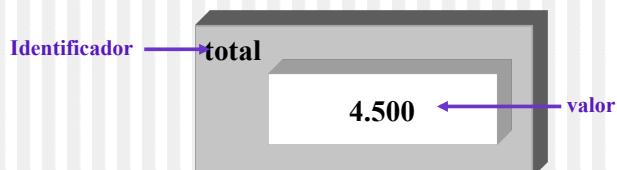


Programación I

Tema 2: Conceptos Básicos de la programación

Variable

- Especie de caja o celda (de memoria), que se distingue por un nombre (**identificador**), en la que se almacena un dato que puede ser modificado.

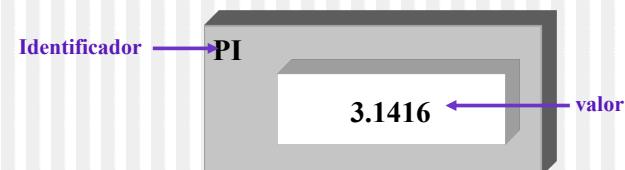


Contenido

- Representación de la información
 - Variables y constantes
 - Tipos de datos
- Instrucciones Básicas. Instrucciones de asignación
- Expresiones
- Estructuras de control
 - Estructura condicional
 - Iteración. Concepto de invariante
 - Utilización de subprogramas
- Escritura de algoritmos. Esquemas básicos
 - Recorrido de una secuencia
 - Búsqueda de un elemento en una secuencia

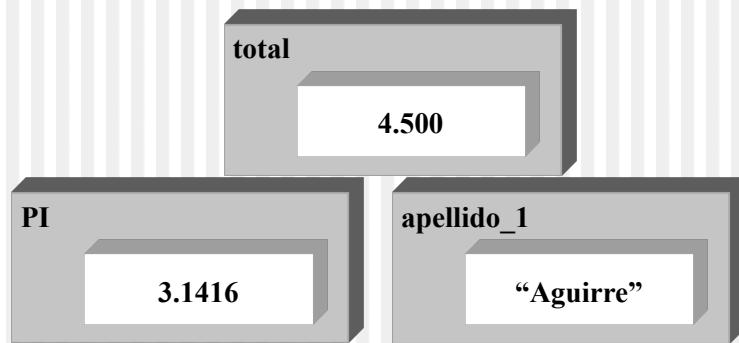
Constante

- Especie de caja o celda (de memoria), que se distingue por un nombre (**identificador**), en la que se almacena un dato que **NO** puede ser modificado. Las constantes deben ser inicializadas en la declaración.



Identificador

- Nombre para distintos elementos de un programa (variables, constantes,...). Cada dato debe tener un identificador distinto



Tipos de datos básicos

- Números enteros (Integer)
 - 4500 -15
- Números reales (Float)
 - 3.1416 9.2E-6
- Carácter (Character)
 - ‘a’ ‘&’
- Booleano (Boolean)
 - TRUE FALSE
- Cadena de caracteres (STRING)
 - “Ander” “Hola”

Tipo de datos entero

- Valores de tipo entero
- Dominio: ..., -3, -2, 1, 0, +1, +2, +3, ...
- Debido a la representación binaria, tiene límites inferior y superior
 - 16 Bits: [-32768, +32767]
 - 32 Bits: [-2.147.483.648, -2.147.483.647]
- Operaciones:
 - Operando y resultado de tipo entero
 - Operadores diádicos
 - Suma, resta, multiplicación, división, resto, (potencia)
 - Operadores monádicos
 - Valor absoluto, negación
- Operadores relacionales
 - Resultado booleano
 - Menor, menor o igual, mayor, mayor o igual, igual, distinto

Tipo de datos real

- Valores de tipo real
- Dominio
 - 0.0 1.5 386473.0 3.141
 - 3.86e5 3.0e-8 0.12e-20
- Operaciones
 - Operando y resultado de tipo real
 - Operadores diádicos
 - Suma, resta, multiplicación, división, (potencia con exponente entero)
 - Operadores monádicos
 - Valor absoluto, negación
- Operadores relacionales
 - Resultado booleano
 - Menor, menor o igual, mayor, mayor o igual, igual, distinto

Tipo de datos booleano

- Dominio
 - TRUE (cierto)
 - FALSE (falso)
- Operadores relacionales
 - Resultado
 - Igual, distinto
- Operadores lógicos
 - Operandos y resultado
 - And (y), Or (o), Xor (o exclusivo), Not (negación)

Tipo de datos carácter

- Símbolos
- **Caracteres entre comilla simple**
- Dominio
 - Caracteres gráficos: ‘a’ ‘A’ ‘3’ ‘@’
 - Caracteres de control: ff cr lf
- Operaciones relacionales
 - Menor, menor o igual, mayor, mayor o igual, igual, distinto
- **Cuidado!!! No es lo mismo el valor 3 que el símbolo ‘3’**

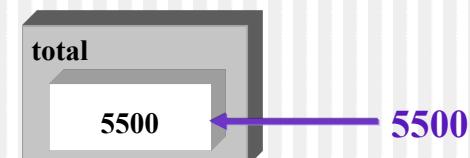
Tipo de datos cadena

- Cadena de caracteres
- Domino
 - **Cadenas de caracteres entre comillas dobles**
 - “Maite”
 - “Kale nagusia 12, 3C”
- **Cuidado!!! “3” 3 y ‘3’ son valores diferentes**
- Operaciones relacionales
 - Menor, menor o igual, mayor, mayor o igual, igual, distinto

Instrucción de Asignación

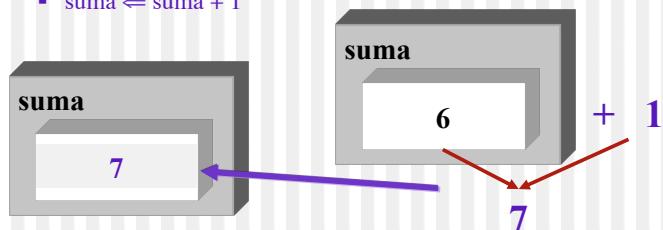
- Es una operación que permite cambiar el contenido de una **variable**
 - Graba un nuevo valor eliminando el anterior

total ⇐ 5500



Expresión

- El valor asignado a una variable puede consistir en el resultado de evaluar una expresión
 - `areaRectangulo` \Leftarrow altura * anchura
- En una expresión puede participar la propia variable a la que vamos a grabar el resultado
 - `suma` \Leftarrow suma + 1

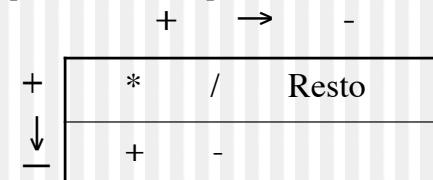


Inicialización de variables

- Hasta que no se asigna un valor a una variable, ésta se encuentra **indefinida** (contiene algún valor “basura” impredecible).
- **Toda variable debe ser inicializada antes de consultar su valor**
 - Asignación de valor

