UTN FRGP
TUP

Técnico Universitario en Programación

UNIVERSIDAD TECNOLÓGICA NACIONAL

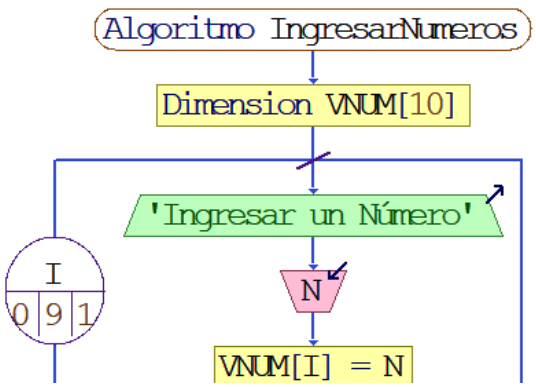
Facultad Regional Gral. Pacheco

Apuntes de clase de la asignatura

Programación I

ARRAYS / VECTORES

ESTRUCTURAS DE DATOS I



2024

Abel Oscar Faure

Lorena Raquel Palermo

INTRODUCCIÓN

En muchos problemas, el uso de tipos simples de datos, como enteros, reales o caracteres, puede complicar la solución. Sin embargo, al emplear tipos estructurados, como los arreglos, se puede simplificar el proceso.

Para ilustrar esto, primero veremos un problema resuelto con tipos simples, lo que nos permitirá notar las dificultades. Luego, presentaremos los arreglos y mostraremos cómo su uso facilita una solución más clara y eficiente.

Los arreglos son estructuras que permiten almacenar y procesar colecciones de datos relacionados, como una lista de calificaciones o una serie de temperaturas.

En este tema, exploraremos el concepto de arreglos unidimensionales y multidimensionales, y cómo utilizarlos en la programación.

Ejemplo – Solución con Datos Simples

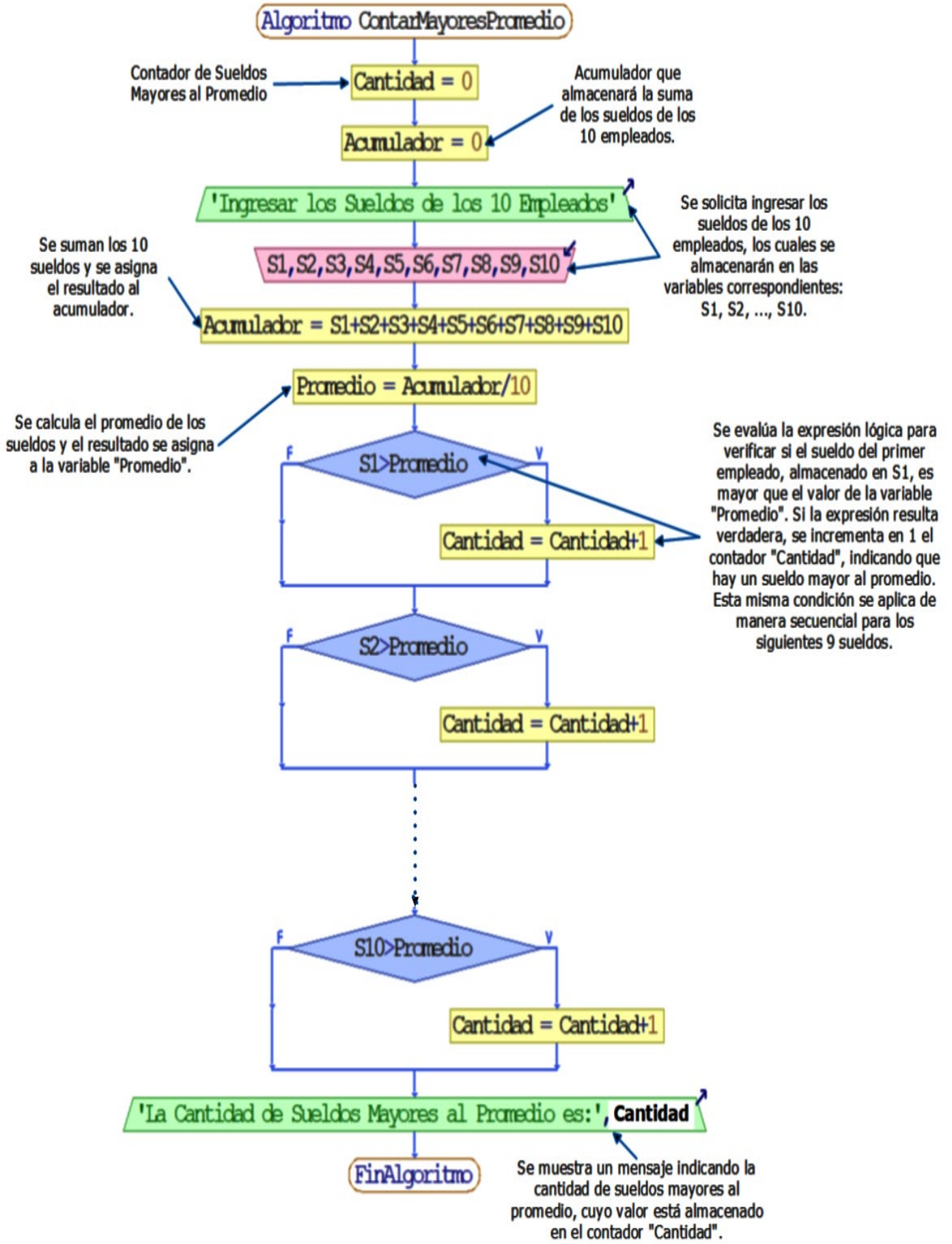
Se tienen los sueldos de un grupo de 10 empleados de una empresa, y se necesita crear un algoritmo que determine cuántos de estos empleados tienen un sueldo superior al promedio del grupo. Este problema puede resolverse mediante un diagrama de flujo que ilustre los pasos necesarios para calcular el promedio y luego comparar los sueldos con dicho valor.

