron
- Leer luego los cuatro números sorteados
- Leer cada boleta de **c:\apueq4.dat** y grabar cada boleta que tenga cuatro aciertos (es decir, cada boleta ganadora) en **c:\gananq4.dat**
- Para calcular la cantidad de aciertos usar una función **aciertos**
- Informar la cantidad de boletas ganadoras y listarlas

```

program quini4(input,output);
type v4=array[1..4] of integer;
    apuesta=record nro:integer;
                  nom:string[6];
                  apu:v4
```

```

        end;
        archivos=file of apuesta;
var apuestas, ganaron: archivos; i, j, n, z: integer; rta: char;
    a: apuesta; lagana: v4;
function aciertos(p, q: v4): integer;
begin z:= 0;
    for i:=1 to 4 do
        for j:=1 to 4 do
            if p[i]=q[j] then z:= z + 1;
        aciertos:=z
    end;
begin assign(apuestas,'c:\apueq4.dat'); rewrite(apuestas);
    assign(ganaron,'c:\gananq4.dat'); rewrite(ganaron);
    write('Ingrese Nro. de apuesta (cero termina) ');
    readln(n);
    while n>0 do
        begin a.nro:= n; write('Ingrese nombre '); readln(a.nom);
            write('Y ahora los 4 nros. de la apuesta ');
            for i:=1 to 4 do read(a.apu[i]);
                write(apuestas, a); writeln;
                write('Ingrese Nro. de apuesta (cero termina) ');
                readln(n)
            end;
        writeln('Se grabaron', filesize(apuestas):3, ' apuestas');
        write('Ingrese los 4 numeros sorteados ');
        for i:=1 to 4 do read(lagana[i]); writeln;
        reset(apuestas);
        while not eof(apuestas) do
            begin read(apuestas,a);
                if aciertos(a.apu , lagana)=4 then write(ganaron,a)
            end;
        if filesize(ganaron)=0
            then writeln('No hubo ganadores')
            else begin writeln(filesize(ganaron):3, ' ganadores');
                reset(ganaron);
                while not eof(ganaron) do
                    begin read(ganaron,a);
                        write(a.nro:4,' ', a.nom);
                        for i:=1 to 4 do write(a.apu[i]:3);
                            writeln
                        end
                    end;
                end;
            write('Pulse...'); read(rta);
            close(apuestas); close(ganaron)
        end.

```

También habíamos dicho que podíamos tener archivos de vectores. Veamos un ejemplo

Armar un archivo que contenga 5 vectores. Los vectores serán de 4 enteros y sus valores deberán ser 1 2 3 4 / 11 12 13 14 / 21 22 23 24 / 31 32 33 34 / 41 42 43 44

Leer luego cada vector, listarlo e indicar la suma de sus elementos.

Finalmente indicar la suma total

```

program ArDeVec(input,output);
type v4=array[1..4] of integer;
var z:file of v4; i, j, k, sp, st:integer; v:v4;
begin assign(z,'c:\ArDV');rewrite(z);
  for i:= 1 to 5 do
    begin for j:= 1 to 4 do v[j]:= 10*(i-1) + j;
           write(z,v)
        end;
    reset(z); writeln;
    st:=0;
    for i:= 1 to 5 do
      begin read(z,v);
            write('Vector',i:2);
            sp:=0;
            for j:= 1 to 4 do begin write(v[j]:4);
                                   sp:=sp+v[j]
                                end;
            writeln(' que suma',sp:4);
            st:=st+sp
          end;
      writeln('y la suma total es',st:4,' ingrese un entero...');
      read(k);
      close(z)
    end.
end.

