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.

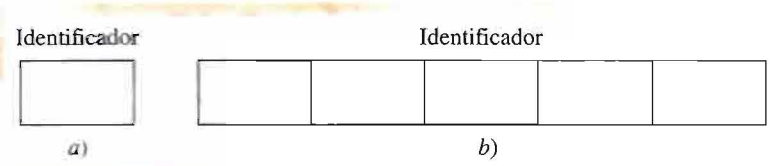


FIGURA 1.1
Tipos de datos simples y estructurados.
a) Dato simple.
b) Dato estructurado.

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 C_1, C_2, C_3, ..., C_{50}
   {Las calificaciones corresponden a los 50 alumnos}

2. Hacer $AC \leftarrow C_1 + C_2 + C_3 + ... + C_{50}$,
   PROM ← AC/50 y CONT ← 0

3. *Si* C_1 > PROM *entonces*
   Hacer CONT ← CONT + 1

4. {Fin del condicional del paso 3}
5. *Si* C_2 > PROM *entonces*
   Hacer CONT ← CONT + 1

6. {Fin del condicional del paso 5}
   ...
100. *Si* C_{50} > 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

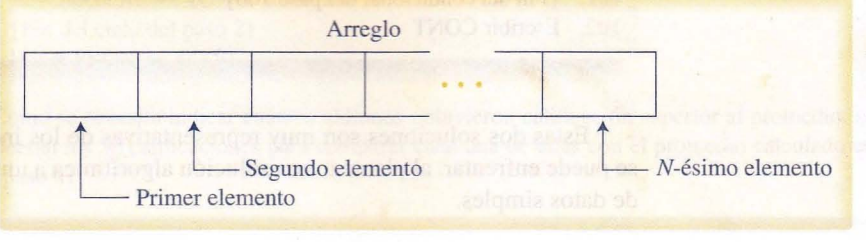
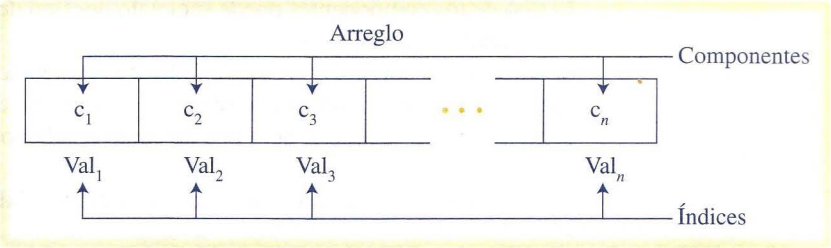


FIGURA 1.2
Representación de arreglos.

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

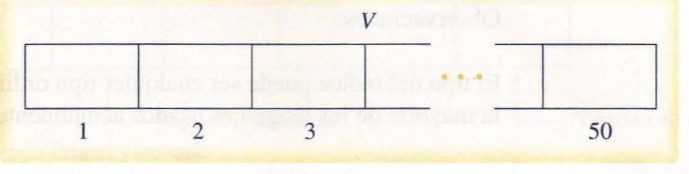
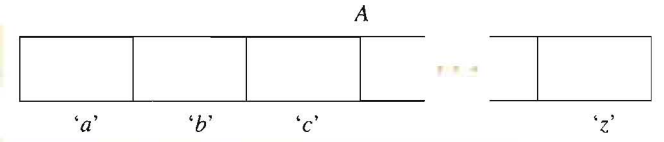


FIGURA 1.5



...
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) + 1) = 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:

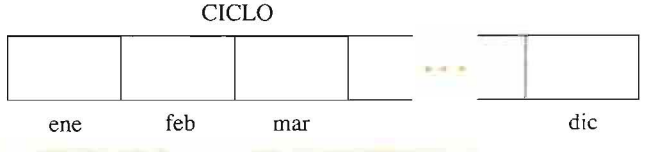
CICLO[ene] hace referencia al elemento de la posición ene (1era.)
CICLO[feb] hace referencia al elemento de la posición feb (2da.)
...
CICLO[dic] hace referencia al elemento de la posición dic (12ava.)

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.

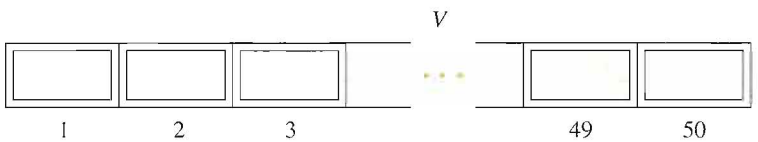


FIGURA 1.7
Lectura de arreglos.

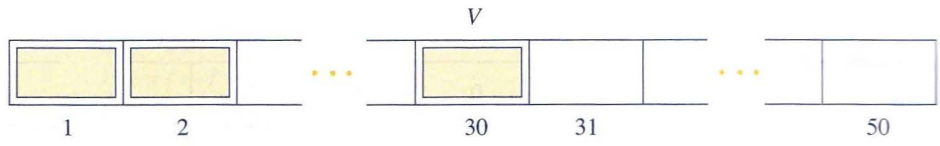


FIGURA 1.8
Lectura de arreglos.

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

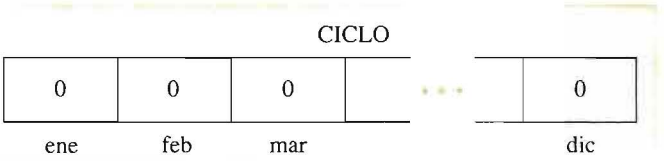
En general, no es posible asignar directamente un valor a todo el arreglo, sino que se debe asignar el valor deseado a cada componente. Enseguida se analizan algunos ejemplos de asignación.

