EnRevision (Rosario).

Algoritmos y estructuras de datos. Entre el nivel medio y la universidad.

Verónica D'Angelo.

Algoritmos y Estructuras de Datos

Entre el nivel medio y la universidad

Verónica D'Angelo

(la siguiente es una vista previa reducida del material de estudio original)

2023

Entrada y salida de datos: LEER y MOSTRAR

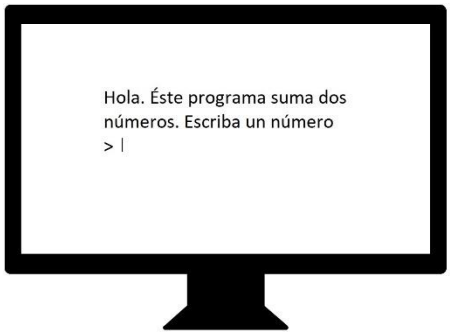
En el capítulo anterior, sobre la capa física, se mencionaron las operaciones de lectura y escritura referidas a las memorias. En Informática, la lectura o la escritura se piensan desde el punto de vista de los dispositivos, no desde el ser humano. Se *lee* y *escribe* información en los dispositivos: en las memorias, en la pantalla, en un disco. No se trata de que el usuario humano lee o escribe (confusión habitual).

En el caso de los lenguajes de programación, hay dos funciones (también las consideramos palabras reservadas del lenguaje) que se utilizan para capturar datos desde el teclado (cuando una persona los escribe) y para mostrar datos por pantalla (para que una persona los lea). Estas funciones se denominan LEER y ESCRIBIR/MOSTRAR.

Estas funciones forman parte de todos los lenguajes (con diferentes expresiones). En este libro utilizaremos LEER y MOSTRAR (En PseInt pueden utilizarse tanto ESCRIBIR como MOSTRAR, son sinónimos).

LEER, en pseudocódigo, significa, detener el funcionamiento del programa en espera a que un usuario ingrese contenido con el teclado. Ese contenido queda depositado en una variable (una posición de memoria con un nombre).

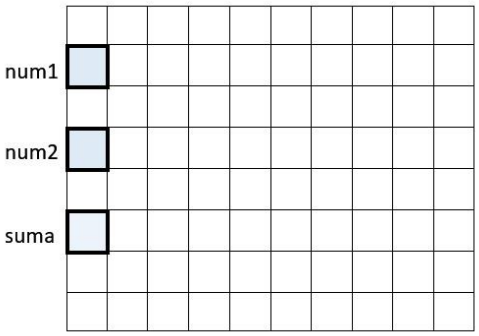
Por ejemplo, supongamos que debemos escribir un programa que le pida al usuario que ingrese dos números y luego le muestra la suma de los números por pantalla.



¿Cómo haría ese programa para comunicarse con el usuario y pedirle eso? Lo único de lo que dispone es una pantalla y un teclado. En pseudocódigo hay ciertas restricciones. No se hacen lecturas del movimiento del mouse ni se muestran botones para que el usuario de clic. Tan sólo se pueden mostrar y capturar palabras o números escritos. Por lo tanto, en primer lugar, el programa le debe mostrar un mensaje al usuario, explicándole que es un programa de suma y pidiéndole el primer número. En pseudocódigo podría ser una instrucción como ésta:

MOSTRAR "Hola. Éste programa suma dos números. Escriba un número"

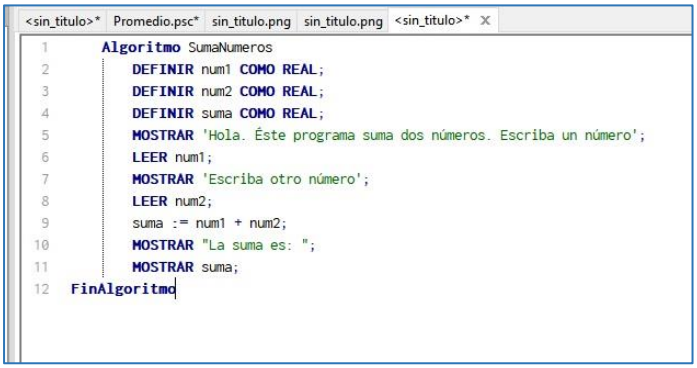
Imaginemos la memoria RAM como una cuadrícula en la que cada celda tiene una dirección. En lugar de recordar su dirección numérica, le colocamos un nombre, y con ese nombre podemos volver a localizar la celda. Para crear esa variable en memoria usamos la función *DEFINIR.*



Por ejemplo, si escribimos en PseInt "***DEFINIR** num1 **como Real***", le estamos pidiendo que reserve una posición en la memoria para la variable num1, que no se puede utilizar ese espacio para otra cosa. Es como si bloqueara esa posición que ahora se denomina *num1.* Esa posición no es un bit ni un byte. El número de bytes que contenga dependerá del tipo de datos. El tipo de datos llamado "Real" ocupa más espacio que el tipo de datos "Entero" o "Carácter". La operación "DEFINIR" asigna el espacio que corresponde a cada variable.

Luego de que están definidas, las variables pueden ser utilizadas en el código.

La memoria RAM es electrónica. Esta memoria es más rápida comparada con la memoria magnética de un disco duro, que tiene partes mecánicas lentas, ya que necesita activar cabezales entre otras cosas. Por eso esta memoria, aunque es bastante frágil ya que depende del suministro eléctrico, es la que utiliza permanentemente el sistema operativo para ciertas operaciones básicas. Por ejemplo, cuando usamos un procesador de texto, antes de que grabemos el documento en el disco duro de manera definitiva, vamos grabando el contenido temporal en la memoria RAM.



LEER num1 es una operación que deja el teclado suspendido hasta que el operador escribe algo. El usuario verá un cursor titilando en espera. Una vez que el usuario pulse enter, habrá terminado la operación de lectura y en la casilla *num1* de la memoria se habrá guardado un número real (ya que esta variable sólo admite números reales).

LEER num2

Guarda un número en la celda *num2*.

Luego, se deben sumar los números y mostrar la suma al usuario.

suma := num1 + num2

Es la operación que suma los números y los guarda dentro de la variable suma. Esa operación se denomina asignación.

MOSTRAR suma Muestra el contenido de la variable suma por la pantalla.

Es importante que el futuro programador pueda imaginar y graficar estos escenarios y estos espacios, como la memoria y sus variables, lo que aparecerá en pantalla, lo que se asigna (cambia de lugar) en una operación.

Descubrir secuencias

Las operaciones más potentes que puede realizar un ordenador son secuenciales, como la iteración (repetición) que le permite realizar miles o millones de veces la misma acción sobre datos distintos, ya que el ordenador no se cansa ni se equivoca como el humano.

Al ser secuenciales los procesos, todos los conjuntos de datos que aparezcan en el enunciado (y que sean relevantes para el problema) tendrán forma de secuencias ("tiras" de datos, o contenedores con filas y columnas). Habrá datos aislados, variables individuales que se utilizan por única vez o pocas veces, pero los más importantes son los conjuntos de datos que pueden ser vistos como secuencias.