Operadores aritméticos

- $+,-,*,/ Resto$
- Ambigüedad: Varias posibles interpretaciones
(Ej: $x \Leftarrow a + b * c$)
 - $x \Leftarrow (a + b) * c$
 - $x \Leftarrow a + (b * c)$
- Reglas de precedencia de operadores



Concordancia de tipos

- Toda variable y constante se declara de un determinado tipo (numérico, carácter, booleano, ...)
- Cada literal tiene implícitamente un tipo
 - `2` \rightarrow numérico
 - “Comisión de apertura” \rightarrow cadena de caracteres

Concordancia de tipos: Ejemplos de errores de concordancia

- Suponiendo que la variable *Número* ha sido declarada de tipo numérico, **las siguientes asignaciones producen errores:**
 - Número \Leftarrow 'a'
 - Número \Leftarrow TRUE
 - Número \Leftarrow "Comisión de apertura"
- Suponiendo que *Nombre* es una variable de tipo cadena y *Encontrado* de tipo booleano, **las siguientes asignaciones son erróneas:**
 - Número \Leftarrow Nombre
 - Nombre \Leftarrow Número
 - Número \Leftarrow Número + Nombre
 - Nombre \Leftarrow Encontrado

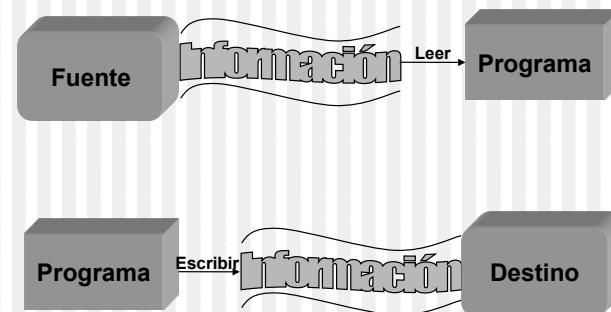
Conversión de tipos (Casting)

- Permite convertir el valor de un tipo (ej. numérico real) a otro tipo (ej: numérico entero)
 - Suponiendo que *nReal* es un real y *nEntero* un entero, *nReal* \Leftarrow (Real) *nEntero*
- **No se puede realizar la conversión entre todos los tipos**
 - *vCadena* \Leftarrow "Cadena"
 - *nReal* \Leftarrow (Entero) *vCadena*
- Algunos lenguajes realizan las conversiones de manera implícita

Instrucciones de Entrada/Salida

- Sirven para tomar valores del “exterior” o devolver valores
- La operación de *lectura* obtiene valores (ej: de un diskete o del teclado)
- Cada vez que se lee un dato se “consume” y la siguiente lectura lee el siguiente dato de la entrada
- Se dispone de una operación para saber si quedan todavía datos por leer
- La operación de *escritura* envía valores al “exterior” (ej: diskete o pantalla)

Instrucciones de Entrada/Salida



Instrucciones de Entrada/Salida

- Abrir el dispositivo/fichero en el que se pretende escribir o desde el que se quiere leer
 - Entrada estándar (teclado) y salida estándar (monitor) **están abiertos**
 - Abrir recurso lectura
 - Abrir recurso escritura [true/false]
 - El último parámetro indica si se añade la información al final o se sobrescribe la información anterior
- Leer un dato
 - leeEntero, leeReal, leeBooleano, leeCadena, leeCaracter
- Escribe un dato
 - escribe

Estructuras de control

- Sirven para determinar el orden de ejecución de las instrucciones
- En ausencia de una estructura de control, las instrucciones se ejecutan en **secuencia** (en el mismo orden en el que están escritas)
 - Num1 ← LeeEntero
 - Num2 ← LeeEntero
 - Escribe "la suma de " Num1 " y " Num2 " es " Num1 + Num2
 - Escribe "y la resta es " Num1 - Num2
- ¿Cómo se puede escribir el máximo de Num1 y Num2?

Estructuras de control

- Las estructuras de control **se basan en condiciones** expresadas mediante operadores relacionales y/o lógicos
- Según se evalúe la condición a cierto o falso, se ejecutará o no una determinada instrucción o grupo de instrucciones

Tipos de condiciones

- Variables booleanas
 - Almacenan un valor booleano (cierto, falso)
- Expresiones con operadores relacionales
- Varias expresiones booleanas combinadas con operadores booleanos

Operadores relacionales

- Devuelven cierto o falso (TRUE|FALSE)
- Sirven para comparar dos elementos (generalmente de tipo numérico)
 - Menor (<)
 - Mayor (>)
 - Menor o igual (<=)
 - Mayor o igual (>=)
 - Igual (==)
 - Distinto (!=)
- Hay que tener en cuenta la concordancia de tipos
- Las comparaciones de booleanos se reducen a *igual* (==) y *distinto* (!=)



Operadores booleanos

- **Or:** la condición combinada es cierta si alguna de las dos subcondiciones es cierta. La condición es falsa sólo si ambas subcondiciones son falsas.

		Subcond1	
OR		T	F
Subcond2	T	T	T
	F	T	F

Operadores booleanos

- **And:** la condición combinada es cierta si y sólo si las dos subcondiciones son ciertas. Basta que una de las condiciones sea falsa para que la combinación con **and** sea falsa.

		Subcond1	
AND		T	F
Subcond2	T	T	F
	F	F	F

Operadores booleanos

- **Not:** El operador **Not** niega el valor de la subcondición.

		Subcond	
NOT		T	F
Subcond	T	F	T
	F	T	F

Operadores booleanos

- **Xor:** la condición combinada es cierta si y sólo una de las dos subcondiciones es cierta. Si ambas subcondiciones son ciertas o falsas la combinación con **xor** es falsa.

		Subcond1	
XOR		T	F
Subcond2	T	F	T
	F	T	F

Operadores booleanos

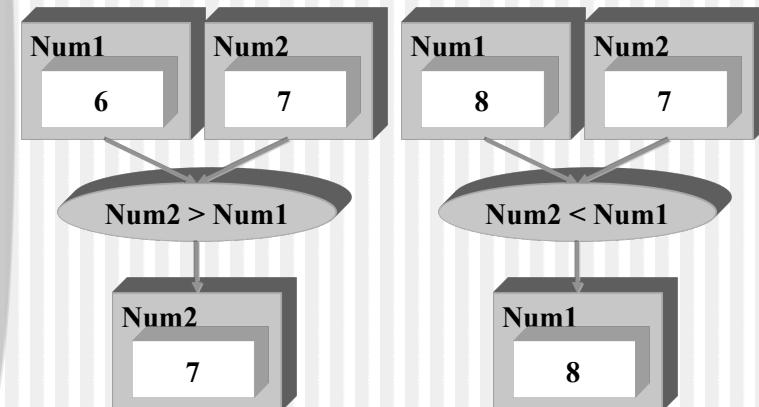
- Precedencia de operadores
 - Not
 - And
 - Or
 - Xor
- Paréntesis