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*.

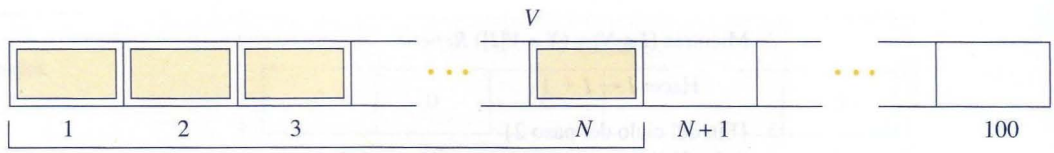


FIGURA 1.10
Actualización de arreglos desordenados.

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*.

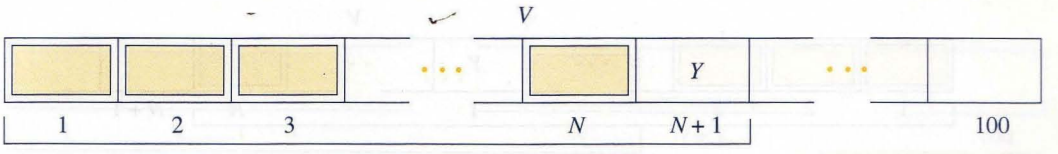


FIGURA 1.10*a*
Inserción en arreglos desordenados.

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"

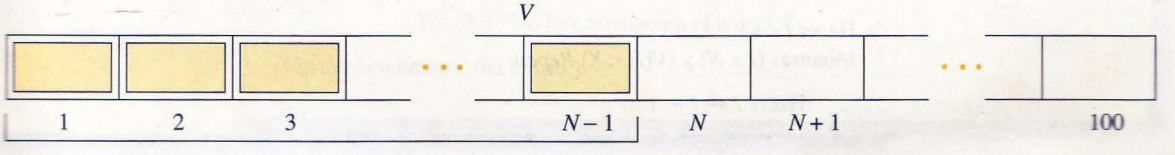
   si no

   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.



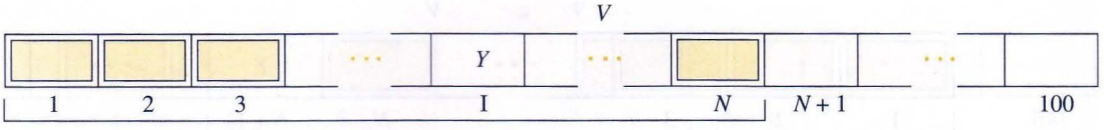


FIGURA 1.10c
Modificación en arreglos desordenados.

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 $I \leftarrow I + 1$

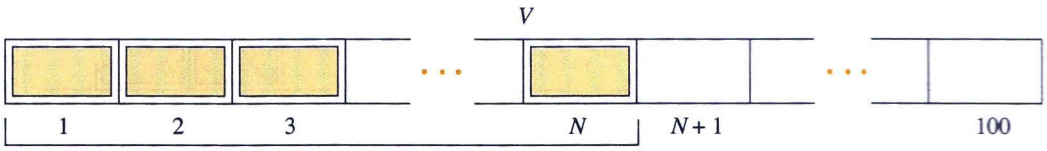


FIGURA 1.11
Actualización de arreglos ordenados.

```
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}
```

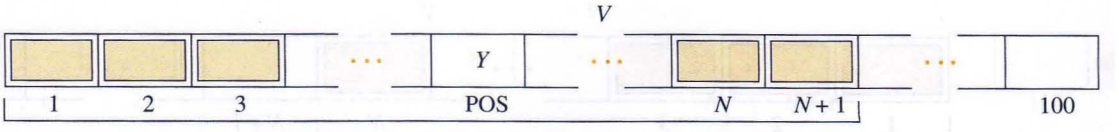


FIGURA 1.11*a*
Inserción en arreglos ordenados.

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-

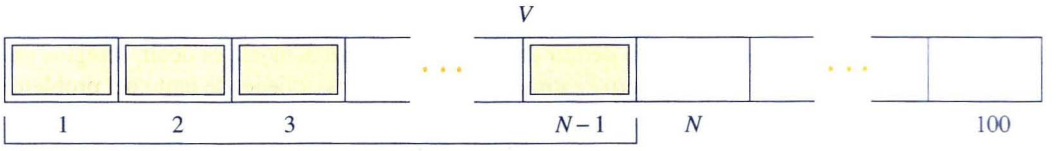


FIGURA 1.11*b*
Eliminación en arreglos ordenados.

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.

Sin embargo, es importante destacar que para la mayoría de los lenguajes de programación se pueden definir arreglos multidimensionales; es decir, arreglos con múltiples índices. El número de dimensiones —índices— depende tanto del problema que se quiera resolver como del lenguaje utilizado.

Se analizarán primero los arreglos bidimensionales, que representan un caso especial de los multidimensionales, por ser los más ampliamente utilizados.

1.3 ARREGLOS BIDIMENSIONALES

Para que el lector entienda mejor la estructura de los arreglos bidimensionales, se presenta el siguiente ejemplo.

Ejemplo 1.5

La tabla 1.1 contiene los costos de producción de cada departamento de una fábrica, correspondientes a los 12 meses del año anterior.