Las secuencias están presentes en diversas actividades y situaciones de la vida cotidiana.

- Fila de clientes en la caja de un supermercado: Los clientes forman una fila para esperar su turno en la caja registradora. Cada cliente se mueve hacia adelante a medida que los clientes anteriores son atendidos y finalmente llega su turno para pagar y salir del supermercado.
- Secuencia de tráfico en un semáforo: Los vehículos se alinean en una secuencia ordenada mientras esperan su turno en un semáforo. Cuando la luz cambia, los vehículos avanzan en la secuencia establecida, deteniéndose y avanzando según las señales del semáforo.
- Secuencia de letras en el abecedario: El alfabeto sigue una secuencia de letras específica, desde la A hasta la Z.
- Secuencia de pasos en un tutorial en video: Los tutoriales en video a menudo presentan una secuencia de pasos para realizar una tarea o aprender una habilidad específica.
- Secuencia de jugadas en un juego de cartas: Los jugadores siguen una secuencia específica de jugadas en juegos de cartas como el póker o el blackjack.

Recuerda que el ordenador es un procesador eficiente de secuencias. Si un enunciado no refiere a secuencias es probable que se trate de un ejercicio trivial. A veces se dan estos ejercicios para practicar ingreso de datos de entrada o mostrar resultados de salida.

Entonces, cuando leas el enunciado de un ejercicio, tratá de detectar en primer lugar cuáles son las estructuras de datos, y dibujalas en papel, para luego poder deducir cómo deberían ser las estructuras de proceso que manipularán esas estructuras de datos.

Cuando el profesor escribe un nuevo ejercicio intenta emular situaciones que podrías encontrar en el mundo real. Los enunciados no dicen exactamente cómo hacer el programa, a veces te cuentan la situación desde el punto de vista de un cliente, de un problema matemático, o de un usuario.

Supongamos un enunciado muy simple que sólo pide lo siguiente:

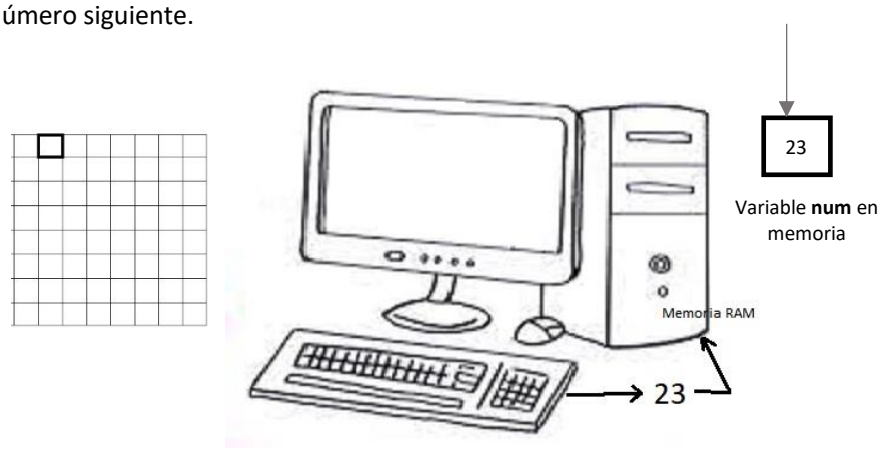
> "Mostrar en pantalla 5 números que ingresan por teclado secuencialmente"

Se pueden deducir muchísimas cosas de este simple enunciado. Habrá un usuario que escribirá números con el teclado. Los va a ingresar secuencialmente, es decir, ingresa un número, pulsa *Enter*, ingresa otro, pulsa *Enter*... Significa que hay un programa en la computadora, que está corriendo (funcionando), y que se encarga de pedir uno a uno los números al usuario. Si el usuario entiende lo que el programa le pide debe ser porque el programa dio alguna señal al respecto. La manera más común de hacerlo es mostrar por pantalla un mensaje que le explique al usuario de qué se trata este programa y qué debe hacer. En este caso, debe ingresar uno a uno 5 números y luego pulsar Enter.

Todo ese proceso lleva bastantes líneas de código, pero en este enunciado no nos han especificado tantos detalles. Sólo nos piden mostrar por pantalla los números que ingresa el usuario imaginario. Tampoco dice que haya que almacenar los números y luego mostrarlos todos juntos.

Se supone que no deberíamos almacenar nada en la memoria a menos que sea necesario para uso futuro. Por lo tanto, aunque parezca algo sin sentido, este simple programa sólo debe "recibir" el número que escribe el usuario, guardarlo temporalmente en algún lugar para que no se pierda, sólo unos segundos, y luego mostrar en la pantalla el mismo número.

Una vez que el número fue mostrado, el lugar de memoria puede quedar libre para guardar el número siguiente.



Cuando el usuario escribe el número, desde el teclado, el número se dirige a la memoría RAM y allí queda en una variable que en este ejemplo llamamos *num*. También, hay una operación que ocurre por defecto cuando se usa el teclado, lo escrito se muestra por pantalla. Pero el enunciado del programa pide mostrar nuevamente el número por pantalla (se verá dos veces). Para que el número se vea nuevamente por pantalla falta la instrucción MOSTRAR(num).

```
LEER(num)
MOSTRAR(num)
LEER(num)
MOSTRAR(num)
LEER(num)
MOSTRAR(num)
LEER(num)
MOSTRAR(num)
LEER(num)
MOSTRAR(num)
```

¿Cinco veces las mismas instrucciones?

```
HACER 5 VECES
    LEER(num)
    MOSTRAR(num)
```

Si el enunciado indicara “Ingresar cinco números por teclado y luego mostrarlos todos juntos”, la repetición no se aplicaría a leer la misma variable, porque aparece una nueva restricción: si usamos la misma variable para leer los cinco números, cada número ingresado sobre escribiría el contenido del anterior. Por lo tanto harían falta cinco variables. Y la solución es bien distinta:

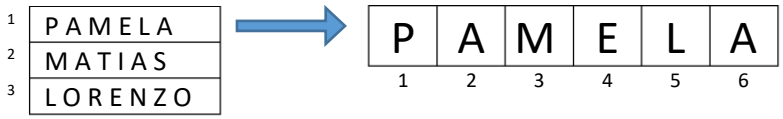
```
LEER(num1)
LEER(num2)
LEER(num3)
LEER(num4)
```

```
LEER(num5)
MOSTRAR(num1, num2, num3, num4, num5)
```

Esas cinco lecturas de diferentes variables es una estrategia aceptable para leer cinco variables. Pero ¿qué pasaría si fueran quinientas o más?

En esos casos se utiliza una estructura de control denominada iteración (también conocida como repetición, ciclo o blucle). En la primer solución a este ejercicio se usa una repetición que pide al microprocesador que haga cinco veces lo mismo: LEER y MOSTRAR num.