Leyes de “de Morgan”

- Negación de una condición compuesta por el operador **AND**
 - $\text{Not}(\text{A and B}) \iff (\text{Not A}) \text{ or } (\text{Not B})$
- Negación de una condición compuesta por el operador **OR**
 - $\text{Not}(\text{A or B}) \iff (\text{Not A}) \text{ and } (\text{Not B})$

Estructuras condicionales

- Devolver el mayor valor de *Num1* y *Num2*



Estructuras condicionales

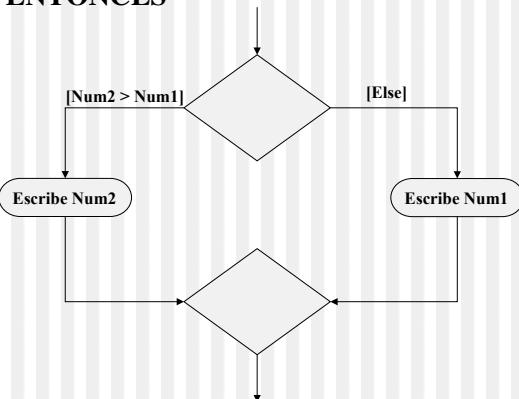
SI Num2 > Num1 **ENTONCES**

Escribe Num2

SI NO

Escribe Num1

FIN SI



Estructuras condicionales

SI condición **ENTONCES**

 grupo_de_instrucciones_1

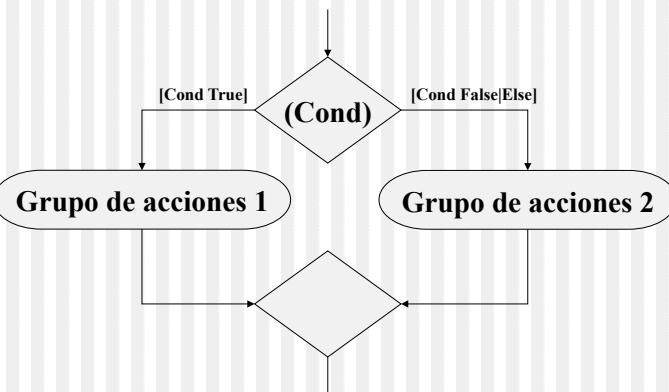
SI NO

 grupo_de_instrucciones_2

FIN SI

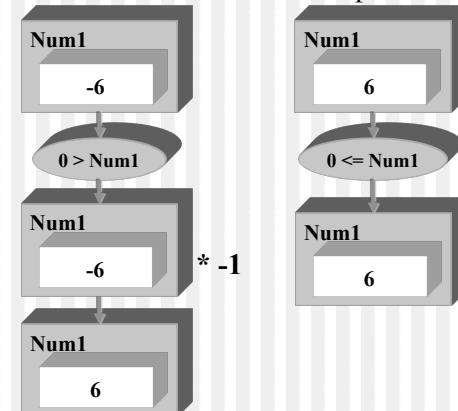
Hay que indicar el punto final de la estructura condicional (**FIN SI**) para delimitar el grupo de instrucciones

Estructuras condicionales



Estructuras condicionales

- Convertir el valor de una variable en positivo



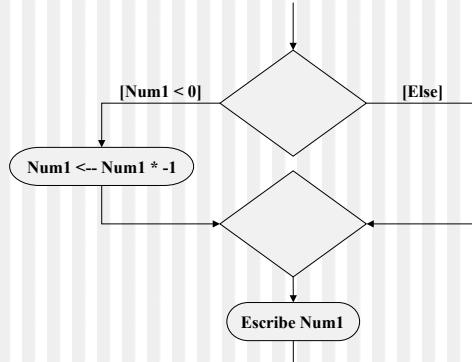
Estructuras condicionales

SI Num1 < 0 **ENTONCES**

 Num1 <-= Num1 * -1

FIN SI

Escribe Num1



Estructuras condicionales

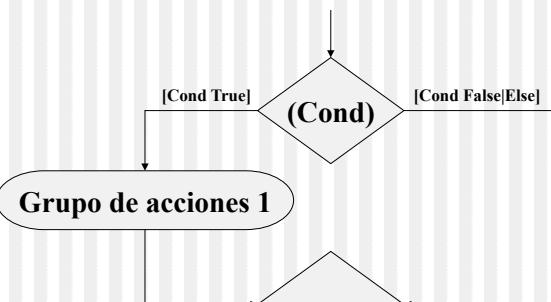
SI condición **ENTONCES**

 grupo_de_instrucciones_1

FIN SI

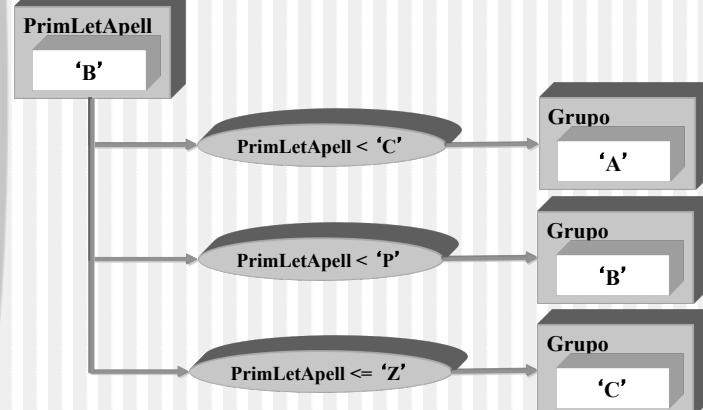
Hay que indicar el punto final de la estructura condicional (**FIN SI**) para delimitar el grupo de instrucciones

Estructuras condicionales



Estructuras condicionales

- Clasificar en función del valor de la variable



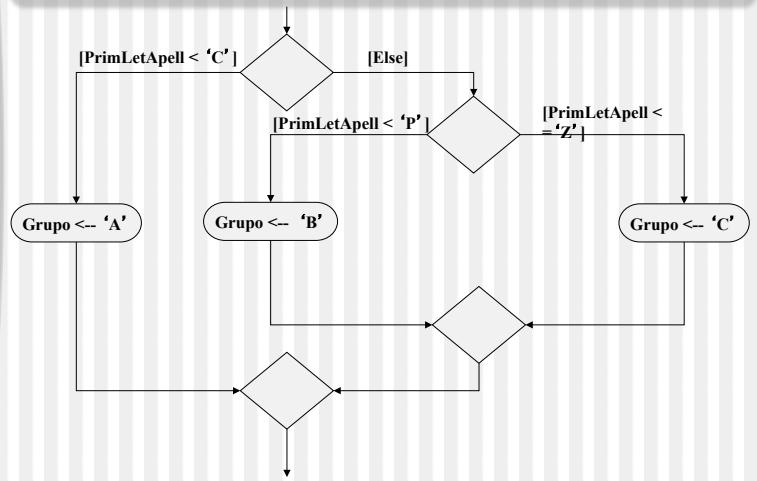
Estructuras condicionales

```
SI PrimLetApell < 'C' ENTONCES  
    Grupo <- 'A'  
SI NO SI PrimLetApell < 'P' ENTONCES  
    Grupo <- 'B'  
SI NO SI PrimLetApell <= 'Z' ENTONCES  
    Grupo <- 'C'  
FIN SI
```

