CAPÍTULO 7

PROPUESTA DE EJERCICIOS

A lo largo de este capítulo recogemos una serie de ejercicios que sirvan para poner en práctica los conceptos y técnicas presentados en los capítulos anteriores. Además, pueden servir para intentar implementar esos algoritmos en un programa escrito en un lenguaje concreto, como Java ó C.

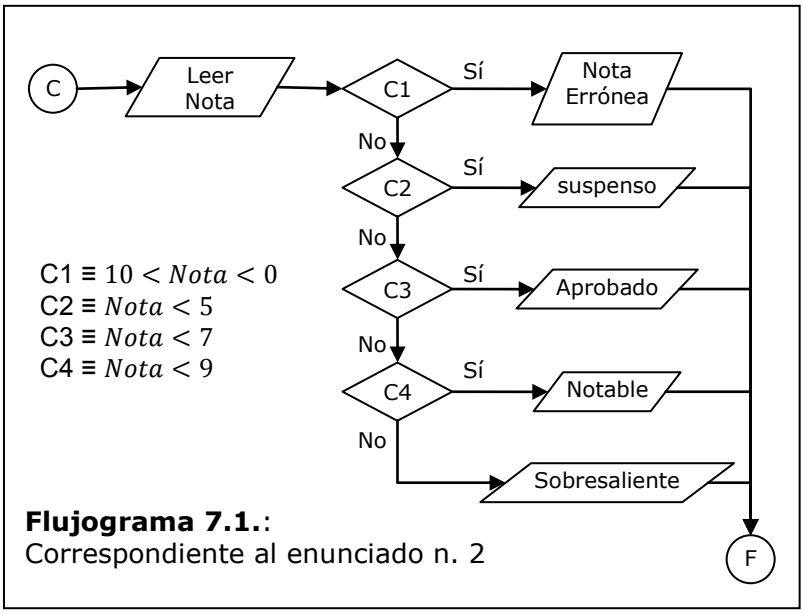
Ejercicios propuestos y resueltos.

1. Presentar el algoritmo para un programa que resuelva una ecuación de primer grado ($a \cdot x + b = 0$). El programa solicita los valores de los parámetros a y b y debe mostrar el valor resultante de la variable x.

1. [Entrada de datos] **Leer** a y b.
2. [Cálculos] **Si** $a \neq 0$ **Entonces** [Mostrar resultado]: **Mostrar** $-b/a$.
   Sino [es decir, $a = 0$]
   2.1. **Entonces** [Mostrar mensaje]: **Mostrar** "No hay Ecuación"

3. **Fin**.

2. Presentar el algoritmo para un programa que recibe del usuario una nota numérica (entre 0 y 10) y devuelve por pantalla la calificación académica correspondiente: suspenso, notable, etc.



Flujograma 7.1.:
Correspondiente al enunciado n. 2

Algoritmo muy sencillo, que se construye con una sucesión anidada de estructuras de control condicionales:

1. [Entrada de la nota]: **Leer** $Nota$.
2. **Si** $10 < Nota < 0$ **Entonces** [Mensaje]: "Nota incorrecta".
   Si no, Entonces:
   2.1. **Si** $Nota < 5$, **Entonces** [Mensaje]: "Suspenso".
   Si no, Entonces:
   2.1.1. **Si** $Nota < 7$, **Entonces** [Mensaje]: "Aprobado".
   Si no, **Entonces**:
   2.1.1.1. **Si** $Nota < 9$, **Entonces** [Mensaje]: "Notable".
   2.1.1.2. **Si no**, **Entonces** [Mensaje]: "Sobresaliente".
3. **Fin**.

Podemos ver el flujograma de este algoritmo en el Cuadro que hemos llamado Flujograma 7.1.

3. Escriba un programa que calcule a qué distancia caerá un proyectil lanzado desde un cañón. El programa recibe desde teclado el ángulo inicial de salida del proyectil (α) y su velocidad inicial (V_0). Se supone que el cañón está en el suelo y que el proyectil cae más allá, pero a la misma altura: es decir, que el suelo es horizontal.

Tenga en cuenta las siguientes ecuaciones que definen el comportamiento del sistema descrito:

(1) $V_x = V_0 . \cos \alpha$ ***(2)*** $V_y = V_0 . sen\ \alpha - g \cdot t$

(3) $x = V_0 . t \cdot cos\ \alpha$ ***(4)*** $y = V_0 . t \cdot sen\ \alpha - \frac{1}{2} \cdot g \cdot t^2$.

Este problema es muy sencillo. Pero hay que saber resolverlo. Como en muchos de los problemas a los que se enfrenta un programador, la miga o el reto no está en el código a escribir, sino en conocer un camino que resuelva el problema.

Por eso, antes de comenzar siquiera a pensar en un algoritmo, debemos resolver el problema desde un punto de vista matemático.

De las cuatro expresiones presentadas, la (1) no nos sirve para nada. Averiguamos la distancia calculando, mediante la expresión (2), el tiempo para que $V_y = 0$ (cuando llega a la máxima altura) y multiplicando por dos; o calculando, mediante la expresión (4), el tiempo para que y vuelva a ser igual a cero. Una vez calculado el tiempo, se sustituye éste en la expresión (3) y obtenemos el alcance. Y para decidir que eso es así, no hay que aprender a programar: es una decisión basada en los conocimientos de cinemática.

Tomamos el segundo camino indicado. Hemos quedado que suponemos que el cañón está a la misma altura que el suelo donde caerá el proyectil. En ese momento tendremos que $y = 0$. Buscamos en qué

instantes ocurre esta igualdad. Para ello, resolviendo la ecuación (4) llegamos a que

$y = 0$ $t = 0$: Instante inicial cuando se dispara el proyectil.
$t = 2 \cdot V_0 \cdot sen\ \alpha / g$: Instante en que cae el proyectil.

Desde luego, en el instante inicial el proyectil está a la altura del suelo. A nosotros nos interesa el segundo momento en que vuelve a estar en el suelo. Y en ese instante, sustituyendo el valor de t en la expresión (3), calcular el valor de la x:

$x = V_0 \cdot \cos\alpha \cdot t \implies x = 2 \cdot V_0^2 \text{ " } sen\ \alpha \cdot \cos\alpha / g$.

Así, con esta expresión, el algoritmo es muy sencillo:

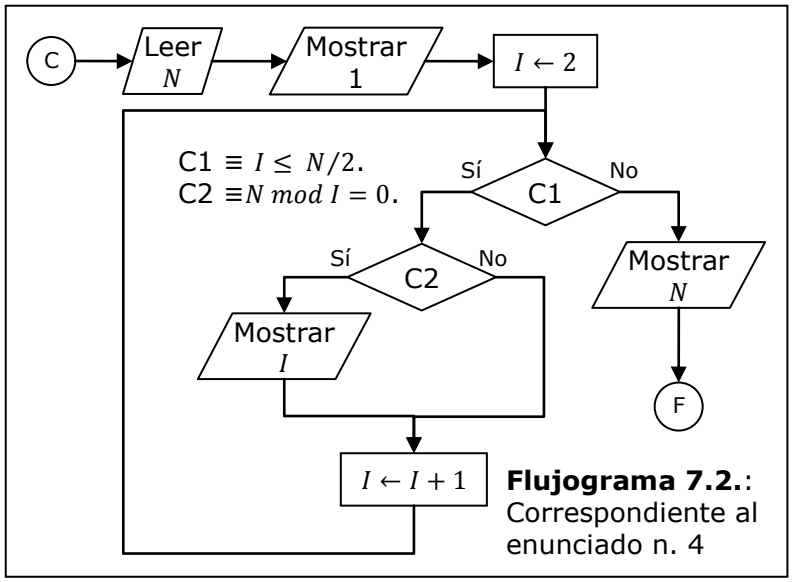
1. [Entrada de datos]: **Leer** V_0 y α.
2. [Cálculos]
   2.1. $x \leftarrow V_0^2 \cdot sen\ \alpha \cdot \cos\alpha$.
   2.2. $x \leftarrow x/g$.
3. [Mostrar resultados]: **Mostrar** x.
4. **Fin**.

