Capítulo 1

ESTRUCTURAS FUNDAMENTALES DE DATOS

1.1 INTRODUCCIÓN

La importancia de las computadoras radica fundamentalmente en su capacidad para procesar información. Esta característica les permite realizar actividades que antes sólo las realizaban los humanos.

Con el propósito de que la información sea procesada, se requiere que ésta se almacene en la memoria de la computadora. De acuerdo con la forma en que los datos se organizan, se clasifican en:

- Tipos de datos simples.
- Tipos de datos estructurados.

La principal característica de los tipos de datos simples consiste en que ocupan sólo una casilla de memoria (fig. 1.1*a*); por tanto, una variable simple hace referencia a un único valor a la vez. En este grupo de datos se encuentran: números enteros y reales, caracteres, booleanos, enumerados y subrangos. Cabe señalar que los dos últimos no existen en algunos lenguajes de programación.

Por otra parte, los tipos de datos estructurados se caracterizan por el hecho de que con un nombre —identificador de variable estructurada— se hace referencia a un grupo de casillas de memoria (fig. 1.1*b*). Es decir, un tipo de dato estructurado tiene varios componentes. Cada uno de éstos puede ser un tipo de dato simple o estructurado. Sin embargo, los componentes básicos, los del nivel más bajo, de cualquier tipo de datos estructurado son siempre tipos de datos simples.

El estudio de las estructuras de datos constituye una de las principales actividades para llegar al desarrollo de grandes sistemas de software. En este capítulo se tratarán las estructuras de datos básicos que son útiles para la mayoría de los lenguajes de programación. Éstas son: arreglos y registros.

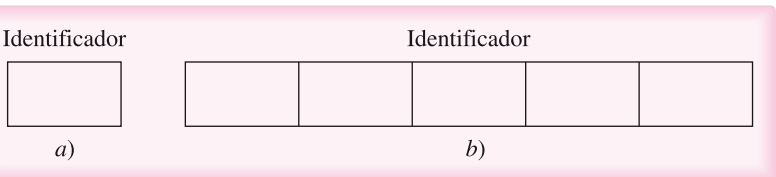


FIGURA 1.1
Tipos de datos simples y estructurados.
a) Dato simple.
b) Dato estructurado.

1.2 ARREGLOS

Con frecuencia se presentan en la práctica problemas cuya solución no resulta fácil —a veces es imposible— si se utilizan tipos de datos simples.

Con el propósito de ilustrar esta dificultad, a continuación se presentarán un problema y dos de sus posibles soluciones mediante tipos simples de datos. El objetivo de este ejemplo es demostrar lo complejo que resulta un algoritmo de solución para ciertos problemas, si no se utilizan tipos de datos estructurados. Finalmente, y luego de presentar los arreglos, se ofrecerá una solución al problema mencionado en primer término usando arreglos.

Ejemplo 1.1

Consideremos que en una universidad se conocen las calificaciones de un grupo de 50 alumnos. Se necesita saber cuántos de éstos tienen calificación superior al promedio del grupo.

¿Cómo resolver este problema?

Primera solución
Algoritmo 1.1 Doble_lectura

Doble_lectura

{Este algoritmo resuelve el problema planteado en el ejemplo 1.1, realizando dos veces la lectura de los datos}

{*I* y CONT son variables de tipo entero. *AC*, PROM y *C* son variables de tipo real}

1. Hacer $AC \leftarrow 0$ e $I \leftarrow 1$
2. Mientras ($I \leq 50$) *Repetir*
    Escribir "Ingrese la calificación", *I*
    Leer *C*
    Hacer $AC \leftarrow AC + C$ e $I \leftarrow I + 1$
3. {Fin del ciclo del paso 2}
4. Hacer PROM $\leftarrow AC/50$

{Como se necesita indicar cuántos alumnos obtuvieron calificación superior al promedio, se releerán las 50 calificaciones para comparar cada una de ellas con el promedio calculado en el paso 4}

```
   Hacer CONT ← 0 e I ← 1
5. Mientras (I ≤ 50) Repetir
        Escribir "Ingrese la calificación", I
        Leer C
   5.1 Si C > PROM entonces
           Hacer CONT ← CONT + 1
   5.2 {Fin del condicional del paso 5.1}
           Hacer I ← I + 1
6. {Fin del ciclo del paso 5}
7. Escribir CONT
```

Segunda solución

Algoritmo 1.2 Muchas_variables

Muchas_variables

{Este algoritmo resuelve el problema planteado en el ejemplo 1.1, pero ahora mediante muchas variables}

{CONT es una variable de tipo entero. PROM, AC y C_i son variables de tipo real}

```
1.  Leer C1, C2, C3, ..., C50
    {Las calificaciones corresponden a los 50 alumnos}

2.  Hacer AC ← C1 + C2 + C3 + ... + C50,
    PROM ← AC/50 y CONT ← 0

3.  Si C1 > PROM entonces
        Hacer CONT ← CONT + 1

4.  {Fin del condicional del paso 3}
5.  Si C2 > PROM entonces
        Hacer CONT ← CONT + 1

6.  {Fin del condicional del paso 5}
    ...
100. Si C50 > PROM entonces
         Hacer CONT ← CONT + 1

101. {Fin del condicional del paso 100}
102. Escribir CONT
```

Estas dos soluciones son muy representativas de los inconvenientes a los que uno se puede enfrentar, al plantear una solución algorítmica a un problema al usar sólo tipos de datos simples.