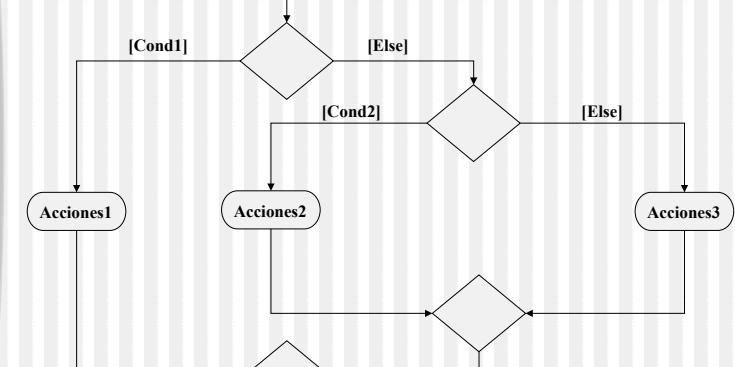
Estructuras condicionales

```
SI condición ENTONCES  
    grupo_de_instrucciones_1  
SI NO SI condición2 ENTONCES  
    grupo_de_instrucciones_2  
....  
[SI NO  
    grupo_de_instrucciones_n]  
FIN SI
```

Estructuras condicionales

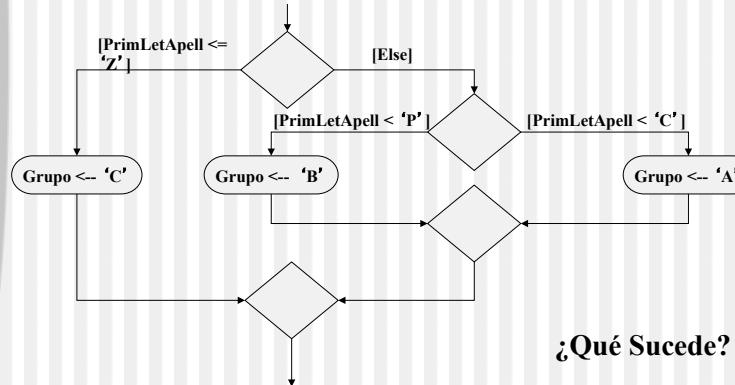


Estructuras condicionales

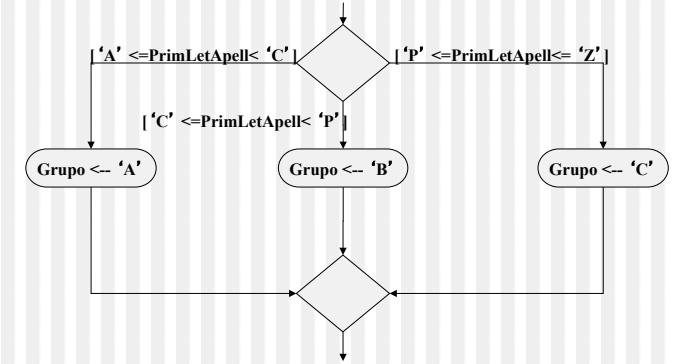


Estructuras condicionales

- Estructura de multiples condiciones
 - Cuidado con el orden de las condiciones!!!!



Estructuras condicionales

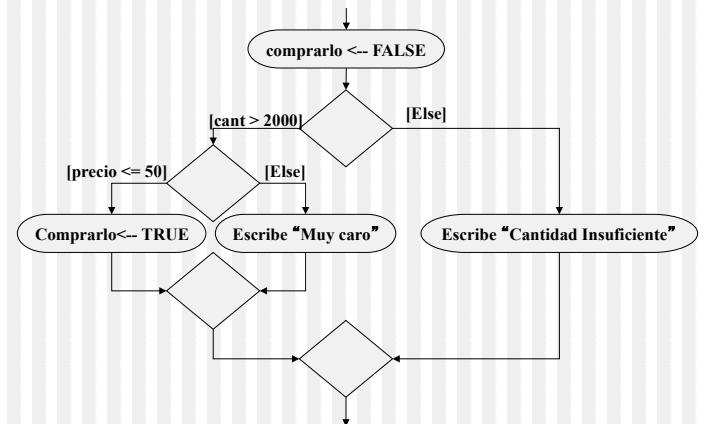


Estructuras condicionales

```
SI VALOR expresión ES  
VALORES Rango_valores  
Acciones  
...  
POR DEFECTO  
Acciones  
FIN SI
```

Estructuras condicionales

- Condiciones anidadas



Estructuras condicionales

- Condiciones anidadas

Comprarlo <-- FALSE

SI cant > 2000 **ENTONCES**

SI precio <= 50 **ENTONCES**

 comprarlo <- TRUE

SI NO

 Escribe "Muy caro"

FIN SI

SI NO

 Escribe "cantidad insuficiente"

FIN SI

Estructuras iterativas

- Con las instrucciones que hemos visto hasta el momento, las acciones identificadas se ejecutan 1 única vez o no se ejecutan si no se cumple la condición de control

- *¿Cómo calcular la nota media de una asignatura?*

- **PROBLEMA:** *¿de cuántos alumnos?*

Estructuras iterativas

- **Bucle o ciclo:** método de estructurar instrucciones de forma que éstas **se repitan** una serie finita o infinita de veces
- **Los bucles también se controlan mediante expresiones booleanas**
- **¡¡¡Cuidado con los bucles infinitos!!!**
 - Programas que nunca finalizan

Estructuras iterativas ("Repetir")

cont <-- 0

REPETIR

SALIR SI cont >= numAlum

 notaAlum <- LeeEntero

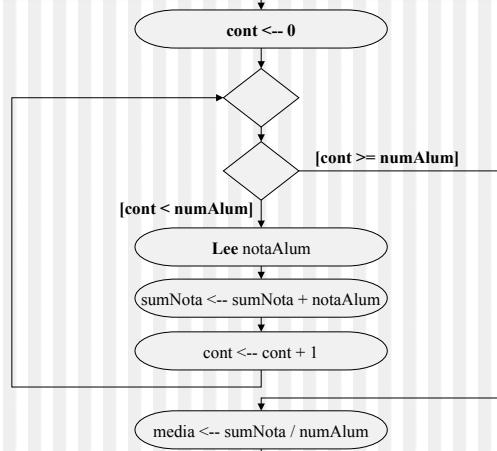
 sumNota <- sumNota + notaAlum

 cont <- cont + 1

FIN REPETIR

 media <- sumNota / numAlum

Estructuras iterativas ("Repetir")

