3

ESTRUCTURA GENERAL DE UN PROGRAMA

La descripci·n de los elementos b§sicos de programaci·n que se encontrar§n en casi todos los programas (interruptores, contadores, acumuladores), as² como las normas elementales para la escritura de algoritmos en pseudoc·digo constituyen el contenido de este cap²tulo.

3.1. ESTRUCTURA DE UN PROGRAMA

Como ya se ha indicado en otras ocasiones el pseudocódigo es la herramienta más adecuada para la representación de algoritmos. El algoritmo en pseudocódigo debe tener una estructura muy clara y similar a un programa, de modo que se facilite al máximo su posterior codificación. Interesa por tanto conocer las secciones en las que se divide un programa, que habitualmente son:

- La *cabecera.*
- El *cuerpo* del programa:
  - — Bloque de declaraciones.
  - — Bloque de instrucciones.

La *cabecera* contiene el nombre del programa.

El *cuerpo* del programa contiene a su vez otras dos partes: el bloque de declaraciones y el bloque de instrucciones. En el bloque de declaraciones se definen o declaran las constantes con nombre, los tipos de datos definidos por el usuario y también las variables. Suele ser conveniente seguir este orden.

La *declaración de tipos* suele realizarse en base a los tipos estándar o a otros definidos previamente, aunque también hay que considerar el método directo de enumeración de los valores constituyentes.

El *bloque de instrucciones* contiene las acciones a ejecutar para la obtención de los resultados. Las instrucciones o acciones básicas a colocar en este bloque se podrían clasificar del siguiente modo:

- **De inicio/fin**. La primera instrucción de este bloque será siempre la de inicio y la última la de fin.
- **De asignación**. Esta instrucción se utiliza para dar valor a una variable en el interior de un programa.

- **De lectura**. Toma uno o varios valores desde un dispositivo de entrada y los almacena en memoria en las variables que aparecen listadas en la propia instrucción.
- **De escritura**. Envía datos a un dispositivo de salida.
- **De bifurcación**. Estas instrucciones no realizan trabajo efectivo alguno, pero permiten controlar el que se ejecuten o no otras instrucciones, así como alterar el orden en el que las acciones son ejecutadas. Las bifurcaciones en el flujo de un programa se realizarán de modo condicional, esto es en función del resultado de la evaluación de una condición. El desarrollo lineal de un programa se interrumpe con este tipo de instrucciones y, según el punto a donde se bifurca, podremos clasificarlas en bifurcaciones hacia adelante o hacia atrás. Las representaremos mediante estructuras selectivas o repetitivas.

Además, se recomienda que los programas lleven comentarios.

3.2. ESTRUCTURA GENERAL DE UN ALGORITMO EN PSEUDOCÓDIGO

La representación de un algoritmo mediante pseudocódigo se muestra a continuación. Esta representación es muy similar a la que se empleará en la escritura al programar.

```
algoritmo <nombre_algoritmo>
const
   <nombre_de_constante1> = valor1
   ...
var
   <tipo_de_dato1>: <nombre_de_variable1> [, <nombre_de_variable2>, .....]
   ...
   //Los datos han de ser declarados antes de poder ser utilizados
inicio
   <acción1>
   <acción2>
   //Se utilizará siempre la sangría en las estructuras selectivas y
   //repetitivas.
   .........
   <acciónN>
fin
```

Como se observa, después de la cabecera se coloca el bloque de declaraciones, donde se han declarado constantes con nombre y variables. Para declarar las constantes con nombre el formato ha sido:

```
const
   <nombre_de_constante1> = <valor1>
   ...
```

Al declarar las variables hay que listar sus nombres y especificar sus tipos de la siguiente forma:

```
var
   <tipo_de_dato1>: <lista_de_variables>
      ...
```

En el bloque de instrucciones, marcado por `inicio` y `fin` se sitúan las sentencias ejecutables, por ejemplo las operaciones de cálculo y lectura/escritura. El formato de las operaciones de lectura, cuando el dispositivo es el dispositivo estándar de entrada (teclado) es:

```
leer(<lista_de_variables>)
```