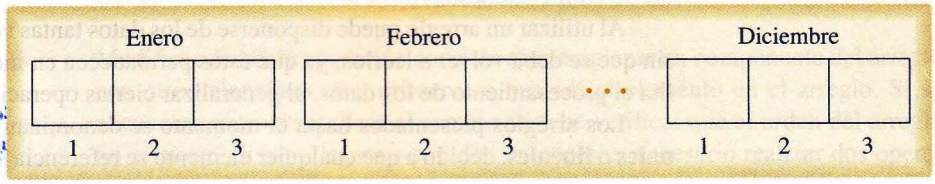
La tabla se interpreta de la siguiente manera: dado un mes, se conocen los costos de producción de cada uno de los departamentos de la fábrica; y dado un departamento, se conocen los costos de producción mensuales. Si se quisiera almacenar esta información utilizando los arreglos unidimensionales, se tendrían dos alternativas:

1. Definir 12 arreglos de tres elementos cada uno. En este caso, cada arreglo almacenará la información relativa a un mes.

TABLA 1.1
Costos mensuales por departamentos

Meses/Deptos.	Dulces	Conservas	Bebidas
Enero	100	300	120
Febrero	400	200	200
Marzo	350	250	210
Abril	280	300	200
Mayo	300	320	300
Junio	250	300	350
Julio	200	280	300
Agosto	180	300	400
Septiembre	500	400	450
Octubre	350	420	220
Noviembre	400	450	360
Diciembre	600	550	531

FIGURA 1.12
Almacenamiento de la información por mes.



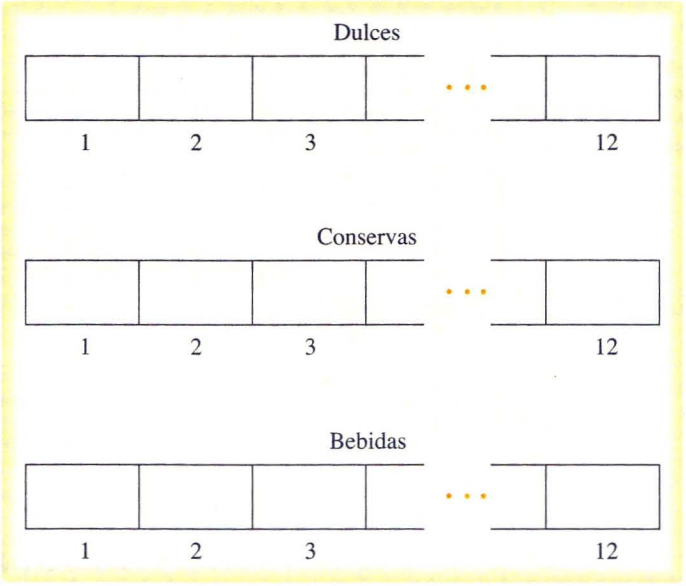


FIGURA 1.13
Almacenamiento de la información por departamento.

2. Definir tres arreglos de 12 elementos cada uno. De esta forma, cada arreglo almacenará la información relativa a un departamento a lo largo del año.

Sin embargo, no resulta muy práctico adoptar alguna de las dos alternativas. Se necesita una estructura que permita manejar los datos considerando los meses —renglones de la tabla—, y los departamentos —columnas de la tabla—; es decir, una estructura que trate a la información como un todo. La estructura que tiene esta característica se denomina **arreglo bidimensional**.

Un arreglo bidimensional es una colección **homogénea**, **finita** y **ordenada** de datos, en la que se hace referencia a cada componente del arreglo por medio de dos índices. El primero se utiliza para indicar el renglón, y el segundo para señalar la columna. Un arreglo bidimensional también se puede definir como un arreglo de arreglos. En la figura 1.14 se presenta un arreglo de tipo bidimensional.

El arreglo $A(M \times N)$ tiene M renglones y N columnas. Un elemento $A[I, J]$ se localiza en el renglón I, y en la columna J. Internamente en memoria se reservan $M \times N$ posiciones consecutivas para almacenar todos los elementos del arreglo.

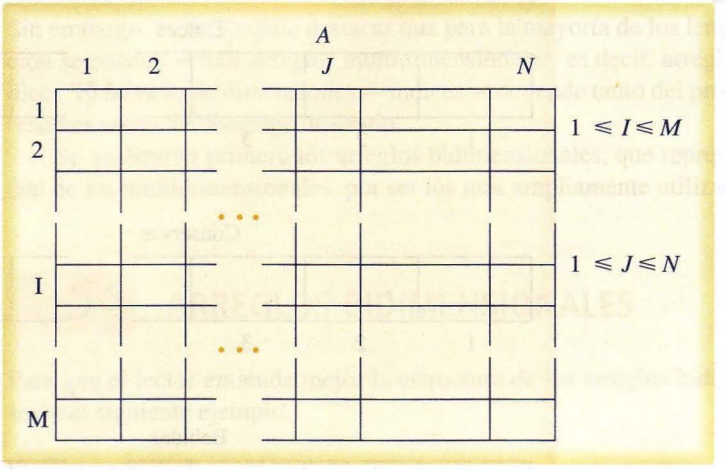
1.3.1 Declaración de arreglos bidimensionales

Los arreglos bidimensionales se declaran cuando se especifican el número de renglones y el número de columnas, junto con el tipo de dato de los componentes.

id_arreglo = ARREGLO [líminfr..límsupr,líminfc..límsupc] DE tipo

Con **líminfr** y **límsupr** se declara el tipo de dato del índice de los renglones y cuántos renglones tendrá el arreglo. Asimismo, con **líminfc** y **límsupc** se declara el tipo

FIGURA 1.14
Representación de un arreglo bidimensional.



de dato del índice de las columnas y cuántas columnas tendrá el arreglo. Con **tipo** se declara el tipo de datos de todos los componentes del arreglo.