4. Presentar un algoritmo que reciba un entero y muestre todos los enteros que lo dividen. Tener en cuenta que los divisores de un entero cualquiera se encuentran entre el 1 y la mitad del entero del que se buscan los divisores; finalmente se añade a la lista el mismo número, que es siempre divisor de sí mismo.

Con las indicaciones recogidas en el enunciado, el algoritmo tiene fácil presentación:

1. [Entrada de datos]: **Leer** N (El número a factorizar)
2. [Mostrar Divisor]: **Mostrar** 1.
3. **Para** $I \leftarrow 2$ **Hasta** $N/2$, **Repetir**:
   3.1. **Si** $N\ mod I = 0$ **Entonces** [Mostrar Divisor]: **Mostrar** I.
   3.2. $I \leftarrow I + 1$.
4. [Mostrar Divisor]: **Mostrar** N.
5. **Fin**.

El Flujograma de este algoritmo puede verse en el Cuadro que hemos llamado Flujograma 7.2.



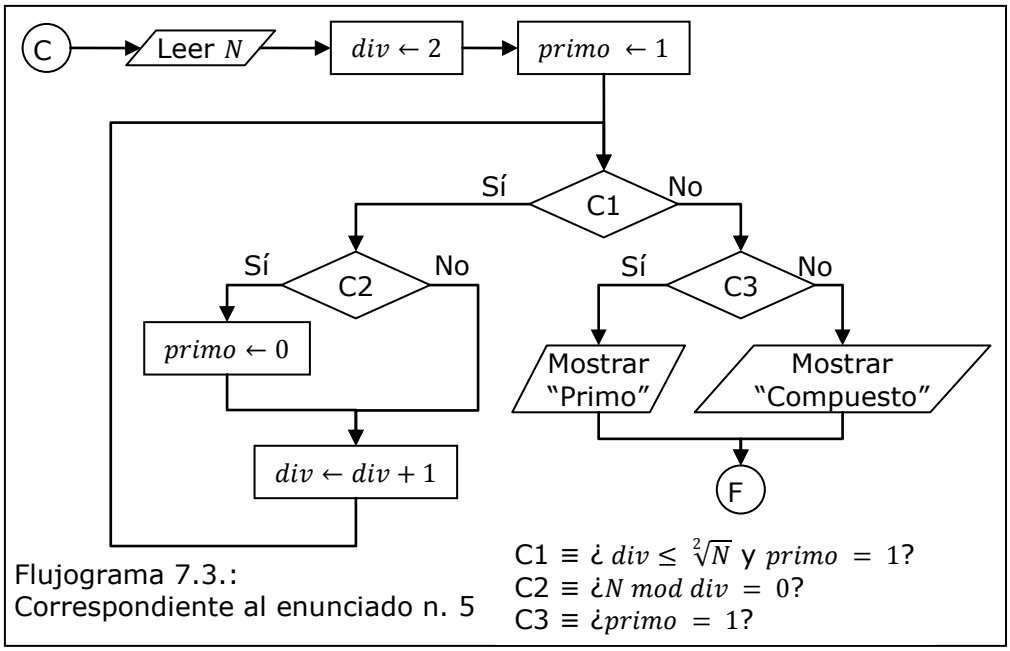
Flujograma 7.2.: Correspondiente al enunciado n. 4

5. Presentar un algoritmo que reciba un entero por teclado y diga si es primo o compuesto.

El procedimiento más usado para determinar esa propiedad de los números, cuando estos son menores de diez millones, es la búsqueda exhaustiva de un entero que resulte ser divisor propio del número analizado. Si entre el 2 y la raíz cuadrada del entero analizado no hay ningún divisor, entonces ese número es primo.

1. [Entrada de datos]: **Leer** N.
2. [Inicializar variables] $div \leftarrow 2$, $primo \leftarrow 1$.
3. **Mientras** $div \leq \sqrt[2]{N}$ **y** $primo = 1$ **Repetir**:
   3.1. **Si** $N \; mod \; div = 0$ **Entonces**:
      3.1.1. $primo \leftarrow 0$
   3.2. $div \leftarrow div + 1$
4. **Si** $primo = 1$ **Entonces**
   4.1. [Mostrar Resultado] **Mostrar** "N es Primo."
   Sino Entonces:
   4.2. **[**Mostrar Resultado] **Mostrar** "N es Compuesto."
5. **Fin**.

El Flujograma de este algoritmo puede verse en el Cuadro que hemos llamado Flujograma 7.3.



Flujograma 7.3.:
Correspondiente al enunciado n. 5

La variable primo hace el papel de una variable chivato: su valor sólo sirve para determinar si ha pasado algo dentro de una estructura de control iterada una vez ya hemos salido de ella.

El realidad esa variable no hubiera sido necesaria. El algoritmo funcionaría de la misma manera si eliminamos la condición C2 y la sentencia que condiciona. La condición de permanencia en la iteración (la condición C1) sería $div \leq \sqrt[2]{N}$ y $N \ mod \ div \neq 0$. Y la condición C3 sería: $N \ mod \ div \neq 0$.

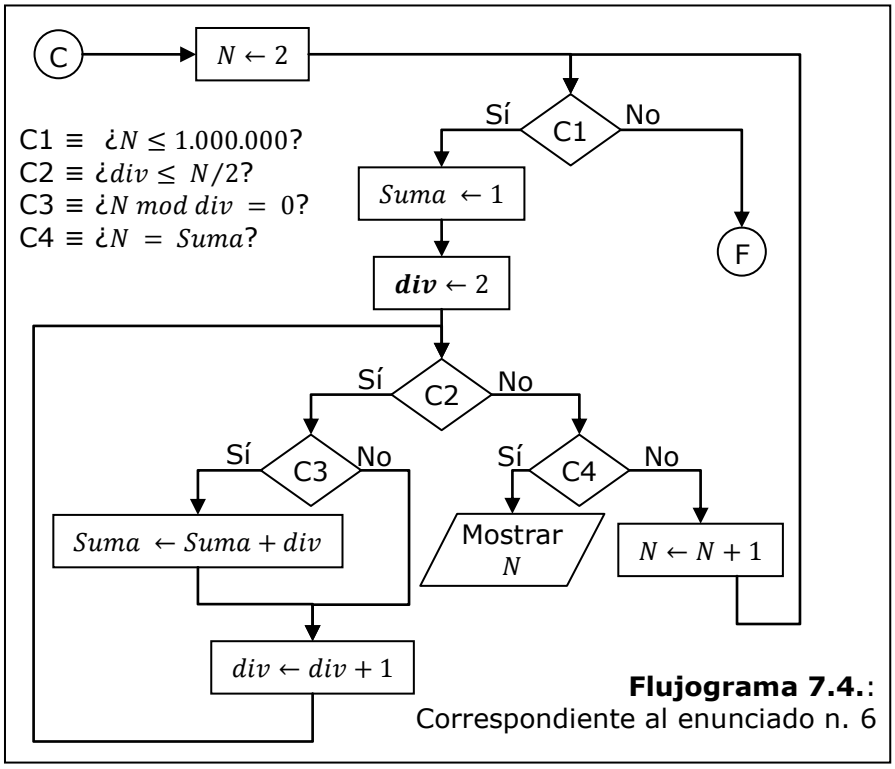
Queda como trabajo personal rehacer el nuevo flujograma y el nuevo pseudocódigo.