La operación de escritura, cuando los datos los enviemos al dispositivo estándar (pantalla), tiene el siguiente formato:

```
escribir(<lista_de_expresiones>)
```

3.3. LA OPERACIÓN DE ASIGNACIÓN

Esta operación se utiliza para dar valor a una variable en el interior de un algoritmo y permite almacenar en una variable el resultado de evaluar una expresión, perdiéndose cualquier otro valor previo que la variable pudiera tener. Su formato es:

```
<nombre_de_variable> ← <expresión>
```

Una expresión puede estar formada por una única constante, variable o función. La variable que recibe el valor final de la expresión puede intervenir en la misma, con lo que se da origen a contadores y acumuladores.

Se supone que se efectúan conversiones automáticas de tipo cuando el tipo del valor a asignar a una variable es compatible con el de la variable y de tamaño menor. En estos casos se considera que el valor se convierte automáticamente al tipo de la variable. También se interpreta que se efectúa este tipo de conversiones cuando aparecen expresiones en las que intervienen operandos de diferentes tipos.

3.3.1. Contadores

Un *contador* es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción. Los contadores se utilizan en las estructuras repetitivas con la finalidad de contar sucesos o acciones internas del bucle.

Con los contadores deberemos realizar una operación de inicialización y, posteriormente, las sucesivas de *incremento* o *decremento* del contador.

La *inicialización* consiste en asignarle al contador un valor. Se situará antes y fuera del bucle.

```
<nombre_del_contador> ← <valor_de_inicialización>
```

En cuanto a los *incrementos* o *decrementos* del contador, puesto que la operación de asignación admite que la variable que recibe el valor final de una expresión intervenga en la misma, se realizarán a través de este tipo de instrucciones de asignación, de la siguiente forma:

```
<nombre_del_contador> ← <nombre_del_contador> + <valor_constante>
```

dicho `<valor_constante>` podrá ser positivo o negativo. Esta instrucción se colocará en el interior del bucle.

3.3.2. Acumuladores

Son variables cuyo valor se incrementa o decrementa en una cantidad determinada. Necesitan operaciones de:

- *Inicialización*

```
<nombre_acumulador> ← <valor_de_inicialización>
```

- *Acumulación*

  <nombre_acumulador> ← *<nombre_acumulador>* + *<nombre_variable>*

  Hay que tener en cuenta que la siguiente también sería una operación de acumulación:

  <nombre_acumulador> ← *<nombre_acumulador>* * *<valor>*

3.3.3. Interruptores

Un interruptor o bandera (*switch)* es una variable que puede tomar los valores **verdad** y **falso** a lo largo de la ejecución de un programa, comunicando así información de una parte a otra del mismo. Pueden ser utilizados para el control de bucles y estructuras selectivas. En este caso también es frecuente que la variable que recibe el valor final de una expresión intervenga en la misma, por ejemplo

En primer lugar el interruptor (*switch)* se inicializa a **verdad** o **falso**

<nombre_del_interruptor> ← *<valor_de_inicialización>*

En determinadas condiciones el interruptor conmuta

<nombre_del_interruptor> ← **no** *<nombre_del_interruptor>*

3.4. EJERCICIOS RESUELTOS

3.1. *Se desea calcular independientemente la suma de los números pares en impares comprendidos entre 1 y 200.*

Análisis del problema

El algoritmo no necesitaría ninguna variable de entrada, ya que no se le proporciona ningún valor. Como dato de salida se tendrán dos variables (`sumapar` y `sumaimpar`) que contendrían los dos valores pedidos. Se necesitará también una variable auxiliar (`contador`) que irá tomando valores entre 1 y 200.

Después de inicializar el contador y los acumuladores, comienza un bucle que se ejecuta 200 veces. En ese bucle se controla si el contador es par o impar, comprobando si es divisible por dos con el operador **mod**, e incrementando uno u otro acumulador.