En ejercicios simples, es probable que la secuencia de datos sea sólo una. Por ejemplo, una lista de números como en el enunciado anterior, o una lista de palabras a las cuales se les aplica una determinada operación. Pero en problemas que implican la organización de gran cantidad de información, las secuencias de datos suelen estar ocultas entre otras secuencias.



Por ejemplo, en una secuencia de nombres se pueden identificar, a su vez, una secuencia de letras dentro de cada nombre. Una secuencia de números que ingresan al ordenador puede representarse como una tira (izquierda), si esos números se guardan en memoria y tienen un orden, puede indicarse con un subíndice (derecha). Estos son simplemente representaciones gráficas de los datos que facilitan poder pensar en ellos.

Por fortuna, las estructuras de control que debemos utilizar en el programa sirven justamente para procesar secuencias de datos. Es decir que una vez identificada la secuencia de datos se hace más fácil identificar el bloque de procesamiento adecuado. Por eso es bueno dibujar en primer lugar las estructuras de datos.

Existen estructuras de control para procesar repeticiones de acciones, y también para la selección de alternativas.

Para dar un marco más claro a las estructuras, especialmente cuando hay anidamiento (una estructura dentro de otra), en esta etapa de tu aprendizaje hemos utilizado

diagramas que se conocen como (NSD o Chapin). Te sugerimos que los pruebes, pero son opcionales. Es sólo un marco alrededor del pseudocódigo, las instrucciones no varían al agregar diagramas.

La estructura de iteración PARA puede producir, *para cada elemento* de una secuencia, el mismo tratamiento (acción o acciones). La iteración MIENTRAS puede producir la acción mientras se cumpla una condición (si no se cumple al inicio no habrá ningún ciclo de procesamiento), la estructura REPETIR MIENTRAS, repite el mismo mientras no se cumpla la condición de fin (habrá como mínimo un ciclo de procesamiento).

El foco de este apunte está puesto en ayudarte a realizar el análisis de los enunciados. Cuando un programa está terminado es recomendable que el programador haya colocado comentarios al lado de las instrucciones para que otros programadores comprendan lo que hizo. Pero antes de llegar a ese punto, mientras se está construyendo el programa en borrador también es muy importante que el programador explique lo que significan las instrucciones para su propio seguimiento de la lógica del proceso. Ese análisis de los enunciados es lo que hemos escrito al lado de las instrucciones de los siguientes problemas resueltos y te invitamos a que hagas lo mismo con otros problemas que tengas que resolver en las prácticas que te han entregado en la universidad.

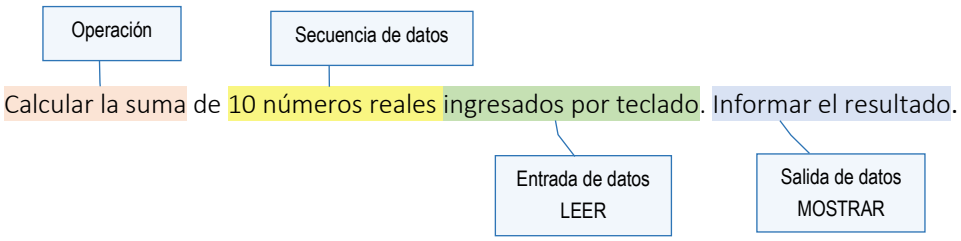
Una salvedad que hacemos en este apunte es que no hemos declarado variables al principio de los programas en todas las secciones, hemos dado por sentado que esa acción está realizada y las variables creadas, sí hemos hecho la inicialización de las variables, es decir, cargar las variables con valores iniciales que sean adecuados para su tratamiento posterior.

En la sección B “Ejercicios para completar” hay ejercicios y un espacio para que expliques la solución.

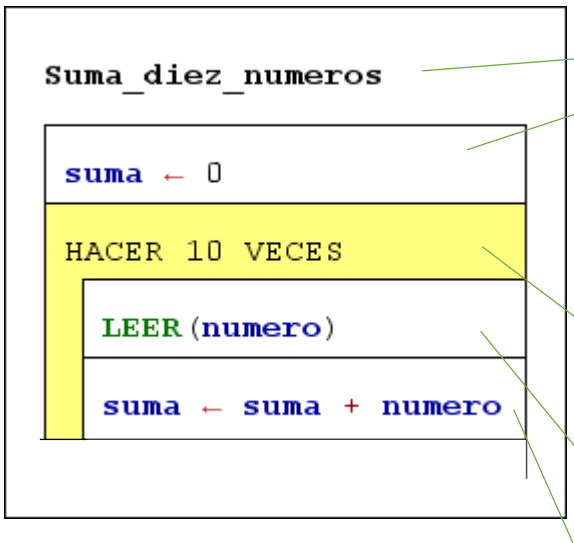
SECCIÓN A: EJERCICIOS ANALIZADOS

1. Calcular la suma de 10 números reales ingresados por teclado. Informar el resultado.

ANÁLISIS DEL PROBLEMA



ANÁLISIS DEL PROGRAMA



Nombre del programa

Inicializa la variable *suma* que será utilizada en la operación de acumulación. De lo contrario daría lugar a error. Aclaración: Inicializar no es declarar (o definir) la variable para el compilador. Aquí sólo asignamos los valores mínimos que el algoritmo necesita para trabajar sin errores. Por ejemplo, no se inicializa *numero*, ya que su primer valor será leído desde teclado.

Se repetirán 10 veces las acciones contenidas en este bloque. HACER no necesita manipular una variable extra para contar la cantidad de iteraciones, ya que el número de iteraciones es conocido

La palabra reservada LEER deja en espera el procesador hasta que el usuario escribe el número, luego lo guarda en la variable *numero*

La variable suma acumula en cada pasada el contenido del nuevo número, sumándolo al anterior. En la última pasada habrá acumulado los 10 valores. No se pide mostrarlo.

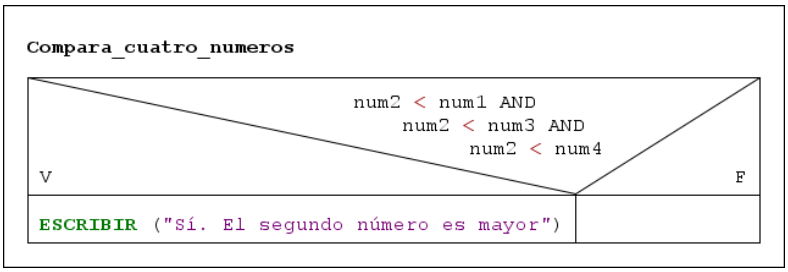
---

2. Dado un conjunto de cuatro números decidir si el segundo es mayor que los otros tres (se supone que los números están contenidos en variables ya inicializadas).

ANÁLISIS DEL PROBLEMA

Sólo se desea conocer la relación entre el segundo número y los demás, por lo tanto sólo habrá una operación de comparación. Los cuatros números no se ingresan, se supone que este paso fue dado en otra parte del programa.