El número total de componentes (NTC) de un arreglo bidimensional está determinado por la expresión:

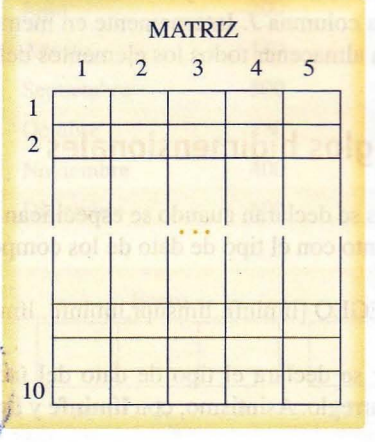
NTC = (límsupr − líminfr + 1) * (límsupc − líminfc + 1) **Fórmula 1.2**

Al igual que en el caso de los arreglos unidimensionales, los índices pueden ser cualquier tipo de dato ordinal (escalar, entero, carácter), mientras que los componentes pueden ser de cualquier tipo (reales, enteros, cadenas de caracteres, etc.). A continuación se analizan algunos ejemplos de arreglos bidimensionales.

Ejemplo 1.6

Sea MATRIZ un arreglo bidimensional de números reales con índices enteros. Su representación se muestra en la figura 1.15.

FIGURA 1.15



MATRIZ = ARREGLO[1..10,1..5] DE reales

- NTC = (10 −1 + 1)*(5 − 1 + 1) = 10 * 5 = 50
- Cada componente de MATRIZ será un número real. Para hacer referencia a cada uno de ellos se usarán dos índices y el nombre de la variable tipo arreglo: MATRIZ[*i*,*j*]

Donde: $1 \leq i \leq 10$
$1 \leq j \leq 5$

Ejemplo 1.7

Sea COSTOS un arreglo bidimensional de números reales con índices de tipo escalar. Su representación se muestra en la figura 1.16.

meses = (ene, feb, mar, abr, may, jun, jul, ago, set, oct, nov, dic)
departamentos = (dulces, conservas, bebidas)

COSTOS = ARREGLO[meses, departamentos] DE reales

- NTC = (ord(dic) − ord(ene) + 1) * (ord(bebidas) − ord(dulces) + 1)
  = (11 − 0 + 1) * (2 − 0 + 1) = 12 * 3 = 36
- Cada componente de COSTOS será un real. Para hacer referencia a cada uno de ellos usaremos dos índices y el nombre de la variable tipo arreglo COSTOS[*i*, *j*]

Donde: ene ≤ *i* ≤ dic
dulces ≤ *j* ≤ bebidas

Ejemplo 1.8

Sea MAT un arreglo bidimensional de cadenas de caracteres con índices para los renglones de tipo carácter y para las columnas de tipo entero. Su representación se muestra en la figura 1.17.

MAT = ARREGLO['*a*'..'*z*', − 5..5] DE cadena-de-caracteres

FIGURA 1.16

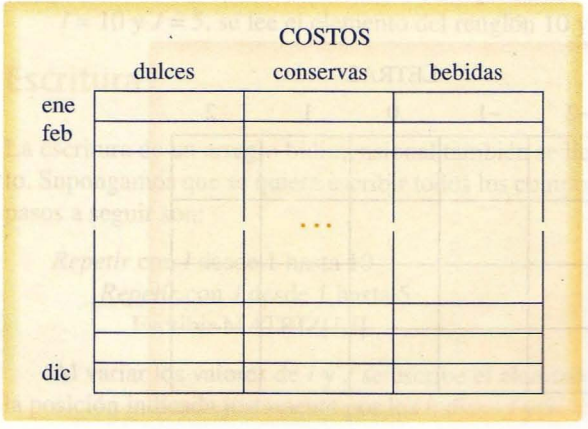
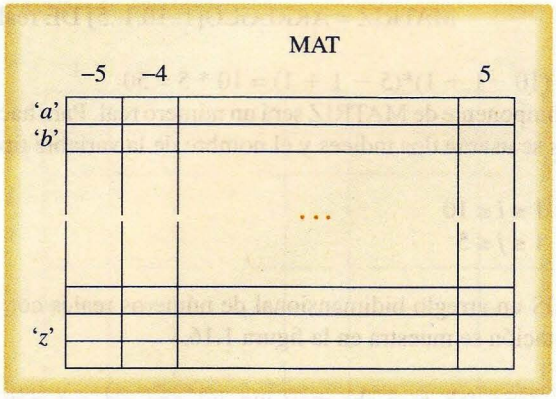


FIGURA 1.17



- NTC = (ord('z') − ord('a') + 1) * (5 − (−5) + 1)
  = (122 − 97 + 1) * (5 + 5 + 1) = 26 * 11 = 286

- Cada componente de MAT será un valor de tipo cadena de caracteres. Para hacer referencia a cada uno de ellos, se usarán dos índices y el nombre de la variable tipo arreglo: MAT[*i*,*j*]