Diseño del algoritmo

```
algoritmo ej_3_1
var
   entero : contador,sumapar,sumaimpar
inicio
   contador ← 0
   sumapar ← 0
   sumaimpar ← 0
   repetir
      contador ← contador + 1
      si contador  mod 2 = 0 entonces
         sumapar ← sumapar + contador
      si_no
         sumaimpar ← sumaimpar + contador
```

```
      fin_si
   hasta_que contador = 200
   escribir (sumapar,sumaimpar)
fin
```

3.2. *Leer una serie de números enteros positivos distintos de 0 (el último número de la serie debe ser el –99) obtener el número mayor.*

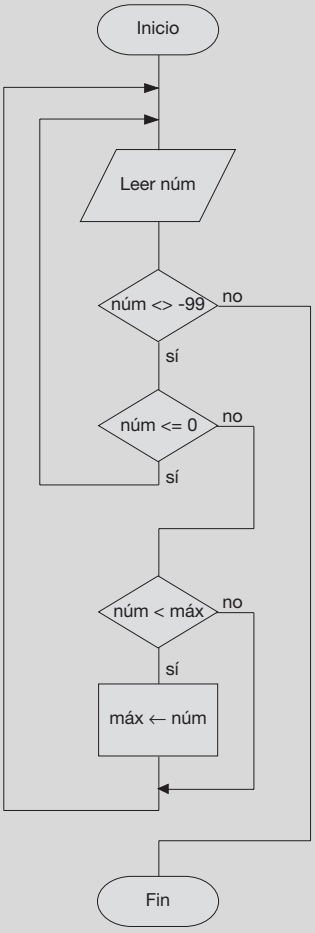
Análisis del problema

DATOS DE SALIDA: máx (el número mayor de la serie)
DATOS DE ENTRADA: núm (cada uno de los números que introducimos)

Después de leer un número e inicializar máx a ese número, se ejecutará un bucle mientras el último número leído sea distinto de -99. Dentro del bucle se debe controlar que el número sea distinto de 0. Si el número es 0 se vuelve a leer hasta que la condición sea falsa. También hay que comprobar si el último número leído es mayor que el máximo, en cuyo caso el nuevo máximo será el propio número.

Diseño del algoritmo

TABLA DE VARIABLES:
entero : máx,núm



3.3. *Calcular y visualizar la suma y el producto de los números pares comprendidos entre 20 y 400, ambos inclusive.*

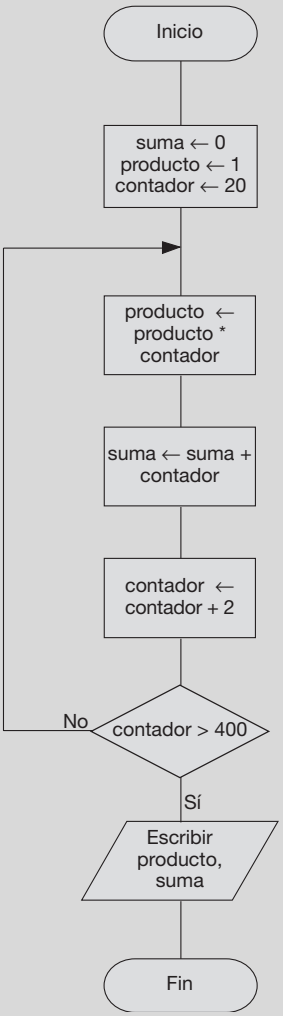
Análisis del problema

DATOS DE SALIDA: `suma`, `producto`
DATOS AUXILIARES: `contador`

Para solucionar el algoritmo, se deben inicializar los acumuladores `suma` y `producto` a 0 y la variable `contador` a 20 (puesto que se desea empezar desde 20) e implementar un bucle que se ejecute hasta que la variable `contador` valga 200. Dentro del bucle se irán incrementando los acumuladores `suma` y `producto` con las siguientes expresiones: `suma ← suma+contador` y `producto ← producto*contador`. Una vez realizadas las operaciones se debe incrementar el contador. Como se desea utilizar sólo los números pares, se usará una expresión como `contador ← contador+2`.