Solución 1:

En la solución presentada en el siguiente diagrama de flujo, se utilizan 10 variables en memoria. Sin embargo, este enfoque tiene el inconveniente de que el manejo de estas variables puede volverse difícil de controlar si su número aumenta considerablemente.

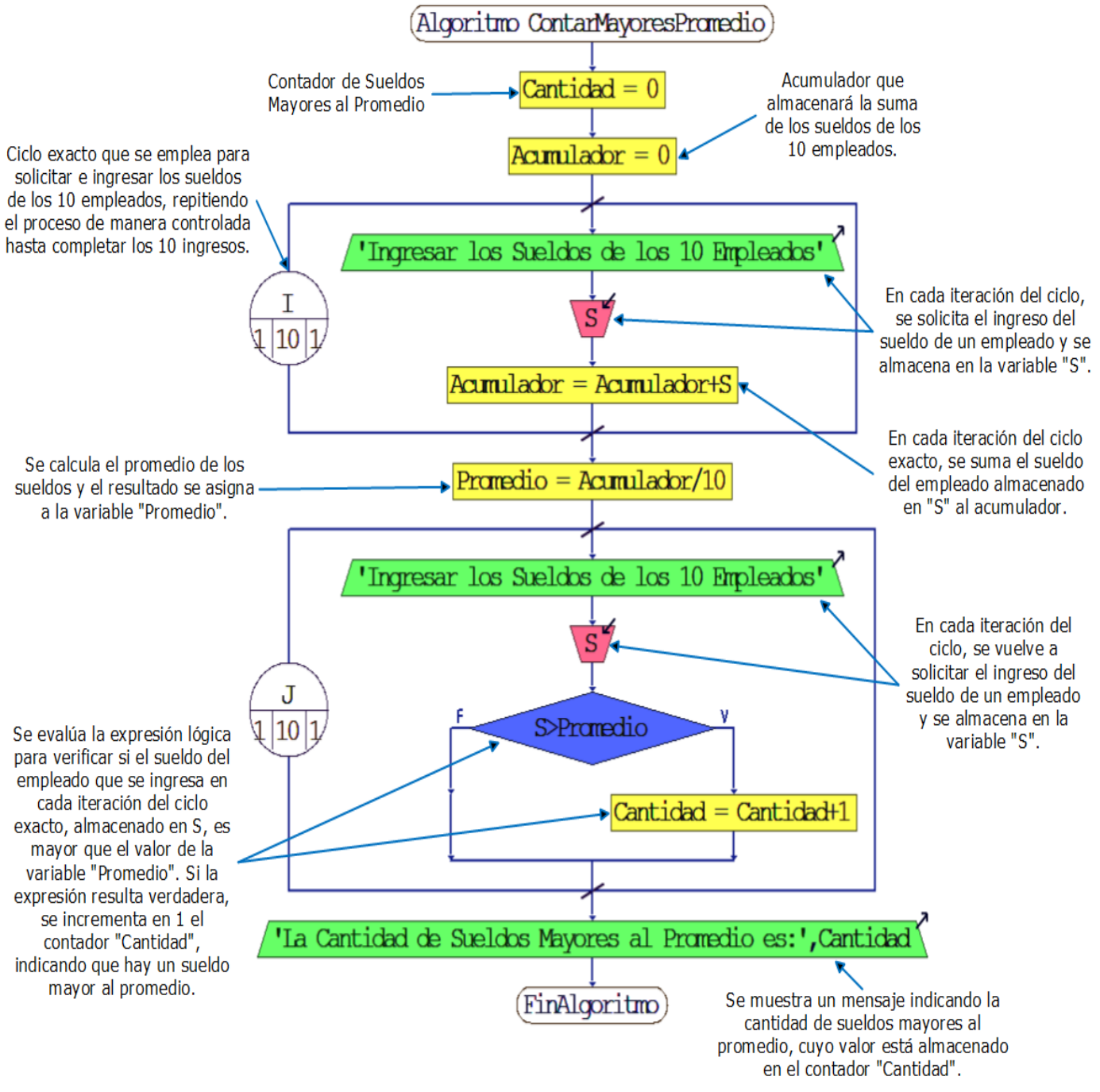
Además, algunos pasos del algoritmo se repiten innecesariamente, ya que no pueden generalizarse. Esta repetición no solo aumenta el trabajo, sino que también incrementa las posibilidades de cometer errores.

Es bien sabido que ejecutar la misma tarea repetidamente, como escribir los mismos pasos varias veces, disminuye la atención y facilita la aparición de errores.