  Donde: '*a*' ≤ *i* ≤ 'z'
  −5 ≤ *j* ≤ 5

Ejemplo 1.9

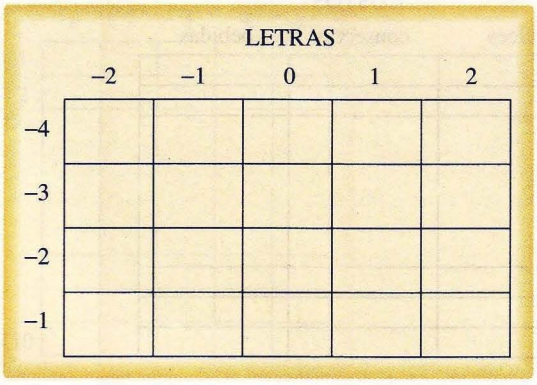
Sea LETRAS un arreglo bidimensional de caracteres con índices enteros. Su representación se muestra en la figura 1.18.

LETRAS = ARREGLO [−4..−1, −2..2] DE caracteres

- NTC = (−1 − (−4) + 1) * (2 − (−2) + 1) = 4 * 5 = 20
- Cada componente de LETRAS será un valor tipo carácter. Para hacer referencia a cada uno de ellos, se usarán dos índices y el nombre de la variable tipo arreglo: LETRAS[*i*, *j*]

  Donde: −4 ≤ *i* ≤ −1
  −2 ≤ *j* ≤ 2

FIGURA 1.18



1.3.2. Operaciones con arreglos bidimensionales

Las operaciones que se pueden realizar con arreglos bidimensionales son:

- Lectura/Escritura
- Asignación
- Actualización: Inserción
  Eliminación
  Modificación
- Ordenación
- Búsqueda

Los arreglos bidimensionales se consideran una generalización de los unidimensionales, por lo que se presentará una revisión rápida de algunas de las operaciones mencionadas. Para ilustrarlas se utilizarán los ejemplos anteriores.

Lectura

Cuando se presentó la operación de lectura en arreglos unidimensionales, se mencionó que con la ayuda de un ciclo se iban leyendo y asignando valores a cada uno de los componentes. Lo mismo sucede con los arreglos bidimensionales. Sin embargo, como sus elementos deben indicarse por medio de dos índices, normalmente se usan dos ciclos para lograr la lectura de elementos consecutivos.

Supongamos, por ejemplo, que se desea leer todos los elementos del arreglo bidimensional MATRIZ. Los pasos a seguir son:

```
Repetir con I desde 1 hasta 10
    Repetir con J desde 1 hasta 5
        Leer MATRIZ[I, J]
```

Al variar los índices de *I* y *J*, cada elemento de MATRIZ que se lee se asigna al lugar que le corresponde en el arreglo, según la posición de los índices *I* y *J*.

Para *I* = 1 y *J* = 1, se lee el elemento del renglón 1 y columna 1.
I = 1 y *J* = 2, se lee el elemento del renglón 1 y columna 2.
...
I = 10 y *J* = 5, se lee el elemento del renglón 10 y columna 5.

Escritura

La escritura de un arreglo bidimensional también se lleva a cabo elemento tras elemento. Supongamos que se quiera escribir todos los componentes del arreglo MATRIZ. Los pasos a seguir son:

```
Repetir con I desde 1 hasta 10
    Repetir con J desde 1 hasta 5
        Escribir MATRIZ[I,J]
```

Al variar los valores de *I* y *J* se escribe el elemento de MATRIZ correspondiente a la posición indicada justamente por los índices *I* y *J*.

Para $I = 1$ y $J = 1$, se escribe el elemento del renglón 1 y columna 1.
$I = 1$ y $J = 2$, se escribe el elemento del renglón 1 y columna 2.
...
$I = 10$ y $J = 5$, se escribe el elemento del renglón 10 y columna 5.

Asignación

La asignación de valores a un arreglo bidimensional se realiza de diferentes formas. La forma depende del número de componentes involucrados. Observemos a continuación dos alternativas diferentes.

1. Se asignan valores a todos los elementos del arreglo: en este caso se necesitarán dos ciclos para recorrer todo el arreglo.

```
Repetir con I desde 1 hasta 10
    Repetir con J desde 1 hasta 5
        MATRIZ[I,J] ← 0
```

Al variar los valores de I y J se asigna el 0 al elemento de MATRIZ correspondiente a la posición indicada por los índices I y J.

Para $I = 1$ y $J = 1$, se asigna el valor 0 al elemento del renglón 1 y columna 1.
$I = 1$ y $J = 2$, se asigna el valor 0 al elemento del renglón 1 y columna 2.
...
$I = 10$ y $J = 5$, se asigna el valor 0 al elemento del renglón 10 y columna 5.

En la figura 1.19 se presenta cómo queda el arreglo bidimensional cuando se asigna el valor 0 a cada una de las casillas.

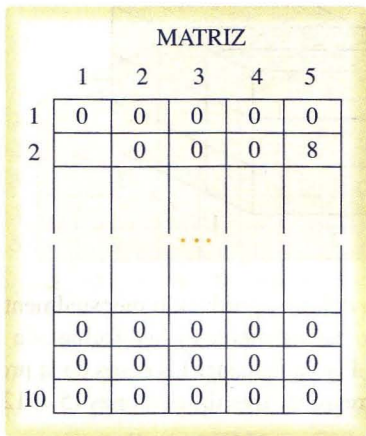
2. Se asigna un valor a un elemento en particular del arreglo: en este caso la asignación es directa y se debe indicar el renglón y la columna del componente involucrado. Por ejemplo, para asignar el valor 8 al elemento del renglón 2 y columna 5 se procede de la siguiente manera:

FIGURA 1.19
Asignación de arreglos.