Diseño del algoritmo

TABLA DE VARIABLES:
`entero` : `contador,suma,producto`



3.4. *Leer 500 números enteros y obtener cuántos son positivos.*

Análisis del problema

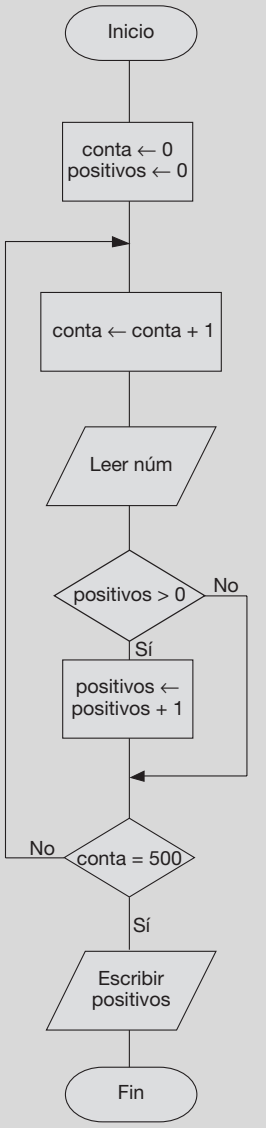
DATOS DE SALIDA: `positivos` (contiene la cantidad de números positivos introducidos)
DATOS DE ENTRADA: `núm` (los números que introducimos)
DATOS AUXILIARES: `conta` (se encarga de contar la cantidad de números introducidos)

Se diseña un bucle que se ejecute 500 veces, controlado por la variable `conta`. Dentro del bucle se lee el número y se comprueba si es mayor que 0, en cuyo caso se incrementa el contador de positivos (`positivos`).

Diseño del algoritmo

TABLA DE VARIABLES:

```
entero : positivos, núm, conta
```



3.5. *Se trata de escribir el algoritmo que permita emitir la factura correspondiente a una compra de un artículo determinado del que se adquieren una o varias unidades. El IVA (Impuesto de Valor Añadido) a aplicar es del 12% y si el precio bruto (precio de venta + IVA) es mayor de 50.000 pesetas, se aplicará un descuento del 5%.*

Análisis del problema

DATOS DE SALIDA: `bruto` (el precio bruto de la compra, con o sin descuento)
DATOS DE ENTRADA: `precio` (precio sin IVA), `unidades`
DATOS AUXILIARES: `neto` (precio sin IVA), `iva` (12% del neto)

Después de leer el precio del artículo y las unidades compradas, se calcula el precio `neto (precio * unidades)`. Se calcula también el `IVA(neto * 0.12)` y el `bruto (neto + iva)`. Si el `bruto` es mayor que `50.000` pesetas, se le descuenta un `5% (bruto ← bruto * 0.95)`.

Al multiplicar el valor `neto` por un valor real `(0.12)` para obtener el `IVA`, y calcular el `bruto` a partir del `IVA`, ambos deben ser datos reales.

Diseño del algoritmo

```
algoritmo ej_3_5
var
   entero : precio,neto,unidades
   real : iva,bruto
inicio
   leer (precio,unidades)
   neto ← precio * unidades
   iva ← neto * 0.12
   bruto ← neto + iva
   si bruto < 50000 entonces
      bruto ← bruto * 0.95
   fin_si
   escribir (bruto)
fin
```