Algoritmo ContarMayoresPromedio
Contador de Sueldos Mayores al Promedio
Cantidad = 0
Acumulador que almacenará la suma de los sueldos de los 10 empleados.
Acumulador = 0
'Ingresar los Sueldos de los 10 Empleados'
Se solicita ingresar los sueldos de los 10 empleados, los cuales se almacenarán en las variables correspondientes: S1, S2, ..., S10.
S1,S2,S3,S4,S5,S6,S7,S8,S9,S10
Se suman los 10 sueldos y se asigna el resultado al acumulador.
Acumulador = S1+S2+S3+S4+S5+S6+S7+S8+S9+S10
Promedio = Acumulador/10
Se calcula el promedio de los sueldos y el resultado se asigna a la variable "Promedio".
F
S1>Promedio
V
Cantidad = Cantidad+1
Se evalúa la expresión lógica para verificar si el sueldo del primer empleado, almacenado en S1, es mayor que el valor de la variable "Promedio". Si la expresión resulta verdadera, se incrementa en 1 el contador "Cantidad", indicando que hay un sueldo mayor al promedio. Esta misma condición se aplica de manera secuencial para los siguientes 9 sueldos.
F
S2>Promedio
V
Cantidad = Cantidad+1
F
S10>Promedio
V
Cantidad = Cantidad+1
'La Cantidad de Sueldos Mayores al Promedio es:', Cantidad
FinAlgoritmo
Se muestra un mensaje indicando la cantidad de sueldos mayores al promedio, cuyo valor está almacenado en el contador "Cantidad".

Solución 2:

En la solución presentada en el siguiente diagrama de flujo, se ha reducido el número de variables en memoria y se han eliminado muchas instrucciones repetitivas. Sin embargo, el usuario de este algoritmo debe ingresar dos veces el conjunto de datos, lo cual es completamente impráctico, especialmente si el número de datos es mayor que 10. Además, esta solución es ineficiente, ya que la operación de lectura, sea de forma interactiva o desde un archivo, debe repetirse, lo que ocasiona una pérdida innecesaria de tiempo.



Se puede observar que ninguna de las dos soluciones propuestas es práctica ni eficiente. Para abordar este tipo de problemas de manera más adecuada, es necesario utilizar un tipo de datos que facilite el manejo de grandes cantidades de información de forma más organizada y eficiente. Los tipos de datos estructurados, como los arrays (arreglos o vectores), son ideales para resolver este tipo de problemas, ya que permiten almacenar y procesar conjuntos de datos relacionados de manera más efectiva.

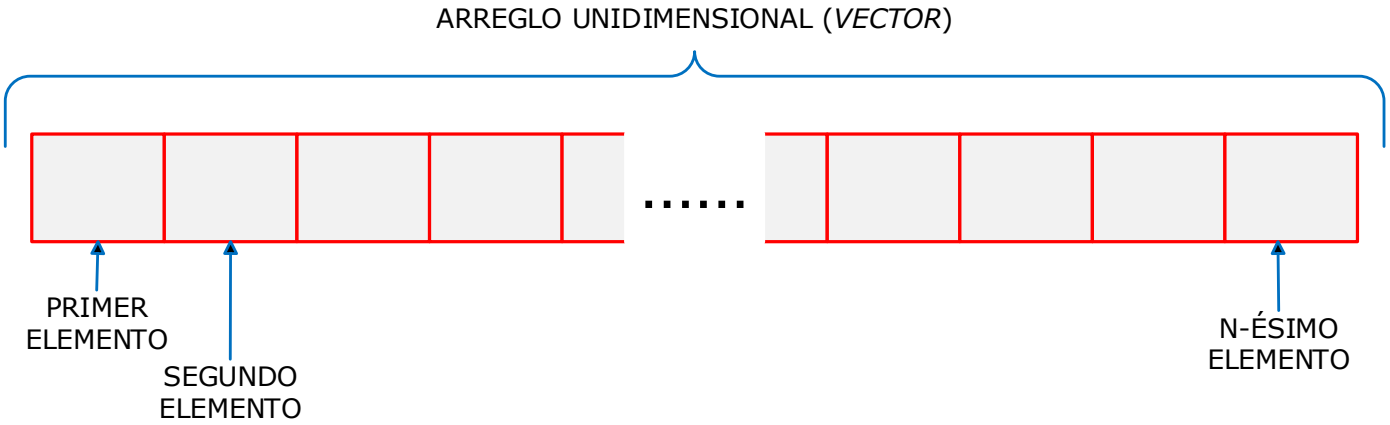
ARRAYS (Arreglos/Vectores) UNIDIMENSIONALES

Un array o arreglo (también llamado matriz o vector) es una colección finita, homogénea y ordenada de elementos. Esto significa que:

- **Finita:** Todo arreglo tiene un límite, es decir, debemos determinar el número máximo de elementos que puede contener.
- **Homogénea:** Todos los elementos del arreglo son del mismo tipo de datos, como enteros, reales o cadenas, pero nunca una combinación de diferentes tipos.
- **Ordenada:** Los elementos dentro del arreglo tienen una posición específica. Podemos identificar claramente el primer elemento, el segundo, el tercero, y así sucesivamente hasta el enésimo elemento.

El tipo más simple de arreglo es el array unidimensional o vector, que se organiza en una sola fila de datos.

Un arreglo puede representarse de manera gráfica como se muestra en la siguiente imagen.