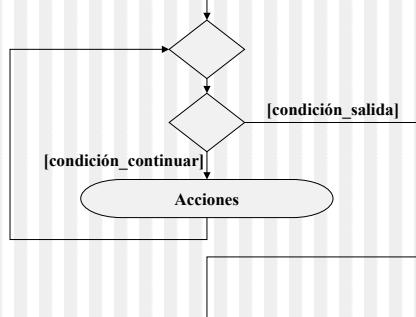


Estructuras iterativas

REPETIR MIENTRAS condición_continuar

Acciones

FIN REPETIR



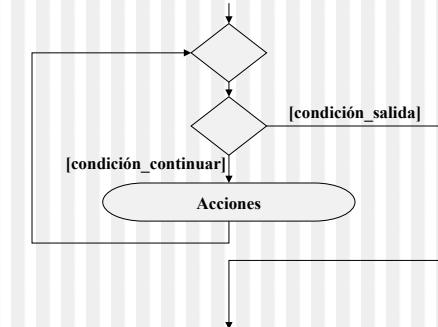
Estructuras iterativas

REPETIR

SALIR SI condición_salida

Acciones

FIN REPETIR



Estructuras iterativas

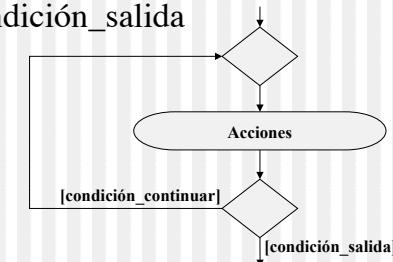
Las acciones se ejecutan al menos una vez

REPETIR

Acciones

SALIR SI condición_salida

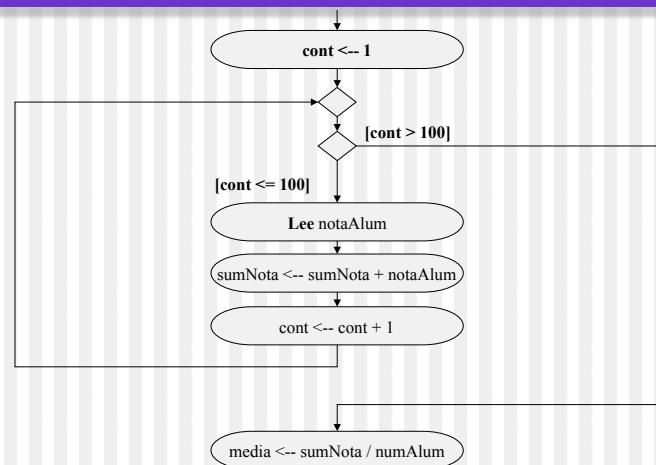
FIN REPETIR



Estructuras iterativas (“Para cada”)

- Cuando se conoce el número de veces que se ejecutará la repetición
- Ejemplos:
 - Calcular la nota media de una asignatura de 100 alumnos
 - Escribir los números entre 1 y 100 que son divisibles por 2 y por 5

Estructuras iterativas (“Para cada”)



Estructuras iterativas (“Para cada”)

PARA CADA cont **ENTRE** 1..100

```
notaAlum ← leeEntero  
sumNota ← sumNota + notaAlum
```

FIN PARA CADA

```
media ← sumNota / numAlum
```

El valor de *cont* se incrementa automáticamente

Estructuras iterativas (“Para cada”)

- Cuando se conoce el número de veces que se ejecutará la repetición
- Ejemplos:
 - Escribir los números entre 1 y 100 que son divisibles por 2 y por 5

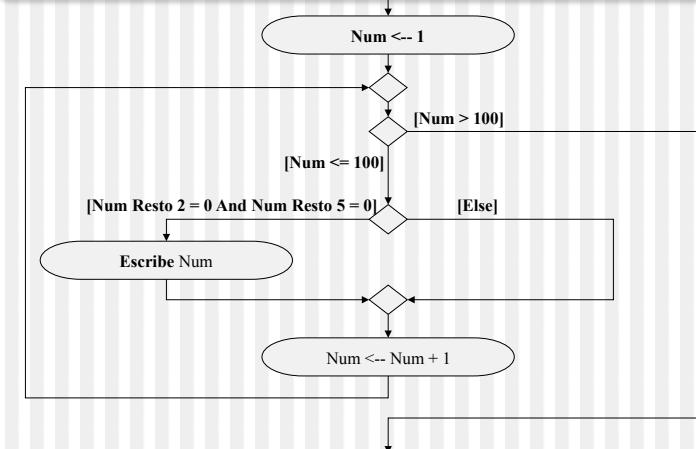
Estructuras iterativas (“Para cada”)

```
PARA CADA Num ENTRE 1..100
  SI Num Resto 2 = 0 AND Num Resto 5 = 0 ENTONCES
    Escribe Num
  FIN SI
FIN PARA CADA
```

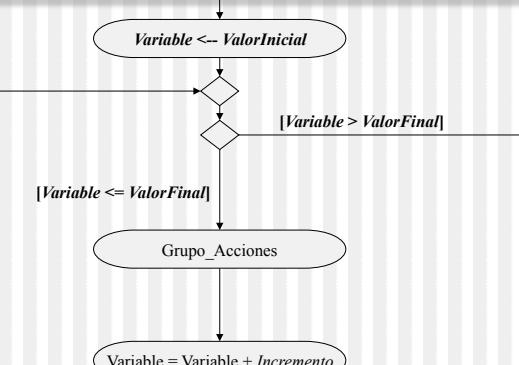
Estructuras iterativas (“Para cada”)

```
PARA CADA Variable ENTRE ValorInicial Y ValorFinal
  (CON INCREMENTO Incremento)
  Grupo_de_acciones
FIN PARA CADA
```

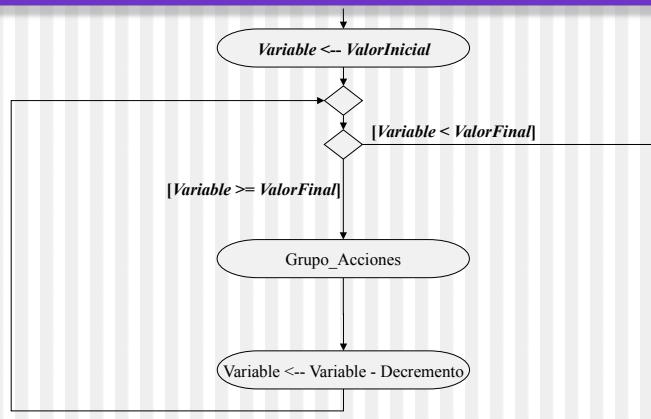
Estructuras iterativas (“Para cada”)



Estructuras iterativas (“Para cada”)



