Fundamentos de Programación I

3. Estructuras de control

Luis Rodríguez Baena (luis.rodriguez@upsam.es)

Universidad Pontificia de Salamanca
Escuela Superior de Ingeniería y Arquitectura

Programación modular

- Dividir un programa en distintos componentes independientes.
  - Cada uno de esos componentes formaría un **módulo**.
- Cada módulo se codificaría por separado y realizaría una única tarea.
  - Si la tarea es compleja se podrá dividir a su vez en otros módulos.
- Una vez finalizados, los módulos se integrarán en un programa principal que indicará el orden de ejecución.

Programación estructurada

- Conjunto de técnicas enunciadas por Edsger Dijkstra
  - 1968: *El GOTO como elemento perjudicial en la programación (GOTO Statement Considered Harmfull).*
  - Dijkstra y otros autores proponen un conjunto de construcciones lógicas con las que puede construirse cualquier programa.
    - ✓ Cada construcción tendrá una estructura lógica predecible con un único punto de entrada y un único punto de salida.
    - ✓ Facilita el diseño del programa, minimiza la complejidad, reduce los errores.

Programación estructurada (II)

- Teorema de la programación estructurada (teorema de Bohm y Jacopini, 1966).
  - Cualquier programa propio puede ser escrito utilizando únicamente tres tipos de estructuras.
    - ✓ Secuenciales. Las instrucciones se ejecutan una única vez una a continuación de otra y sin posibilidad de variar el orden de ejecución.
    - ✓ Alternativas. Se puede ejecutar una u otra instrucción en virtud del valor de una expresión.
    - ✓ Repetitivas. Las instrucciones se ejecutan un numero determinado de veces en virtud del valor de una expresión.

Nota: Un programa propio es aquel que:

- Posee un único punto de entrada y uno de salida.
- Existen caminos desde la entrada a la salida que se pueden seguir y que pasan por todas las partes del programa.
- Todas las instrucciones son ejecutables y no existen lazos o bucles sin fin.

Programación estructurada (III)

- Supone diseñar los programa de acuerdo a las siguientes reglas:
  - El programa debe tener un diseño modular.
    - ✓ Dividir el problema en pequeñas partes independientes que se desarrollan por separado (módulos) y que luego se integran en una única aplicación.
  - Diseñar los módulos de modo descendente (diseño top-down).
    - ✓ Diseñar cada módulo partiendo del problema general e ir descomponiéndolo en subproblemas más simples mediante refinamientos sucesivos.
  - Uso de recursos abstractos.
    - ✓ Complemento del diseño descendente.
    - ✓ Consiste en suponer en cada uno de los subproblemas ya está resuelto.
      - Más adelante, si es necesario, se resolverán.
  - Limitar las construcciones lógicas del programa.
    - ✓ Utilizar únicamente tres tipos de estructuras:
      - Secuenciales
      - Alternativas
      - Repetitivas.
    - ✓ Esas estructuras están presentes con los lenguajes de alto nivel, por lo que si se respetan, el paso del algoritmo a la codificación será mucho más simple.

Control del flujo de un programa

- Flujo de control.
  - Orden en que se ejecutan las instrucciones dentro de un programa.
- Proceso de ejecución de un programa.
  - Carga del programa en memoria.
  - Reserva de espacio en memoria para almacenar los datos.
  - Ejecución de la primera sentencia.
  - La ejecución continúa hasta llegar a la declaración de final del programa.
- La ejecución se realiza de forma secuencial.
  - Las instrucciones de control se encargan de variar esa ejecución secuencial.
- Las instrucciones de control permiten desarrollar los tres tipos básicos de construcciones lógicas propuestas por la programación estructurada.
  - Desde el punto de vista del procesador, todas presentan alguna bifurcación que permiten saltar de una instrucción a otra.
  - Esas bifurcaciones o saltos se encapsulan en los lenguajes de programación de alto nivel en lo que se llama **estructuras de control**.

Estructura secuencial

- Repite las instrucciones una detrás de otra sin posibilidad de modificar el orden en que se ejecutan.
  - La salida de una instrucción es la entrada de la siguiente.
- Representación algorítmica.

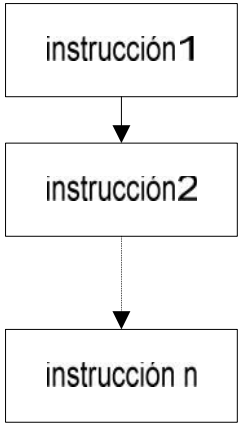


Diagrama de flujo

```
instrucción 1
instrucción 2
      …
instrucción n
```

Pseudocódigo

En C, la estructura secuencial estaría representada por los bloques de instrucciones delimitados por las llaves

```
{
instrucción 1;
instrucción 2;
…
instrucción n;
}
```

C

Estructura secuencial (II)

❑ Ejemplo 3.1.

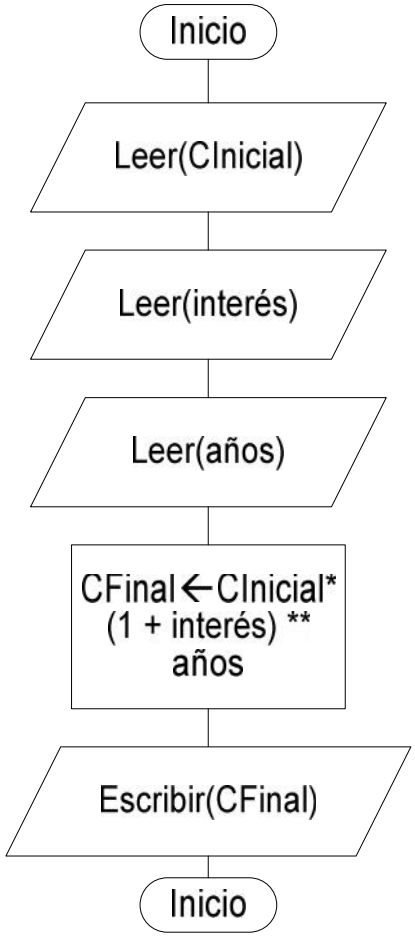
Se desea calcular el capital final de una cantidad de euros colocada en un banco a un interés compuesto determinado durante un periodo de años. Tanto el capital inicial, como el tipo de interés y el número de años se introducirán mediante teclado.

- ***Análisis del problema***
  - ✓ **Datos de salida:** Capital final (variable CFinal de tipo real)
  - ✓ **Datos de entrada:** Capital inicial (variable CInicial de tipo real), interés (variable interés de tipo real) y el número de años (variable años de tipo entero)

Para resolver el problema habrá que aplicar la fórmula:

$$CFinal = CInicial(1 + interés)^{años}$$

Estructura secuencial (III)



Estructuras selectivas

- Permiten ejecutar o no un conjunto de sentencias según el valor de una expresión.
- Dependiendo del número de conjuntos distintos a ejecutar existirán:
  - Estructura selectiva simple.
  - Estructura selectiva doble.
  - Estructura selectiva múltiple.

Estructura selectiva simple

- Evalúa una expresión lógica.
- En función del valor de la expresión ejecuta o no una instrucción o grupo de instrucciones.
- Representación algorítmica.

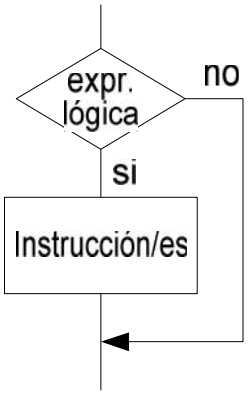


Diagrama de flujo

```
si exp.lógica entonces
   instrucción/es
fin_si
```

Pseudocódigo

```
if (exp. lógica) {
   bloque sentencias;
}
```

C

Estructura selectiva simple (II)

- **Ejemplo 3.2.**

  Determinar si un número entero positivo es par sacando un mensaje en caso afirmativo.

  - ***Análisis del problema***
    - ✓ Datos de salida: El dato de salida sería un mensaje indicando si el número es par.
    - ✓ Datos de entrada: El número a comprobar (variable num de tipo entero)

  Un número es par si es divisible entre 2, por lo que habrá que utilizar el operador **mod** y comprobar si el resto es 0.

```
algoritmo EsNúmeroPar
var
   entero : num
inicio
   leer(num)
   si num mod 2 = 0 entonces
      escribir('es par')
   fin_si
fin
```

```
#include <stdio.h>

int main(void){
    int num;

    printf("Numero entero positivo: ");
    scanf("%i",&num);

    if(num % 2 == 0){
          printf("\nNúmero par");
    }
    return 0;
}
```

Estructura selectiva doble

- Evalúa una expresión lógica y selecciona entre dos grupos de acciones en función del valor de la expresión.