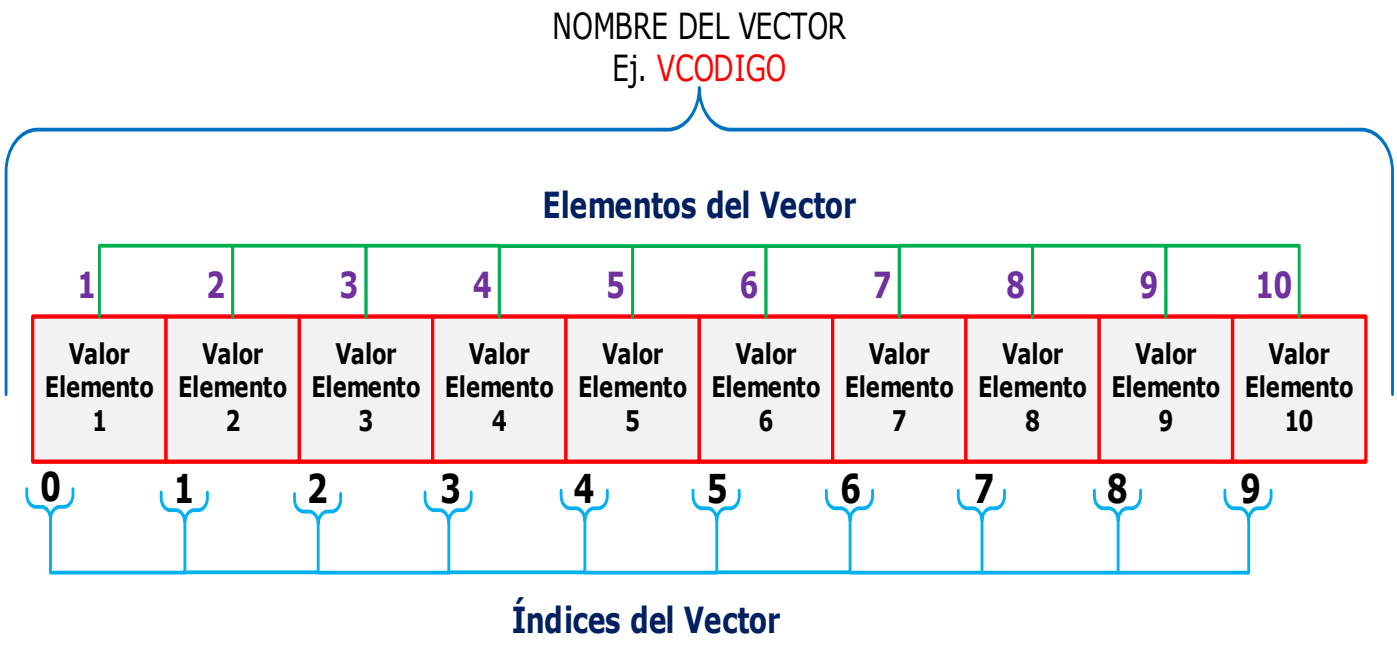
Un vector tiene la capacidad de almacenar N elementos del mismo tipo y permite acceder a cada uno de ellos de manera individual. Para entender mejor cómo funciona, podemos distinguir dos partes fundamentales:

Elementos:

Son los valores que se almacenan en cada posición o casilla del vector. Estos valores pueden ser de cualquier tipo, como enteros, caracteres o decimales, pero todos deben ser del mismo tipo dentro de un vector.

Índices:

Son los números que identifican la posición de cada elemento dentro del vector. Los índices permiten acceder a los elementos de forma individual, y generalmente comienzan en 0 en la mayoría de los lenguajes de programación. Por ejemplo, en un vector de 10 elementos, los índices van de 0 a 9.



En resumen, los elementos son los datos almacenados en el vector, mientras que los índices permiten acceder a estos datos de manera precisa y ordenada.

Para hacer referencia a un elemento de un vector, se necesita:

- El nombre del vector.
- El índice del elemento.

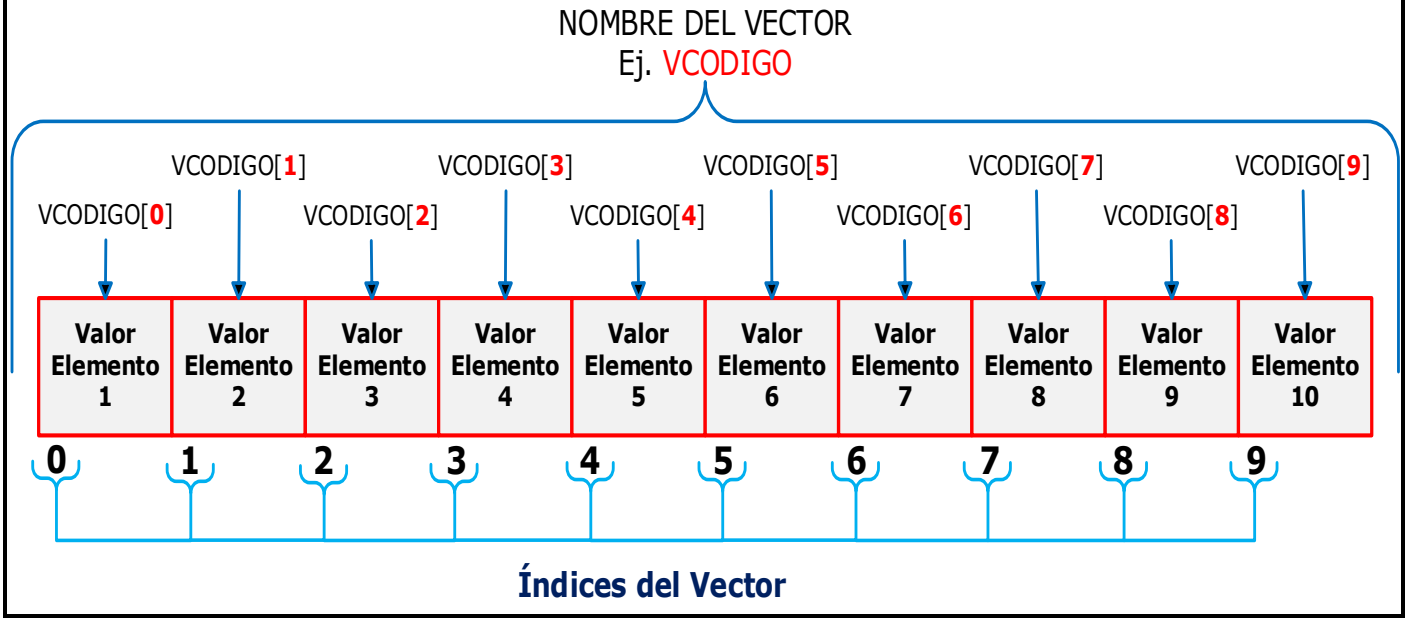
En un vector, los corchetes [ ] se utilizan en el subíndice para acceder o modificar un elemento específico. El valor o expresión que se coloca dentro de los corchetes representa el índice del elemento al que se desea acceder.