```

Finalmente hagamos un ejemplo con un archivo de matrices

Leer por filas tres matrices enteras de 2 filas y 4 columnas con el siguiente criterio:
 leer cada matriz y grabarla en el archivo **c:\ArDeMat.dat**, el archivo quedará entonces con tres registros, donde cada registro contendrá una matriz de 2x4.
 volver al principio del archivo, leer una a una las matrices, mostrarlas por filas en la pantalla, indicando además, que matriz es la que leen (*el orden en que fue ingresada*), el número del registro y la suma de cada fila

```

program ArDeMat(input,output);
type m24=array[1..2,1..4] of integer;
var i, j, k, q, tf: integer; z: file of m24; m:m24;
begin assign(z,'c:\aldo\ArDeMat.dat'); rewrite(z);
  for k:=1 to 3 do begin writeln('Ingrese una matriz por filas');
                        for i:=1 to 2 do for j:=1 to 4 do read(m[i,j]);
                        write(z,m)
                      end;
  reset(z); k:=0;
  while not eof(z) do begin q:= filepos(z); k:= k+1; read(z,m);
                           writeln('Matriz',k:2,' grabada en reg.',q:2);
                           for i:=1 to 2 do begin tf:=0;
                                                 for j:=1 to 4 do
                                                   begin tf:=tf+m[i,j];
                                                         write(m[i,j]:6)
                                                   end;
                           writeln(' fila',i:2,' suma',tf:6)
                           end

```



```

end;
write('Ingrese un numero:');read(k)
end.

```

Existe una cierta similitud entre la ubicación lógica de los elementos de un archivo y los elementos de un arreglo unidimensional (ya sean dichos elementos registros, o enteros, o reales, etc.)

Por lo tanto muchas veces se pueden usar en archivos técnicas que fueron explicadas en arreglos.

Supongamos que tenemos

v: array [1..N] of t donde **N** es una constante entera y **t** es cualquier tipo
a: file of t pensemos también que **a** tiene **N** elementos
supongamos que **aux** es una variable de tipo **t**

pensemos que queremos asignar el elemento de posición **i** del arreglo a la variable **aux**, en el programa deberíamos escribir

```
aux := v[i]
```

```

|
|
|
|
|
|
|

```

si queremos asignar el elemento **i**-ésimo del archivo a la variable **aux**, y siempre recordando que el primer elemento del archivo es el de posición 0, haríamos

```
seek(a,i-1)
read(a,aux)
```

Algo que debemos tener en cuenta es que el arreglo puede tener índice no-entero, y aún siendo entero, puede no empezar en 1 (aunque esto es raro). En este caso habría que modificar el algoritmo para tratar el archivo.

Ordenamiento de Archivos:

Vamos a usar el método de ordenamiento por **selección del mínimo**.

Vamos a ejemplificar sobre un archivo de enteros, pero la metodología es la misma si se quiere ordenar por un determinado campo en un archivo de registros.

```

program OrdeArch(input,output);
var a:file of integer;i,j,min,aux,posmin,k,n:integer;
begin assign(a,'c:\ArPaOrd.dat');reset(a);n:=filesize(a);
  for i:=0 to n-2 do begin seek(a,i);
    read(a,k);
    aux:=k;
    min:=k;
    posmin:=filepos(a)-1;
    for j:=i+1 to n-1 do
      begin seek(a,j);
        read(a,k);
        if k<min then begin min:=k;
                      posmin:=filepos(a)-1
                    end
      end;
    seek(a,i);
    write(a,min);
    seek(a,posmin);

```

```

                write(a,aux)
            end;
        reset(a);writeln('El archivo quedo asi');
        while not eof(a) do begin read(a,k);
                                write(k:4)
                            end;
        writeln;write('Ingrese un entero');read(j)
    end.

```

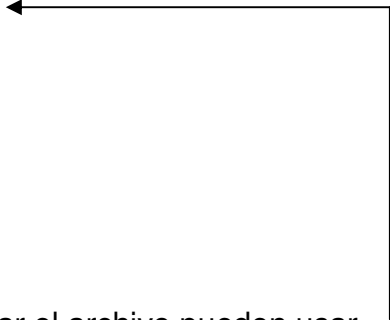
Si desean probar el programa anterior o hacerle modificaciones, tengan presente que el programa anterior modifica el archivo ya que lo ordena sobre si mismo. Entonces sería bueno generar el archivo nuevamente cada vez que vayan a probar el programa. Para ello, tienen el programa siguiente que arma un archivo con 11 enteros a saber

35 22 -6 -9 0 12 17 -60 33 1 -19

```

program ArPaOrd(input,output);
var v:array[1..11] of integer; a:file of integer;i,j,k:integer;
begin assign(a, 'c:\ArPaOrd.dat');rewrite(a);
    v[1]:= 35; v[2]:= 22; v[3]:= -6; v[4]:= -9; v[5]:= 0; v[6]:= 12;
    v[7]:= 17; v[8]:=-60; v[9]:= 33; v[10]:= 1; v[11]:=-19;
    for i:=1 to 11 do write(a,v[i]);reset(a);
    writeln('El archivo quedo asi');
    while not eof(a) do begin read(a,k);
                            write(k:4)
                        end;
    writeln;write('Ingrese un entero');read(j)
end.