6. Se llama número perfecto a aquel que es igual a la suma de sus divisores (sin contar, claro está, consigo mismo) Por ejemplo, el

número 6 tiene como divisores los enteros 1, 2 y 3, que, efectivamente, suman 6; el número 28 tiene como divisores el 1, 2, 4, 7 y 14, que de nuevo efectivamente, suman 28; el 18 tiene como divisores el 1, 2, 3, 6 y 9, que suman 21, por lo que este último número no es perfecto; los dos primeros sí lo han sido.

Presentar un algoritmo que muestre los números perfectos entre el 1 y un millón.

Este algoritmo requiere buscar todos los divisores de un entero. Ya hemos visto más arriba cómo se hace esto. Ahora, además de buscar los divisores, lo que debemos hacer es sumarlos.



Flujograma 7.4.: Correspondiente al enunciado n. 6

El algoritmo queda de la siguiente manera.

1. **Para** $N \leftarrow 2$ **Hasta** 1.000.000 **Repetir**:
   1.1. $Suma \leftarrow 1$ (ya tomamos sumado el primer divisor)
   1.2. **Para** $\boldsymbol{div} \leftarrow 2$ **y Mientras** $div \leq N/2$ **Repetir**:

1.2.1. **Si** $N \; mod \; div \; = \; 0$ **Entonces**:
1.2.1.1. $Suma \leftarrow Suma + div$
1.2.2. $div \leftarrow div + 1$
1.3. **Si** $N \; = \; Suma$ **Entonces**:
[Mostrar Valor]: **Mostrar** N (es número perfecto)
1.4. $N \leftarrow N + 1$
2. **Fin**.

El Flujograma de este algoritmo puede verse en el Cuadro que hemos llamado Flujograma 7.4.

7. Cinco marineros llegan, tras un naufragio, a una isla desierta con un gran número de cocoteros y un pequeño mono. Dedican el primer día a recolectar cocos, pero ejecutan con tanto afán este trabajo que acaban agotados, por lo que deciden repartirse los cocos al día siguiente.

Durante la noche un marinero se despierta y, desconfiando de sus compañeros, decide tomar su parte. Para ello, divide el montón de cocos en cinco partes iguales, sobrándole un coco, que regala al mono. Una vez calculada su parte la esconde y se vuelve a acostar.

Un poco más tarde otro marinero también se despierta y vuelve a repetir la operación, sobrándole también un coco que regala al mono. En el resto de la noche sucede lo mismo con los otros tres marineros.

Al levantarse por la mañana procedieron a repartirse los cocos que quedaban entre ellos cinco, no sobrando ahora ninguno.

¿Cuántos cocos habían recogido inicialmente? Mostrar todas las soluciones posibles menores de 1 millón de cocos.

Vamos a presentar la solución más general: Supondremos que tenemos N marineros y que, cada vez que uno de ellos se levanta durante la

noche para hacer N partes sobran m cocos que se le dan al mono. Así, el caso del enunciado no es más que un caso particular de la solución que aquí mostramos.

Este problema tiene poco de informática y bastante de pensar cómo dar con la solución. Su programación es sencilla… una vez se ha logrado saber cómo dar con la respuesta. Por eso, sería útil que quien quiera aprender a programar una aplicación que resuelva este ejercicio, antes lo intentara resolver en un papel.

El número de cocos (lo llamaremos C) debe ser tal que al llegar el primer marinero ocurra que haga N montones y sobren m cocos para el mono. Es decir, C es tal que $C \; mod \; N = m$.

Cuando el primer marinero se acuesta de nuevo, se ha llevado y escondido una porción de los cocos. Además ha entregado m al mono. Por lo tanto, los cocos que quedan son $C \leftarrow (N-1) \cdot (C-m)/N$.

Ahora se levanta el segundo marinero. Y se repite la faena: Vuelve a hacer N montones, vuelven a sobrar m cocos, que se los da al mono, y se lleva su parte, dejando los restantes en el menguado montón inicial. Quedan, por tanto, $C \leftarrow (N-1) \cdot (C-m)/N$ cocos, donde ahora C es el número de cocos que ha dejado el marinero primero, y no el número total de cocos recolectado.

Y le toca ahora el turno al tercer marinero. Y vuelve a hacer N montones, vuelven a sobrar m cocos, que se los da al mono, y se lleva su parte, dejando los restantes en el cada vez más menguado montón inicial. Quedan, por tanto, $C \leftarrow (N-1) \cdot (C-m)/N$ cocos, donde ahora C es el número de cocos que ha dejado el marinero segundo, y no el número total de cocos recolectado.

Y así, van desfilando todos los marineros. Y todos realizan la misma operación. Y siempre ocurre que sobran, después de hacer N montones, los m cocos para el mono.

Al final, cuando ya han pasado los N marineros, se hace de día, y todos se levantan como si tal cosa, y reparten los cocos en N montones, y esta vez el mono se queda sin coco alguno. Es decir, al final, después de N veces en que se verifica que $C \; mod \; N = m$, y después de menguar el valor de C de acuerdo con la expresión $C \leftarrow (N-1) \cdot (C-m)/N$, ahora se cumple que $C \; mod \; N = 0$.

Vamos a expresar esto con un algoritmo:

1. [Entrada de datos] **Leer** N y m.
2. $cocos \leftarrow 1$.
3. **Mientras** $cocos < 1.000.000$ **Repetir**
   3.1. $C \leftarrow cocos$.
   3.2. **Para** $I \leftarrow 1$ **Hasta** N **Repetir**:
      3.2.1. **Si** $C \; mod \; N = m$ **Entonces**
         3.2.1.1. $C \leftarrow (N-1) \cdot (C-m)/N$.
         3.2.1.2. $I \leftarrow I + 1$.
      3.2.2. **Sino** [Es decir, $C \; mod \; N \neq m$] **Entonces**
         3.2.2.1. [Abandonar la iteración]: Ir a paso 3.4.
   3.3. **Si** $C \; mod \; N = 0$ **y** $I > N$ **Entonces**
      3.3.1. [Mostrar valor]: **Mostrar** $cocos$.
   3.4. $cocos \leftarrow cocos + 1$.
4. **Fin**.