En la solución planteada en el algoritmo 1.1 el usuario debe ingresar dos veces el conjunto de datos. Esto último tiene varias desventajas: es totalmente molesto —considere que el número de datos puede ser mayor a 50—, ineficiente —la operación de lectura, ya sea de manera interactiva con el usuario o desde un archivo, se debe repetir, lo que ocasiona pérdida de tiempo— y causa de errores —en los casos donde la entrada de datos se haga de forma manual—.

Por otra parte, en la solución planteada en el algoritmo 1.2 se manejan 50 variables en memoria. Esta solución presenta el inconveniente de que el manejo de las variables se puede tornar incontrolable, sobre todo si su número crece en forma considerable. Además, algunos pasos especificados en el algoritmo, que posteriormente serán instrucciones de algún lenguaje de programación, se repiten, ya que no se pueden generalizar. Esta característica no sólo provoca más trabajo, sino también posibles errores. Es sabido que ejecutar una tarea en forma repetida, en este caso escribir un mismo paso varias veces, resta interés en la acción que se está llevando a cabo, y puede propiciar más errores.

Se observa, entonces, que ninguna de las dos soluciones resulta práctica ni eficiente. Es necesario un tipo de dato que permita manejar mucha información, generalizando sus operaciones. Los tipos de datos estructurados que ayudan a resolver problemas como éste son los arreglos.

Un **arreglo unidimensional** se define como una colección finita, homogénea y ordenada de elementos.

- **Finita:** todo arreglo tiene un límite; es decir, se debe determinar cuál será el número máximo de elementos que formarán parte del arreglo.
- **Homogénea:** todos los elementos de un arreglo son del mismo tipo. Es decir, todos enteros, todos booleanos, etcétera, pero nunca una combinación de distintos tipos.
- **Ordenada:** se puede determinar cuáles son el primero, el segundo, el tercero, ... y el enésimo elementos.

Un arreglo unidimensional se puede representar gráficamente como se muestra en la figura 1.2.

Si un arreglo tiene la característica de que puede almacenar a *N* elementos del mismo tipo, entonces deberá permitir la recuperación de cada uno de ellos. Como consecuencia, se distinguen dos partes fundamentales en los arreglos:

- Los componentes.
- Los índices.

Los primeros hacen referencia a los elementos que forman el arreglo; es decir, a los valores que se almacenan en cada una de sus casillas (fig. 1.3). Considerando el

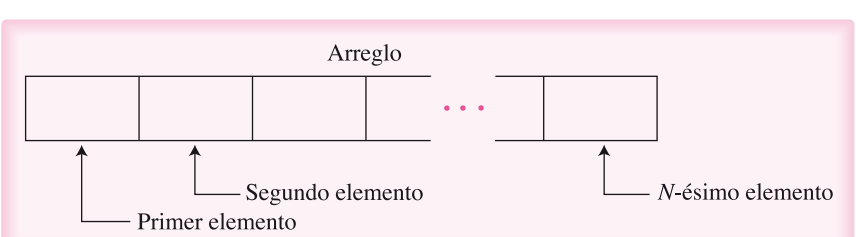


FIGURA 1.2
Representación de arreglos.

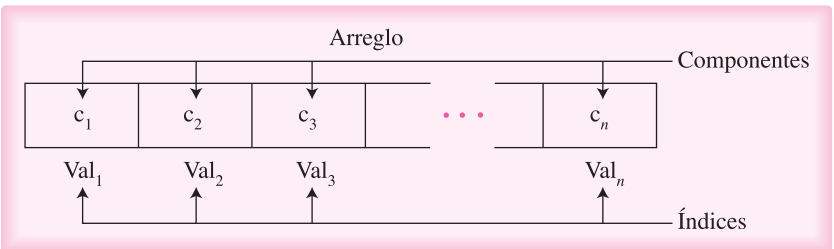


FIGURA 1.3
Índices y componentes de un arreglo.

ejemplo anterior, cada una de las 50 calificaciones será un componente de un arreglo "calificaciones". En este contexto, los índices especifican cuántos elementos tendrá el arreglo y además de qué modo podrán recuperarse esos componentes. Los índices también permiten hacer referencia a los componentes del arreglo en forma individual; es decir, distinguirán entre sus elementos. Por tanto, para hacer referencia a un elemento de un arreglo se debe utilizar:

- El nombre del arreglo.
- El índice del elemento.