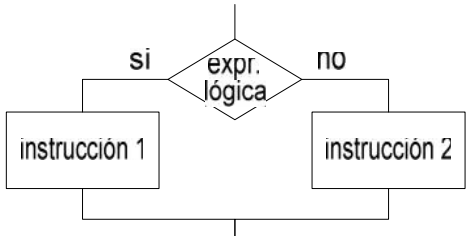


Diagrama de flujo

```
si exp.lógica entonces
   instrucción 1
si_no
   instrucción 2
fin_si
```

Pseudocódigo

```
if (exp. lógica){
   bloque sentencias;
}
else{
   bloque sentencias;
}
```

C

Estructura selectiva doble (II)

- **Ejemplo 3.3.**

  Resolver una ecuación de primer grado ax + b = 0 para dos coeficientes a y b introducidos por teclado. El algoritmo deberá devolver el valor de x y tener en cuenta los casos en los que no tiene solución real (a = 0)

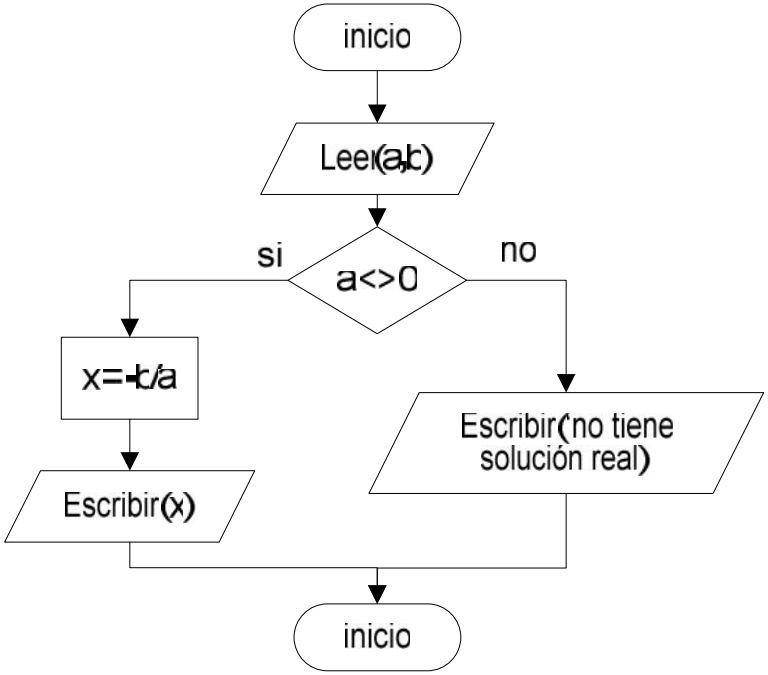
  - ***Análisis del problema***
    - ✓ Datos de salida: El valor de x (una variable real) o el mensaje cuando no hay solución
    - ✓ Datos de entrada: Las variables a y b (de tipo entero)

    El valor de x, siempre que a sea distinto de 0; vendrá determinado por

$$x = \frac{-b}{a}$$

    Si a es 0, sacará un mensaje indicando que no tiene solución real.

Estructura selectiva doble (III)




```
//Ecuación de primer grado
#include <stdio.h>

int main(void){

    int a,b;
    float x;

    printf("Primer número: ");
    scanf("%i",&a);

    printf("Segundo número: ");
    scanf("%i",&b);

    if(a!=0){
           x = (float)-b /a;
           printf("\nPara la ecuación %ix+%i=0"
                  " el valor de x es %f",a,b,x);
    }else{
           printf("\nPara la ecuación %ix+%i=0"
              " el valor de x no tiene solución",a,b);
    }

    return 0;
}
```

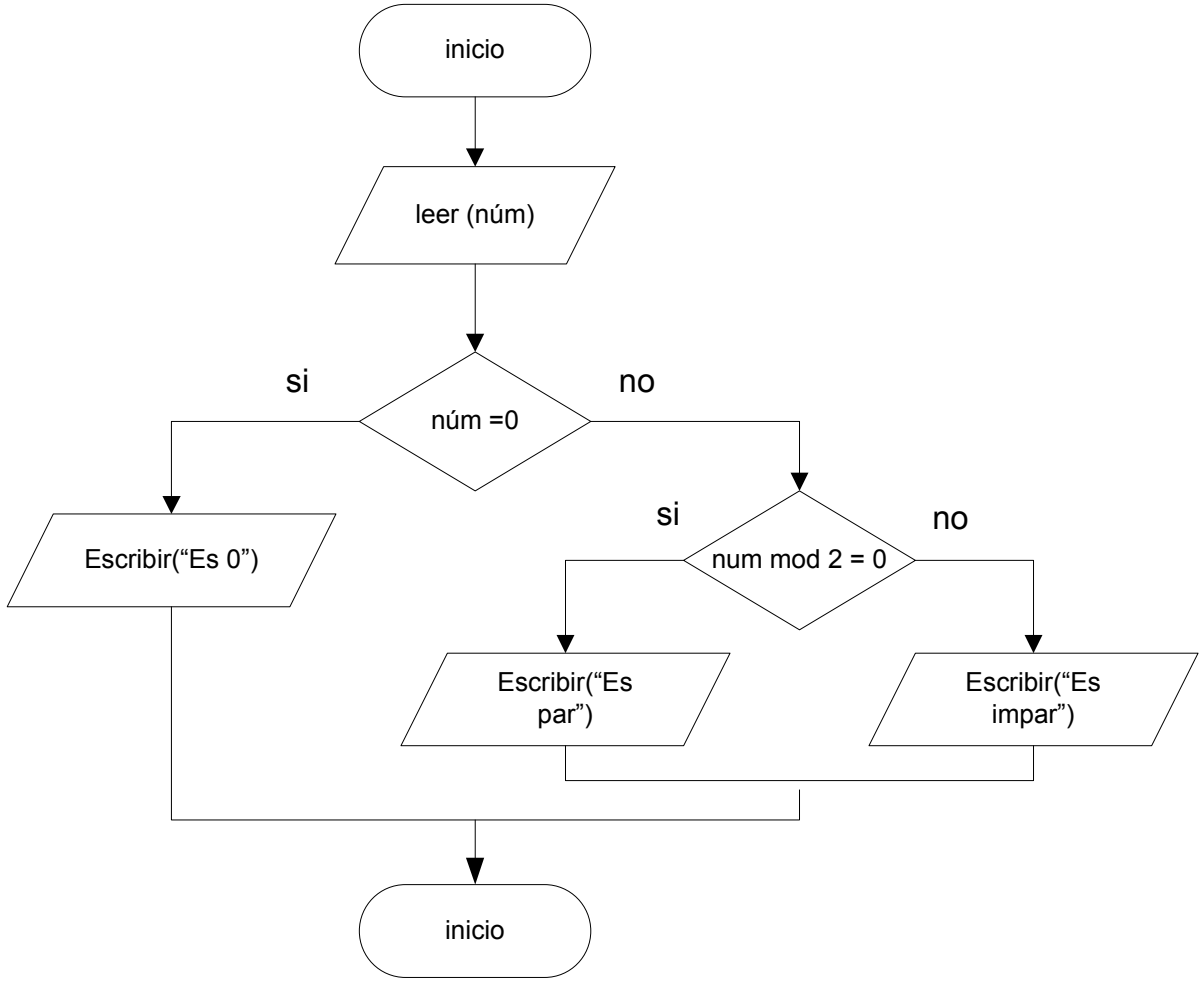
Estructura selectiva doble
Estructuras selectivas anidadas

- **Ejemplo 3.4.**

  Determinar si un número entero es par, impar o 0.

  - ***Análisis del problema***
    - ✓ Datos de salida: El mensaje, una constante de cadena indicando si el número es par, impar o 0.
    - ✓ Datos de entrada: Un número entero.
    - ✓ Proceso: Una vez que se ha determinado que el número es distinto de 0, se comprueba si es par o impar mediante el operador de resto.

Estructura selectiva doble
Estructuras selectivas anidadas (II)



Estructura selectiva múltiple

- Evalúa una expresión no necesariamente de tipo lógico.
  - La expresión puede tomar más de dos valores.
- Dependiendo de los n valores que devuelve la expresión se ejecutarán alguno de los n conjuntos de sentencias que presenta.
- Se utiliza cuando se dispone de varias selecciones múltiples anidadas.
- En la mayoría de los lenguajes la expresión debe ser de tipo entero o algún otro tipo de tipo de dato ordinal.
  - Un tipo de dato ordinal es aquel en el que cada valor tiene un elemento anterior y un elemento siguiente.