```



Ahora hagamos un ordenamiento de un archivo pero por burbuja.

Podría ser el mismo enunciado que el anterior, y si quieren generar el archivo pueden usar pero cambiando el nombre ArPaOrd.dat por burbarch.dat

```

program burbarch(input, output);
var b: file of integer; ant, post, i, j, n: integer;
begin assign(b, 'c:\burbarch.dat');
    reset(b);
    writeln('Archivo antes de ordenar');
    writeln('-----');
    while not eof(b) do begin read(b, ant);
                            write(ant:5)
                        end; writeln;

    n:= filesize(b);
    for i:=0 to n-2 do
        for j:=n-2 downto i do
            begin seek(b, j);
                read(b, ant);
                read(b, post);
                if ant>post then begin seek(b, j);
                                write(b,post);
                                write(b,ant)
                            end;
            end;
        end;
    reset(b);
    writeln('Archivo despues de ordenar');
end.

```

```
writeln('-----');
while not eof(b) do begin read(b, ant);
                        write(ant:5)
                        end; writeln;

close(b); readln
end.
```

Búsqueda en Archivos:

En la gran mayoría de los ejemplos anteriores hemos realizado búsquedas secuenciales.

Veamos entonces como haríamos una búsqueda dicotómica.

Recordemos que la búsqueda dicotómica funciona si donde queremos buscar está ordenado.

Podríamos buscar en el archivo anterior, después de haberlo ordenado. Es decir en

-60 -19 -9 -6 0 1 12 17 22 33 35

Vamos a ejemplificar sobre un archivo de enteros, pero la metodología es la misma si se quiere buscar un determinado campo en un archivo de registros. *La **búsqueda secuencial** ya la hicimos varias veces (en algunos ejemplos de **altas**, **bajas** y **modificaciones**)*

```
program BusEnArc(input,output);
var a:file of integer; me,imin,imax,imedio,k,i:integer;
begin assign(a,'c:\ArPaBus.dat');reset(a);
      writeln('El archivo esta asi');
      while not eof(a) do begin read(a,k);
                              write(k:4)
                              end;reset(a);

      writeln;write('Ingrese el entero a buscar');readln(k);
      imin:=0;imax:=filesize(a)-1;imedio:=(imin+imax) div 2;
      seek(a,imedio);read(a,me);
      while (k<>me) and (imin<=imax) do
        begin if k<me then imax:=imedio-1
              else imin:=imedio+1;
              imedio:=(imin+imax) div 2;
              seek(a,imedio);read(a,me)
        end;
      if k=me then writeln('Esta en el registro',imedio:4)
        else writeln('No esta');
        write('Ingrese un entero ');readln(k)
end.
```

Intercalación de Archivos:

Vamos a ejemplificar sobre un ejercicio del examen del 07 / 12 / 2010.

Examen Algoritmos y Estructuras de Datos

Una empresa tiene dos sucursales. De cada sucursal dispone de un archivo **C:\SUCU1.DAT** y **C:\SUCU2.DAT**. Cada archivo contiene los datos de los artículos de esa sucursal, a saber:

- Número del artículo (entero)
- Nombre del artículo (20 caracteres)
- Precio unitario (real)
- Cantidad en stock (entero)

Artículos iguales tienen el mismo número, nombre y precio en ambas sucursales, pero pueden tener distinto stock. Puede haber artículos que figuren en el archivo de una sucursal y en el de otra no, y artículos que figuren en los archivos de ambas.

Los números de artículos no son consecutivos.

Ambos archivos están ordenados por número de artículo creciente.

Se pide diagrama de Chapin y programa Pascal para:

1) Generar un archivo **C: \ EMPRESA.DAT** que contenga los datos de todos los artículos, también ordenado por número de artículo creciente, un solo registro por artículo y stock totalizado.