ANÁLISIS DEL PROGRAMA



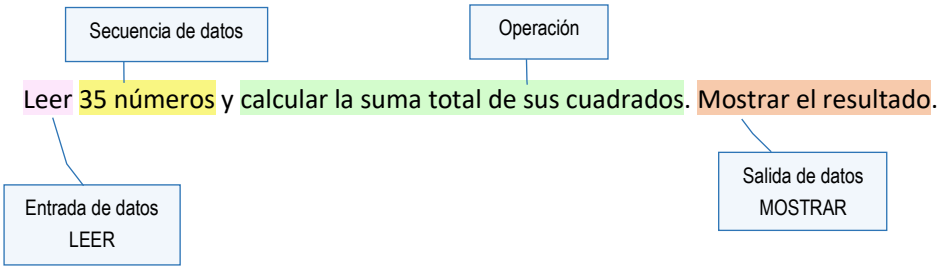
Nombre del programa

Comparaciónn entre num2 y los números restantes

---

3. Leer 35 números y calcular la suma de sus cuadrados. Mostrar el resultado.

ANÁLISIS DEL PROBLEMA



La secuencia de datos más importante son los 35 números que ingresan. Se desconoce la secuencia (los números) pero se conoce cuántas veces debe permitirse al usuario ingresar un número. Llenar 35 variables es poco eficiente, por lo tanto, en una única variable se guardará el número ingresado, se calculará su cuadrado y se acumulará en otra variable totalizadora.

ANÁLISIS DEL PROGRAMA

```
Suma_35_cuadrados

numero ← 0
acumula ← 0
HACER 35 VECES
    LEER(numero)
    acumula ← acumula + numero*numero
MOSTRAR(acumula)
```

Inicializa la variable *numero* que contendrá los números ingresados desde teclado

Inicializa la variable *acumula*, que contendrá las sumas de los cuadrados

Repite 35 veces las acciones: Leer el número que ingresa el usuario. El cuadrado del numero se suma al valor anterior de acumula.

Se muestra el resultado en pantalla

---

4. Calcular y mostrar en forma ascendente los múltiplos de 4 menores que *n*.

ANÁLISIS DEL PROBLEMA

La secuencia de múltiplos de 4 se obtiene multiplicando 4 por cada uno de los números naturales hasta llegar a un múltiplo cercano a 60.

1	2	3	4	5	6	7	...

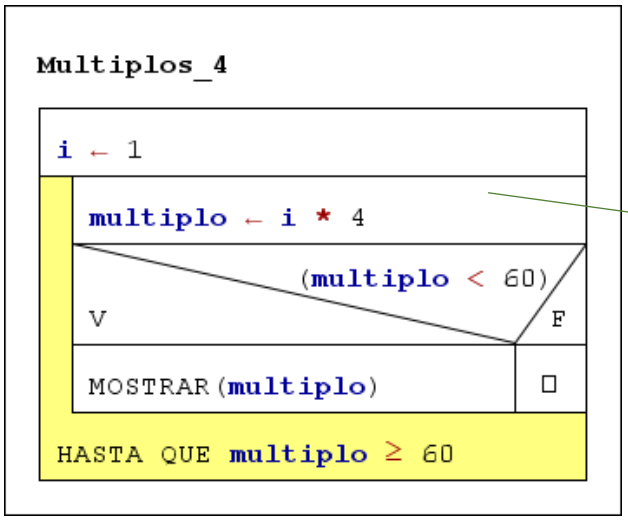
X 4

4	8	12	16	20	24	28	...

Si bien el programador conoce la secuencia de números, se pide que el programa los calcule de a uno y los vaya mostrando en forma creciente, suponiendo que se desconoce el último. La condición será que el múltiplo obtenido sea menor que 60.

Habrá por lo menos una iteración, y no se necesita validar ese primer dato, por lo tanto, la estructura adecuada es REPETIR.

ANÁLISIS DEL PROGRAMA



Inicializa la variable *i* que contendrá una secuencia de números naturales

En cada pasada de la secuencia se multiplica por 4 el valor de i, obteniendo el múltiplo

Sólo se muestran los múltiplos que no superan 60

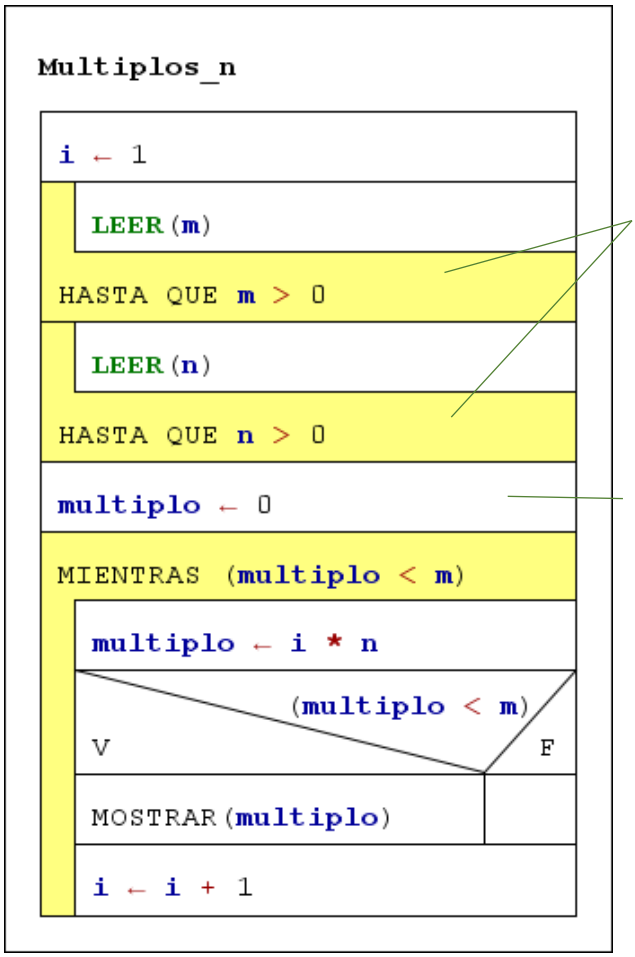
---

5. Calcular y mostrar en forma ascendente los múltiplos de *m* menores que *n*.

ANÁLISIS DEL PROBLEMA

Como *m* y *n* serán ingresados por teclado, deberá validarse su ingreso para evitar valores que produzcan error. En los siguientes casos, no habrá resultados válidos: n = m, n > m, m = 0, n =0

ANÁLISIS DEL PROGRAMA



Inicializa la variable *i* que contendrá una secuencia de números naturales

Se valida el ingreso para evitar que m o n contengan números negativos o cero. La estructura ideal para cualquier validación es REPETIR.
El esquema mental del programador es: REPETIR hasta que sea válido.

Asigna 0 a la variable que contendrá los sucesivos múltiplos.