NOMBRE_VECTOR [VALOR INDICE]

El nombre asignado al arreglo o vector, sigue las mismas reglas que se utilizan para nombrar una variable.

El valor del índice del elemento del arreglo o vector al que se desea acceder debe ser un número entero.

Ejemplo de cómo acceder a cada uno de los 10 elementos que contiene el vector de nombre **VCODIGO**.



Los subíndices de un vector pueden ser números enteros, variables de tipo entero o expresiones que resulten en un valor entero.

Subíndice con valor numérico literal. (Número fijo y explícito)

VCODIGO[5]

Subíndice con variable. (Del tipo entero)

I=3

VCODIGO[I]

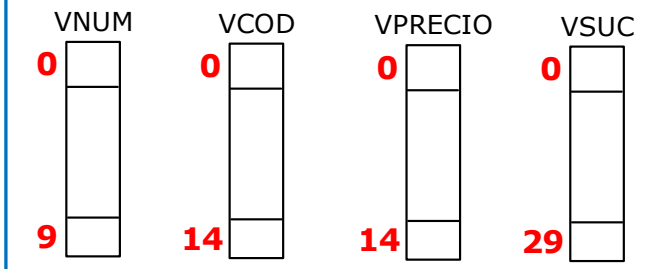
Subíndice con una expresión aritmética. (El resultado de la expresión debe ser un valor entero)

I=0

VCODIGO[I+1]

En los diagramas de flujo, no es necesario declarar un vector de forma explícita como se haría en un lenguaje de programación, ya que los diagramas se centran más en la lógica del proceso que en la implementación técnica. Sin embargo, puedes representar el uso de un vector de manera clara sin requerir una declaración formal. Al inicio del algoritmo, puedes incluir una nota que indique algo como "Vector de N elementos", sin la necesidad de detallar su declaración como en un código.

Formas de anotar vectores en papel.



VNUM[10]

VCOD[15]

VPRECIO[15]

VSUC[30]

En PseInt, es necesario declararlo de la siguiente forma.

```
Dimension NUM[10]
Dimension COD[15]
Dimension PRECIO[15]
Dimension SUC[30]
```

OPERACIONES BÁSICAS CON VECTORES

Estas son algunas de las operaciones más comunes con vectores:

- Lectura (Entrada).
- Escritura (Salida).
- Asignación.

Lectura:

En esta instrucción de lectura (Entrada), el valor ingresado por el usuario se almacena en el elemento del vector que indica el índice.

```
NOMBRE_VECTOR [VALOR INDICE]
```

Escritura:

En esta instrucción de escritura (Salida), se muestra el valor almacenado en el elemento del vector que indica el índice.

```
NOMBRE_VECTOR [VALOR INDICE]
```

Asignación:

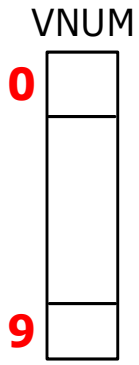
En esta operación, se asigna el valor 5 al elemento del vector que indica el valor del índice.

```
NOMBRE_VECTOR [VALOR INDICE] = 5
```

ACCESO SECUENCIAL AL VECTOR (recorrido)

El proceso de lectura de un vector consiste en asignar un valor a cada uno de sus elementos.