Es decir que podríamos tener una situación como la siguiente:

<u>SUCU1.DAT</u>	<u>SUCU2.DAT</u>	<u>EMPRESA.DAT</u>
35 GOMAS 3.75 45	35 GOMAS 3.75 12	35 GOMAS 3.75 57
56 LAPICES 8.40 83	44 BROCHES 4.40 60	44 BROCHES 4.40 60
80 RESMAS 22.80 14	75 CLIPS 3.65 30	56 LAPICES 8.40 83
120 TINTA 9.90 8	80 RESMAS 22.80 33	75 CLIPS 3.65 30
.	80 RESMAS 22.80 47
.	120 TINTA 9.90 8
.

2) Ingresar por teclado un entero que será el porcentaje de descuento que se efectuará a los artículos cuyo stock sea superior a 50 unidades.

3) Modificar el archivo **C: \ EMPRESA.DAT** según el criterio enunciado en el punto anterior.

4) Listar **C: \ EMPRESA.DAT** completo.

NOTA: No se conoce la cantidad de registros de SUC1.DAT y SUC2.DAT pero no están vacíos

```

program ExaDic10(input,output);
type artic=record nro:integer;
                  nombre:string[10];
                  precio:real;
                  stock:integer
end;
  archi=file of artic;
var a,b,c:archi; rc,ra,rb:artic; n,m,i,j, t:integer;
begin assign(a,'c:\SUCU1.DAT');assign(b,'c:\SUCU2.DAT');
      assign(c,'c:\EMPRESA.DAT');rewrite(c);reset(a);reset(b);
      i:=1; j:=1; n:=filesize(a); m:=filesize(b); read(a,ra); read(b,rb);
      while (i<=n) and (j<=m) do
        begin
          if ra.nro<rb.nro then begin
            write(c,ra);
            if not eof(a) then read(a,ra);
            i:=i+1
          end;
          if rb.nro<ra.nro then begin
            write(c,rb);
            if not eof(b) then read(b,rb);
            j:=j+1
          end;
          if ra.nro=rb.nro then begin rc:=rb;

```

```

        rc.stock:=ra.stock+rc.stock;
        if not eof(a) then read(a,ra);
        if not eof(b) then read(b,rb);
        write(c,rc);
        i:=i+1;
        j:=j+1
    end
end;
if i>n then for t:=j to m do begin write(c,rb);
                                if not eof(b) then read(b,rb)
                                end
    else for t:=i to n do begin write(c,ra);
                                if not eof(a) then read(a,ra)
                                end;
reset(c); write('Ingrese porcentaje a rebajar:'); read(n);
while not eof(c) do begin read(c,rc);
                        if rc.stock>50 then begin rc.precio:=rc.precio*(1-n/100.0);
                                                seek(c,filepos(c)-1);
                                                write(c,rc)
                                                end
                        end;
writeln('El archivo EMPRESA.DAT quedo asi'); reset(c);
while not eof(c) do begin read(c,rc);
                        writeln(rc.nro:4,rc.nombre,rc.precio:6:2,rc.stock:6)
                        end;
close(a);close(b);close(c)
end.

```

Se ha tratado de asimilar al algoritmo de intercalación usado para arreglos. El alumno puede practicar cambiando los ($i \leq n$), etc. por $\text{not eof}(a)$, etc y también cambiando algunos while o for por repeat.

A continuación vamos a resolver un ejemplo similar (el examen del 07/12/11), pero en vez de resolverlo como el anterior haciendo una analogía con intercalación de archivos, lo resolveremos trabajando directamente con $\text{eof}()$, etc. Trabajando de esta manera, hay que tomar algunas precauciones extra, ya que podría omitirse garabar el o los últimos registros de los archivos, etc.

Algoritmos y Estructuras de Datos – Examen – 07 de Diciembre de 2.011 .-

Una Facultad tiene registrado (entre otros archivos de datos) las Notas de los exámenes de sus alumnos. Por cada alumno que rindió un examen se guardó:

Legajo del alumno: entero / **Código de la asignatura:** string[6] / **Nota:** real
(también tiene otros archivos: alumnos, materias, etc. pero no son necesarios para este problema)

Los datos antes mencionados están guardados en dos archivos:

Los exámenes del Ciclo Básico en **c:\basico.dat**

Los exámenes del Ciclo Profesional en **c:\profesio.dat**

Un mismo alumno puede tener varios registros en ambos archivos. Ambos archivos están ordenados por Legajo del alumno, en forma no-decreciente (por lo tanto todos los registros de un mismo alumno estarán en posiciones adyacentes).

Existe además un archivo, ya creado, pero vacío **c:\juntos.dat**.