Dependiendo que el múltiplo obtenido sea menor que el máximo múltiplo permitido,

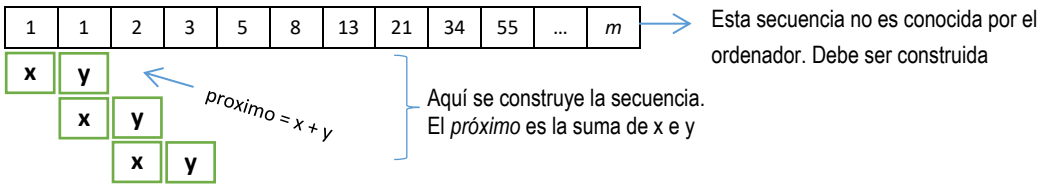
obtiene un múltiplo

---

6. Mostrar los primeros m términos de la sucesión de Fibonacci.

ANÁLISIS DEL PROBLEMA

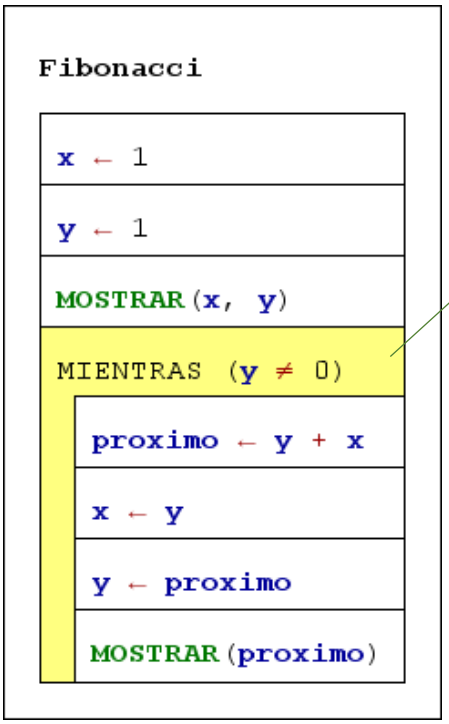
La secuencia de datos principal es la sucesión de Fibonacci, donde cada número se forma por las suma de los dos números que le preceden. Imaginemos otra secuencia de parejas x, y que van conteniendo los sucesivos valores a sumar. En principio, x e y contienen a 1 y 1, la suma es 2. En un segundo ciclo, *x* e *y* cambian sus valores por los que siguen, *x* contendrá el contenido de *y*, y el próximo número se guarda en *y*.



Esta secuencia no es conocida por el ordenador. Debe ser construida

Aquí se construye la secuencia.
El *próximo* es la suma de x e y

ANÁLISIS DEL PROGRAMA



Asigna a x el primer valor de la sucesión

Asigna a y el segundo valor de la sucesión

Se utiliza MIENTRAS porque es posible que el usuario ingrese 0 como primer valor, con lo cual no habría ciclo

Se obtiene el próximo como la suma de los anteriores

Una vez mostrados los valores de x e y, ya pueden ser reemplazados.

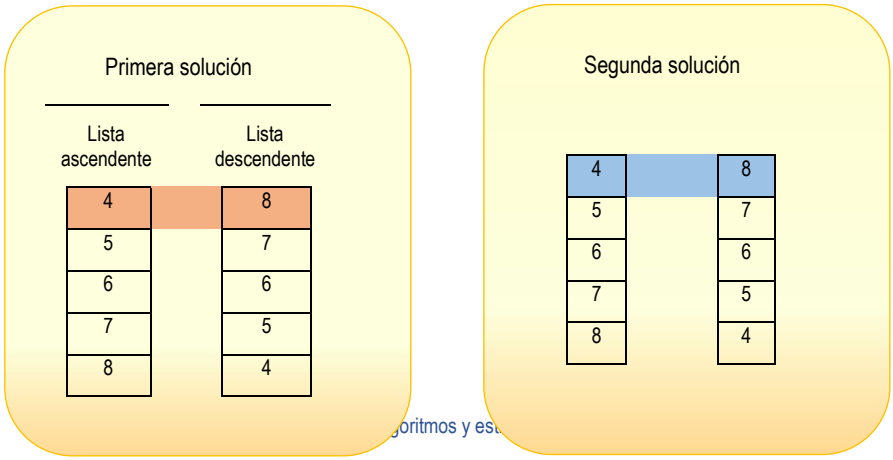
Se asigna el siguiente

---

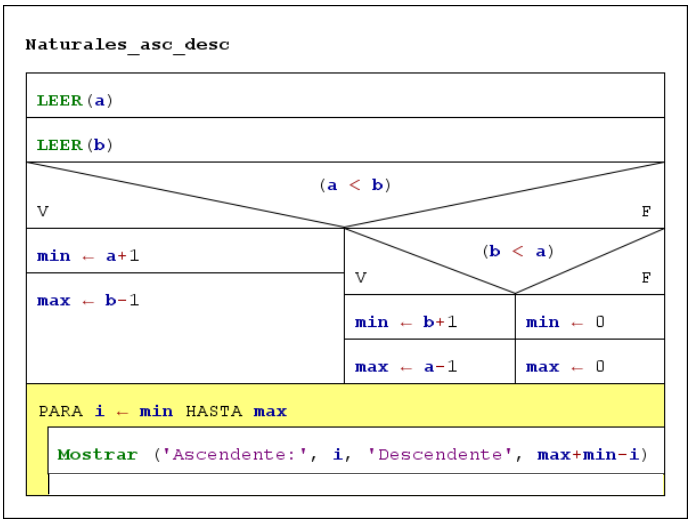
7. Ingresar dos números e informar los números naturales comprendidos entre ellos en orden ascendente y descendente utilizando un solo ciclo.

ANÁLISIS DEL PROBLEMA

La secuencia de datos es un intervalo abierto ( ]a;b[ ), es decir, excluyendo de la sucesión a estos valores. Se pide diferenciar entre orden ascendente y descendente, lo cual implica averiguar cuál de los números es menor que el otro. Una segunda solución más simple los muestra en pantalla como dos listas invertidas desconociendo el procesador cuál es la ascendente y cuál la descendente.



ANÁLISIS DEL PROGRAMA



Lee el primer número

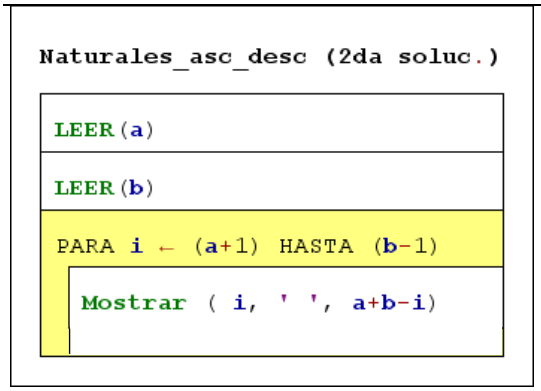
Lee el segundo número

Se pregunta cuál es menor. O si son iguales. En este caso los vuelve a 0 y culmina el programa. Si a es menor que b,

Asigna a *x* el primer valor de la sucesión

Asigna a *y* el segundo valor de la sucesión

Desde el valor mas pequeño al mayor, *i* recorre los naturales comprendidos en orden ascendente y descendente, mostrando una fila de cada lista por vez (los números 4 y 8 en la figura anterior).