MATRIZ

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
			...		
	0	0	0	0	0
	0	0	0	0	0
10	0	0	0	0	0

FIGURA 1.20
Asignación de arreglos.



MATRIZ[2,5] ← 8

El arreglo se muestra en la figura 1.20.

Es importante aclarar que las operaciones de lectura, escritura y asignación a todos los elementos de un arreglo bidimensional se pueden hacer tanto por renglones como por columnas.

1.4 ARREGLOS DE MÁS DE DOS DIMENSIONES

Un arreglo multidimensional —*N* dimensiones— se define como una colección finita, homogénea y ordenada de $K_1 \times K_2 \times ... \times K_N$ elementos. Para hacer referencia a cada componente de un arreglo de *N* dimensiones, se usarán *N* índices, uno para cada dimensión.

El arreglo *A* de *N* dimensiones se declara de la siguiente manera:

$$A = \text{ARREGLO}[LI_1..LS_1, LI_2..LS_2,...,LI_N..LS_N] \text{ DE tipo}$$

El total de componentes de *A* será:

$$NTC = (LS_1 - LI_1 + 1) * (LS_2 - LI_2 + 1) * ... * (LS_N - LI_N + 1) \qquad \textbf{Fórmula 1.3}$$

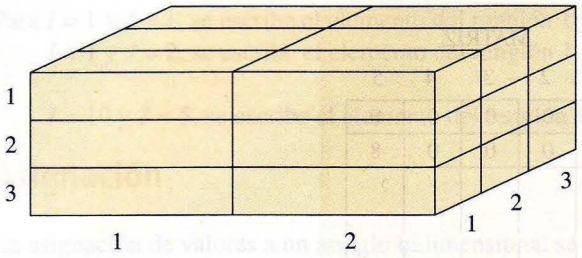
Por ejemplo, el arreglo tridimensional A[1...3, 1...2, 1...3] tendrá:

$$(3 - 1 + 1) * (2 - 1 + 1) * (3 - 1 + 1) = 3 * 2 * 3 = 18 \text{ elementos}$$

Gráficamente el arreglo *A* se puede representar como se muestra en las figuras 1.21 y 1.22:

A continuación se presenta un ejemplo de un arreglo tridimensional.

FIGURA 1.21
Representación de arreglos de más de dos dimensiones.



Ejemplo 1.10

Una empresa lleva un registro del total producido mensualmente por cada departamento. La empresa consta de cinco departamentos y la información se ha registrado a lo largo de los últimos cuatro años. Para almacenar los datos de la producción de la empresa, se requiere entonces de un arreglo de tres dimensiones ($5 \times 12 \times 4 = 240$ elementos), como el de la figura 1.23.

A = ARREGLO [1..5, 1..12, 1..4] DE reales

Supongamos que la empresa necesita obtener la siguiente información:

a) El total mensual de cada departamento durante el segundo año. Para obtener la información solicitada se deben realizar los siguientes pasos:

```
Repetir con I desde 1 hasta 5
   Repetir con J desde 1 hasta 12
      Escribir A[I,J,2]
```

Observe que para este caso se asigna la constante 2 al tercer índice —el de los años— y se hace variar a los otros dos índices. De esta manera se escribirán las producciones mensuales.

b) El total de la producción durante el primer año. Para obtener la información solicitada se deben realizar los siguientes pasos:

FIGURA 1.22
Representación de arreglos de más de dos dimensiones.

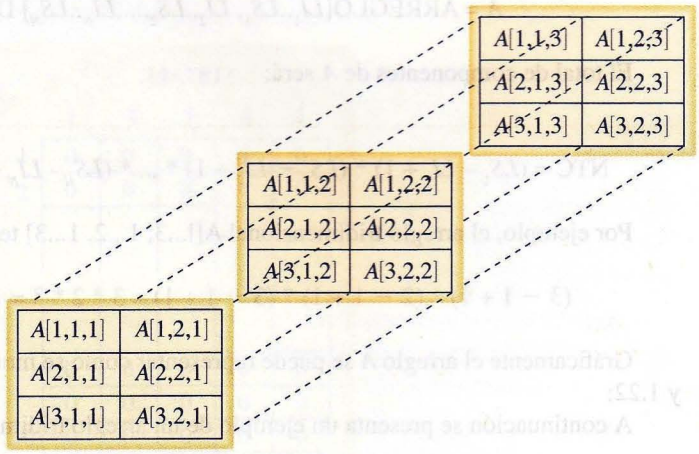
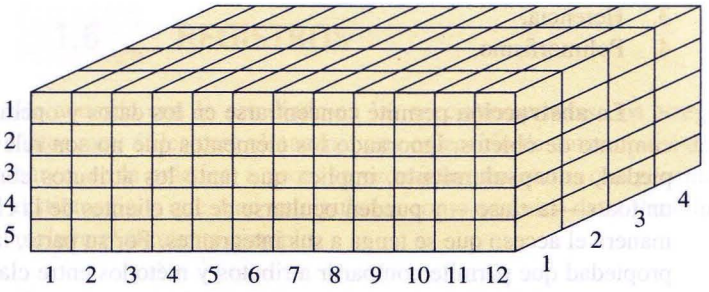