Se pide **Chapin** y programa **Pascal** para:

- 1- que **c:\juntos.dat** contenga los datos de **c:\basico.dat** y **c:\profesio.dat** también ordenado por Legajo creciente. (si, por ejemplo un alumno tiene 4 registros en **c:\basico.dat** y 2 registros en **c:\profesio.dat**, deberá tener 6 en **c:\juntos.dat**). Listarlo completo (todos los registros).
- 2- Hacer un listado del archivo **c:\juntos.dat** que contenga Legajo y Promedio de cada alumno, una sola línea por alumno. Por promedio se entiende la suma de las Notas del alumno dividida por la cantidad de Materias rendidas por ese alumno.

Por ejemplo, suponiendo que los archivos contienen:

basica.dat						profesio.dat		
17	B111	6.0	←	1er. reg.	→	17	P043	6.0
17	B112	7.5	...	2do. reg.	...	22	P100	9.0
45	B111	6.5	...	3er. reg.	...	22	P103	8.5
64	B064	8.5	...	4to. reg.	...	64	P053	8.5
74	B222	4.0	←	5to. reg.	→	64	P031	6.0

La salida debería ser algo así

Listado de juntos.dat

```
-----
17    B111    6.0
17    B112    6.0
17    P043    7.5
22    P100    9.0
22    P103    8.5
45    B111    6.5
64    B064    8.5
64    P053    6.0
64    P031    8.5
74    B222    4.0
```

no deben preocuparse por el orden en que
aparecen los registros para legajos iguales
lo importante es que sea por legajo no-decreciente

Listado de promedios

```
-----
17    6.50
22    8.75
45    6.50
64    7.67
74    4.00
```

```
program ExaDic11(input,output);
type alu=record lega:integer;
               materia:string[6];
               nota: real
           end;
   facu=file of alu;
var a,b,c:facu; rc,ra,rb,aux:alu; legant,sulega,n,m,i,j,t:integer;
    prome, suma: real; puseldea, puseldeb: boolean;
begin assign(a,'c:\basica.dat');assign(b,'c:\profesio');
      assign(c,'c:\juntos.dat');rewrite(c);rewrite(a);rewrite(b);
      aux.lega:=17;
      aux.materia:='B111';
      aux.nota:=6.0;
      write(a,aux);
      aux.lega:=17;
      aux.materia:='B112';
      aux.nota:=7.5;
      write(a,aux);
      aux.lega:=45;
      aux.materia:='B111';
      aux.nota:=6.5;
      write(a,aux);
      aux.lega:=64;
      aux.materia:='B064';
      aux.nota:=8.5;
```

```

write(a,aux);
aux.lega:=74;
aux.materia:='B064';
aux.nota:=8.5;
write(a,aux);
aux.lega:=74;
aux.materia:='B068';
aux.nota:=8.5;
write(a,aux);
aux.lega:=88;
aux.materia:='B033';
aux.nota:=8.5;
write(a,aux);
aux.lega:=17;
aux.materia:='P043';
aux.nota:=6.0;
write(b,aux);
aux.lega:=22;
aux.materia:='P100';
aux.nota:=9.0;
write(b,aux);
aux.lega:=22;
aux.materia:='P103';
aux.nota:=8.5;
write(b,aux);
aux.lega:=64;
aux.materia:='P053';
aux.nota:=8.5;
write(b,aux);
aux.lega:=64;
aux.materia:='P031';
aux.nota:=6.0;
write(b,aux);
reset(a); reset(b);
writeln('Listado del archivo basica.dat');
while not eof(a) do begin read(a,rc);
                        writeln(rc.lega,' ',rc.materia,' ',rc.nota:6:2)
                        end;
reset(c); writeln('Listado del archivo profesio.dat');
while not eof(b) do begin read(b,rc);
                        writeln(rc.lega,' ',rc.materia,' ',rc.nota:6:2)
                        end;
write('Pulse...'); readln;
reset(a); reset(b); read(a,ra); read(b,rb);
puseldea:=false; puseldeb:=false;
while not ((eof(a) and puseldea) or (eof(b) and puseldeb)) do
    if ra.lega<=rb.lega then begin
        sulega:=ra.lega;
        write(c,ra);
        puseldea:=true;
        puseldeb:=false;
        read(a,ra);
        if eof(a) then
            if ra.lega=sulega then write(c,ra)
        end
    else begin
        sulega:=rb.lega;
        write(c,rb);
        puseldeb:=true;
        puseldea:=false;
        read(b,rb);
        if eof(b) then
            if rb.lega=sulega then write(c,rb)
        end;
    if eof(a) and puseldea then write(c,rb);