Supongamos que queremos ingresar todos los valores de un vector de forma consecutiva. Por ejemplo, tenemos un vector llamado VNUM con 10 elementos, y en cada uno vamos a ingresar un número entero. Esto podría hacerse de la siguiente manera:




```
VNUM [0],VNUM [1],VNUM [2], ......,VNUM [9]
```

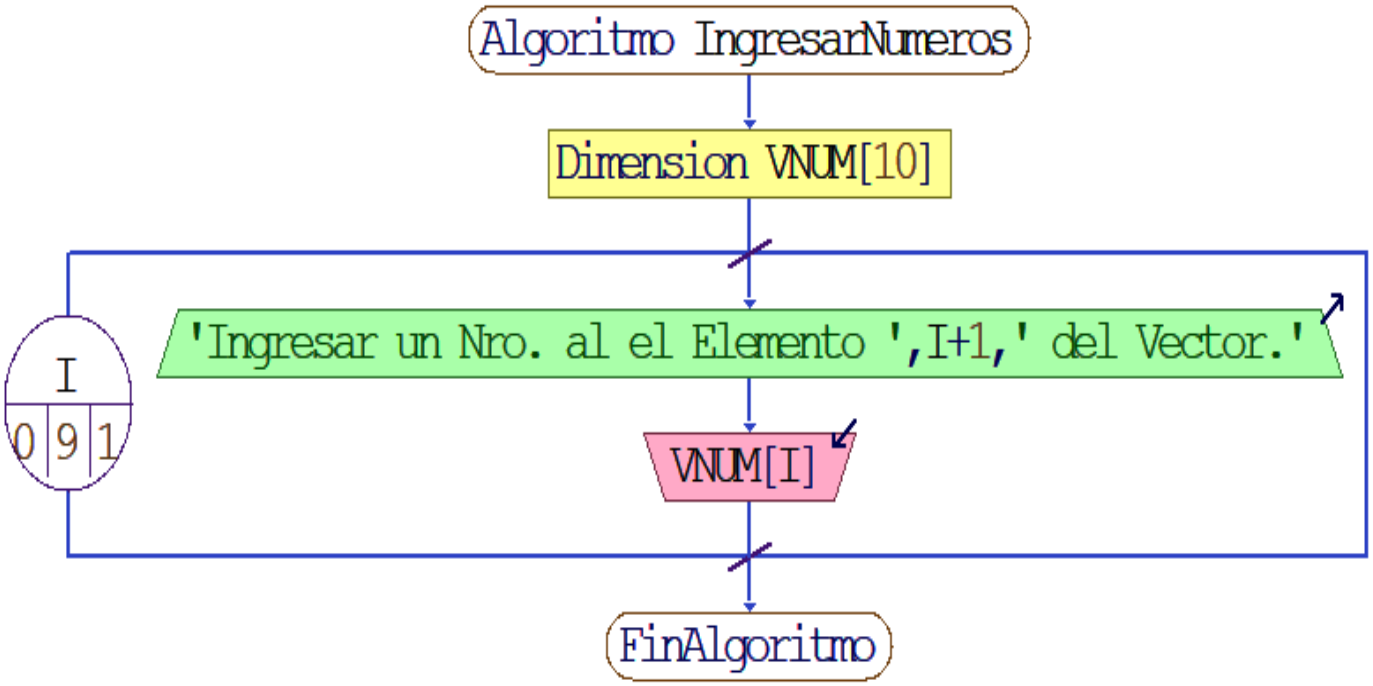
Lo mismo puede hacerse para mostrar el contenido de cada elemento, aunque hacerlo manualmente no es práctico.

Para trabajar con los elementos de un vector, ya sea para ingresar datos o para visualizar su contenido, es más eficiente utilizar un método práctico.

Este proceso de aplicar una acción a todos los elementos de un vector se llama "recorrido del vector". Estas operaciones se realizan utilizando estructuras repetitivas, donde las variables de control (por ejemplo, I) actúan como índices del vector (como VNUM[I]). A medida que el contador del ciclo aumenta, se acceden a los elementos del vector de forma consecutiva para ingresar datos o visualizar los mismos.

Tomemos como ejemplo un algoritmo que solicite y almacene 10 números enteros en un vector de 10 elementos.

El algoritmo recorrerá cada posición del vector, pidiendo al usuario que ingrese un número, el cual se guardará en la posición correspondiente del vector. Este proceso se repetirá hasta completar todas las posiciones, asegurando que cada uno de los 10 números quede almacenado correctamente.