3.6. *Calcular la suma de los cuadrados de los 100 primeros números naturales.*

Análisis del problema

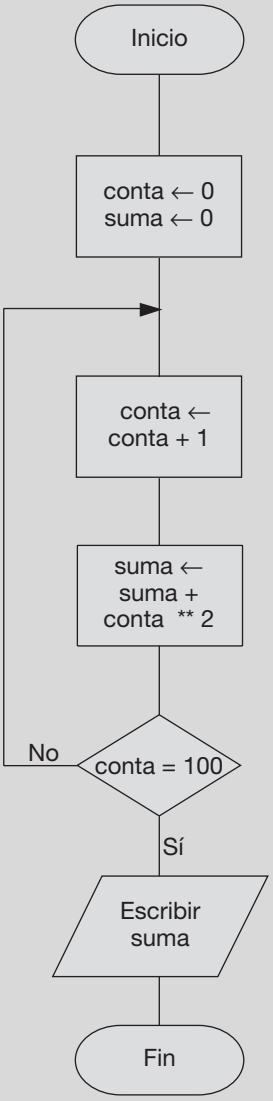
DATOS DE SALIDA: `suma` (acumula los cuadrados del número)
DATOS AUXILIARES: `conta` (contador que controla las iteraciones del bucle)

Para realizar este programa es necesario un bucle que se repita `100` veces y que se controlará por la variable `conta`. Dentro de dicho bucle se incrementará el contador y se acumulará el cuadrado del contador en la variable `suma`.

Diseño del algoritmo

TABLA DE VARIABLES:

```
entero : suma, conta
```



3.7. *Sumar los números pares del 2 al 100 e imprimir su valor.*

Análisis del problema

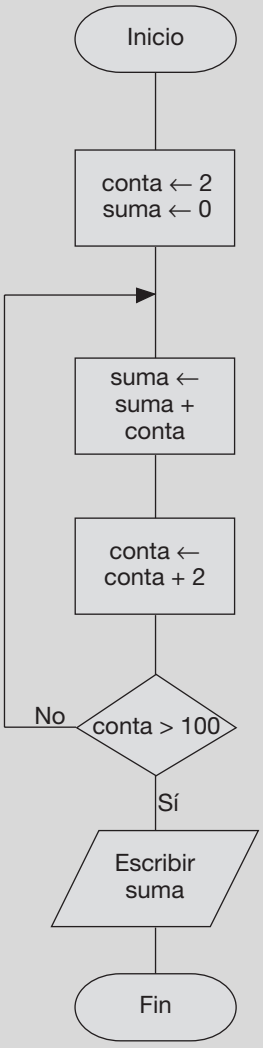
DATOS DE SALIDA: `suma` (contiene la suma de los números pares)
DATOS AUXILIARES: `conta` (contador que va sacando los números pares)

Se trata de hacer un bucle en el que un contador (`conta`) vaya incrementando su valor de dos en dos y, mediante el acumulador `suma`, y acumulando los sucesivos valores de dicho contador. El contador se deberá inicializar a 2 para que se saquen números pares. El bucle se repetirá hasta que `conta` sea mayor que 100.

Diseño del algoritmo

TABLA DE VARIABLES:

```
entero : conta,suma
```



3.8. *Sumar 10 números introducidos por teclado.*

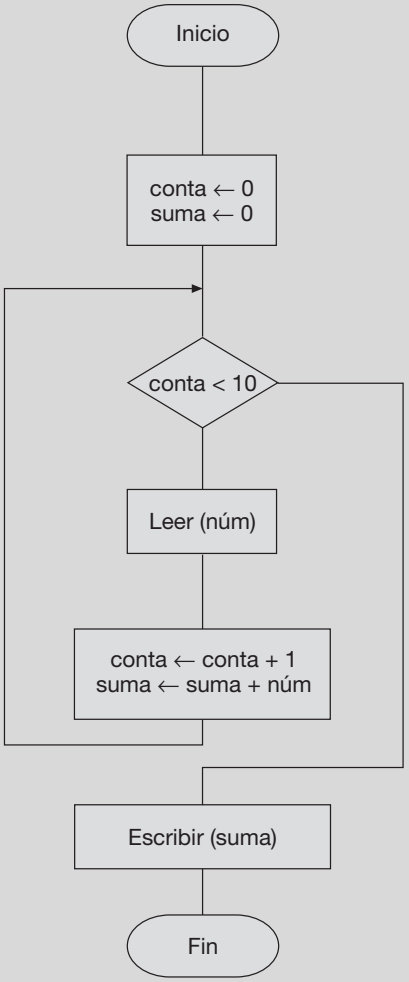
Análisis del problema

DATOS DE SALIDA: `suma` (suma de los números)
DATOS DE ENTRADA: `núm` (los números que introducimos por teclado)
DATOS AUXILIARES: `conta` (contador que controla la cantidad de números introducidos)