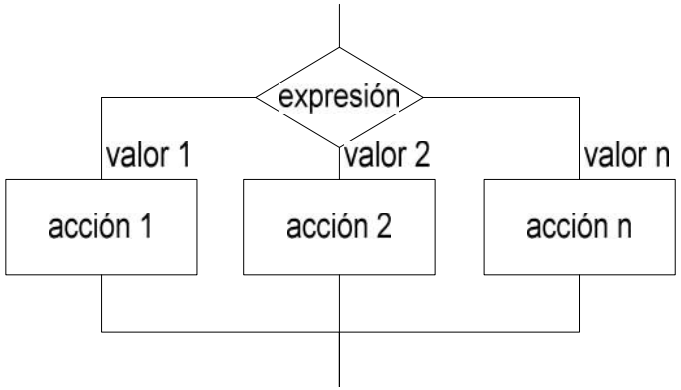


Diagrama de flujo

```
según_sea expresión hacer
   listaValores1: acción 1
   listaValores2: acción 2
   ...
   listaValoresn: acciónn
[si_no
   otras acciones]
fin_según
```

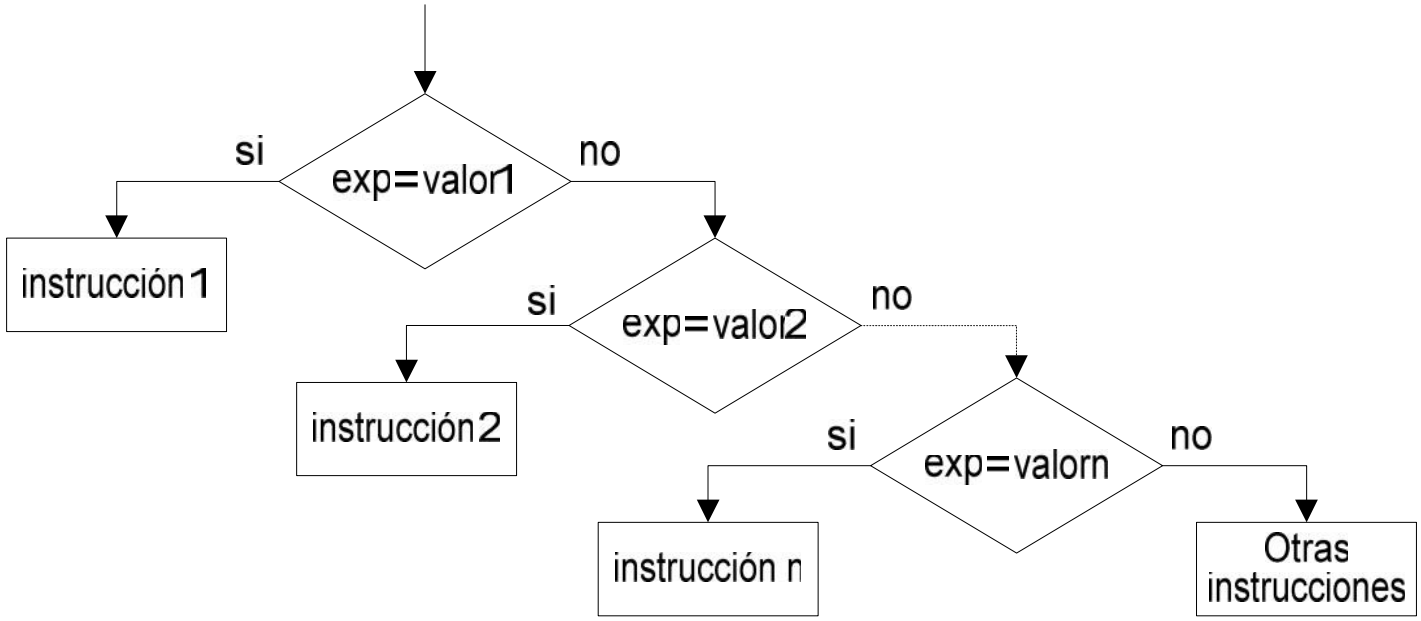
Pseudocódigo

Estructura selectiva múltiple (II)

- Modo de funcionamiento:
  - Se evalúa el valor de la expresión.
  - Si coincide con el valor de alguno de los casos (p.e. *`valor1`*) ejecuta el conjunto de acciones asociadas a él.
  - En caso contrario comprueba el siguiente caso.
  - Puede existir alguna acción por omisión (si no se cumple ninguno de los casos).
    - ✓ La cláusula **`si_no`**, indicaría las acciones a realizar por omisión.
  - Si no existe ninguna acción por omisión o el valor de la expresión no coincide con ninguno de los casos no hace nada.

Estructura selectiva múltiple (III)

- Su modo de funcionamiento equivale a varias estructuras alternativas dobles anidadas.



Estructura selectiva múltiple (IV)

- En C, se utiliza la sentencia `switch`.
- El tipo de expresión debe ser de tipo entero.
- Cada caso sólo puede tener un valor.
  - Podemos seleccionar varios mediante varios `case`:

    `case` *valor1*`: case` *valor2*`: case` *valor3* `:` *instrucciones*
- Además, en C se evalúan todos los valores.
  - Para evitar esto se utiliza una sentencia `break` al final de cada valor.

```
switch(expresión){
   case valor1: instrucciones;
                break;
   case valor2: instrucciones;
                break;
   ...
   case valorn: instrucciones;
                break;
   [default:
       otras instrucciones;]
       break;
}
```

Estructura selectiva múltiple (V)

- **Ejemplo** 3.5

  Diseñar un algoritmo que lee un número correspondiente a un mes del año 2011 y devuelve el número de días que tiene.

  - ***Análisis del problema***
    - ✓ Datos de salida: Un mensaje con el número de días del mes u otro indicando que es un número de mes erróneo.
    - ✓ Datos de entrada: El mes a comprobar (variable mes de tipo entero)

      Se tendrá que determinar si es un mes de 30, de 31 o de 28 (el año 2011 no es bisiesto). Si no se trata de ninguno de estos será un error.

```
algoritmo NúmeroDeDíasDelMes
var
   entero : mes
inicio
   leer(mes)
   según_sea mes hacer
      4,6,9,11 : escribir('tiene 30 días') //Meses de 30 días
      1,3,5,7,8,10,12: escribir('tiene 31 días') //Meses de 31 días
      2 : escribir('tiene 28 días') //Febrero
   si_no
      escribir('número de mes erróneo')
   fin_según
fin
```

Estructura selectiva multiple (VI)