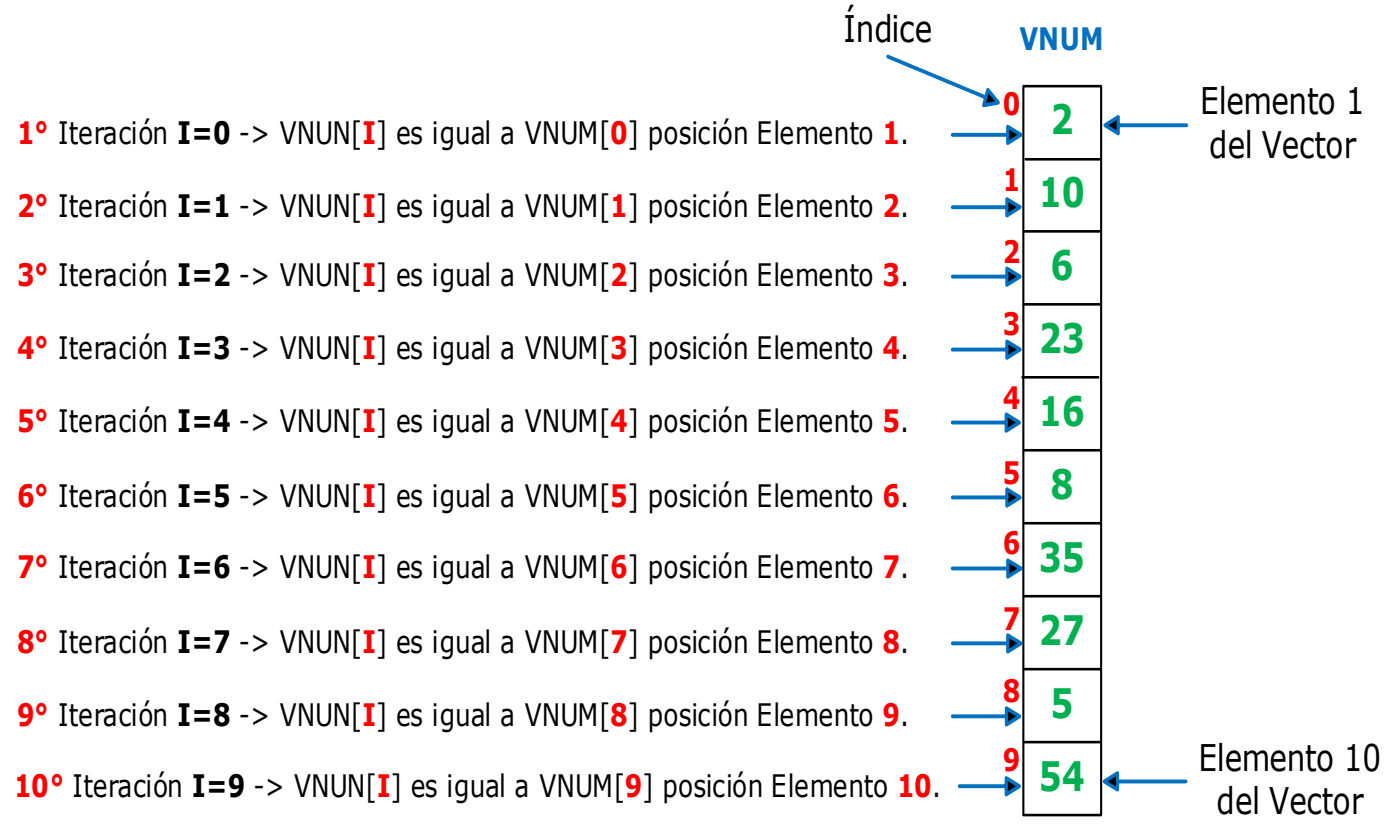


En este ejemplo, utilizamos un vector llamado VNUM con 10 elementos. Recordemos que las posiciones en el vector comienzan desde cero.

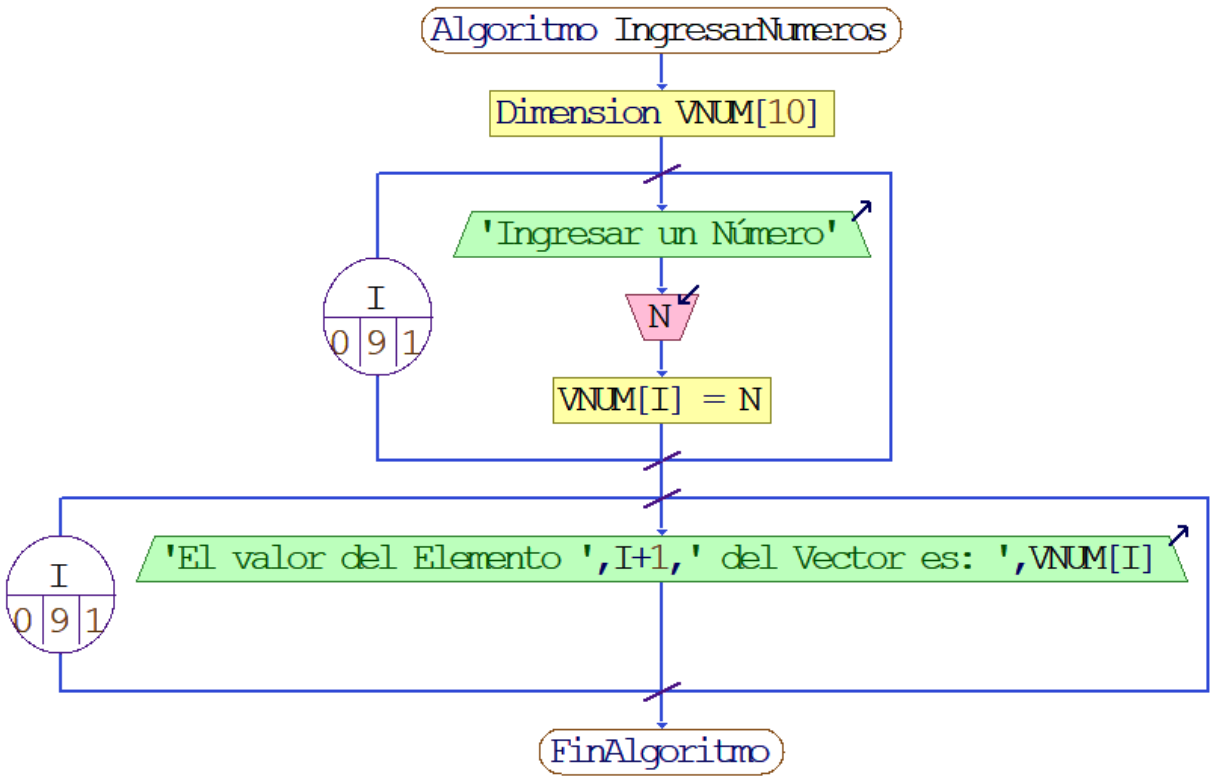
Para recorrerlo, usamos un ciclo exacto con una variable de control, I, que se inicializa en cero y cuyo límite es 9, es decir, el ciclo se ejecutará mientras I sea menor o igual a 9.