Complete con los valores sucesivos de *i* en cada paso del ciclo:

---

8. Se cuenta con el listado de notas de un alumno. En un lugar aparte figura la cantidad de aplazos. Se deben informar dos promedios: el promedio con aplazos y el promedio sin aplazos.

Secuencia de datos:

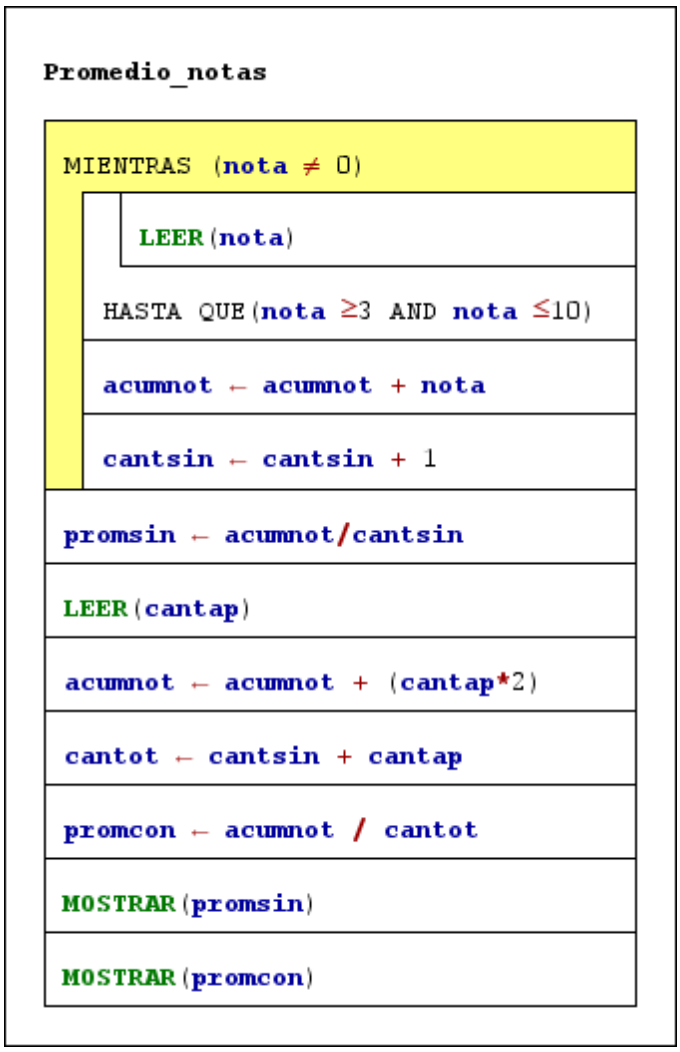
Notas con valores >=3 y <=10

4	6	7	8	6	9	10

Variables sugeridas:

nota: contiene el valor de cada nota de la lista
acumnot: acumula el valor de todas las notas para calcular el promedio.
cantsin: contador de notas sin aplazos
promsin: promedio sin aplazos

promcon: promedio con aplazos
cantap: cantidad de aplazos (ya se conoce)



El 0 (cero) será un indicador de que el usuario desea finalizar el programa. Si la nota no es cero se supone que intenta ingresar una nota, se ingresa al ciclo de validación para evitar valores incorrectos como nota=15.

Se acumula cada nota sin considerar los aplazos

Se incrementa en 1 el contador de notas sin aplazos.

Finalizado el ciclo se calcula el promedio sin aplazos como la suma de notas dividida la cantidad de notas sin aplazos.
Se lee la cantidad de aplazos

Se acumula la suma de las notas anteriores más la suma de los aplazos. Cada aplazo vale 2.

Se obtiene la cantidad total de notas. Con y sin aplazos.

El promedio con aplazos es la acumulación total de notas sobre la cantidad total de notas.

Muestra ambos promedios

---

9. Calcular el factorial de un número (sin recursividad). Validar que el número esté comprendido entre 1 y 40 y calcular la suma de los naturales menores que él.

```
Factorial_no_rec
    LEER(n)
HASTA QUE(n >0) AND (n<40)
MIENTRAS (n≠0)
    fac ← fac * (n-1)
    sum ← sum + n-1
    n ← n-1
MOSTRAR(fac)
MOSTRAR(sum)
```

Se valida que sólo ingrese un número mayor que cero y menor que 40

Habrá con seguridad un primer ingreso al ciclo.

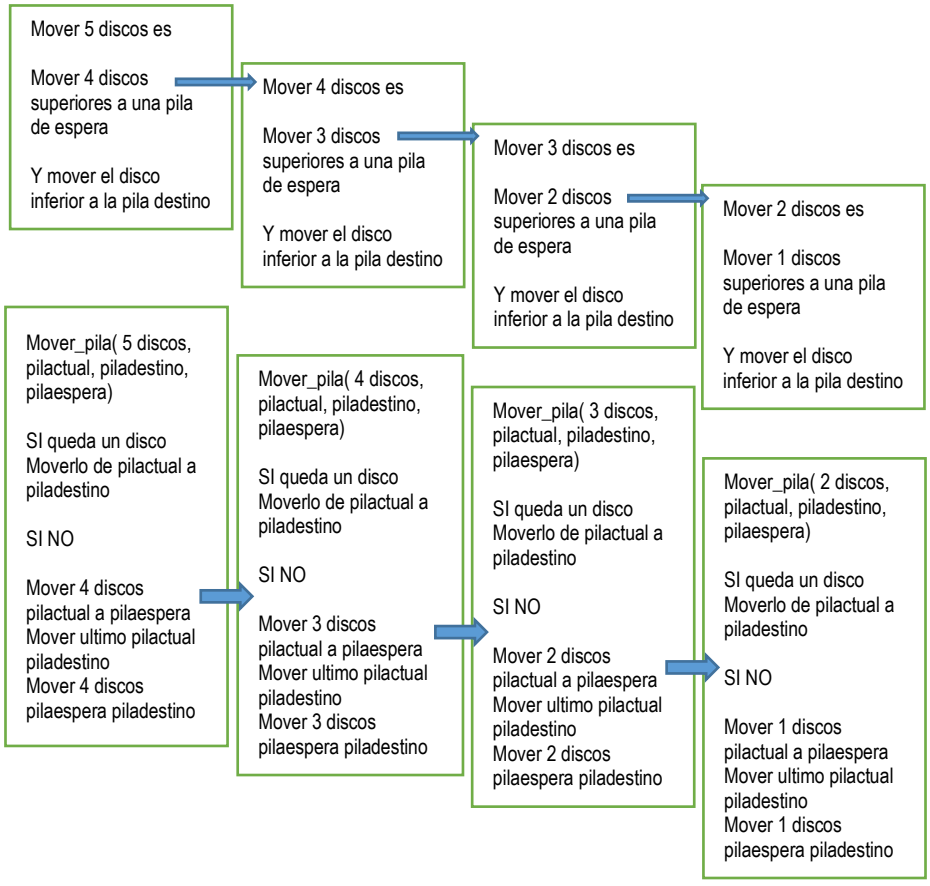
Inicialmente, fac contiene el número ingresado, luego contendrá el número multiplicado por su antecesor, ej. 5*4
Se suman el numero y su antecesor, ej. 5+4