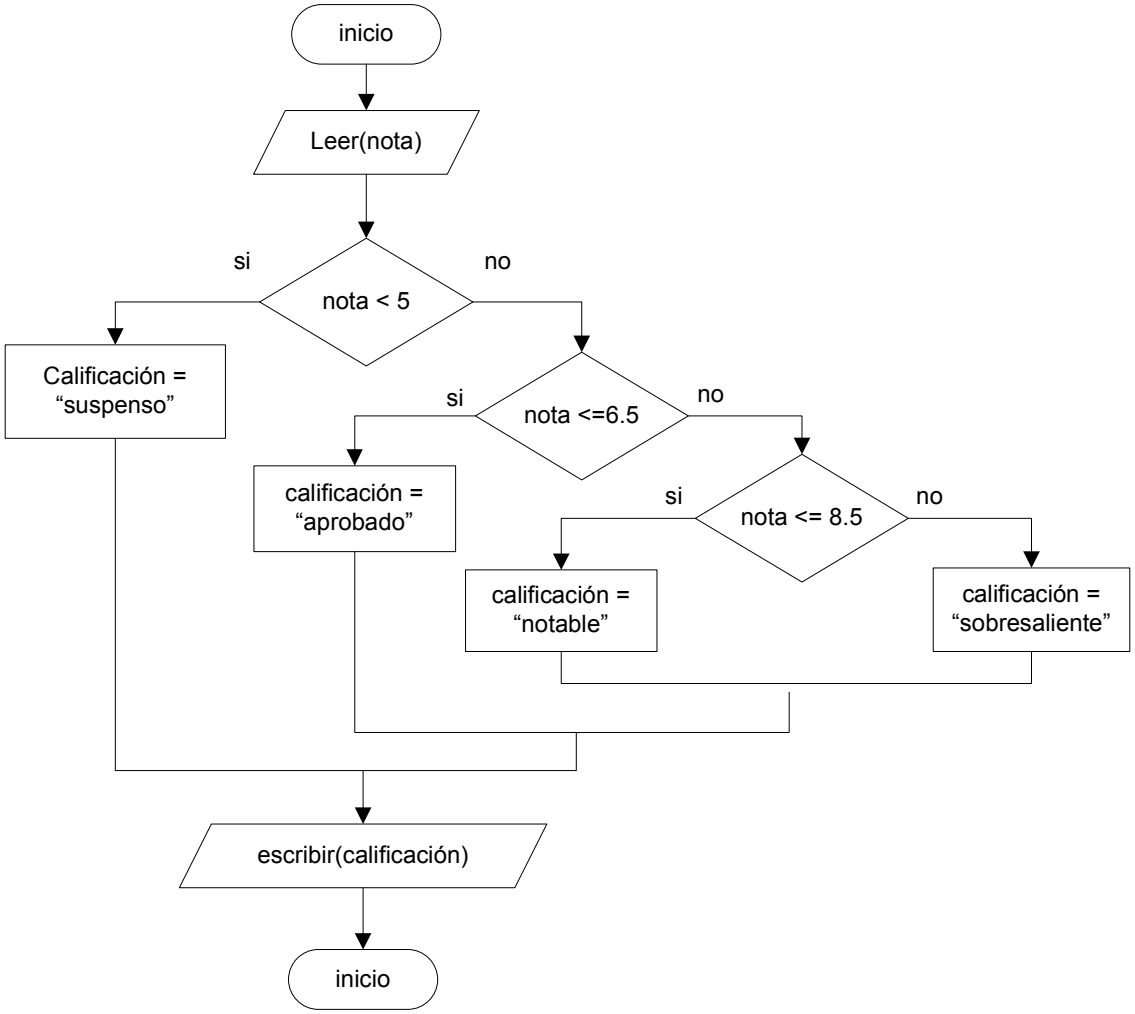
```
//Ejercicio 3.5
//Calcular el número de días del mes
#include <stdio.h>

int main(void){
    int mes;

    printf("Numero del mes:");
    scanf("%i",&mes);

    switch(mes){
          case 4: case 6: case 9: case 11:
                    printf("30 dias");
                    break;
          case 2:
                    printf("28 dias");
                    break;
          case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                    printf("31 dias");
                    break;
          default:
                    printf("No es un mes valido");
    }
}
```

Estructuras selectivas: ejemplo

- Ejemplo 3.6.
  - Diseñar un algoritmo que lea una nota y devuelva su calificación.
    - ✓ La calificación será suspenso (menor que 5), aprobado (entre 5 y 6.5), notable (mayor que 6.5 y menor o igual que 8.5) o sobresaliente (mayor que 8.5).
    - ✓ **Datos salida**: calificación, un dato de tipo cadena.
    - ✓ **Datos de entrada**: nota, un dato real.

¿Qué estructura selectiva se debe utilizar?

Estructuras selectivas: ejemplo (II)



Estructuras repetitivas

- Repiten un conjunto de acciones un número determinado de veces.
- Forman bucles.
  - Cada bucle tendría:
    - ✓ Cuerpo del bucle.
      - Las instrucciones que se repiten.
    - ✓ Condición de entrada o salida del bucle.
  - Para evitar entrar en bucles infinitos.
    - ✓ El resultado de la condición podrá variar en alguna de las iteraciones.
- Dos grandes grupos de instrucciones repetitivas.
  - El cuerpo del bucle se ejecuta de 1 a n veces.
  - El cuerpo del bucle se ejecuta de 0 a n veces.
- Al mismo según la forma en que se controla la ejecución existen:
  - Bucles controlados por contador.
  - Bucles controlados por centinela.

Bucles de 1 a N veces

- Ejecuta un conjunto de instrucciones de 1 a n veces.
- Modo de funcionamiento:
  - Ejecuta el conjunto de acciones que forma el cuerpo del bucle.
  - Evalúa una expresión lógica.
    - ✓ Si es verdadera sale del bucle, en caso contrario vuelve a ejecutar el cuerpo del bucle (estructura **`repetir`**).
    - ✓ En C, si es falsa sale del bucle, en caso contrario vuelve a ejecutar el cuerpo del bucle (instrucción `do-while, o` **`hacer-mientras`**).

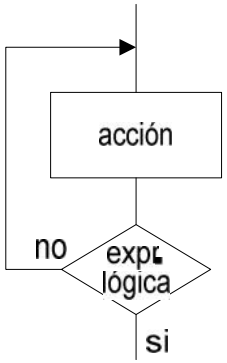


Diagrama de flujo

```
repetir
     acción
hasta_que expresiónLógica
```

Pseudocódigo

```
do{
 bloque de instrucciones;
}while (exp. lógica);
```

C

Bucles controlados por contador

- Un contador es una expresión del tipo:

  variable ← *variable + incrementoConstante*

  - Cada vez que el flujo del programa ejecuta la sentencia la variable se incrementa en una unidad.

```
x ← 0      La variable x se inicializa a 0
x ← x +1   La variable x toma el valor 1
x ← x +1   La variable x toma el valor 2
x ← x +1   La variable x toma el valor 3
```

El valor de la variable equivale al número de veces que ha pasado por la expresión

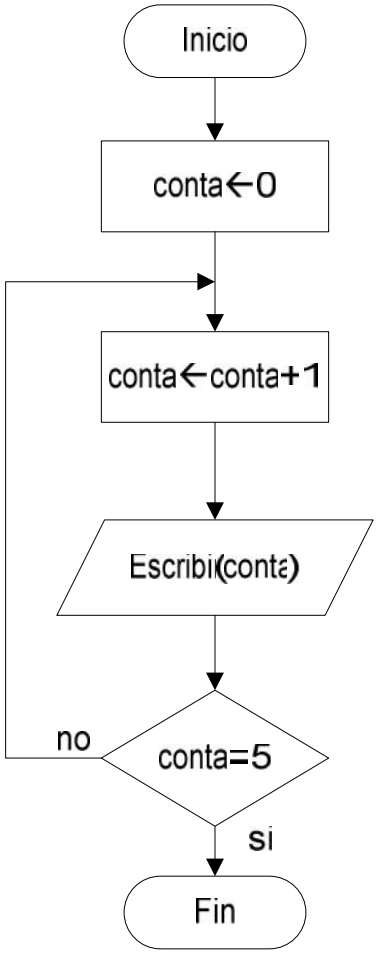
Bucles controlados por contador (II)

- **Ejemplo 3.7.**

  Diseñar un algoritmo que escriba los números entre 1 y 5.

  - ***Análisis del problema***
    - ✓ Datos de salida: Cada uno de los números representados por un contador (variable conta de tipo entero).

      Será necesario utilizar un bucle que se repita 5 veces, inicializando el contador a 1 y comprobando en cada iteración si el número es igual a 5, es decir, si ha escrito los cinco números.

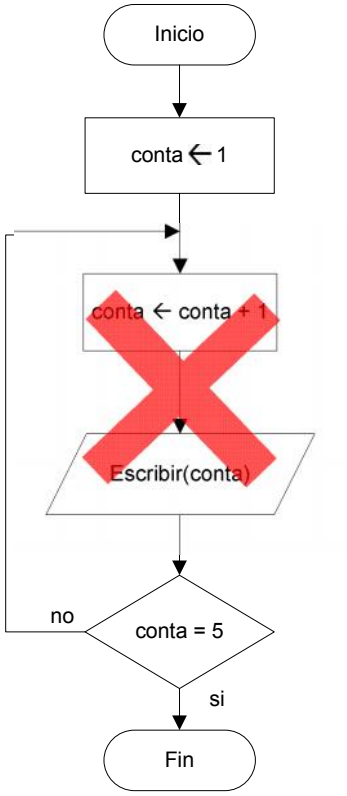


Bucles controlados por contador (III)