```

hemos puesto las sentencias sombreadas para generar nuevamente los archivos cada vez que se corre el programa, por si desean probar diferentes soluciones. también habría que cambiar rewrite por reset

hemos agregado mas registros que los que figuran en el ejemplo del enunciado

```

if eof(b) and puseldeb then write(c,ra);
if eof(a) and puseldea then begin
    while not eof(b) do begin
        read(b,rb);
        write(c,rb)
    end
end
else begin
    while not eof(a) do begin
        read(a,ra);
        write(c,ra)
    end
end;
reset(c); writeln('Listado del archivo juntos.dat');
while not eof(c) do begin read(c,rc);
    writeln(rc.lega,' ', rc.materia, rc.nota:6:2)
end;
reset(c); write('Pulse...'); readln;
read(c,rc); legant:= rc.lega; suma:= rc.nota; n:= 1;
while not eof(c) do begin read(c,rc);
    if rc.lega<>legant then begin
        prome:= suma/n;
        n:= 1;
        writeln(legant,prome:6:2);
        suma:= rc.nota;
        legant:= rc.lega
    end
else
    begin n:= n+1;
        suma:= suma+rc.nota
    end
end;
prome:= suma/n; writeln(legant, prome:6:2);
close(a);close(b);close(c);write('Pulse...'); readln
end.

```

Los archivos también pueden ser parámetros de funciones. Supongamos que tenemos dos archivos del mismo tipo, **curso1.dat** y **curso2.dat**, y que en ambos deseamos calcular el promedio de las notas de los alumnos, es decir la suma de las notas de **curso1.dat** dividido por la cantidad de alumnos, y la suma de las notas de **curso2.dat** dividido por la cantidad de alumnos. Vamos a hacer una única función que calcule ese promedio para un archivo parámetro formal **x**, y después usamos como parámetros actuales los archivos mencionados.

```

program ArParFun(input, output);
type alu=record nom: string[4];
    nota: integer
end;
archi= file of alu;
var aux: alu; a, b:archi; m: real;
function prome(var x: archi):real;
var w:alu; t:real;
begin reset(x); t:= 0;
    while not eof(x) do begin read(x, w);
        t:= t + w.nota
    end;
    prome:= t / filesize(x)
end;
begin assign(a,'c:\aldo\curso1.dat');

```



```

assign(b, 'c:\aldo\curso2.dat');
m:= prome(a); writeln('Curso1 tiene promedio', m:6:2);
m:= prome(b); writeln('Curso2 tiene promedio', m:6:2);
close(a); close(b); readln
end.

```

Notarán que hemos puesto **var** delante del parámetro formal **x** de la función, esto lo exige Pascal, y es lógico, ya que los archivos son en general de gran tamaño y no sería práctico (o sería imposible) trabajarlos como parámetros por valor, es decir haciendo una copia de ellos en memoria.

Para terminar, veamos un ejercicio mas sobre archivos

El objeto del siguiente ejemplo es simplemente que tengan mas ejercicios sobre archivos y presentar una forma de interactuar con el sistema operativo desde un programa Pascal.

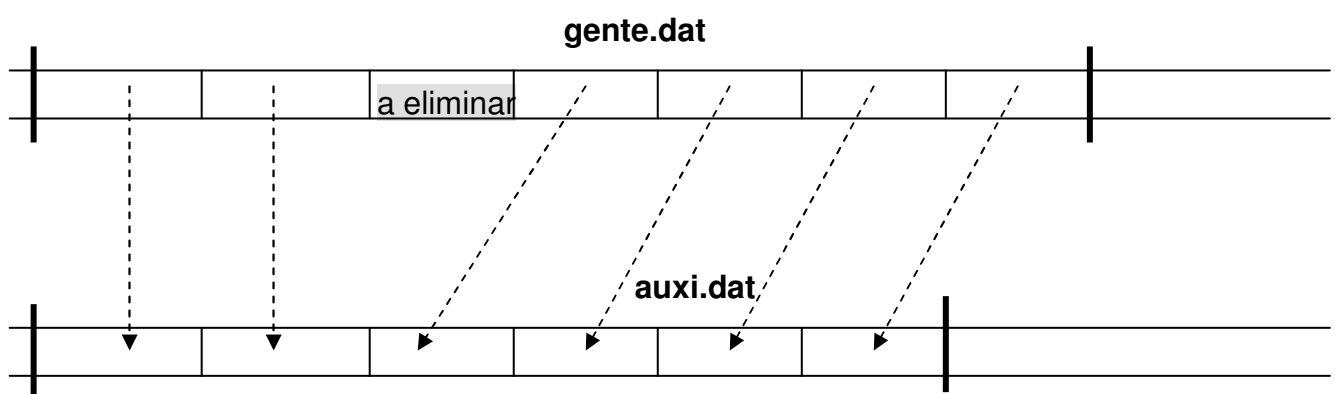
¿Cómo podemos hacer para dar una baja física?