Estructuras iterativas (“Para cada”)



Estructuras iterativas

• “Para cada” vs “Repetir”

- “Repetir” es más general
- Todo lo que se puede hacer con “Para cada” se puede hacer con “Repetir”
- **No** todo lo que se puede hacer con “Repetir” se puede hacer con “Para cada”

Construcción de estructuras iterativas

Tareas:

- A. Flujo de control
 1. Condición de parada
 2. Inicialización de la condición
 3. Modificación de la condición
- B. Proceso que realiza el bucle
 1. Cuál es el proceso a repetir
 2. Inicialización del proceso
- C. Estado de salida del bucle

Ejemplo:

Diseña un programa que indique cuántos números, entre los 100 primeros de la secuencia de entrada, están entre 5 y 10 ambos inclusive

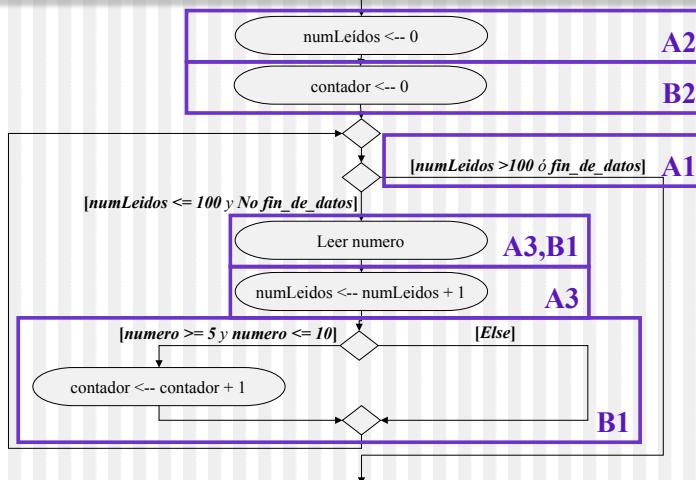
Ejemplo:

- **Flujo de control**
 - **Condición de parada:** Números leídos > 100 ó fin de datos
 - **Inicialización de la condición:** Números leídos <- 0
 - **Modificación de la condición:** Leer nuevo número, Números leídos <- Números leídos + 1
- **Proceso que realiza el bucle**
 - **Proceso a repetir:** Leer nuevo número, incrementar contador si número pertenece a [5,10]
 - **Inicialización del proceso:** Contador <- 0
- **Estado a la salida del bucle:**
 - El contador contiene el resultado

Ejemplo:

```
numLeidos <- 0          A2
contador <- 0           B2
REPETIR
    SALIR SI numLeidos > 100 ó fin de datos      A1
    Leer numero                         A3, B1
    numLeidos <- numLeidos + 1        A3
    SI numero >= 5 y numero <= 10 ENTONCES
        contador <- contador + 1      B1
    FIN SI
FIN REPETIR
```

Ejemplo:



Bucles anidados

- También se pueden anidar estructuras iterativas
- Ejemplo: Dada una serie de números leídos de la entrada estándar, escribir la tabla de multiplicar de cada uno de ellos.
 - Suponiendo que los números leídos son el 1 y el 3, la salida debe ser la siguiente:

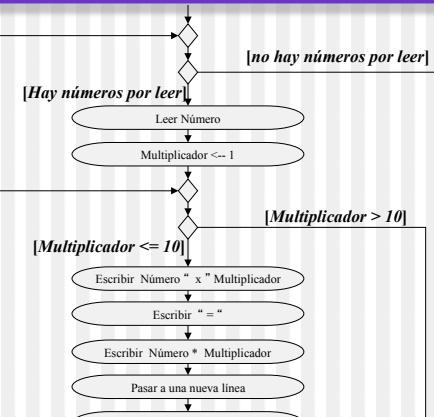
$$\begin{aligned} 1 \times 1 &= 1 \\ 1 \times 2 &= 2 \\ \dots \\ 3 \times 8 &= 24 \\ 3 \times 9 &= 27 \\ 3 \times 10 &= 30 \end{aligned}$$

Bucles anidados

REPETIR

```
SALIR SI no hay números por leer  
    Leer Número  
    PARA CADA Multiplicador ENTRE 1 Y 10  
        Escribir Número " x " Multiplicador  
        Escribir " = "  
        Escribir Número * Multiplicador  
        Pasar a nueva línea  
    FIN PARA CADA  
FIN REPETIR
```

Bucles anidados



Algoritmos. Esquemas básicos

- Introducción a los esquemas de programas
- Hay muchos problemas “parecidos”
 - Obtener la suma de una secuencia de enteros acabada en cero
 - Obtener el número de caracteres de una secuencia acabada en punto

Escritura de algoritmos: Esquemas básicos

- Esquemas de algoritmos para problemas “similares”
 - **Enumeración** de una secuencia: se deben recorrer todos los elementos de la secuencia.
 - Se conoce a priori el número de elementos de la secuencia.
Ej.: sumar $1 + 2 + \dots + N$
 - No se conoce el número de elementos de la secuencia.
 - **Secuencia con centinela:** Existe un elemento especial para indicar el final de la secuencia. **Ej.:** secuencia de enteros acabada en cero
 - **Secuencia sin centinela:** No existe un elemento especial para indicar el final de la secuencia. Se debe detectar el final de la secuencia de alguna otra forma.
 - **Búsqueda** en una secuencia: No siempre es necesario recorrer todos los elementos de la secuencia.

Esquemas básicos

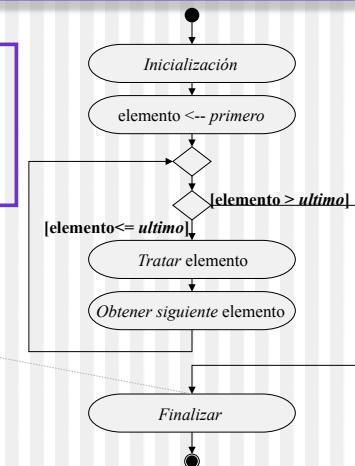
1. Esquema de enumeración en una secuencia cuyo número de elementos es conocido

- Condiciones de aplicación: se deben conocer el primer y el último valor de la secuencia a recorrer

Esquemas básicos

```
Inicialización  
PARA CADA elemento entre el primero y el último  
    Tratar elemento  
FIN PARA  
-- Se han tratado todos los elementos  
Finalización
```

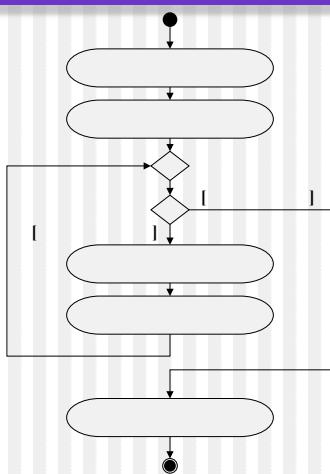
Finalización



Ejemplo:

```
PARA CADA _____ ENTRE _____ Y _____  
FIN PARA  
-- Se han tratado todos los elementos
```