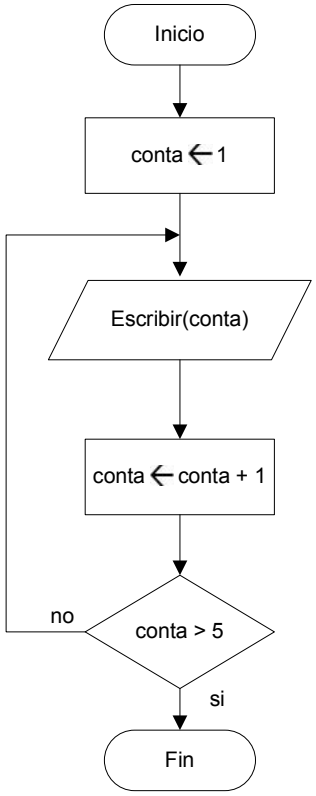
- **Ejemplo 3.7.** Tabla de seguimiento

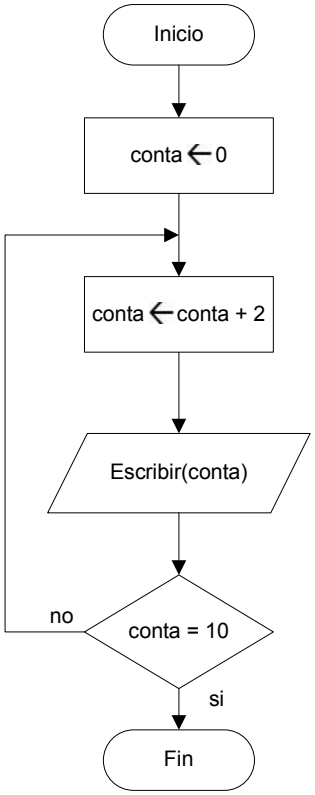
Instrucciones	Variables del algoritmo	Observaciones
	conta	
1. conta ← 0	0	
2. conta ← conta + 1	1	Escribe 1. Cómo conta <> 5 repite el paso 2
2. conta ← conta + 1	2	Escribe 2. Cómo conta <> 5 repite el paso 2
2. conta ← conta + 1	3	Escribe 3. Cómo conta <> 5 repite el paso 2
2. conta ← conta + 1	4	Escribe 4. Cómo conta <> 5 repite el paso 2
2. conta ← conta + 1	5	Escribe 5. Cómo conta = 5 repite el paso 2

Bucles controlados por contador (IV)

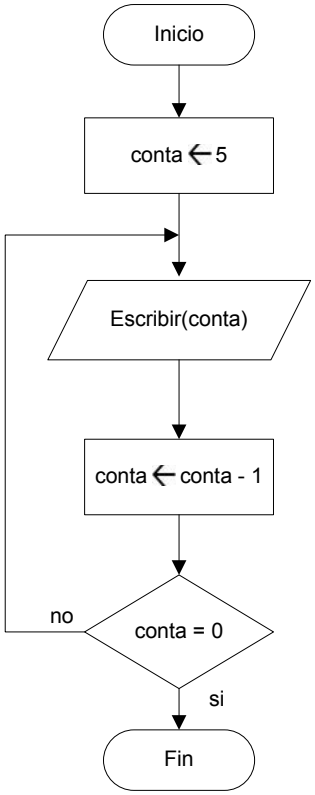


El mismo resultado con el contador inicializado a 1



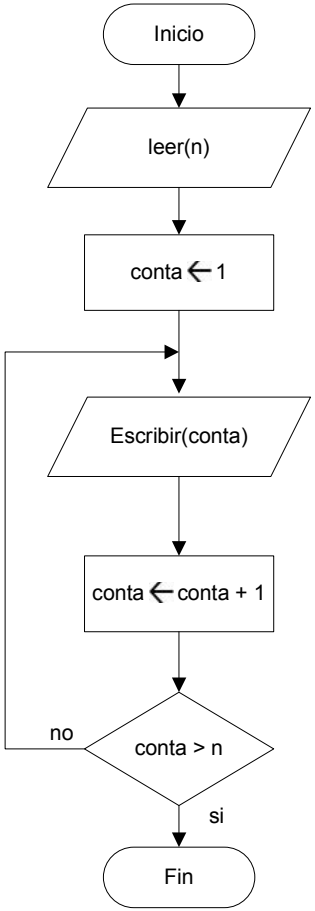


Sacar los 5 primeros números pares. El contador se incrementa de 2 en 2

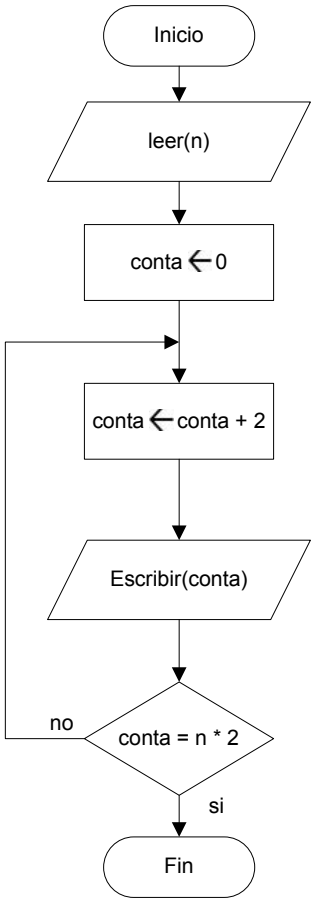


Sacar los número de 5 a 1. El contador se inicializa a 5 y se decrementa

Bucles controlados por contador (V)



Sacar los n primeros números enteros positivos



Sacar los n primeros números pares

Acumuladores

- Un acumulador o sumador es una expresión del tipo:

  variable ← *variable* + *incrementoVariable*

  `media ← media + nota`

- Almacena resultados parciales de una operación.
- Al igual que en los sumadores, la variable aparece en la expresión de la derecha de la sentencia de asignación, por lo que es necesario inicializarla, normalmente, al elemento neutro de la operación que se realiza.
- La operación puede variar, siempre que la expresión sea válida:

  `stock ← stock – unidades`

  `dato ← dato + producto`

  `frase ← frase & palabra`

Ejemplo:
Contadores y acumuladores

- **Ejemplo 3.8.**

  Se desea calcular la nota media de una clase de n alumnos. n es un número entero positivo que introducirá el usuario por teclado. Cada una de las notas también se introducirá por teclado.

  - ***Análisis del problema***
    - ✓ Datos de salida: La media de la clase (variable media de tipo real).
    - ✓ Datos de entrada: El número de alumnos (n, de tipo entero) y cada una de las notas a introducir (variable nota de tipo real).
    - ✓ Datos auxiliares: Es necesario utilizar un contador (variable conta de tipo entero) para poder realizar el número de iteraciones adecuado.

  Para calcular la nota media será necesario acumular cada una de las notas y dividirlas entre el número de notas determinado por n. El bucle se ejecutará hasta que el número de notas introducidas sea igual a n.

Ejemplo:
Contadores y acumuladores (II)

```
algoritmo MediaAlumnos
var
   entero : conta, n
   real : nota,media
inicio
   conta ← 0
   media ← 0
   leer(n)
   repetir
      leer(nota)
      media ← media + nota
      conta ← conta + 1
   hasta_que conta = n
   media ← media / n
   escribir(media)
fin
```

```
int main(void){
    int conta=0,n;
    float nota,media=0;

    printf("Numero de alumnos:");
    scanf("%i",&n);

    do{
          printf("Nota: ");
          scanf("%f",&nota);
          media = media + nota;
       conta = conta + 1;
    }while(conta < n);
    media = media / n;
    printf("Nota media: %.2f",media);

    return 0;
}
```

Bucles de 0 a N veces

- En ocasiones es posible que el cuerpo del bucle tenga que ejecutarse 0 veces.
- La estructura **`mientras`** ejecuta un conjunto de instrucciones de 0 a n veces.
- Modo de funcionamiento:
  - Evalúa una expresión lógica.
  - Si es verdadera ejecuta el conjunto de acciones, en caso contrario no las ejecuta.
  - El cuerpo del bucle se repite mientras la expresión es verdadera.

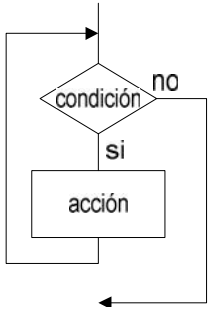


Diagrama de flujo

```
mientras expresiónLógica hacer
  acción
fin_mientras
```

Pseudocódigo

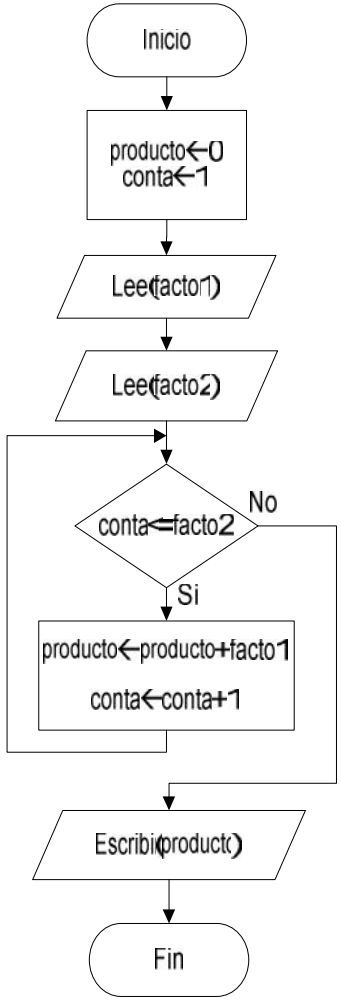
```
while (exp. lógica){
 bloque de instrucciones;
};
```