En la figura 1.3 se representa un arreglo unidimensional y se indican tanto sus componentes como sus índices.

1.2.1 Declaración de arreglos unidimensionales

No es el propósito de este libro seguir la sintaxis de algún lenguaje de programación en particular; un arreglo unidimensional se define de la siguiente manera:

ident_arreglo = ARREGLO [líminf .. límsup] DE tipo

Con los valores **líminf** y **límsup** se declara el tipo de los índices, así como el número de elementos que tendrá el arreglo. El número total de componentes (NTC) que tendrá el arreglo unidimensional se calcula con

$$NTC = límsup - líminf + 1 \qquad \textbf{Fórmula 1.1}$$

Con **tipo** se declara el tipo de datos para todos los componentes del arreglo unidimensional. El tipo de los componentes no tiene que ser el mismo que el de los índices. En general, los lenguajes de programación establecen restricciones al respecto.

Observaciones:

a) El tipo del índice puede ser cualquier tipo ordinal: carácter, entero, enumerado. En la mayoría de los lenguajes usados actualmente se permite sólo números enteros.

b) El tipo de los componentes puede ser cualquier tipo de datos —entero, real, cadena de caracteres, registro, arreglo, etcétera—.
c) Se utilizan los corchetes "[]" para indicar el índice de un arreglo. Entre [] se debe escribir un valor ordinal; puede ser una variable, una constante o una expresión tan compleja como se quiera, pero que dé como resultado un valor ordinal.

Enseguida se verán algunos ejemplos de arreglos unidimensionales:

Ejemplo 1.2

Sea *V* un arreglo unidimensional de 50 elementos enteros con índices enteros. Su representación se indica en la figura 1.4.

V = ARREGLO[1..50] DE enteros

- NTC = (50 – 1 + 1) = 50.
- Cada componente del arreglo unidimensional V será un número entero, al cual se tendrá acceso por medio de un índice que será un valor comprendido entre 1 y 50.

Por ejemplo:

V[1] hace referencia al elemento de la posición 1.
V[2] hace referencia al elemento de la posición 2.
...
V[50] hace referencia al elemento de la posición 50.

Los índices de tipo entero no necesariamente deben tener un límite inferior igual a cero o a uno. Podrían usarse valores negativos [−10..10] o valores mayores a uno [100..200].

Ejemplo 1.3

Sea *A* un arreglo de 26 elementos booleanos con índices de tipo carácter. Su representación se muestra en la figura 1.5.

A = ARREGLO ['*a*'.. '*z*'] DE booleanos

- NTC = (ord('*z*') - ord('*a*') + 1) = 122 − 97 + 1 = 26.
- Cada componente del arreglo unidimensional *A* será uno de los dos posibles valores lógicos (VERDADERO o FALSO) al cual se tendrá acceso por medio de un índice, que será un valor comprendido entre los caracteres '*a*' y '*z*'.

Por ejemplo:

A['*a*'] hace referencia al elemento de la posición '*a*' (1era.)
A['*b*'] hace referencia al elemento de la posición '*b*' (2da.)

FIGURA 1.4

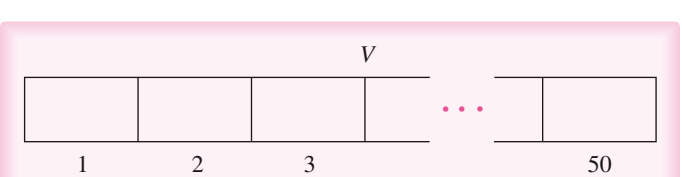
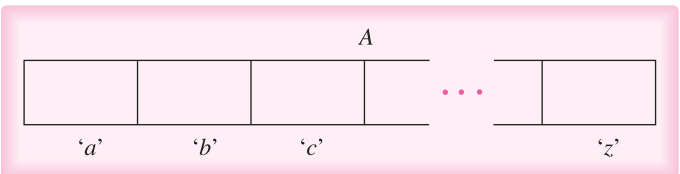


FIGURA 1.5



...
A['z'] hace referencia al elemento de la posición 'z' (26)

Ejemplo 1.4

Sea CICLO un arreglo de 12 elementos reales con índices de tipo escalar o enumerados. Su representación se muestra en la figura 1.6.

meses = (ene, feb, mar, abr, may, jun, jul, ago, sept, oct, nov, dic)

CICLO = ARREGLO [meses] DE reales

- NTC = (ord(dic) − ord(ene) + l) = 11 – 0 + 1 = 12.
- Cada componente del arreglo unidimensional CICLO será un número real, al cual se tendrá acceso por medio de un índice, que será un valor comprendido entre ene y dic.