Vamos a ejemplificar como dar de baja (eliminar, en caso de baja física) a **un** registro. Es decir queremos que el registro a dar de baja “desaparezca” del archivo y que por lo tanto el archivo quede “**un registro mas corto**”.

Vamos a programar el siguiente método.

Supongamos que nuestro archivo se llama **gente.dat** y queremos eliminar un registro. Creamos un archivo vacío (o hacemos **rewrite**), **auxi.dat** y copiamos en él uno a uno los registros de **gente.dat** excepto el que queremos dar de baja.

Entonces va a quedar como pretendemos que quede nuestro archivo al dar una baja física



Pero ahora tenemos un problema, nuestro archivo se llama **auxi.dat**, pero debería llamarse **gente.dat**.

Borramos entonces nuestro **gente.dat** y luego a **auxi.dat** lo renombramos como **gente.dat**.

Esto lo podríamos conseguir saliendo del Pascal y luego, desde Windows hacemos el borrado y el renombrado que mencionamos antes.

Sin embargo es mas seguro hacerlo desde el propio programa Pascal. Esta interacción con los sistemas operativos, se puede hacer desde cualquier lenguaje, en particular desde Pascal, tenemos dos instrucciones, **erase** y **rename**, de muy fácil comprensión y que se incluyen directamente en el programa siguiente.

En el programa, se tratará de dar de baja a un registro, al que identificaremos por el campo **nro**, el cual se ingresará por teclado.

En realidad, el archivo **gente.dat**, debería existir, por lo que usamos **reset (a)**.

Sin embargo en el programa hemos puesto las sentencias sombreadas para generarlo cada vez que se corra el programa por si desean hacerle algunas mejoras, como por ejemplo:

- informar los datos del registro a borrar para pedir confirmación de borrado (en el ejemplo se informa que fue borrado)
- avisar si no se encontró
- etc...

```

program bajafisi(input, output);
type gistro= record nro: integer;
                  nombre: string[4]
                end;
  chivo= file of gistro;
var abajar, p: integer; a, b, z: chivo; c: gistro;
begin assign(a, 'c:\gente.dat');
      assign(b, 'c:\auxi.dat');
      rewrite(a); rewrite(b);
      c.nro:=111; c.nombre:='AAAA'; write(a,c);
      c.nro:=222; c.nombre:='BBBB'; write(a,c);
      c.nro:=333; c.nombre:='CCCC'; write(a,c);
      c.nro:=444; c.nombre:='DDDD'; write(a,c);
      c.nro:=555; c.nombre:='EEEE'; write(a,c);
      c.nro:=666; c.nombre:='FFFF'; write(a,c);
      reset(a);
      writeln('El archivo contiene los siguientes registros');
      while not eof(a) do begin read(a,c);
                              writeln(c.nro, ' ', c.nombre)
                            end;
      write('Indique el nro. de la persona a dar de baja :');
      readln(abajar); reset(a);
      while not eof(a) do begin read(a,c);
                              if c.nro=abajar then
                                  writeln('Se da de baja a ', c.nombre)
                              else write(b,c)
                              end;
      close(a); close(b);
      erase(a); rename(b, 'c:\gente.dat');
      assign(z, 'c:\gente.dat'); reset(z); writeln('Y ahora quedo asi');
      while not eof(z) do begin read(z,c);
                              writeln(c.nro, ' ', c.nombre)
                            end;
      write('Ingrese un entero '); read(p); close(z)
end.

```