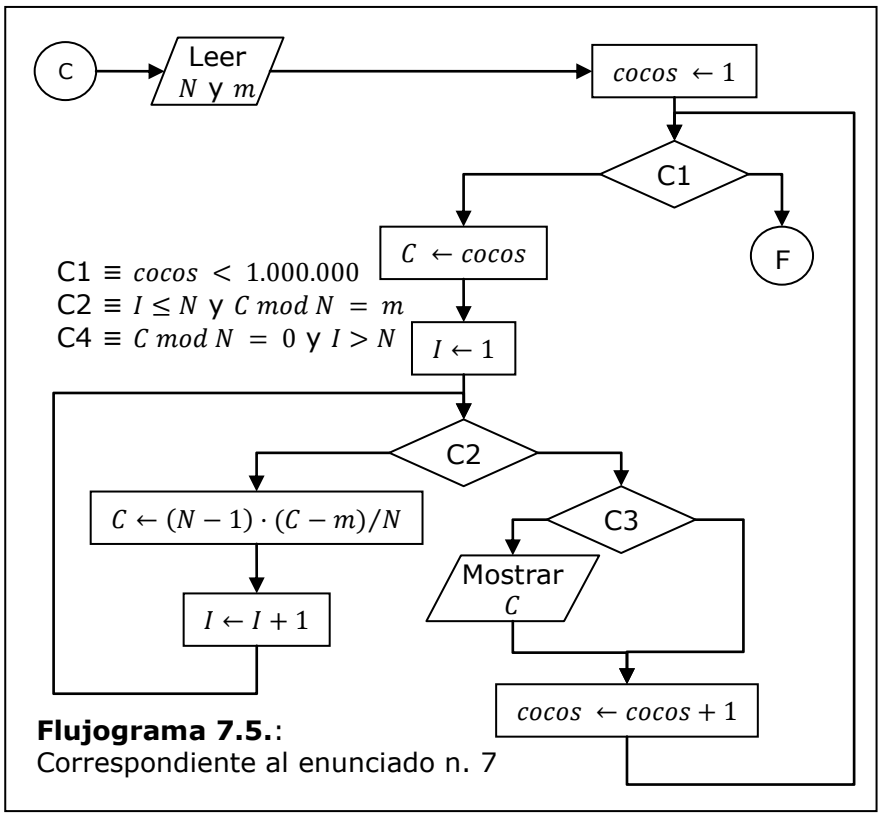
La sentencia 3.3.2.1. no es un salto de los prohibidos. Existen en muchos lenguajes (y C y Java entre ellos) sentencias de salto que se emplean para abandonar un proceso iterativo y saltar a la siguiente instrucción después de la iteración. Esta sentencia de salto nos saca de la iteración controlada por la variable I y nos deja en la siguiente instrucción o sentencia: la 3.4. En esa sentencia se evalúan dos cosas: primero si $I > N$; y luego si $C \; mod \; N = 0$. La segunda evaluación es la última condición que se debe verificar para el número de cocos: después de haber pasado todos los marineros durante la noche, todos se reúnen por la mañana y, a la hora de hacer los N montones, tenemos ahora que no sobra ningún coco.

Para la primera condición, conviene ver el funcionamiento del algoritmo. La variable I estará a un valor igual a $N + 1$ si y sólo si en ningún momento ha pasado por la sentencia 3.3.2.1. Si en alguno de los

sucesivos valores de C no se cumple que $C \, mod \, N = m$, entonces se abandona la iteración controlada por la I porque el valor actual de cocos que se está evaluando no ha cumplido todas las condiciones exigidas y no es por tanto solución a nuestro problema. Y al producirse ese abandono la variable I no alcanza el valor $N + 1$. Es decir, si $I \leq N$ quiere decir que el valor de cocos ha fallado en algún momento algunas de las muchas condiciones que se le deben exigir. A una variable que, además de controlar la ejecución de una iteración, de alguna manera informa del comportamiento dentro del bloque iterado se la llama variable **centinela**.

Una forma más reducida el algoritmo, sin la sentencia de abandono de la estructura de iteración, es la siguiente:

1. [Entrada de datos] **Leer** N y m.
2. $cocos \leftarrow 1$.
3. **Mientras** $cocos < 1.000.000$ **Repetir**



Flujograma 7.5.:
Correspondiente al enunciado n. 7

3.1. $C \leftarrow cocos$.
3.2. **Para** $I \leftarrow 1$ **Hasta** N **y Mientras** $C \bmod N = m$ **Repetir**:
3.2.1. $C \leftarrow (N-1) \cdot (C-m)/N$.
3.2.2. $I \leftarrow I + 1$.
3.3. **Si** $C \bmod N = 0$ **y** $I > N$ **Entonces**
3.3.1. [Mostrar valor]: **Mostrar** $cocos$.
3.4. $cocos \leftarrow cocos + 1$.
4. **Fin**.

Éste es el algoritmo que queda representado en el Flujograma 7.5.

8. El calendario juliano (debido a julio Cesar) consideraba que el año duraba 365.25 días, por lo que se estableció que los años tendrían una duración de 365 días y cada cuatro años se añadiese un día más (año bisiesto).

Sin embargo se comprobó que en realidad el año tiene 365.2422 días, lo que implica que el calendario juliano llevase un desfase de unos once minutos. Este error es relativamente pequeño, pero, con el transcurrir del tiempo, el error acumulado puede ser importante.

El papa Gregorio XII, en 1582, propuso reformar el calendario juliano para evitar los errores arrastrados de años anteriores. Los acuerdos tomados entonces, que son por los que nos aún nos regimos, fueron los siguientes:

- ***Para suprimir el error acumulado por el calendario juliano, se suprimieron diez días. De tal manera que el día siguiente al 4 de octubre de 1582 fuel el día 15 del mismo mes.***
- ***La duración de los años sería de 365 días o 366 en caso de ser bisiesto.***
- ***Serán bisiestos todos los años que sean múltiplos de 4, salvo los que finalizan en 00, que sólo lo serán cuando también sean múltiplos de 400, por ejemplo 1800 no fue bisiesto y el 2000 sí.***

- ***Con esta reforma el desfase existente entre el año civil y el año real se reduce a menos de treinta segundos anuales.***

Definir un algoritmo que solicite al usuario una fecha introducida mediante tres datos: día, mes y año; ese programa debe validar la fecha: es decir comprobar que la fecha es correcta cumpliendo las siguientes reglas:

- ***El año debe ser mayor que 0.***
- ***El mes debe ser un número entre uno y doce.***
- ***El día debe estar entre 1 y 30, 31,28 ó 29 dependiendo el mes de que se trate y si el año es bisiesto o no.***

Una vez se hay introducido el valor del día (variable *dd*), del mes (variable *mm*) y del año (variable *aa*), el algoritmo decidirá si esa fecha es válida o no, de acuerdo con las indicaciones recogidas en el enunciado. El algoritmo queda de la siguiente forma:

1. **Si** C1 **Entonces**: $B \leftarrow True$. (Año bisiesto)
2. **Si no**, **Entonces** $B \leftarrow False$. (Año no bisiesto)
3. **Si** C2 **Entonces**:
   [Mostrar Mensaje]: "Fecha Errónea".
   Si no, **Entonces**:
   3.1.1. **Si** C3 **Entonces**:
   [Mostrar Mensaje]: "Fecha Errónea".
   Si no, **Entonces**:
   3.1.1.1. **Si** C4 **Entonces**:
   [Mostrar Mensaje]: "Fecha Errónea".
   Si no, **Entonces**
   [Mostrar Mensaje]: "Fecha Correcta".
4. **Fin**.

Donde las condiciones resumidas son las recogidas a continuación.