Después de inicializar `conta` y `suma` a 0, se debe implementar un bucle que se ejecute 10 veces. En dicho bucle se incrementará el contador `conta` en una unidad, se introducirá por teclado `núm` y se acumulará su valor en `suma`. El bucle se ejecutará mientras que `conta` sea menor que 10.

Diseño del algoritmo

TABLA DE VARIABLES
`entero` `: suma, núm, conta`



3.9. *Calcular la media de 50 números introducidos por teclado y visualizar su resultado.*

Análisis del problema

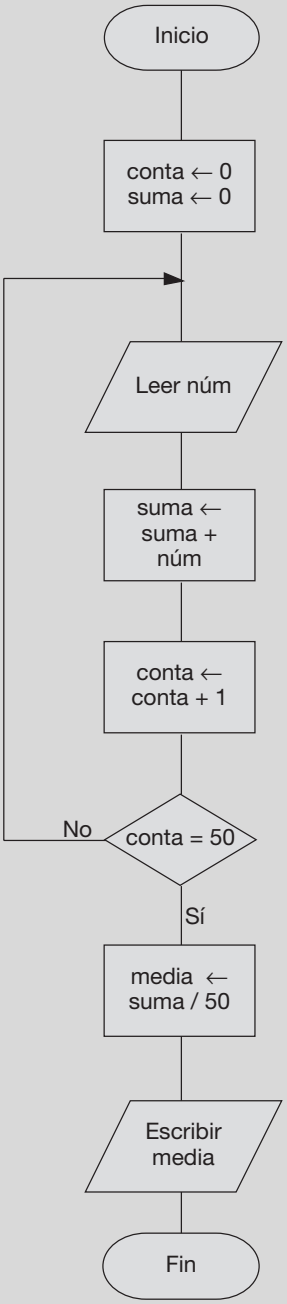
DATOS DE SALIDA: `media` (contiene la media de los cincuenta números)
DATOS DE ENTRADA: `núm` (cada uno de los cincuenta números introducidos por teclado)
DATOS AUXILIARES: `conta` (contador que controla la cantidad de números introducidos), `suma` (acumula el valor de los números)

Después de inicializar `conta` y `suma`, se realiza un bucle que se repetirá 50 veces. En dicho bucle se lee un número (`núm`), se acumula su valor en `suma` y se incrementa el contador `conta`. El bucle se repetirá hasta que `conta` sea igual a 50. Una vez fuera del bucle se calcula la media (`suma/50`) y se escribe el resultado.

Diseño del algoritmo

TABLA DE VARIABLES

```
entero : núm,conta,suma
real : media
```



3.10. Visualizar los múltiplos de 4 comprendidos entre 4 y N, donde N es un número introducido por teclado.

Análisis del problema

DATOS DE SALIDA: `múltiplo` (cada uno de los múltiplos de 4)
DATOS DE ENTRADA: `N` (número qué indica hasta que múltiplo vamos a visualizar)
DATOS AUXILIARES: `conta` (contador que servirá para calcular los múltiplos de 4)