Se resta 1 al número y se vuelve a efectuar la operación hasta que el número sea 0

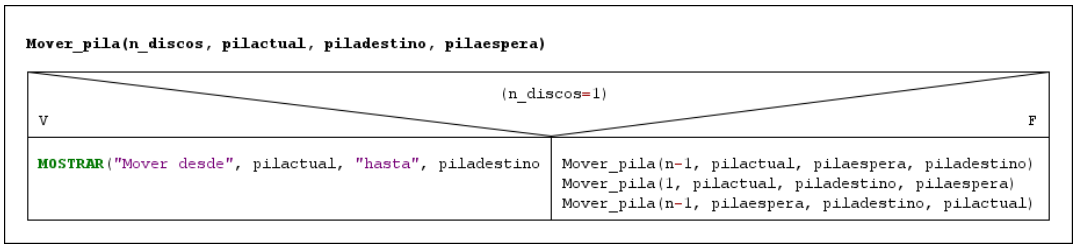
El factorial es el resultado de todas las operaciones,

igual que la suma

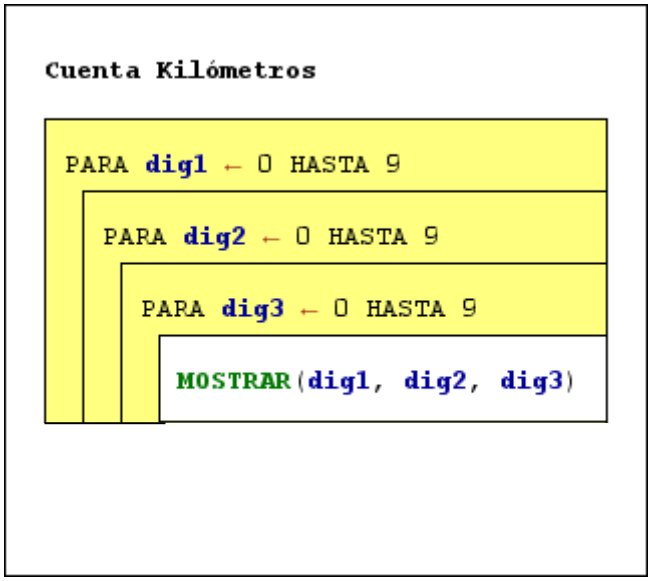
10. Resolver el problema de las Torres de Hanoi para cinco discos.



Mover una pila completa desde la posición actual a una posición destino significa mover primero los discos superiores de la pila hacia una pila de espera, mover el último a la posición destino y luego mover los discos en espera a la posición destino. Pero mover los discos en espera implica realizar el mismo procedimiento con un disco menos (n-1).[2]



11. Representar el movimiento de un cuenta kilómetros desde 000 a 999.



El primer dígito varía de 0 a 9

El segundo dígito varía de 0 a 9

El tercero dígito varía de 0 a 9. En principio todos muestran 0. Cuando el dígito 3 de las unidades avance hasta 9, se incrementará en 1 el dígito 2 de las decenas. Sólo cuando el dígito 2 avance hasta 9, se incrementará en 1 el dígito 3 de las centenas.

[2] Solución basada en: Eric Grimson, and John Guttag. 6.00 Introduction to Computer Science and Programming. Fall 2008. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

12. Formar las combinaciones posibles de dos dígitos distintos tomadas de a 3. Mostrar las que tengan los tres dígitos iguales.

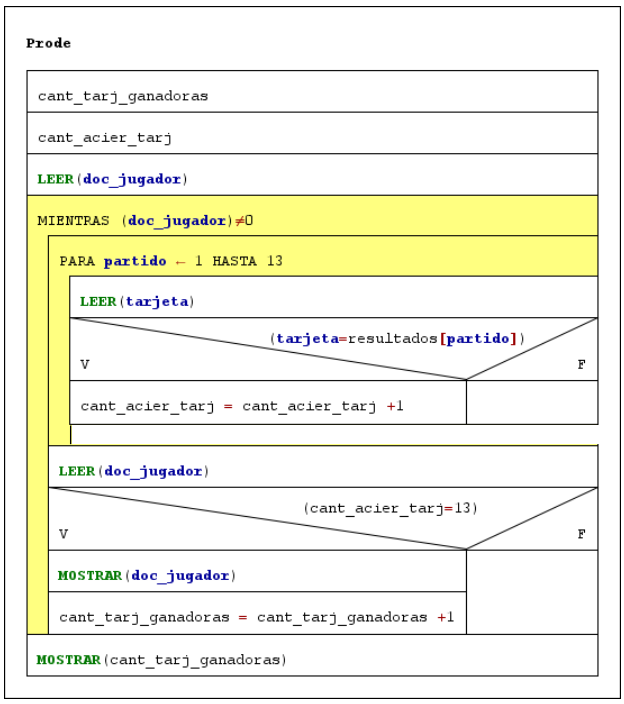
```
Combinaciones_de_a_3
    LEER(dig1)
HASTA QUE(dig1>0 AND dig1<10)
    LEER(dig2)
HASTA QUE (dig2>0 AND dig2<10 AND dig2≠dig1)
PARA i ← dig1 HASTA dig2
    PARA i2 ← dig1 HASTA dig2
        PARA i3 ← dig1 HASTA dig2
            MOSTRAR(i, i2, i3)
```

Se fuerza al operador a sólo ingresar números positivos menores que 10

Además, el enunciado especifica que sean distintos ambos dígitos.

Varía cada desde el primer número ingresado hasta el segundo, en caso de que el primero sea mayor que el segundo la secuencia tendrá un orden inverso (ej. 9, 8, 7)

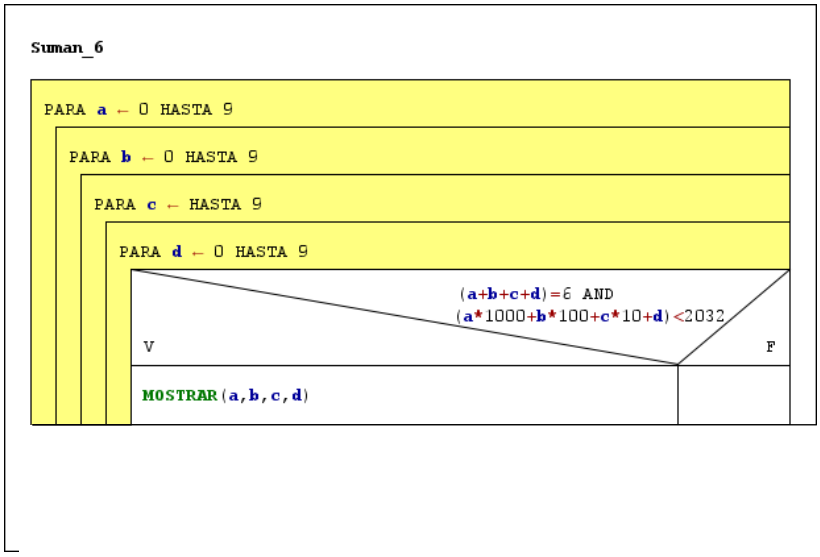
13. Informar la cantidad de ganadores del prode y sus números de documentos. Los resultados tienen el formato de la tarjeta: son listas de 13 elementos cada una.