C1 ≡ $aa \bmod 4 = 0$ y ($aa \bmod 100 \neq 0$ ó ($aa \bmod 100 = 0$ y $aa \bmod 400 \neq 0$)).
C2 ≡ ($aa < 0$) ó ($mm < 0$) ó ($mm > 12$) ó ($dd < 0$) ó ($dd > 31$) ó
($(aa = 1582)$ y ($mm = 10$) y ($dd \leq 14$)).
C3 ≡ $m \in \{4, 6, 9, 11\}$ y $d = 31$
C4 ≡ $m = 2$ y ($d > 29$ ó ($B = False$ y $d > 28$))

Queda pendiente, como trabajo a resolver por el alumno, la confección del flujograma.

9. Defina un algoritmo que recibe por teclado las calificaciones de cada uno de los alumnos de una clase. Las notas deber ser siempre comprendidas entre el cero (0) y el diez (10). Cualquiera otra nota no la tomará en consideración. La introducción de datos terminará cuando se introduzca la nota menos 1 (-1).

Al terminar la introducción de datos, el programa debe mostrar por pantalla la siguiente información:

- ***Número de aprobados (nota igual o mayor que cinco) y porcentaje del total de evaluados.***
- ***Número de suspensos y porcentaje del total de evaluados.***
- ***Nota media entre los aprobados.***
- ***Nota media entre los suspendidos.***

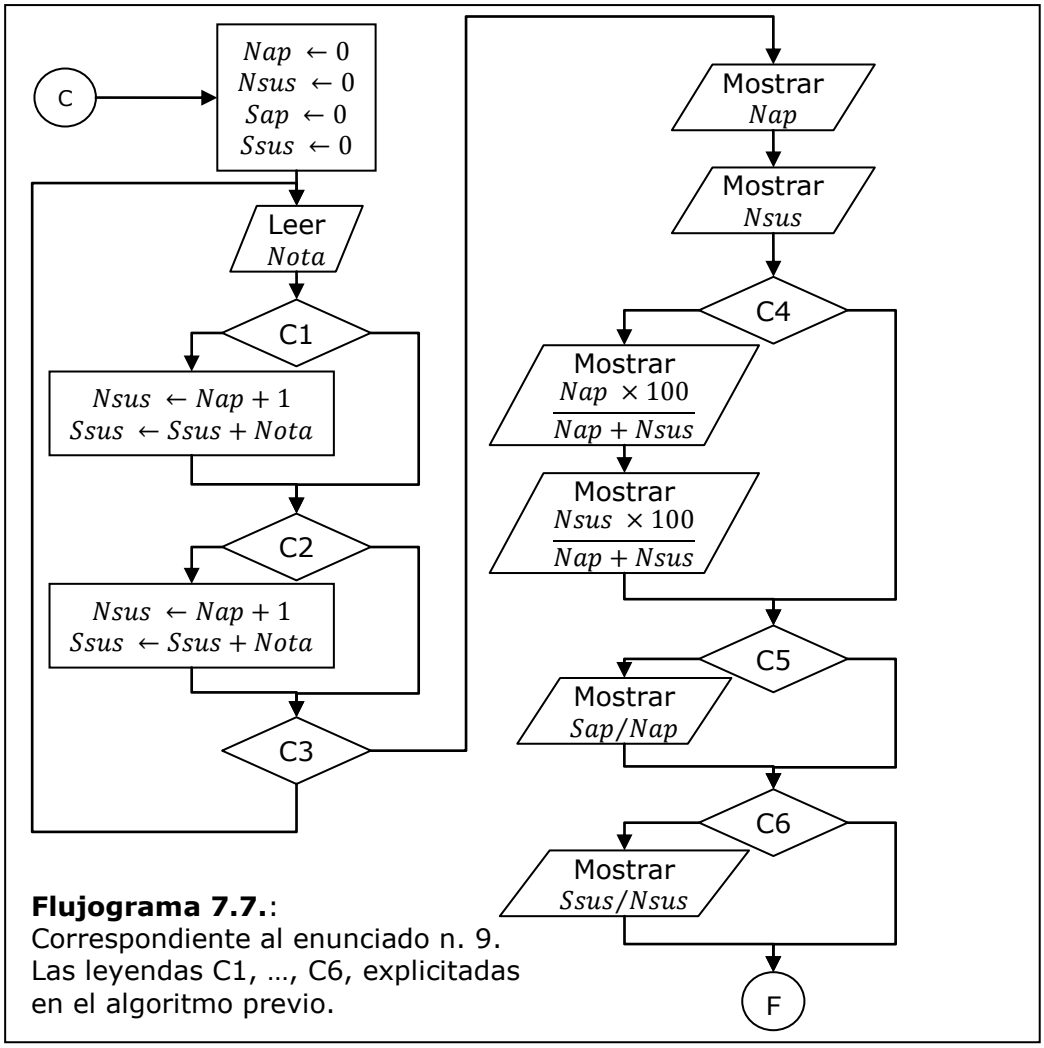
Una vez se ha introducido el valor del día (variable dd), del mes (variable mm) y del año (variable aa), el algoritmo debe decidir si esa fecha es válida o no, de acuerdo con las indicaciones recogidas en el enunciado. Un posible algoritmo de respuesta al problema podría ser el siguiente:

1. [Inicializar las variables]: $Nap \leftarrow 0$, $Nsus \leftarrow 0$, $Sap \leftarrow 0$, $Ssus \leftarrow 0$.
2. **Repetir**:
   2.1. [Leer entrada]: **Leer** $Nota$.
   2.2. **Si** C1: $(0 \leq Nota < 5)$ **Entonces**:
      2.2.1. $Nsus \leftarrow Nsus + 1$.
      2.2.2. $Ssus \leftarrow Ssus + Nota$.
      Si no, **Entonces**:
      2.2.3. **Si** C2: $(5 \leq Nota \leq 10)$ **Entonces**
         2.2.3.1. $Nap \leftarrow Nap + 1$.
         2.2.3.2. $Sap \leftarrow Sap + Nota$.

Mientras que C3: ($Nota \neq -1$).

3. [Mostrar Resultados]
   3.1. [Número de aprobados]: **Mostrar** Nap.
   3.2. [Número de aprobados]: **Mostrar** $Nsus$.
   3.3. **Si** C4: ($Nap + Nsus \neq 0$) **Entonces**:
      3.3.1. [Porcentaje aprobados]: **Mostrar** $Nap \times 100/(Nap + Nsus)$.
      3.3.2. [Porcentaje aprobados]: **Mostrar** $Nsus \times 100/(Nap + Nsus)$.
   3.4. **Si** C5: ($Nap \neq 0$) **Entonces** [Media aprobados]: **Mostrar** Sap/Nap.
   3.5. **Si** C6: ($Nsus \neq 0$) **Entonces** [Media aprobados]: **Mostrar** $Ssus/Nsus$.
4. **Fin**.

Donde hemos llamado: Nap la variable que cuenta el número de



Flujograma 7.7.:
Correspondiente al enunciado n. 9.
Las leyendas C1, …, C6, explicitadas en el algoritmo previo.

aprobados; $Nsus$ a la que cuenta el número de suspensos; Sap a la variable que suma las notas de los aprobados; y $Ssus$ a la que suma las notas de los suspensos.

El Flujograma de este algoritmo puede verse en el Cuadro que hemos llamado Flujograma 7.7.