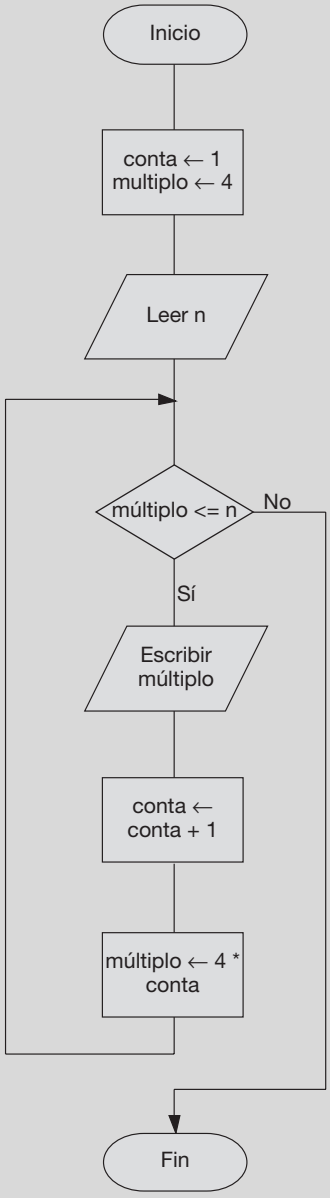
Para obtener los múltiplos de cuatro se pueden utilizar varios métodos. Un método consiste en sacar números correlativos entre 4 y `N` y para cada uno comprobar si es múltiplo de 4 mediante el operador **`mod`**. Otro método sería utilizar un contador que arrancando en 4 se fuera incrementado de 4 en 4. Aquí simplemente se irá multiplicando la constante 4 por el contador, que tomará valores a partir de 1.

Para realizar el algoritmo, después de inicializar `conta` a 1 y `múltiplo` a 4 y leer el número de múltiplos que se desean visualizar, se debe ejecutar un bucle mientras que `múltiplo` sea menor que `N`. Dentro del bucle hay que visualizar `múltiplo`, incrementar el contador en 1 y calcular el nuevo múltiplo (`4 * conta`).

Diseño del algoritmo

TABLA DE VARIABLES

```
entero : múltiplo,N,conta
```



3.11. Realizar un diagrama que permita realizar un contador e imprimir los 100 primeros números enteros.

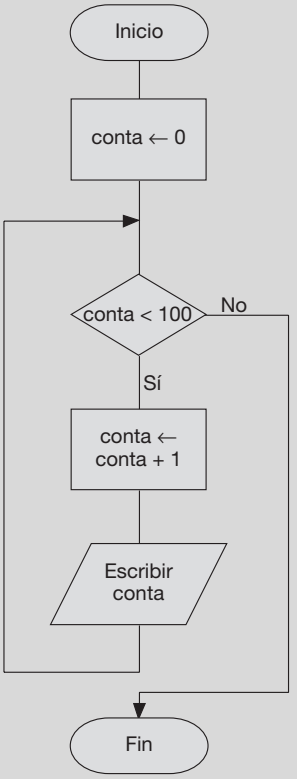
Análisis del problema

DATOS DE SALIDA: Los números enteros entre 1 y 100
DATOS AUXILIARES: `conta` (controla el número de veces que se ejecuta el bucle)

Se debe ejecutar un bucle 100 veces. Dentro del bucle se incrementa en 1 el contador y se visualiza éste. El bucle se ejecutará mientras `conta` sea menor que 100. Previamente a la realización del bucle, `conta` se inicializará a 0.

Diseño del algoritmo

TABLA DE VARIABLES
`entero` : `conta`



3.12. *Dados 10 números enteros que introduciremos por teclado, visualizar la suma de los números pares de la lista, cuántos números pares existen y cuál es la media aritmética de los números impares.*

Diseño del algoritmo

DATOS DE SALIDA: `spar` (suma de pares), `npar` (cantidad de números pares), `media` (media de números impares)
DATOS DE ENTRADA: `núm` (cada uno de los números introducidos por teclado)
DATOS AUXILIARES: `conta` (contador que controla la cantidad de números introducidos), `simpar` (suma de los números impares), `nimpar` (cantidad de números impares)