Por ejemplo:

CICLO[ene] hace referencia al elemento de la posición ene (1era.)
CICLO[feb] hace referencia al elemento de la posición feb (2da.)
...
CICLO[dic] hace referencia al elemento de la posición dic (12ava.)

1.2.2 Operaciones con arreglos unidimensionales

Como ya se mencionó, los arreglos se utilizan para almacenar datos. Por tanto, resulta necesario leer, escribir, asignar o simplemente modificar datos en un arreglo. Asimismo, al considerar que es una estructura, a una colección de elementos se deben incorporar nuevos elementos, así como eliminar algunos de los ya almacenados. Las operaciones válidas en arreglos son las siguientes:

- Lectura/Escritura.
- Asignación.

FIGURA 1.6

- Actualización: Inserción.
  Eliminación.
  Modificación.
- Ordenación.
- Búsqueda.

Como los arreglos son tipos de datos estructurados, muchas de estas operaciones no se pueden llevar a cabo de manera global; es decir, tratando al arreglo como un todo, sino que se debe trabajar sobre cada componente.

A continuación se analizará cada una de estas operaciones. Cabe destacar que las dos últimas, ordenación y búsqueda, serán tema de estudio en próximos capítulos. Para ilustrarlas se utilizarán los ejemplos presentados anteriormente.

Lectura

El proceso de lectura de un arreglo consiste en leer y asignar un valor a cada uno de sus componentes. Suponga que se desea leer todos los elementos del arreglo unidimensional *V* en forma consecutiva. Se podría hacer de la siguiente manera:

Leer *V*[1],
Leer *V*[2],
...
Leer *V*[50]

Pero es importante que el lector observe que de esta forma no resulta práctico. Por tanto, se usará un ciclo para leer todos los elementos del arreglo unidimensional.

Repetir con *I* desde 1 hasta 50
  Leer *V*[*I*]

Al variar el valor de *I*, cada elemento leído se asigna al correspondiente componente del arreglo según la posición indicada por *I*.

Para *I* = 1, se lee *V*[1]
  I = 2, se lee *V*[2]
  ...
  I = *N*, se lee *V*[*N*]

Al finalizar el ciclo de lectura se tendrá asignado un valor a cada uno de los componentes del arreglo unidimensional *V*. El arreglo se muestra en la figura 1.7.

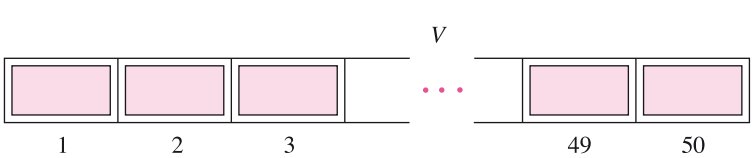


FIGURA 1.7
Lectura de arreglos.

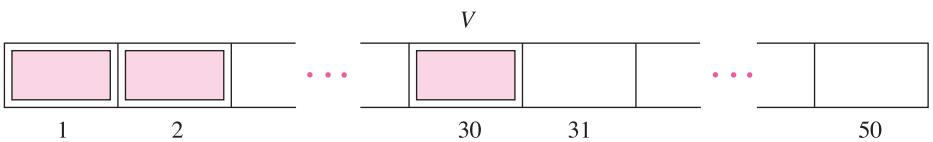


FIGURA 1.8
Lectura de arreglos.

Puede suceder que no se necesiten leer todos los componentes del arreglo, sino solamente alguno de ellos. Supongamos que se deben leer los elementos con índices comprendidos entre el 1 y el 30. A continuación se muestra el ciclo que se necesita para realizar esta operación:

Repetir con *I* desde 1 hasta 30
  Leer *V*[*I*]

El arreglo se muestra en la figura 1.8.

Escritura

El caso de la operación de escritura es similar al de lectura. Se debe escribir el valor de cada uno de los componentes. Supongamos que se desea escribir los primeros *N* componentes del arreglo unidimensional *V* en forma consecutiva. Los pasos a seguir son:

Repetir con *I* desde 1 hasta *N*
  Escribir *V*[*I*]

Al variar el valor de *I* se escribe el elemento del arreglo unidimensional *V*, correspondiente a la posición indicada por *I*.

Para *I* = 1, se escribe el valor de *V*[1]
  I = 2, se escribe el valor de *V*[2]
  ...
  I = *N*, se escribe el valor de *V*[*N*]

Asignación

En general, no es posible asignar directamente un valor a todo el arreglo, sino que se debe asignar el valor deseado a cada componente. Enseguida se analizan algunos ejemplos de asignación.

Observe que en los dos primeros casos se asigna un valor a una determinada casilla del arreglo, en el primero a la señalada por el índice ene, y en el segundo a la indicada por el índice mar.

CICLO[ene] ← 123.89
CICLO[mar] ← CICLO[ene]/2

En el tercer caso se asigna el 0 a todas las casillas del arreglo, con lo que éste queda como se muestra en la figura 1.9.