10. Calcule el valor del número e. El número e se define como el límite de la sucesión de término general:

$$\left(1+\frac{1}{n}\right)^n$$

También se puede calcular como la suma de los inversos de todos los factoriales:

$$e=\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots$$

Para el cálculo del número e quizá bastaría con calcular el valor de $(1+1/n)^n$ para un valor de n suficientemente grande. Esa operación no requiere ningún algoritmo y es una simple instrucción.

El algoritmo para el cálculo del número e a partir de la propiedad presentada podría ser el siguiente:

1. [Inicializar variables]: $e \leftarrow 1$, $invF \leftarrow 1$, $n \leftarrow 10.000$.
2. **Para** $I \leftarrow 1$ **Hasta** n **Repetir**:
   2.1. $i\text{nvF} \leftarrow \text{invF} \times (1/I)$.
   2.2. $e \leftarrow e + invF$.
   2.3. $I \leftarrow I + 1$.
3. [Mostrar resultado]: **Mostrar** e
4. **Fin**.

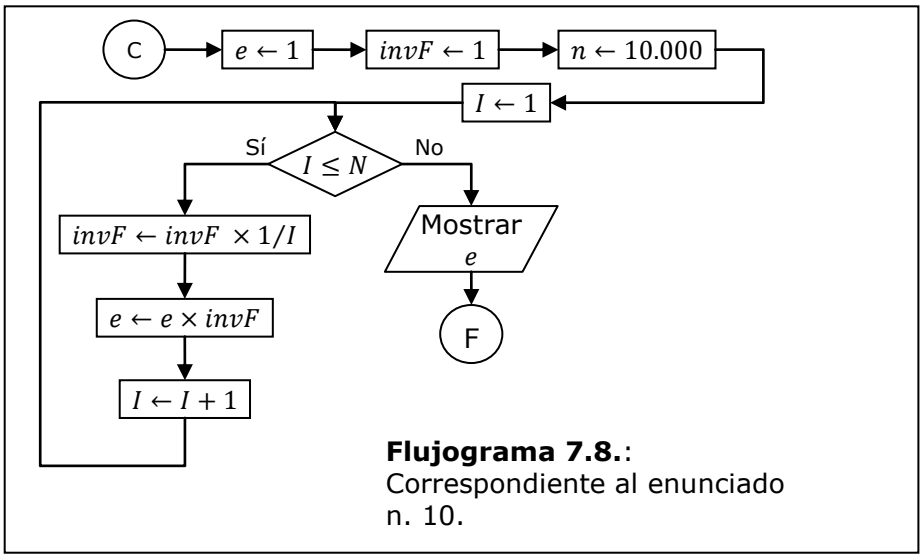
La variable e la inicializamos a 1: ya le hemos sumado el primero de los sumandos, el $1/0!$.

La variable $invF$ almacena el valor del inverso de los factoriales. Lo inicializamos al valor de $1/0!$.

Se repite tantas veces como indique la variable n el proceso de sumar elementos del sumatorio. La variable e va guardando esa suma realizada

paso a paso según la estructura de control iterativa gobernada por la variable I. La variable $invF$ almacena los sucesivos valores $1/I!$.

El flujograma de este algoritmo queda recogido en Flujograma 7.8.



Flujograma 7.8.: Correspondiente al enunciado n. 10.

11. Juego de las 15 cerillas: "Participan dos jugadores. Inicialmente se colocan 15 cerillas sobre una mesa y cada uno de los dos jugadores toma, alternativamente 1, 2 o 3 cerillas pierde el jugador que toma la última cerilla".

Buscar el algoritmo ganador para este juego, generalizando: inicialmente hay N cerillas y cada vez se puede tomar hasta un máximo de k cerillas. Los valores N y k son introducidos por teclado y decididos por un jugador (que será el jugador perdedor), el otro jugador (que será el ordenador, y que siempre debe ganar) decide quien empieza el juego.

El algoritmo que permite programar este juego es muy sencillo. Pero hay que saber llegar hasta él.

Este es de nuevo un buen ejemplo para mostrar que el reto de la programación no está tanto en conocer un determinado lenguaje, sino en saber expresar mediante sentencias sucesivas y con el control que ofrecen las estructuras condicionales y las iterativas, el modo de lograr que la máquina gane siempre al usuario del programa.

A lo largo de la explicación del juego llamaremos jugador G al ganador, es decir, al ordenador; y jugador P al perdedor, es decir, al usuario.

Supongamos que hemos tomado un valor $k = 3$: es decir, en cada jugada, el jugador puede retirar 1, ó 2, ó 3 cerillas. En el cuadro 7.1. queda recogido la cantidad de cerillas que se pueden llegar a retirar cada vez que juega un turno los jugadores G y P.

		Jugador P		
		1	**2**	**3**
Jugador G	**1**	2	3	4
	2	3	4	5
	3	4	5	6

Cuadro 7.1.: Cerillas que se retiran después de un turno de juego.

Está claro, y se ve reflejado en el cuadro, que sea cual sea el número de cerillas que retire el primer jugador (el jugador P), el segundo (el jugador G) siempre puede retirar un número tal que en todos los turnos de juego siempre se retire el mismo número de cerillas. Entre las cantidades de cerillas que se pueden retirar en un turno hay un valor que se repite en todas las filas y columnas: el 4. Ése es el valor al que debe jugar el que retira cerillas en segundo lugar (que será el jugador G). Si el primero (jugador P) ha retirado 1, entonces el ganador retira 3; si el primero retira 2 cerillas, entonces el ganador retira también 2. Si el primero retira 3 cerillas, entonces el ganador retira sólo una. Y, en general, si tenemos k cerillas, el número de cerillas que siempre se

podrán retirar en un turno de jugada es $k + 1$. Si el jugador P retira $c \leq k$ cerillas, entonces el jugador G deberá retirar, si de verdad quiere ser el ganador, $k + 1 - c$ cerillas.

Por lo tanto, la primera regla del jugador ganador es retirar las cerillas siempre después del jugador que va a perder, y retirar siempre tantas cerillas como sean necesarias para lograr que en cada turno se retiren $k + 1$ cerillas.

Figura 7.1.: Cerillas. $N = 15$ y $k = 3$.

Si, por ejemplo, tenemos $N = 15$ y $k = 3$, entonces hemos quedado que en cada turno el ganador logrará que se retiren 4 cerillas. En la figura 7.1. se ven las cerillas agrupadas en bloques de 4.

El objetivo del juego es lograr que después de la jugada del ganador quede sobre la mesa únicamente una cerilla. Si el jugador ganador va logrando que después de cada turno se retiren $k + 1$ cerillas (es decir, 4 cerillas), lo que importa es que al inicio del juego sobre la mesa haya una cantidad de cerillas múltiplo de $k + 1$, más una cerilla más. Pero en nuestro juego no ocurre así: de hecho hay una cantidad múltiplo de 4 más TRES cerillas más.

Esa es la segunda regla del algoritmo ganador: el jugador ganador (jugador G) debe decidir quién comienza a jugar. Y lo hace de la siguiente manera:

1. Calcula el resto de dividir $(N + 1)$ por $(k + 1)$: $r \leftarrow (N - 1)\, mod\, (k + 1)$. Ya hemos dicho que el objetivo es que al principio el número de cerillas sea tal que $(N + 1)$ sea múltiplo de $(k + 1)$.