Inicializa el contenido de los acumuladores que contarán la cantidad de tarjetas ganadoras y la cantidad de aciertos por tarjeta.
Lee el documento de cada jugador, que puede contener letras y dígitos.
Mientras haya más documentos de jugadores a ingresar.
Para cada documento hay una tarjeta que consiste en 13 posiciones, una por cada partido. Por cada posición se pregunta si fue un acierto o no. En caso positivo lo contabiliza.
Tarjeta y partido son arreglos de 13 elementos cada uno. Por ser del mismo tipo de datos pueden compararse.

Si obtuvo 13 aciertos lo cuenta como un ganador.

14. Generar todos aquellos números menores que 2032 tal que sus dígitos sumen 16.



Cada dígito en a se combina con cada dìgito en b, en c y en d.

Y para cada combinación se pregunta si la suma de los dígitos es 6 y el número menor que 2032.

14. Una cátedra tiene clasificados 277 ejercicios según aspectos distintos: ejercicios para razonar, para aplicar definiciones, para desarrollar habilidades prácticas, para investigar, para pasar el rato. Cada ejercicio tiene una calificación del 1 al 10 en cada aspecto. Se desea realizar consultas interactivas a dichos datos. Ingresando dos condiciones (razonar y

pasar el rato), se pide listar aquellos cinco mejores ejercicios para esas dos condiciones (el criterio para clasificar se obtiene en base al promedio de las notas de cada ejercicio respecto a esas condiciones). Se conoce además que dichos ejercicios no varían como así tampoco sus calificaciones. Indicar una estructura de datos adecuada para resolver el problema teniendo en cuenta que el tiempo se constituye en un factor crítico.

ANÁLISIS DEL PROBLEMA

La cantidad de ejercicios está pre fijada, así como los aspectos a ser evaluados. Cada nota clasificatoria se aplica a la combinación ejercicio+aspecto, por ejemplo, suponiendo que el ejercicio 2 es muy útil para aplicar definiciones y para pasar el rato, mientras que el ejercicio 1, es bueno para razonar aunque no es necesariamente entretenido como pasatiempo, así que en este ultimo aspecto tendrá una nota más bien baja.

ARREGLO DE ASPECTOS

ARREGLO DE EJERCICIOS

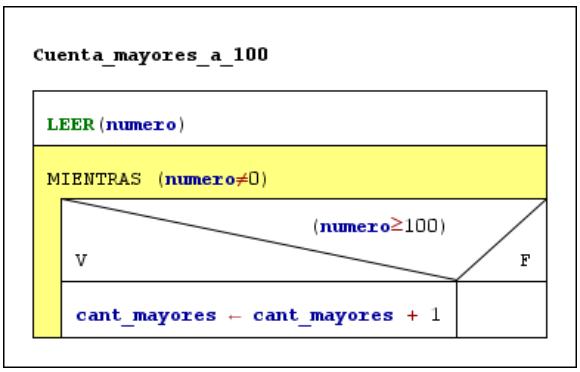
	1	2	3	4	5	6	7	…	277
Raz	10	6	5	8	8	9	10	…	8
Def	8	10	9	10	2	10	10	…	3
Pra	5	10	9	5	3	10	10	…	8
Inv	9	8	9	5	10	7	2	…	9
Pas	4	8	8	9	8	7	3	…	9

La estructura de datos para contener las calificaciones podría ser un arreglo de arreglos (tabla de dos dimensiones): un arreglo de 277 ejercicios (considerando la fila en verde como subíndice) cada uno de los cuales es un arreglo de 5 notas cada una calificando una habilidad: *razonar*, *definir*, *practica*, *investigar*, *pasatiempo*.

SECCIÓN B: EJERCICIOS PARA COMPLETAR

1. Dado un conjunto de números naturales, determinar cuántos de ellos son mayores o iguales a 100. Un número igual a 0 indica fin de datos.

ANÁLISIS DEL PROBLEMA

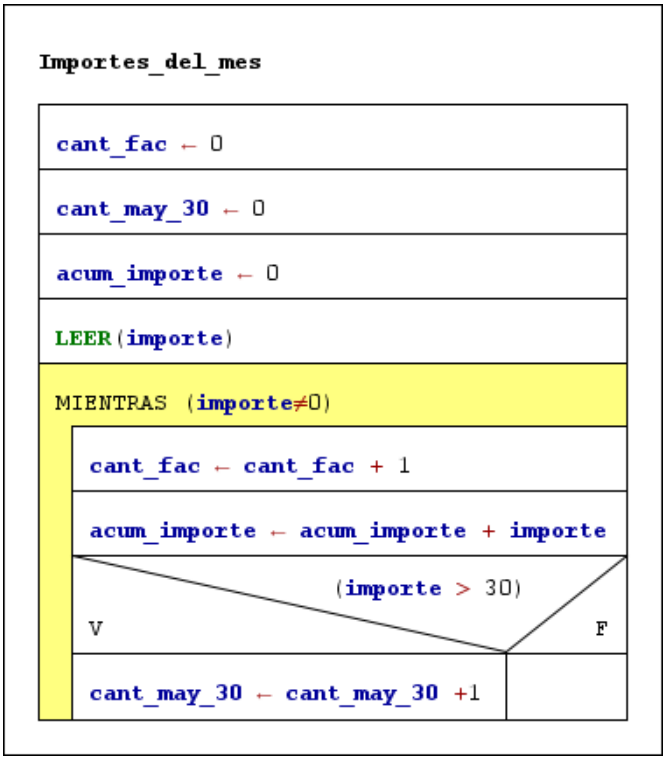


2. Se tiene como datos los importes de todas las facturas correspondientes al mes que acaba de finalizar de un comercio (no se sabe cuántas son). Se desea conocer:

a. Cuántas facturas se realizaron.
b. Importe promedio de las mismas
c. Cuántos son los importes que superan los 30 pesos.

ANÁLISIS DEL PROBLEMA

Secuencia de datos:


Importes_del_mes
cant_fac ← 0
cant_may_30 ← 0
acum_importe ← 0
LEER(importe)
MIENTRAS (importe≠0)
cant_fac ← cant_fac + 1
acum_importe ← acum_importe + importe
(importe > 30)
V
F
cant_may_30 ← cant_may_30 +1

SECCIÓN C: EJERCICIOS COMPARADOS (PSEUDOCODIGO Y C)