FIGURA 1.23




```
Hacer SUMA ← 0
Repetir con I desde 1 hasta 5
    Repetir con J desde 1 hasta 12
        Hacer SUMA ← SUMA + A[I,J,1]
Escribir SUMA
```

Observe que este caso es similar al anterior. La diferencia radica en que las cantidades mensuales no se escribirán, sino que se acumularán obteniendo el total anual.

c) El total de la producción del departamento 3 a lo largo del último año. Para obtener la información solicitada será necesario ejecutar los siguientes pasos:

```
Hacer SUMA ← 0
Repetir con J desde 1 hasta 12
    Hacer SUMA ← SUMA + A[3,J,4]
Escribir SUMA
```

Note que en este caso se tienen dos índices constantes, el de departamentos y el de años, y se hace variar solamente el índice de meses. Concluido el ciclo se escribirá el total producido por el departamento 3 durante el cuarto año.

1.5 LA CLASE *ARREGLO*

Para entender la clase arreglo, se requiere primero conocer algunos conceptos básicos relacionados con el paradigma de la programación orientada a objetos (POO).

Una **clase** define a un **objeto** por medio de la descripción de sus datos, conocidos como **atributos** y de su comportamiento, representado por métodos. Se dice que los atributos y los métodos son **miembros** de la clase.

Una clase puede representar a los alumnos de una escuela. En este caso los datos son los atributos que caracterizan a un alumno, por ejemplo, nombre, fecha de nacimiento, dirección, teléfono, etcétera, mientras que el comportamiento hace referencia a las operaciones que pueden realizarse sobre esos datos, por ejemplo, cambiar dirección o teléfono del alumno.

La programación orientada a objetos tiene cuatro propiedades:

1. Abstracción.
2. Encapsulamiento u ocultamiento de la información.

3. Herencia.
4. Polimorfismo.

La **abstracción** permite concentrarse en los datos y operaciones que definen a un conjunto de objetos, ignorando los elementos que no son relevantes. La segunda propiedad, **encapsulamiento**, implica que tanto los atributos como los métodos forman un todo —la clase— y pueden ocultarse de los clientes de la clase, al controlar de esta manera el acceso que se tenga a sus integrantes. Por su parte, la **herencia** representa la propiedad que permite compartir atributos y métodos entre clases. Por último, el **polimorfismo** ofrece la facilidad de que ciertos métodos puedan adoptar distintas formas.

La **clase *Arreglo*** tendrá atributos y métodos. Los atributos constituirán la colección de elementos y el tamaño. Los métodos serán todas las operaciones analizadas en las secciones previas: lectura, inserción, eliminación, etcétera. Gráficamente la clase *Arreglo* puede verse como se muestra en la figura 1.24.

Un **objeto** es una instancia de una clase. Es decir, esta última representa a un conjunto de objetos, a un concepto general, por ejemplo, los alumnos de una escuela o los arreglos, mientras que los primeros son ocurrencias de la clase. Considerando la clase *Arreglo*, un ejemplo de objeto será el arreglo de calificaciones de un grupo de alumnos.

En los lenguajes de programación orientada a objetos más conocidos se usa la notación de puntos para tener acceso a los miembros no privados de un objeto.

<objeto>.<miembro>

Dentro de un método de una clase, la referencia a cualquiera de sus otros miembros no requiere el uso de esta notación. Asumiendo que la variable CALIF es un objeto de la clase *Arreglo*, se pueden tener las siguientes instrucciones:

```
CALIF.Tamaño = CALIF.Tamaño −2
CALIF.Datos[6] = 10
```

Para el caso de que las instrucciones fueran parte de un método, se puede omitir el nombre del objeto y el punto, y usar directamente el atributo.

```
Datos[6] = 10
```

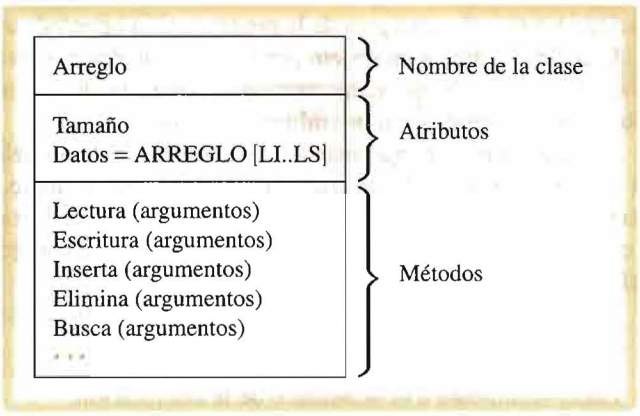


FIGURA 1.24
Clase Arreglo.

1.6 REGISTROS

De acuerdo con lo estudiado en las secciones previas, los arreglos son estructuras de datos muy útiles para almacenar una colección de datos, todos del mismo tipo. Sin embargo, en la práctica, a veces se necesitan estructuras que permitan almacenar datos de distintos tipos que sean manipulados como un único dato. Para ilustrar este problema se incluye el siguiente ejemplo.

Ejemplo 1.11

Una compañía tiene por cada empleado la siguiente información:

- Nombre (cadena de caracteres)
- Dirección (cadena de caracteres)
- Edad (entero)
- Sexo (carácter)
- Antigüedad (entero)

Si se quisiera almacenar estos datos no sería posible usar un arreglo, ya que sus componentes deben ser todos del mismo tipo. La estructura que puede guardar esta información de manera efectiva se conoce como **registro** o **estructura**.