2. Si $r = 0$, entonces el jugador ganador cede gentilmente el turno al otro jugador, que será quien comenzará la partida. Si $r \neq 0$ entonces será el jugador ganador quien comenzará a jugar, retirando r cerillas. A partir de ese momento tiene una cantidad de cerillas adecuada, y sin más que lograr retirar en cada turno $k + 1$ llevará inevitablemente a la derrota a su contrincante. En el ejemplo de la figura 7.1. se obtiene el valor $r = 2$, y entonces el jugador G lo que debe hacer, si desea ganar, es empezar retirando esas dos cerillas. Y a partir de este momento, el jugador G juega siempre detrás del jugador P, y retira siempre $k + 1 - c = 4 - c$ cerillas, donde c es el número de cerillas que ha retirado en la última jugada el jugador P.

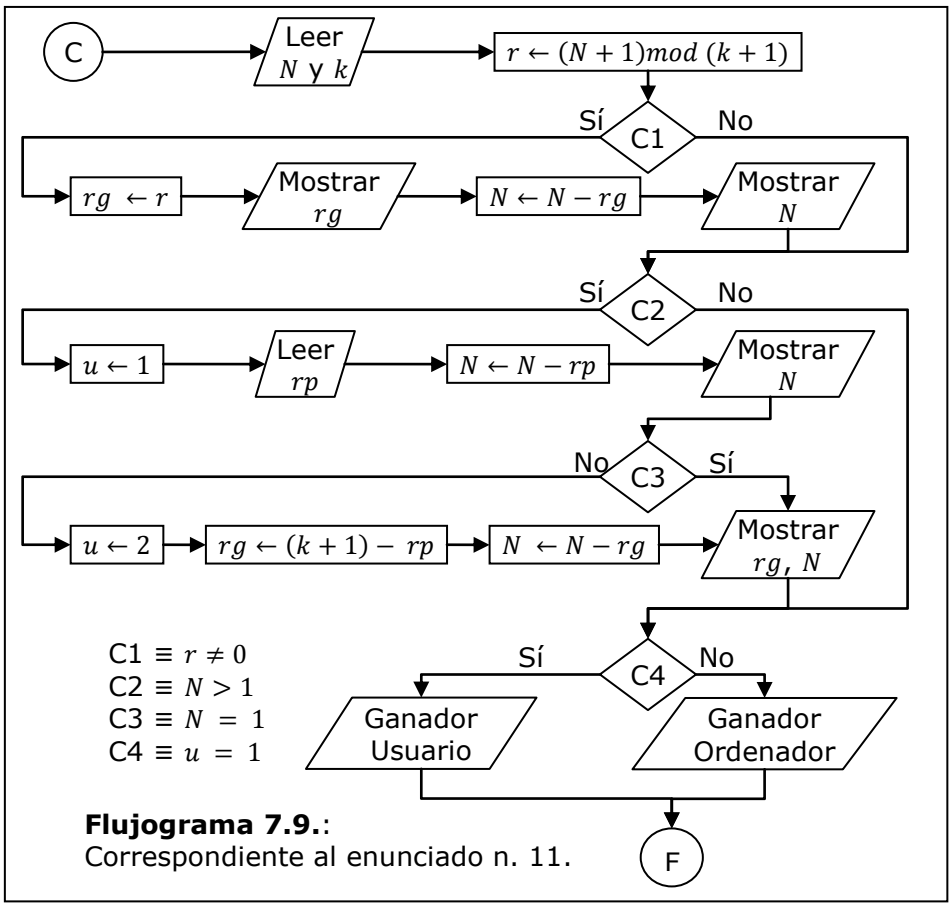
Quien deduce estas reglas del juego, ya tiene resuelto el algoritmo, que toma la siguiente forma:

1. [entrada de parámetros]: **Leer** N y k.
2. [Calcular]: $r \leftarrow (N + 1)\ mod\ (k + 1)$.
3. **Si** $r \neq 0$ **Entonces**:
   [Turno del ordenador]:
   3.1. $\mathrm{rg} \leftarrow \mathrm{r}$ (rg: cerillas que retira el ganador)
   3.2. $N \leftarrow \mathrm{N} - \mathrm{rg}$ (hay que descontar esas cerillas)
   3.3. [Mostrar información]: **Mostrar** rg y N.
4. **Mientras** $N > 1$ **Hacer**:
   4.1. [Turno usuario]: $u \leftarrow 1$
   4.2. [Lectura]: **Leer** rp (cerillas que retira el perdedor)
   4.3. $N \leftarrow N - rp$ (hay que descontar esas cerillas)
   4.4. [Mostrar información]: **Mostrar** N.
   4.5. **Si** $N = 1$ **Entonces**: [Salir de la iteración]: **Ir** al paso 5.
   4.6. [Turno ordenador]: $u \leftarrow 2$.
   4.7. $\mathrm{rg} \leftarrow (k + 1)\text{- rp}$.
   4.8. $N \leftarrow \mathrm{N} - \mathrm{rg}$ (hay que descontar esas cerillas)
   4.9. [Mostrar información]: **Mostrar** rg y N.
5. **Si** $u = 1$ **Entonces**:
   [Mostrar ganador]: **Mostrar** "Ha ganado el usuario.
   Sino Entonces:
   [Mostrar ganador]: **Mostrar** "Ha ganado el ordenador."
6. **Fin**.

En el algoritmo hemos utilizado una variable (que hemos llamado u) que nos indica quién está jugando en cada momento. Si $u = 1$, juega el

usuario, es decir, el jugador P; si $u = 2$, entonces juega el ordenador, es decir, el jugador G.

El Flujograma del algoritmo queda recogido en Flujograma 7.9.



Flujograma 7.9.:
Correspondiente al enunciado n. 11.

12. Población de conejos. Supongamos que tenemos una pareja de conejos (macho y hembra) recién nacidos. Y supongamos que esos conejos llegan al estado de adultos una vez transcurrido un mes de vida. Supongamos que una coneja sólo puede concebir si es adulta.

Supongamos también que toda hembra adulta pare una nueva pareja de conejos (un macho y una hembra) después de un mes

de gestación, y que en cuanto una coneja pare sus crías, la pareja logra de inmediato volver a concebir una nueva pareja.

Supongamos que en nuestra población los conejos nunca se mueren; que en cada nueva cría nacen únicamente dos conejos: uno macho y otro hembra; que todos los conejos, después del primer mes de crecimiento y a partir del segundo mes, ya puede aparearse con la correspondiente coneja y todos los meses cada pareja adulta (mayor de un mes de vida) cría dos nuevos conejillos.

Escriba el algoritmo del programa que calcule la población de conejos después de transcurrir tantos meses como indique el usuario por consola.

En el primer mes nuestra población de conejos está formada por un macho y una hembra no adultos; es decir, una pareja. Ni mueren, ni pueden concebir.

En el segundo mes nuestra población de conejos está formada por un macho y una hembra adultos; es decir, una pareja. La hembra ha concebido un par de conejos, pero éstos aún no han nacido.

En el tercer mes tenemos dos pares de conejos: uno de conejos adultos con la coneja de nuevo en estado de buena esperanza, y una pareja de jóvenes conejos. En el cuadro 7.2. vemos la evolución de las parejas de conejos mes a mes.

En realidad, estamos ante la serie de Fibonacci. Pero vamos a hacer como si no lo supiéramos.