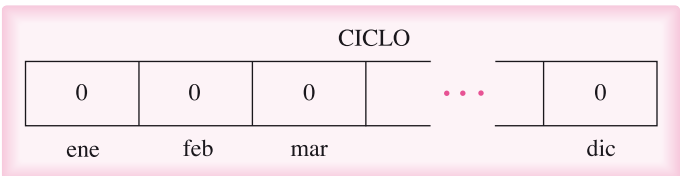


FIGURA 1.9
Asignación de arreglos.

Repetir con MES desde ene hasta dic
  Hacer CICLO[MES] ← 0

Cabe destacar que en algunos lenguajes de programación es posible asignar una variable tipo arreglo a otra del mismo tipo.

$V_1 \leftarrow V$

La expresión anterior es equivalente a realizar lo siguiente:

Repetir con *I* desde 1 hasta 50
  Hacer $V_1[I] \leftarrow V[I]$

Actualización

La actualización es una operación que se realiza en forma frecuente en los arreglos. La cantidad de actualizaciones está relacionada con el tipo de problema que se intente resolver. A diferencia de las otras operaciones estudiadas, la actualización lleva implícita otros tipos de operaciones, como inserción y eliminación de elementos.

Con el propósito de realizar una actualización de manera eficiente, es importante conocer si el arreglo está o no ordenado; es decir, si sus componentes respetan algún orden, ya sea creciente o decreciente. Cabe destacar que las operaciones de inserción, eliminación y modificación serán tratadas en forma separada para arreglos ordenados y desordenados.

Finalmente, es importante señalar que la operación de búsqueda se utiliza como auxiliar en las operaciones de inserción, eliminación y modificación. Esta es la principal razón por la cual a continuación se presenta el algoritmo de búsqueda secuencial en arreglos desordenados. En el capítulo correspondiente a métodos de búsqueda se tratará con mayor detalle este tema.

Algoritmo 1.3 Busca_secuencial_desordenado

Busca_secuencial_desordenado

{El algoritmo busca en forma secuencial un elemento en un arreglo unidimensional que se encuentra desordenado. *V* es un arreglo de 100 elementos, *N* el número actual de elementos y *X* el valor a buscar}
{*I* es una variable auxiliar de tipo entero}

1. Hacer *I* ← 1

2. Mientras ($I \leq N$) y ($X \neq V[I]$) *Repetir*

Hacer $I \leftarrow I + 1$

3. {Fin del ciclo del paso 2}
4. *Si* $I > N$ {No se encontró el valor buscado}

entonces
Escribir "El valor X no está en el arreglo"
si no Escribir "El valor X está en la posición I"

5. {Fin del condicional del paso 4}

Este método de búsqueda es sencillo, aunque no muy eficiente. Consiste en recorrer el arreglo, comparando cada elemento del mismo con el valor a buscar. El proceso se repite hasta que el valor se encuentre —éxito— o hasta que se haya superado el tamaño del arreglo —fracaso—.

a) Arreglos desordenados Considere un arreglo unidimensional V de 100 elementos, como el que se presenta en la figura 1.10. Observe que los primeros N componentes tienen asignado un valor.

a.1) Inserción: Para insertar un elemento Y en un arreglo unidimensional V desordenado, se debe verificar que exista espacio. Si se cumple esta condición, entonces se asignará en la posición $N + 1$ el nuevo elemento y se incrementará en N el total de elementos del arreglo.

A continuación se presenta el algoritmo de inserción en arreglos unidimensionales desordenados.

Algoritmo 1.4 Inserta_desordenado

Inserta_desordenado (V, N, Y)

{El algoritmo inserta un elemento en un arreglo unidimensional desordenado. V es un arreglo de máximo 100 elementos. N es el número actual de elementos. Y representa el valor a insertar}

1. *Si* $N < 100$

entonces
Hacer $N \leftarrow N + 1$ y $V[N] \leftarrow Y$
si no {No hay espacio en el arreglo}
Escribir "El valor Y no se puede insertar. No hay espacio"

2. {Fin del condicional del paso 1}

Luego de la inserción el arreglo unidimensional V queda como se muestra en la figura 1.10*a*.

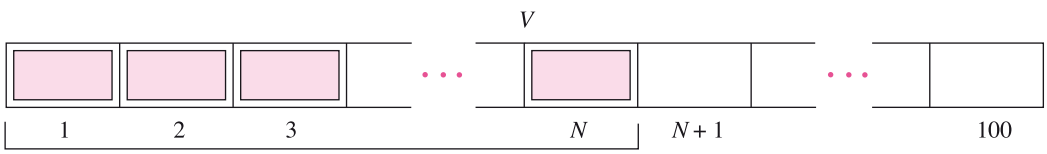


FIGURA 1.10
Actualización de arreglos desordenados.

a.2) Eliminación: Para eliminar un elemento X de un arreglo unidimensional V desordenado, se debe verificar que X se encuentre en el arreglo. Si se cumple esta condición, entonces se procederá a recorrer todos los elementos que están a su derecha una posición a la izquierda, disminuyendo en uno el número de componentes del arreglo.