Diseña un programa que, dado un valor entero positivo N, calcule El sumatorio de 1 a N



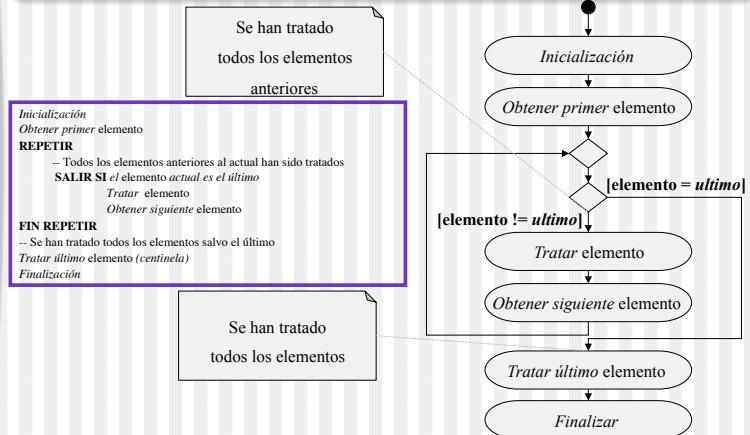
Esquemas básicos

2. Esquema de enumeración en una secuencia con centinela

Condiciones de aplicación:

- Secuencia con centinela
 - En la secuencia existe al menos un elemento (el centinela)
- Tratamiento especial del centinela
 - En general el tratamiento es vacío

Esquemas básicos



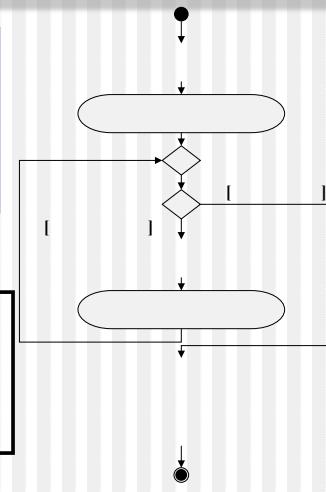
Ejemplo:

REPETIR:

- Todos los elementos anteriores al actual han sido tratados
- SALIR SI _____

FIN REPETIR:

- Se han tratado todos los elementos salvo el último



Diseña un programa que calcule la media de una secuencia de números enteros acabada en cero.

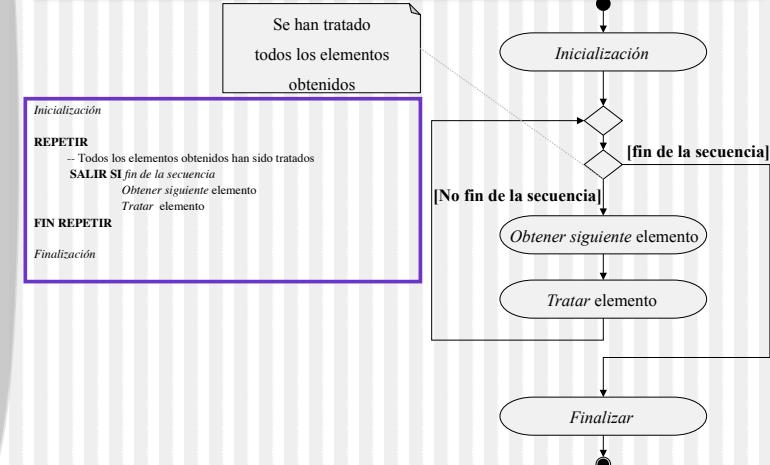
Esquemas básicos

3. Esquema de enumeración en una secuencia sin centinela

Condiciones de aplicación:

- Secuencia sin centinela. No hay elemento especial que marque el final de la secuencia
 - La secuencia puede ser vacía
 - Esquema típico para el tratamiento de ficheros
- No se conoce el número de elementos de la secuencia

Esquemas básicos



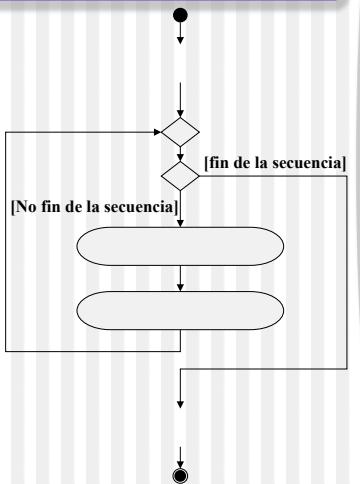
Ejemplo:

```

REPETIR
    -- Todos los elementos obtenidos han sido tratados
    SALIR SI fin de la secuencia
FIN REPETIR

```

Diseña un programa que calcule la media de una secuencia de números enteros.



Esquemas básicos

4. Esquema de búsqueda de un elemento en una secuencia

Condiciones de aplicación:

- Secuencia con centinela.
 - La secuencia contiene al menos un elemento (el centinela)

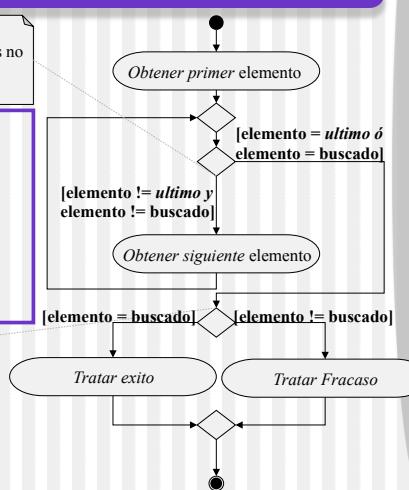
Esquemas básicos

```

Obtener primer elemento
REPETIR
    -- Todos los elementos anteriores al actual no son el buscado
    SALIR SI el elemento actual es el último ó es el buscado
    Obtener siguiente elemento
FIN REPETIR
    - Se ha llegado al último elemento ó se ha encontrado el buscado
    SI el elemento es el buscado ENTONCES
        Tratar búsqueda exitosa
    FIN SI
    SI NO
        Tratar el caso de elemento no encontrado
    FIN SI

```

Se ha llegado al último elemento ó se ha encontrado el buscado

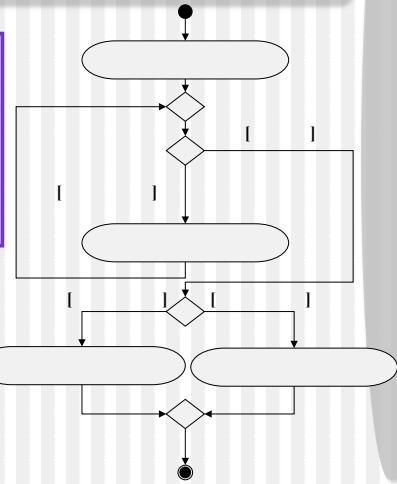


```

REPETIR
    -- Todos los elementos anteriores al actual no son el buscado
    SALIR SI
FIN REPETIR
-- Se ha llegado al último elemento ó se ha encontrado el buscado
SI _____ ENTONCES
SI NO
FIN SI

```

Diseña un programa que, dada una secuencia de enteros acabada en cero, encuentre el primer número par.