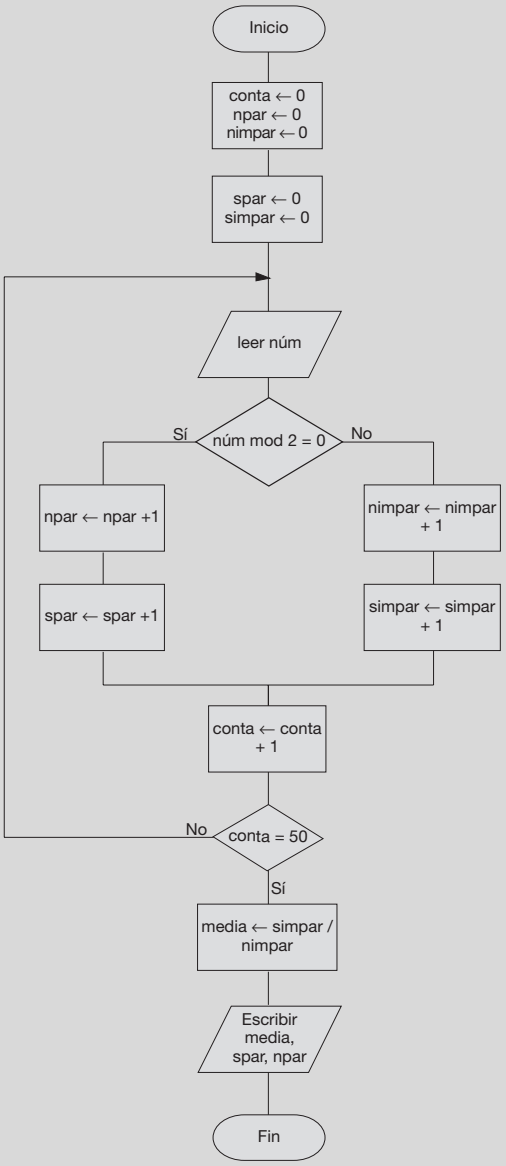
Se ha de realizar un bucle 10 veces. En ese bucle se introduce un número y se comprueba si es par mediante el operador **mod**. Si es par se incrementa el contador de pares y se acumula su valor en el acumulador de pares. En caso contrario se realizan las mismas acciones con el contador y el acumulador de impares. Dentro del bucle también se ha de incrementar el contador `conta` en una unidad. El bucle se realizará hasta que `conta` sea igual a 10.

Ya fuera del bucle se calcula la media de impares (`simpar/nimpar`) y se escribe `spar`, `npar` y `media`.

Diseño del algoritmo

TABLA DE VARIABLES

```
entero : conta,núm,spar,npar,simpar,nimpar
real :   media
```



3.13. Calcular la nota media por alumno de una clase de `a` alumnos. Cada alumno podrá tener un número `n` de notas distinto.

Análisis del problema

DATOS DE SALIDA:	`media` (media de cada alumno que también utilizaremos para acumular las notas)
DATOS DE ENTRADA:	`a` (número de alumnos), `n` (número de notas de cada alumno), `nota` (nota de cada alumno en cada una de las asignaturas)
DATOS AUXILIARES:	`contaa` (contador de alumnos), `contan` (contador de notas)

Para realizar este algoritmo se han de realizar dos bucles anidados. El primero se repetirá tantas veces como alumnos. El bucle interno se ejecutará por cada alumno tantas veces como notas tenga éste.

En el bucle de los alumnos se lee el número de notas y se inicializan las variables `media` y `contan` a 0. Aquí comenzará el bucle de las notas en el que leeremos una nota, se acumula en la variable `media` y se incrementa el contador de notas. Este bucle se ejecutará hasta que el contador de notas sea igual a `n` (número de notas).

Finalizado el bucle interno, pero todavía en el bucle de los alumnos, se calcula la media (`media/n`), se escribe y se incrementa el contador de alumnos. El programa se ejecutará hasta que `contaa` sea igual a `a`.

Diseño del algoritmo

TABLA DE VARIABLES

```
entero : a,n,contaa,contan
real : media,nota
```