A continuación se presenta el algoritmo de eliminación en arreglos desordenados. Cabe destacar que la operación de búsqueda presentada en el algoritmo 1.3 se usa para determinar si el elemento X se encuentra en el arreglo. Para el caso de que la respuesta sea positiva, se obtiene también la posición en que se encuentra. Con el propósito de ofrecer mayor claridad en la solución de este problema, se incluye dentro del algoritmo de eliminación el algoritmo de búsqueda secuencial en arreglos desordenados.

Algoritmo 1.5 Elimina_desordenado

Elimina_desordenado (*V*, *N*, *X*)

{El algoritmo elimina un elemento en un arreglo unidimensional desordenado. V es un arreglo de 100 elementos. N es el número actual de elementos. X es el valor a eliminar}
{I y K son variables de tipo entero}

1. Hacer $I \leftarrow 1$
2. Mientras $(I \leq N)$ y $(X \neq V[I])$ *Repetir*

Hacer $I \leftarrow I + 1$

3. {Fin del ciclo del paso 2}
4. *Si* $(I > N)$ {No se encontró el valor buscado}

entonces
Escribir "El valor X no se encuentra en el arreglo"
si no
1.1 *Repetir* con K desde I hasta $(N - 1)$
Hacer $V[K] \leftarrow V[K + 1]$
1.2 {Fin del ciclo del paso 4.1}
Hacer $N \leftarrow N - 1$

5. {Fin del condicional del paso 4}

Luego de la eliminación, el arreglo unidimensional V queda como se muestra en la figura 1.10*b*.

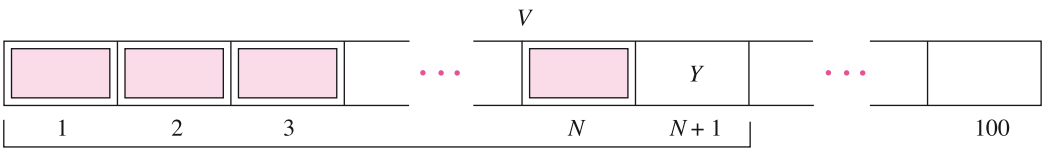


FIGURA 1.10*a*
Inserción en arreglos desordenados.

a.3) Modificación: Para modificar un elemento X de un arreglo unidimensional V desordenado se debe verificar que X se encuentre en el arreglo. Si se cumple esta condición, entonces se procederá a su actualización.

A continuación se presenta el algoritmo de modificación en arreglos desordenados, en el cual se incluye la búsqueda secuencial.

Algoritmo 1.6 Modifica_desordenado

Modifica_desordenado (*V*, *N*, *X*, *Y*)

{El algoritmo modifica un elemento de un arreglo unidimensional desordenado. V es un arreglo de máximo 100 elementos. N es el número actual de elementos. X es el elemento a modificar por el elemento Y}
{I es una variable de tipo entero}

1. Hacer $I \leftarrow 1$
2. Mientras $(I \leq N)$ y $(X \neq V[I])$ *Repetir*

Hacer $I \leftarrow I + 1$

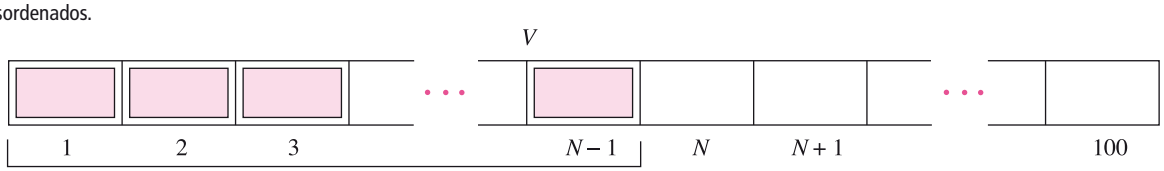
3. {Fin del ciclo del paso 2}
4. *Si* $(I > N)$ {No se encontró el valor buscado}

entonces
Escribir "El valor X no se encuentra en el arreglo"
si no
Hacer $V[I] \leftarrow Y$

5. {Fin del condicional del paso 4}

Luego de la modificación, el arreglo unidimensional V queda como se muestra en la figura 1.10*c*.

FIGURA 1.10*b*
Eliminación en arreglos desordenados.



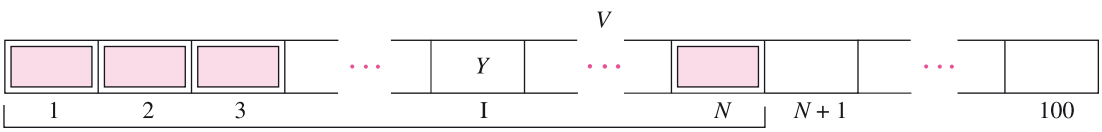


FIGURA 1.10*c*
Modificación en arreglos desordenados.