C

Bucles de 0 a N veces (II)

- Multiplicar dos números mayores o iguales que 0 mediante sumas sucesivas.

```
#include <stdio.h>

int main(void){
    int factor1;
    int factor2;
    int conta=0;
    int producto=0;

    printf("Primer factor:");
    scanf("%i",&factor1);
    printf("Segundo factor:");
    scanf("%i",&factor2);

    while(conta<factor2){
          producto += factor1;
          conta++;
    }

    printf("Resultado: %d",producto);

    return 0;
}
```



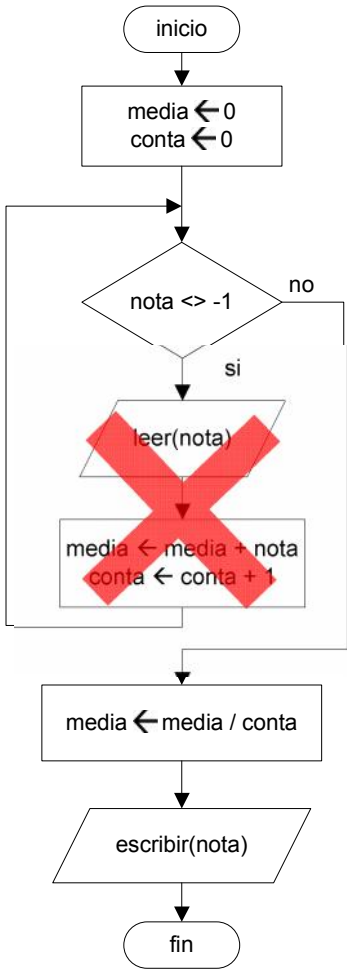
Bucles controlados por centinela

- En ocasiones no se sabe *a priori* el número de iteraciones que debe ejecutarse el cuerpo del bucle.
  - En estos casos el bucle no puede estar controlado por un contador.
  - En su lugar se utiliza un centinela.
- Un centinela es un valor dado a un dato, ya sea de forma arbitraria o por los requisitos de programa, para indicar el fin de las iteraciones.
  - En la condición del bucle, se comprobará si la variable de control tiene el valor centinela asignado para salir del bucle.
  - El centinela puede ser algún valor para los datos de entrada o para cualquier otra variable auxiliar.

Bucles controlados por centinela (II)

- Ejemplo 3.9.
  - Calcular la nota media de un número indeterminado de alumnos.
    - ✓ Cómo las notas deberán tomar valores entre 0 y 10 se podrá utilizar como centinela una nota que no esté dentro de ese rango.
      - Por ejemplo, se podrán introducir notas hasta que la nota tome el valor -1.
        - La condición de salida del bucle será cuando la nota tome el valor -1.

Bucles controlados por centinela (III)

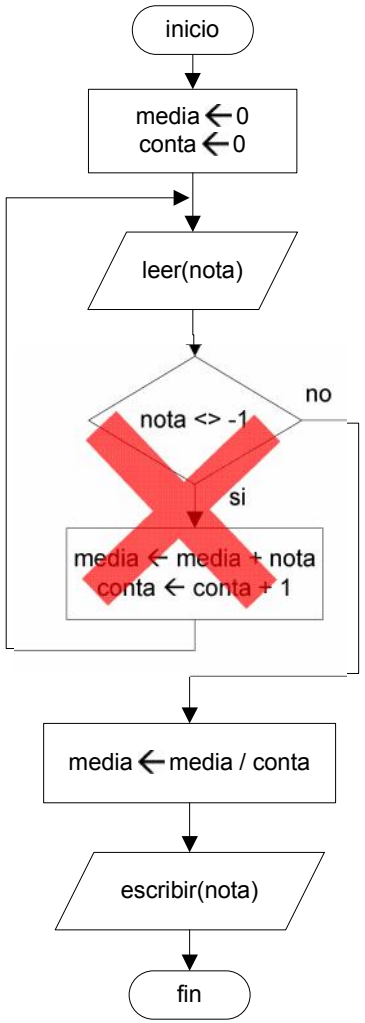


El valor del centinela se incluiría dentro del resultado.

Se saldría del bucle con un valor de la variable media al que se habría acumulado el centinela.

Variables del algoritmo		
media	**conta**	**nota**
0	0	
		3
3	1	
		6
9	2	
		-1
8	3	
2,6		

Bucles controlados por centinela (IV)

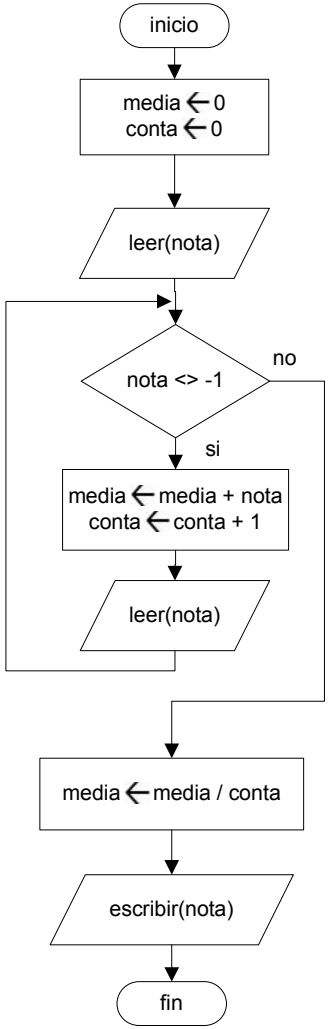


El resultado sería correcto, pero…

¿cuál sería la estructura de control adecuada para codificar esto?

Variables del algoritmo		
media	**conta**	**nota**
0	0	
		3
3	1	
		6
9	2	
		-1
4,5		

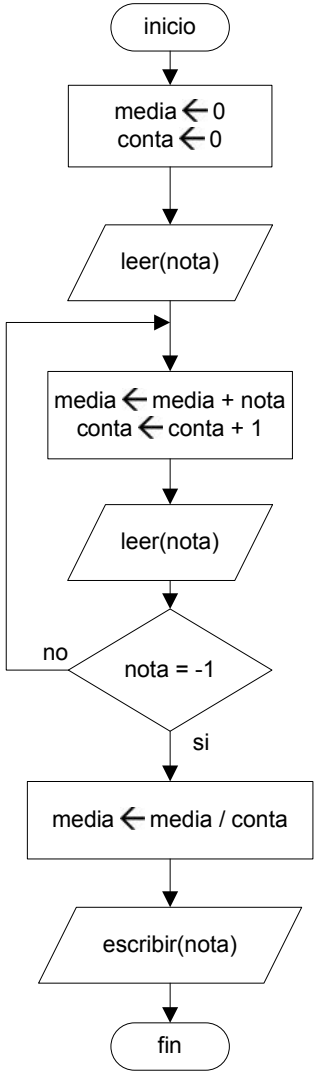
Bucles controlados por centinela (V)



Cuando el centinela es alguno de los datos que intervienen en el resultado se puede utilizar la técnica de lectura anticipada:

Leer una vez antes de entrar en el bucle y otra vez al final del bucle, justo antes de verificarse la condición de salida.

Bucles controlados por centinela (VI)



Utilizar bucles con centinela es independiente del tipo de bucle elegido.

La elección de un bucle controlado por contador o uno controlado por centinela depende de que se conozca o no el número de iteraciones.