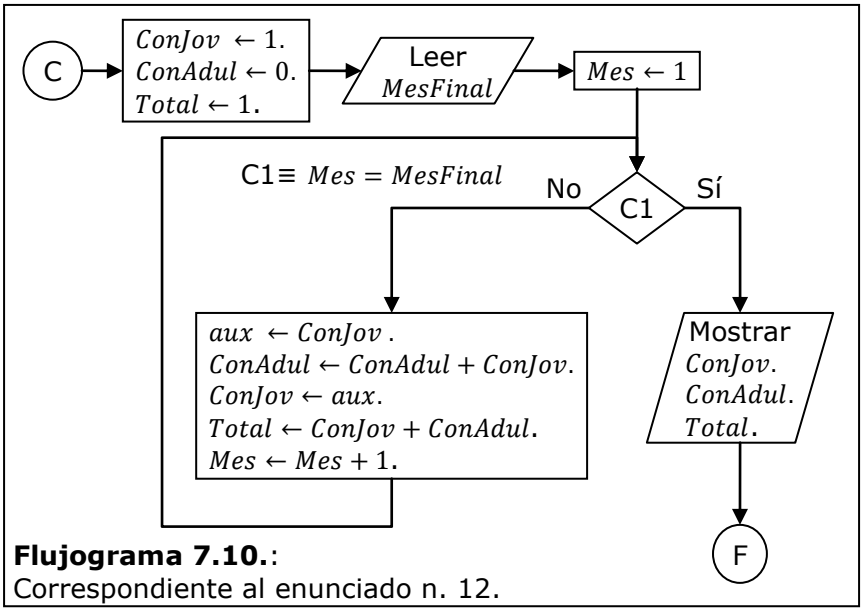
El algoritmo se presta mucho a la recurrencia. El número de conejos jóvenes de cada mes es el número de adultos de la vez anterior. Y el número de adultos es la suma de los jóvenes y adultos del mes anterior.

mes	nuevos	adultos	total
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
9	13	21	34
10	21	34	55

Cuadro 7.2.: Evolución de las parejas de conejos por meses.

El algoritmo simple de este proceso podría ser el siguiente:

1. [Inicializar Variables]: $conJov \leftarrow 1$, $ConAdul \leftarrow 0$, $Total \leftarrow 1$.
2. [Introducir mes]: Leer $MesFinal$.
3. Para $Mes \leftarrow 1$ Hasta $MesFinal$ Repetir:
   [Operaciones]
   3.1. $aux \leftarrow ConAdul$.
   3.2. $ConAdul \leftarrow ConAdul + ConJov$.
   3.3. $ConJov \leftarrow aux$.
   3.4. $Total \leftarrow ConJov + ConAdul$.
   3.5. $Mes \leftarrow Mes + 1$.



Flujograma 7.10.:
Correspondiente al enunciado n. 12.

4. [Mostrar Resultados]: Mostrar $ConJov$, $ConAdul$, $Total$.
5. **Fin**.

El Flujograma del algoritmo queda recogido en Flujograma 7.9.

13. El número áureo (Φ) verifica muchas curiosas propiedades. Por ejemplo:

$\Phi^2 = \Phi + 1$ $\quad$ $\Phi - 1 = 1/\Phi$ $\quad$ $\Phi^3 = (\Phi + 1)/(\Phi - 1)$

$\Phi = 1 + 1/\Phi$ $\quad$ $\Phi = \sqrt[2]{1 + \Phi}$ $\quad$ **y otras…**

La penúltima expresión presentada ($\Phi = 1 + 1/\Phi$) muestra un camino curioso para el cálculo del número áureo:

$$\boldsymbol{\Phi = 1 + \frac{1}{\Phi} \Rightarrow \Phi = 1 + \frac{1}{1 + \frac{1}{\Phi}} \Rightarrow \Phi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\Phi}}}} \ldots}$$

Y, entonces un modo de calcular el número áureo es el siguiente:

$$\boldsymbol{\Phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots \cfrac{1}{1 + \frac{1}{1}}}}}}}$$

Es decir, inicializando Φ al valor 1, se puede ir afinando en el cálculo del valor del número áureo a base de repetir muchas veces que $\Phi = 1 + 1/\Phi$.

Presente el algoritmo para calcular el número áureo mediante este procedimiento haciendo, por ejemplo, la sustitución $\Phi = 1 + 1/\Phi$ mil veces. Mostrar luego el resultado por pantalla.

El enunciado es algo aparatoso. Pero el algoritmo que lo resuelve es muy sencillo. Hay que inicializar la variable Φ a 1 y luego realiza 1000 veces la sustitución $\Phi = 1 + 1/\Phi$. El algoritmo queda como sigue:

1. [Inicializar Variables]: $\Phi \leftarrow 1$.
2. **Para** $I \leftarrow 1$ **Hasta** 1000 **Repetir**:
   2.1. $\Phi = 1 + 1/\Phi$.
3. [Mostrar Resultados]: **Mostrar** Φ.
4. **Fin**.

Siguiendo con el mismo enunciado, también podemos plantearnos calcular el número áureo a partir de la relación $\Phi = \sqrt[2]{1+\Phi}$. De nuevo tenemos:

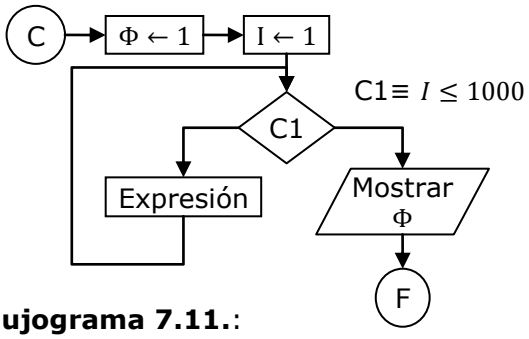
$$\Phi = \sqrt[2]{1+\Phi} = \sqrt[2]{1+\sqrt[2]{1+\Phi}} = \sqrt[2]{1+\sqrt[2]{1+\sqrt[2]{1+\Phi}}}$$

$$= \sqrt[2]{1+\sqrt[2]{1+\sqrt[2]{1+\sqrt[2]{1+\sqrt[2]{1+\cdots\sqrt[2]{1+\Phi}}}}}}$$

eso nos brinda otro algoritmo, casi idéntico al anterior, donde ahora cambiamos la operación cociente por la operación raíz cuadrada.

1. [Inicializar Variables]: $\Phi \leftarrow 1$.
2. **Para** $I \leftarrow 1$ **Hasta** 1000 **Repetir**:
   2.1. $\Phi = \sqrt[2]{1+\Phi}$
3. [Mostrar Resultados]: **Mostrar** Φ.
4. **Fin**.

Los flujogramas de ambos algoritmos quedan recogidos en el cuadro Fujograma 7.11. Si Expresión es $\Phi \leftarrow 1 + 1/\Phi$, entonces estamos en el primer algoritmo; si Expresión es $\Phi \leftarrow \sqrt{1+\Phi}$, entonces estamos en el segundo algoritmo.



Flujograma 7.11.:
Correspondiente al enunciado n. 13.

De todas formas, estos dos cálculos tienen también una solución mediante concurrencia. En las expresiones que hemos tomado, el valor del número áureo depende del valor del número áureo.

El algoritmo de la función recurrente podría tener la siguiente forma:

Función Aureo(Φ Real, I Entero)→ Real
Constantes:
∅.
Variables:
∅.
Acciones:
1. **Si** $I = 0$ **Entonces** $Expresión$.
   Sino Entonces $Aureo \leftarrow Aureo(\Phi, I-1)$
2. **Fin**.