***b)* Arreglos ordenados** Considere el arreglo unidimensional ordenado V de 100 elementos de la figura 1.11. Los primeros N componentes del mismo tienen asignado un valor. En este caso se trabajará con un arreglo ordenado de manera creciente, es decir:

$$V[1] \leq V[2] \leq V[3] \leq \ldots \leq V[N]$$

Cuando se trabaja con arreglos ordenados se debe evitar alterar el orden al insertar nuevos elementos o al modificar los existentes.

*b.*1) Inserción: Para insertar un elemento X en un arreglo unidimensional V ordenado, primero se debe verificar que exista espacio. Luego se encontrará la posición en la que debería estar el nuevo valor para no alterar el orden del arreglo. Cuando se detecte la posición, se procederá a recorrer todos los elementos desde ahí hasta la N-ésima posición, un lugar a la derecha. Finalmente se asignará el valor de X en la posición encontrada. Cabe destacar que el desplazamiento no se lleva a cabo cuando el valor a insertar es mayor que el último elemento del arreglo.

Generalmente, cuando se quiere hacer una inserción se debe verificar que el elemento no se encuentre en el arreglo. En la mayoría de los casos prácticos no interesa tener información duplicada; por tanto, si el valor que se desea insertar ya estuviera en el arreglo, la operación no se llevará a cabo.

Antes de presentar el algoritmo de inserción, se definirá una función de búsqueda auxiliar, para arreglos ordenados, que se utilizará tanto en el proceso de inserción como en el de eliminación. Esta función es una variante de la presentada en el algoritmo 1.3, y da como resultado la posición en la que encontró al elemento X o el negativo de la posición en la que debería estar. Para mayor información sobre algoritmos de búsqueda, consulte el capítulo 9.

Algoritmo 1.7 Busca_secuencial_ordenado

Busca_secuencial_ordenado (V, N, X, POS)

{El algoritmo busca un elemento X en un arreglo unidimensional V de N elementos que se encuentra ordenado crecientemente. POS indica la posición de X en V o la posición en la que estaría X}
{I es una variable de tipo entero}

1. Hacer $I \leftarrow 1$
2. Mientras ($I \leq N$) y ($V[I] < X$) *Repetir*

Hacer $I \leftarrow I + 1$

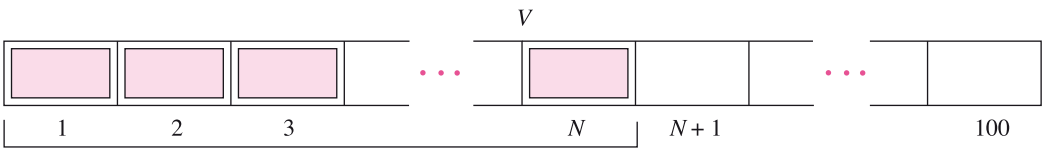


FIGURA 1.11
Actualización de arreglos ordenados.

```
3. {Fin del ciclo del paso 2}
4. Si ((I > N) o (V[I] > X))
       entonces
           Hacer POS ← −I
       si no
           Hacer POS ← I
5. {Fin del condicional del paso 4}
```

A continuación se presenta el algoritmo de inserción en un arreglo unidimensional que se encuentra ordenado en forma creciente.

Algoritmo 1.8 Inserta_ordenado

Inserta_ordenado (*V*, *N*, *Y*)

```
{Este algoritmo inserta un elemento Y en un arreglo unidimensional que se encuentra ordenado
de forma creciente. La capacidad máxima del arreglo es de 100 elementos. N indica el número
actual de elementos de V}
{POS e I son variables de tipo entero}

1. Si (N < 100)
       entonces
           Llamar al algoritmo Busca_secuencial_ordenado con V, N, Y y POS
   1.1 Si POS > 0 {El elemento fue encontrado en el arreglo}
           entonces
               Escribir "El elemento ya existe"
           si no
           Hacer N ← N + 1 y POS ← POS * (−1)
       1.1.1 Repetir con I desde N hasta POS + 1
           Hacer V[I] ← V[I − 1]
       1.1.2 {Fin del ciclo del paso 1.1.1}
           Hacer V[POS] ← Y
   1.2 {Fin del condicional del paso 1.1}
       si no
               Escribir "No hay espacio en el arreglo"
2. {Fin del condicional del paso 1}
```

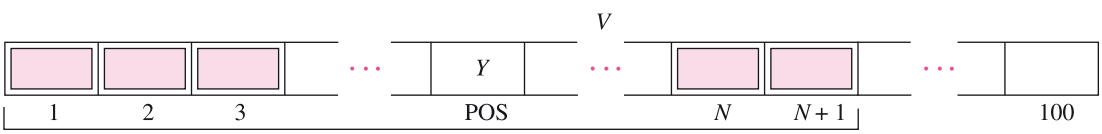


FIGURA 1.11*a*
Inserción en arreglos ordenados.