La elección de bucles que se ejecutan 0 a n veces o 0 a 1 vez depende de las especificaciones del problema

Bucles controlados por centinela: Ejemplos

- Ejemplo 3.10.
  - Diseñar un algoritmo que permita leer números enteros cualesquiera. El algoritmo deberá devolver la cantidad de números leídos y la suma total de los mismos.
    - ✓ Puesto que los datos de entrada no tienen un rango determinado, no se pueden utilizar como centinela.
    - ✓ Es posible utilizar otra nueva variable como centinela, pero independiente de los datos de entrada.
    - ✓ **Datos de salida**: conta y suma (de tipo entero).
    - ✓ **Datos de entrada**: núm (cada uno de los números de tipo entero) y masNúmeros (un dato de tipo cadena, si toma el valor 'S' se continuará en el bucle, si toma el valor 'N' se saldrá del bucle)

Bucles controlados por centinela:
Ejemplos (II)

```
algoritmo SumarYContarNúmeros
var
   entero : núm, conta, suma
   //masNúmeros será una variable de tipo carácter.
   //Mientras tenga valor 'S' se seguirán pidiendo
   //números.
   cadena: masNúmeros
inicio
   conta ← 0
   suma ← 0
   repetir
      leer(núm)
      suma ← suma + num
      conta ← conta + 1
      //Antes de finalizar el bucle es necesario dar un valor a
      //masNúmeros para realizar la comprobación
      escribir('¿Más números (S/N)?')
      leer(másNúmeros)
   hasta_que masNúmeros = 'N'
   escribir(suma,conta)
fin
```

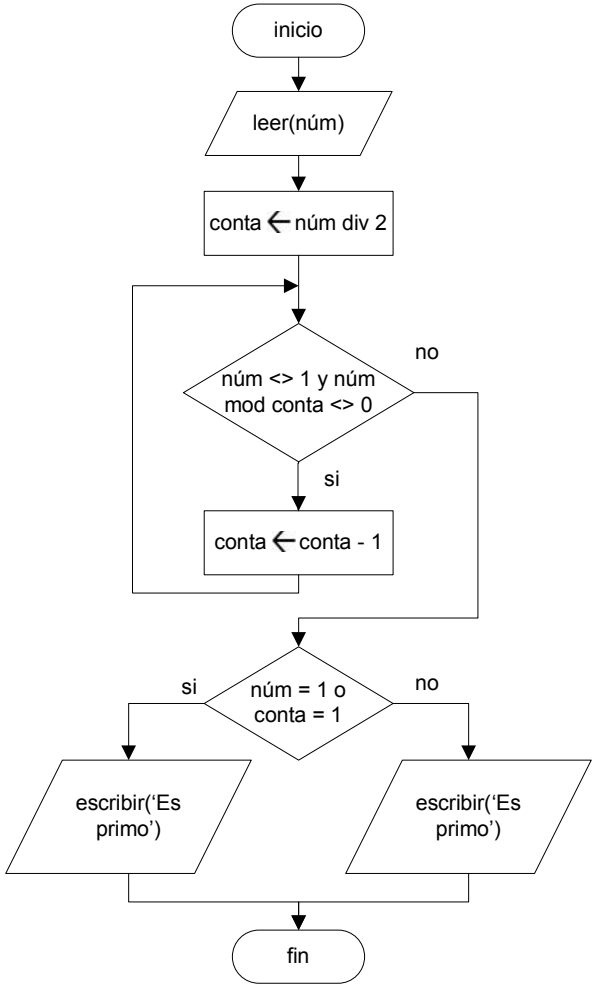
Bucles controlados por centinela:
Ejemplos (III)

```
//Ejemplo 3.10
//Leer una cantidad indefinida de números enteros cualesquiera.
//Obtener la suma y la cantidad de números leídos

#include <stdio.h>

int main(void){
    int num, conta=0, suma=0;
    char masNumeros= ' ';

    do{
          printf("Numero:");
          scanf("%i",&num);
          suma +=num;
          conta++;
          printf("¿Mas numeros?");
          //Sirve para vaciar el buffer de entrada después de pulsar Intro
          fflush(stdin);
          scanf("%c",&masNumeros);
    }while(masNumeros != 'N');

    printf("Numeros leidos: %i\n",conta);
    printf("Suma: %i",suma);

    return 0;
}
```

Bucles controlados por centinela:
Ejemplos (IV)

- Ejemplo 3.11.
  - Diseñar un algoritmo que indique si un número entero positivo es primo.
    - ✓ **Datos de salida**: un mensaje por pantalla ("Es primo" o "No es primo").
    - ✓ **Datos de entrada**: núm (un número entero positivo)
    - ✓ **Proceso**: Un número primo es divisible entre el mismo y la unidad, por lo que si al dividirlo sucesivamente desde la mitad del número por todos los números inferiores no se encuentra ningún divisor, el número es primo. Será necesaria una variable auxiliar, un contador por el que se irán dividiendo los números.

Bucles controlados por centinela:
Ejemplos (V)




```
//Ejemplo 3.11
//Comprueba si un número es primo

#include <stdio.h>

int main(void){
    int conta,num;

    printf("Numero:");
    scanf("%i",&num);

    conta = num / 2;

    while(num != 1 && num % conta != 0){
          conta = conta - 1;
    }

    if(num == 1 || conta == 1){
          printf("Es primo");
    }else{
          printf("No es primo");
    }

    return 0;
}
```

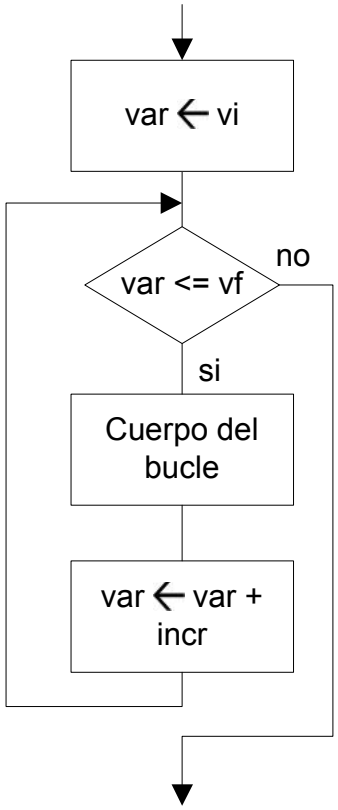
Estructura desde

- Estructura repetitiva controlada por un contador y que se ejecuta de 0 a n veces y que incorpora el contador, su incremento y la condición de salida.
  - En el lenguaje algorítmico UPSAM, su formato es…

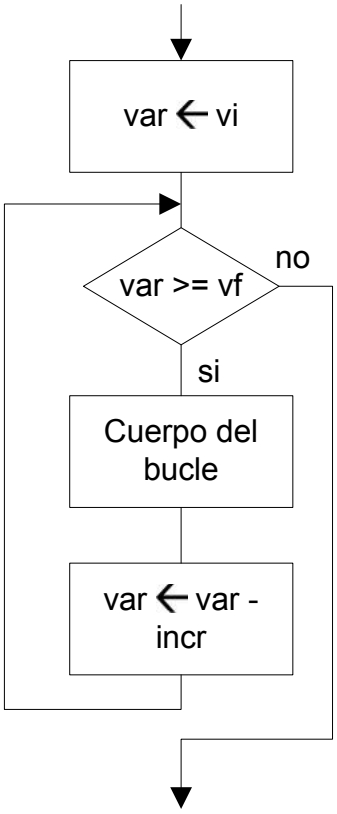
```
desde variable ← vInicial hasta vFinal [incremento vIncremento] hacer
   instrucciones
fin_desde
```

- La variable, el valor inicial, el valor final y el incremento deberían ser variables y expresiones enteras.
- Modo de funcionamiento:
  - Se asigna a la variable el valor inicial que actuará como contador.
  - Si el incremento es positivo se comprueba si el valor es menor o igual al valor final.
    - ✓ En caso afirmativo ejecuta el cuerpo del bucle, en caso contrario no entra en el bucle.
  - Al final del bucle se incrementa la variable y se vuelve a comprobar si el nuevo valor es menor o igual que el valor final.

Estructura desde (II)



Funcionamiento con incremento positivo



Funcionamiento con incremento negativo

Estructura desde (III)

- En C, su formato es:

```
for(expresión inicialización; condición entrada; incremento){
   bloque de instrucciones;
}
```

  - Aunque en C la expresión de inicialización, la condición de entrada y la actualización del contador puede ser cualquier expresión (incluso estar vacías), se recomienda que las tres expresiones inicialicen, comprueben y actualicen.
    - ✓ Si se desea utilizar una condición de salida para un bucle con centinela es mejor utilizar alguna de las otras estructuras repetitivas.

Estructura desde (IV)

```
desde i ← 1 hasta 5 hacer
   escribir(i)
fin_desde
//La salida será 1, 2, 3, 4, 5
```

```
desde i ← 0 hasta 10 incremento 2 hacer
   escribir(i)
fin_desde
//La salida será 0, 2, 4, 6, 8, 10
```

```
desde i ← 1 hasta 5 incremento -1 hacer
   escribir(i)
fin_desde
//No habrá salida, ya que 1 < 5
//y el incremento es negativo
```

```
desde i ← 5 hasta 1 incremento -1 hacer
   escribir(i)
fin_desde
//La salida será 5, 4, 3, 2, 1
```

```
leer(n)
desde i ← 1 hasta n hacer
   escribir(i)
fin_desde
//Si n < 1 no habrá salida
//Si no, la salida será 1, 2, 3, … , n
```

```
leer(n)
desde i ← n mod 2 hasta n + 1 hacer
   escribir(i)
fin_desde
//Si n = 6,
//la salida será 0, 1, 2, 3, 4, 5, 6, 7
```

Estructura desde (V)

- Ejemplo 3.12.
  - Diseñar un algoritmo que permita calcular la potencia de un número x elevado a la potencia y, siendo x e y dos números enteros mayores o iguales que 0.
    - ✓ **Datos de salida**: potencia, un número entero positivo.
    - ✓ **Datos de entrada**: x e y, dos números enteros mayores o iguales que 0.
    - ✓ **Datos auxiliares**: potencia, un número entero, y i, un contador.
    - ✓ **Proceso**: Habrá que multiplicar x tantas veces como indique i. Hay que tener en cuenta que cualquier número elevado a 0 es 1.