En cada iteración, se solicitará un número al usuario y se almacenará en el vector en la posición correspondiente al valor de I en ese momento.

Por ejemplo, si ingresamos la siguiente lista de números en este orden: **{2, 10, 6, 23, 16, 8, 35, 27, 5, 54}**, cada número se guardará en la posición respectiva del vector a medida que avanza el ciclo.

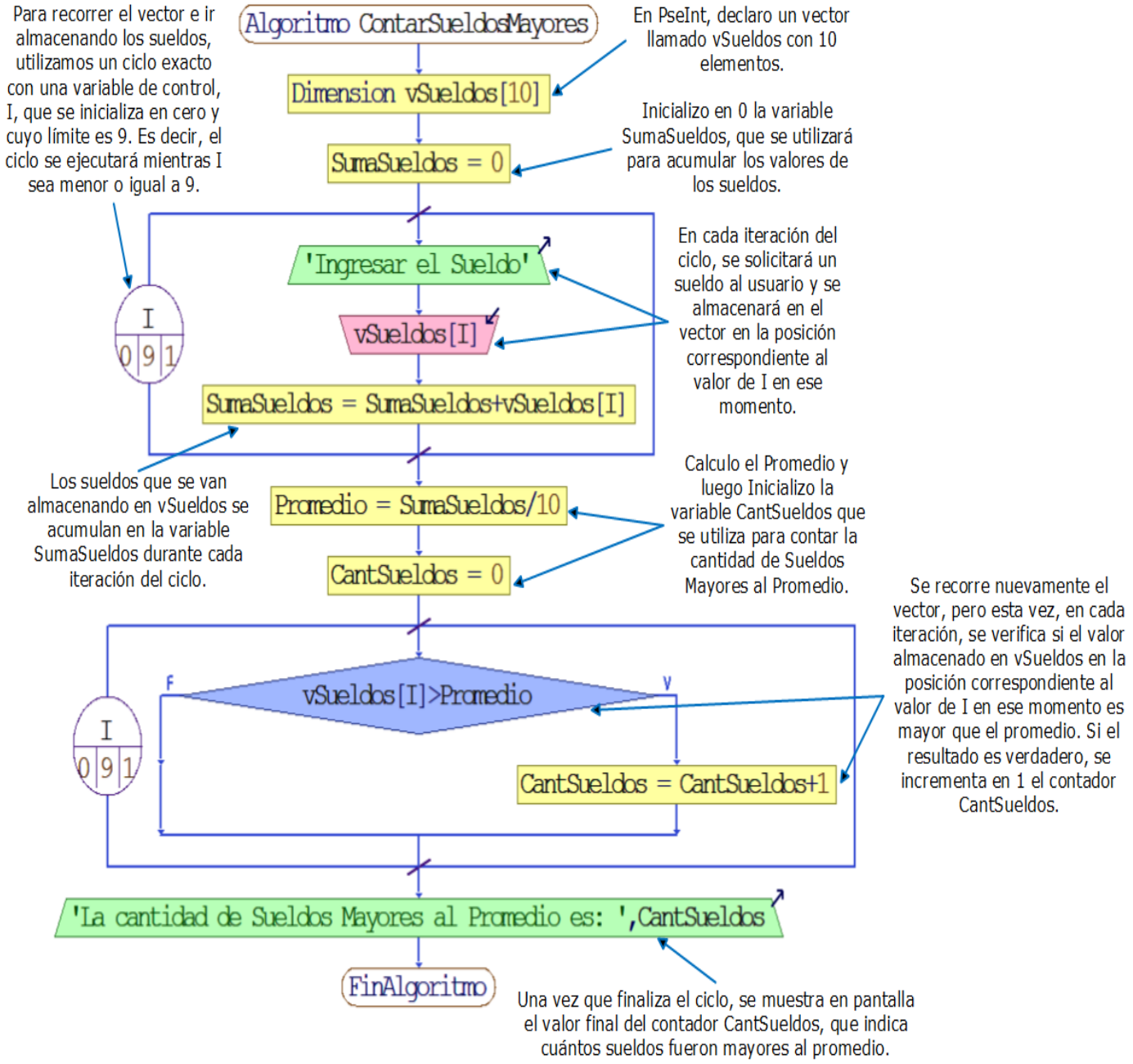


Otra opción es almacenar el número ingresado asignando el valor de N a VNUM en la posición correspondiente al valor de I en ese momento, en lugar de guardar el número directamente al momento de la lectura. Luego, se recorre el vector mostrando los valores almacenados en cada uno de sus elementos.



Sigamos con el ejemplo inicial, donde vimos dos posibles soluciones sin utilizar vectores.

Se tienen los sueldos de un grupo de 10 empleados de una empresa, y se necesita crear un algoritmo que determine cuántos de estos empleados tienen un sueldo superior al promedio del grupo. Este problema puede resolverse mediante un diagrama de flujo que ilustre los pasos necesarios para calcular el promedio y luego comparar los sueldos con dicho valor.



Ahora podemos ver cómo, con el uso de un vector, resolvimos el problema de tener que manejar 10 variables por separado, como se planteaba en la primera solución, o de evitar el doble ingreso de datos (los sueldos) que se mencionaba en la segunda solución.