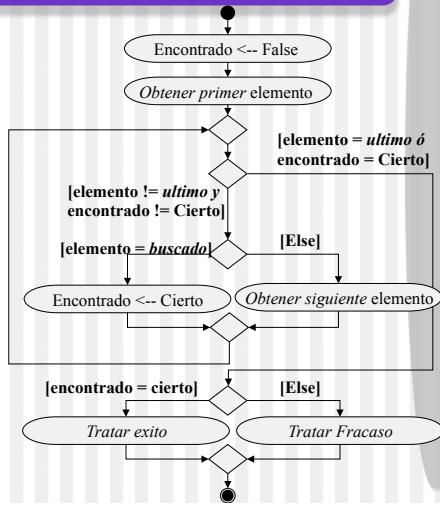


Esquemas básicos

```

Encontrado <- Falso
Obtener primer elemento
REPETIR
    -- Todos los elementos anteriores al actual no son el buscado
    SALIR SI el elemento actual es el último ó encontrado = Ciento
        SI el elemento es el buscado ENTONCES
            encontrado <- Ciento
        FIN SI
    FIN REPETIR
    -- Se ha llegado al último elemento ó se ha encontrado el buscado
    SI encontrado = Ciento ENTONCES
        Tratar búsqueda exitosa
    FIN SI
    Tratar el caso de elemento no encontrado
FIN SI

```



Esquemas básicos

5. Esquema de búsqueda de un elemento en una secuencia

Condiciones de aplicación:

- Secuencia sin centinela.
 - Se puede “detectar” el último elemento
 - La secuencia puede no tener ningún elemento

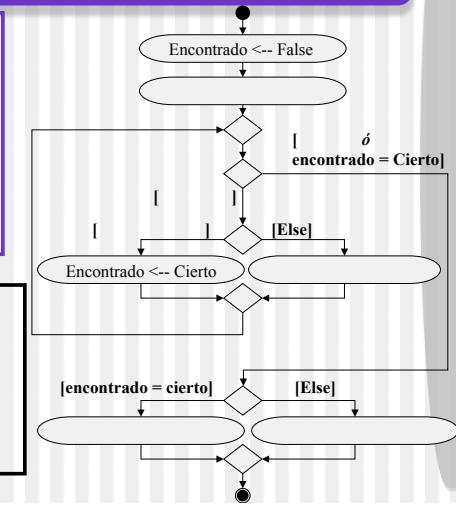
Ejemplo:

```

Encontrado <- Falso
REPETIR
    -- Todos los elementos anteriores al actual no son el buscado
    SALIR SI el elemento actual es el último ó encontrado = Ciento
        SI el elemento es el buscado ENTONCES
            encontrado <- Ciento
        FIN SI
    FIN REPETIR
    -- Se ha llegado al último elemento ó se ha encontrado el buscado
    SI encontrado = Ciento ENTONCES
        Tratar búsqueda exitosa
    FIN SI

```

Diseña un programa que, dada una secuencia de enteros acabada en cero, encuentre el primer número par.



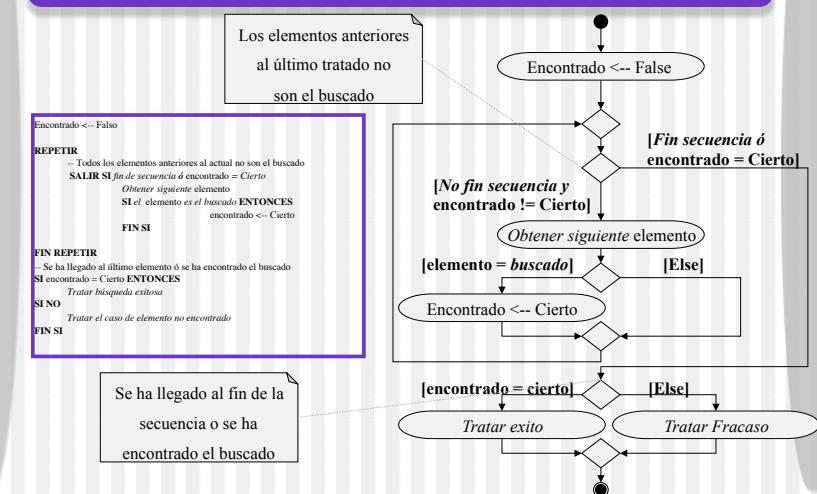
Esquemas básicos

```

Los elementos anteriores al último tratado no son el buscado
Encontrado <- Falso
REPETIR
    -- Todos los elementos anteriores al actual no son el buscado
    SALIR SI fin de secuencia ó encontrado = Ciento
        SI el elemento es el buscado ENTONCES
            encontrado <- Ciento
        FIN SI
    FIN REPETIR
    -- Se ha llegado al último elemento ó se ha encontrado el buscado
    SI encontrado = Ciento ENTONCES
        Tratar búsqueda exitosa
    FIN SI
    Tratar el caso de elemento no encontrado
FIN SI

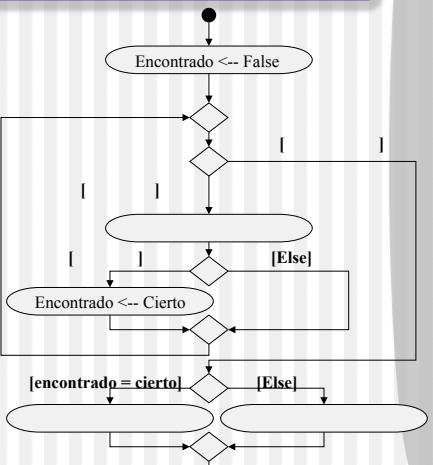
```

Se ha llegado al fin de la secuencia o se ha encontrado el buscado



Ejemplo:

```
encontrado <- Falso
REPETIR
    -- Todos los elementos anteriores al actual no son el buscado
    SALIR SI
        SI _____ ENTONCES
            encontrado <- Ciento
        FIN SI
    FIN REPETIR
    -- Se ha llegado al ultimo elemento ó se ha encontrado el buscado
    SI encontrado = Ciento ENTONCES
        SINO _____
        FIN SI
```



Identificación de problemas con sus esquemas asociados

- **Enumeración de una secuencia cuyo número de elementos es conocido**
 - Esquema 1
- **Enumeración de una secuencia en la que no se conoce el número de elementos**
 - Secuencia con centinela
 - Esquema 2
 - Secuencia sin centinela
 - Esquema 3
- **Búsqueda**
 - Con centinela
 - Esquema 4
 - Sin centinela
 - Esquema 5