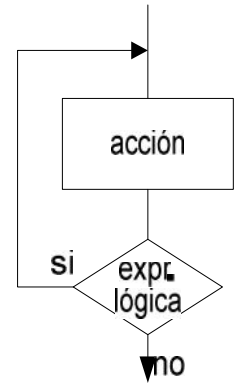
Estructura desde (VI)

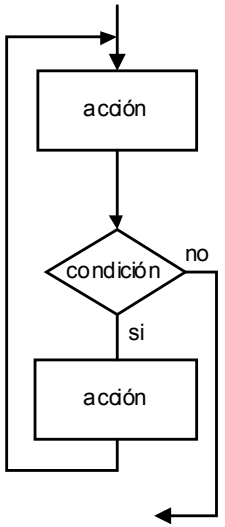
```
algoritmo potencia
var
   entero: x, y, potencia, i
inicio
   potencia ← 1
   leer(x,y)
   desde i ← 1 hasta y hacer
      potencia ← potencia * x
   fin_desde
   escribir(potencia)
fin
```

```
#include <stdio.h>

int main(void){
    int x,y,potencia=1;

    printf("X:");
    scanf("%i",&x);
    printf("Y:");
    scanf("%i",&y);

    for(int i=1;i<=y;i++){
           potencia = potencia * x;
    }

    printf("%i elevado a %i es %i",x,y,potencia);
    return 0;
}
```

Otras estructuras de control

- **`hacer-mientras`.**
  - Estructura mientras que se ejecuta de 1 a n veces.
  - Equivale a la estructura `do-while` de C.
- **`iterar-fin_iterar`.**
  - Estructura repetitiva sin instrucción de entrada o salida definida.
  - La condición de salida aparecerá dentro del cuerpo del bucle mediante alguna sentencia de salto.





Otras estructuras de control (II)

- Sentencias de salto.
  - Algunos lenguajes (C, Java) permiten salir del bucle por cualquier sitio.
  - La sentencia **`break`** (**`salir`**), provoca que el flujo de control salte a la instrucción siguiente al final del bucle.
  - La sentencia **`continue`** (**`continuar`**), provoca que el flujo de control salte al comienzo del bucle.
  - Utilizando de forma controlada estas sentencias es posible implementar bucles que cuya condición de salida no esté ni al comienzo ni al final.

```
...
mientras verdad hacer
   ...
   si condiciónSalidaBucle entonces
      salir
   fint_si
   ...
fin_mientras
...
```

```
...
do{
   ...
   if(condiciónSalidaBucle){
      break;
   }
   ...
}while(1) //La condición siempre es verdad
...
```

Otras estructuras de control (III)

- Sentencia **`goto`** (**`ir_a`**).
  - Transfiere el control a la línea de programa situada después de una etiqueta.
    - ✓ Una etiqueta sirve para referenciar una parte del programa.
  - Cualquier programa propio se puede realizar sin utilizarla.
  - No se debe utilizar nunca.

Ejercicios

1. Diseñe un algoritmo que permita calcular el sueldo neto semanal de un trabajador según los siguientes criterios:
   - El sueldo bruto se calculará según las horas semanales trabadas. Las primeras 40 horas semanales se pagarán a la tarifa normal (35 €). Cada hora extra trabajada se pagará a 1,5 veces la tarifa normal.
   - El sueldo neto se calculará restando al sueldo bruto las retenciones. Los primeros 1000 euros no llevarán retención. El tramo del sueldo bruto entre 1000 y 1500 euros llevará una retención del 12%. Todo el tramo de sueldo que sobrepase los 1500 euros llevará un 18% de retención.
2. Diseñe un algoritmo que permita almacenar 3 números en tres variables y los muestre por pantalla ordenados de menor a mayor.

Ejercicios (II)

3. Diseñe un algoritmo que permita resolver la ecuación de segundo grado $ax^2+bx+c=0$.
   - Comprobar:
     - ✓ Si a = 0, no es una ecuación de segundo grado.
     - ✓ Si el discriminante es 0, sólo tiene una solución.
     - ✓ Si el discriminante es negativo la solución es imaginaria.
     - ✓ En caso contrario calcular los valores de x_1 y x_2.
4. Diseñe un algoritmo que permita introducir el número de un mes y devuelva su nombre.
5. Diseñe un algoritmo que permita sumar n números.
   - Realice dos versiones, una con n > 0 y otra con n >= 0.
6. Diseñe un algoritmo que permita sumar una cantidad desconocida de números enteros positivos.

Ejercicios (III)

7. Diseñe un algoritmo que calcule el factorial de un número entero mayor o igual que 0 introducido por teclado.
8. Diseñe un algoritmo que permita hallar la suma de n números, sumando por separado los números pares y los impares.
9. Diseñe un algoritmo que permita obtener la nota media de un número indeterminado de notas (antes de comenzar el bucle se desconoce el número de notas a procesar, por lo que habrá que utilizar un bucle controlado por centinela).
   - Al final del algoritmo será necesario mostrar la nota media y la calificación (menor que 5, suspenso, mayor o igual que 5 y menor que 7 aprobado, mayor o igual que 7 y menor que 9 notable y mayor o igual que 9 sobresaliente).

Ejercicios (IV)

10. Diseñe un algoritmo que muestre por pantalla el máximo común divisor de dos números enteros positivos introducidos por teclado.
    - Diseñar dos versiones distintas del algoritmo, una realizada mediante tanteo y otra mediante el algoritmo de Euclides.
11. Diseñe un algoritmo que permita sacar por la pantalla el valor mayor de una serie de números positivos introducidos por teclado.
12. Diseñe un algoritmo que saque por pantalla los números primos menores que n. n es un número entero positivo introducido por teclado.
13. Sacar todos los números de tres cifras abc que cumplen la siguiente condición $abc=a^3+b^3+c^3$, es decir la suma de las centenas al cubo, más las decenas al cubo + las unidades al cubo es igual al propio número.
    - 370 cumple la condición, ya que $3^3+7^3+0^3 = 370$
14. Se tiene una cuba con 100 litros de cola y otra con 100 litros de ginebra. Además se tiene un recipiente de un litro. Con el recipiente se desea ir vaciando litro a litro la cuba con cola y llenarla con un litro de ginebra.
    - Diseñe un algoritmo que cuente el número de veces que se repite la operación para que en la mezcla en la cuba de cola tenga más ginebra que refresco.

Ejercicios (V)

15. Diseñe un algoritmo que permita escribir un número al revés.
16. Dada una letra introducida por teclado y secuencia de caracteres también introducidos por teclado, diseñar un algoritmo que indique el número de veces que la letra aparece en la secuencia.
    - Por ejemplo: si la letra es la "s", y la secuencia de caracteres introducida por teclado es "r, t, s, u, s, v, x, s, t", el algoritmo debería sacar un 3, ya que aparecen tres caracteres "s" en la secuencia.
17. Se desea realizar una estadística sobre el peso de los alumnos de un colegio. Diseñe un algoritmo que indique el número de alumnos en cada uno de los siguientes intervalos de peso:
    - Alumnos de menos de 40 kg.
    - Alumnos entre 40 y 60 kg.
    - Alumnos de más de 60 kg.

Ejercicios (VI)

18. Hallar la temperatura media mensual de un observatorio durante un mes del año 2009.
    - Se deberá introducir el número del mes por teclado para saber el número de días del mismo.
    - La temperatura media del mes será la media de las temperaturas medias diarias.
    - Por cada día se introducirá la temperatura máxima y mínima del día y la temperatura media de ese día será el valor medio entre ambas temperaturas.
19. Dada la serie $a_1=0$, $a_2=1$, $a_n=3*a_{n-1}+2*a_{n-2}$, obtener el valor y el rango del primer término <= que 1000.
20. Calcular la nota media de cada alumno en una clase de n alumnos (el algoritmo deberá devolver n notas medias, una por cada alumno). Cada uno de los alumnos tiene una cantidad desconocida de notas.