Luego de la inserción, el arreglo queda como se muestra en la figura 1.11*a*.

b.2) Eliminación: Para eliminar un elemento *X* de un arreglo unidimensional ordenado *V* se debe buscar la posición del elemento a eliminar. Si el resultado de la función es un valor positivo, significa que el elemento se encuentra en el arreglo y, por tanto, se puede eliminar; en caso contrario, no se puede realizar la operación de eliminación.

A continuación se presenta el algoritmo de eliminación en arreglos ordenados.

Algoritmo 1.9 Elimina_ordenado

Elimina_ordenado (*V*, *N*, *X*)

{El algoritmo elimina un elemento *X* de un arreglo unidimensional *V* de *N* elementos que se encuentra ordenado en forma creciente}

{POS e *I* son variables de tipo entero}

```
1. Si (N > 0)
        entonces
        Llamar al algoritmo Busca_secuencial_ordenado con V, N, X y POS
   1.1  Si (POS < 0) {No se puede eliminar porque X no existe}
                entonces
                    Escribir "El elemento no existe"
                si no
                Hacer N ← N – 1
        1.1.1  Repetir con I desde POS hasta N
                        Hacer V[I] ← V[I + 1]
        1.1.2  {Fin del ciclo del paso 1.1.1}
   1.2  {Fin del condicional del paso 1.1}
        si no
                        Escribir "El arreglo está vacío"
2. {Fin del condicional del paso 1}
```

Luego de la eliminación, el arreglo queda como se muestra en la figura 1.11*b*.

b.3) Modificación: Esta operación consiste en reemplazar un componente del arreglo con otro valor. Para ello, primero se buscará el elemento en el arreglo. Si se encuentra, antes de realizar el cambio se debe verificar que el orden del arreglo no se altere. Si esto llegara a suceder, entonces es necesario realizar dos opera-

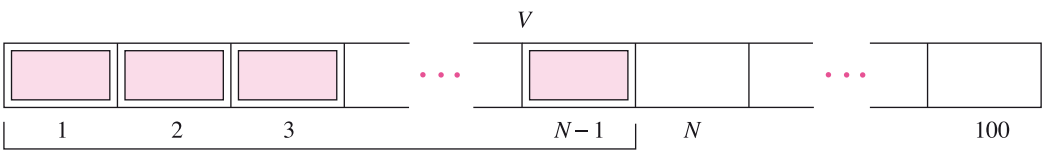


FIGURA 1.11*b*
Eliminación en arreglos ordenados.

ciones; primero se debe eliminar el elemento que se quiere modificar y luego insertar en la posición correspondiente el nuevo valor. Como consecuencia de que las operaciones que se necesitan para realizar una modificación ya han sido presentadas, se deja como tarea la construcción del algoritmo de modificación en arreglos ordenados.

Hasta el momento se ha analizado cómo declarar arreglos y cómo usarlos. Ahora se puede dar solución al problema del ejemplo 1.1 mediante este tipo de estructura de datos.

Algoritmo 1.10 Con_arreglos

Con_arreglos (CAL)

{Este algoritmo resuelve el problema del ejemplo 1.1 al aplicar arreglos unidimensionales. CAL es un arreglo de 50 elementos de números reales}
{*AC*, *I* y CONT son variables de tipo entero. PROM es una variable de tipo real}

1. Hacer *AC* ← 0
2. *Repetir* con *I* desde 1 hasta 50
  Leer CAL[*I*]
  Hacer *AC* ← *AC* + CAL[*I*] e I ← *I* + 1
3. {Fin del ciclo del paso 2}
4. Hacer PROM ← *AC*/50 y CONT ← 0
5. *Repetir* con *I* desde 1 hasta 50
  5.1 *Si* (CAL[*I*] > PROM) *entonces*
    Hacer CONT ← CONT + 1
  5.2 {Fin del condicional del paso 5.1}
6. {Fin del ciclo del paso 5}
7. Escribir CONT

Ésta es una solución más eficiente que las que se presentaron en los algoritmos 1.1 y 1.2. Se realiza una lectura de los datos y además se define una variable para almacenar las 50 calificaciones.

Al utilizar un arreglo puede disponerse de los datos tantas veces como sea necesario sin que se deba volver a leerlos, ya que éstos permanecen en memoria. Además se facilita el procesamiento de los datos, al generalizar ciertas operaciones.

Los arreglos presentados hasta el momento se denominan **arreglos unidimensionales** o **lineales**, debido a que cualquier elemento se referencia solamente con un índice.