Un registro se define como una colección finita y heterogénea de elementos. También representa un tipo de dato estructurado, en el que cada uno de sus componentes se denomina **campo**. Los campos de un registro pueden ser todos de diferentes tipos de datos. Por tanto, también podrán ser registros o arreglos. Cada campo se identifica con un nombre único, el identificador de campo. Otra diferencia importante con los arreglos es que no es necesario establecer un orden entre los campos.

1.6.1 Declaración de registros

Como no es la intención de los autores seguir la sintaxis de algún lenguaje de programación en particular, un registro se declara de la siguiente forma:

ident_registro = REGISTRO
    id_campo_1: $tipo_1$
    id_campo_2: $tipo_2$
    ...
    id_campo_n: $tipo_n$
{Fin de la declaración del registro 1}

Donde: ident_registro es el nombre del dato tipo registro
    id_campo_i es el nombre del campo i
    $id\_campo_i \neq id\_campo_j \; \forall_{i,j} = {}_{1,\ldots,n} e_{i \neq j}$
    $tipo_i$ es el tipo del campo i

Los que siguen son ejemplos de declaraciones de registros, con su correspondiente representación gráfica.

Ejemplo 1.12

Sea FECHA un registro formado por tres campos numéricos. Su representación se muestra en la figura 1.25.

```
FECHA = REGISTRO
    día:  1..31
    mes: 1..12
    año:  0..2100
{Fin de la declaración del registro FECHA}
```

Ejemplo 1.13

Sea DOMICILIO un registro formado por cuatro campos, uno de ellos es numérico y los tres restantes del tipo cadena de caracteres. Su representación se muestra en la figura 1.26.

```
DOMICILIO = REGISTRO
    calle: cadena_de_caracteres
    número: entero
    ciudad: cadena_de_caracteres
    país: cadena_de_caracteres
{Fin de la declaración del registro DOMICILIO}
```

Ejemplo 1.14

Sea CLIENTE un registro formado por cuatro campos, dos del tipo cadena de caracteres, uno del tipo real y el otro del tipo booleano. Su representación se muestra en la figura 1.27.

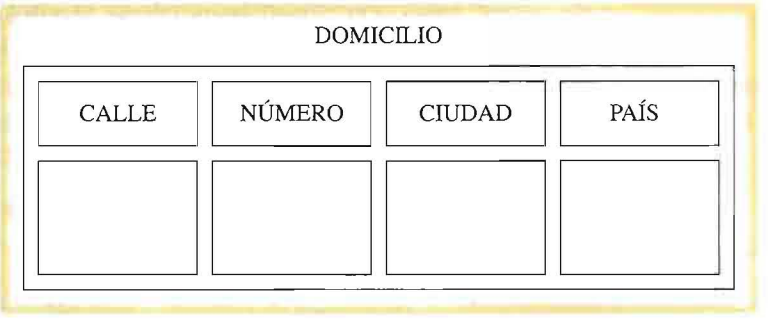
```
CLIENTE = REGISTRO
    nombre: cadena_de_caracteres
    teléfono: cadena_de_caracteres
    saldo: real
    moroso: booleano
{Fin de la declaración del registro CLIENTE}
```

1.6.2 Acceso a los campos de un registro

Como un registro es un tipo de dato estructurado, no se puede tener acceso a él directamente como un único dato, sino que se debe especificar el elemento —campo— del

FIGURA 1.25

FIGURA 1.26



registro que nos interesa. Para ello, en la mayoría de los lenguajes se sigue la siguiente sintaxis:

variable_registro.id_campo

Donde: variable_registro es una variable de tipo registro
id_campo es el identificador del campo deseado

Es decir, se usarán dos identificadores para hacer referencia a un elemento: el nombre de la variable tipo registro y el nombre del campo, separados entre sí por un punto.

De acuerdo con los ejemplos de registros 1.12, 1.13 y 1.14, se presentan a continuación diferentes casos que ilustran el acceso a los campos de un registro.

a) Para leer los tres campos de una variable *F* de tipo FECHA:
Leer *F.día,F.mes,F.año*

b) Para escribir los cuatro campos de una variable *D* de tipo DOMICILIO:
Escribir *D.calle,D.número,D.ciudad,D.país*

c) Para asignar valores a algunos de los campos de una variable *C* de tipo CLIENTE:

C.saldo ← C.saldo + cant
C.moroso ← VERDADERO
C.nombre ← "Juan Pérez"

FIGURA 1.27

En general, como se mencionó anteriormente, el orden en el que se manejan los campos no es importante. Es decir, se podrían haber leído los campos de la variable *F* de la siguiente manera:

Leer *F.año,F.día,F.mes*

Sólo se debe tener en cuenta que los datos proporcionados por el usuario o asignados en un algoritmo se correspondan en tipo con los campos.

1.6.3 Diferencias entre registros y arreglos

Las dos diferencias sustanciales existentes entre registros y arreglos son:

1. Un arreglo puede almacenar *N* elementos del mismo tipo —estructura de datos homogénea—, mientras que un registro puede almacenar *N* elementos de diferentes tipos de datos —estructura de datos heterogénea—.
2. A los componentes de un arreglo se tiene acceso por medio de índices que indican la posición del elemento correspondiente en el arreglo, mientras que a los componentes de un registro, los campos, se tiene acceso por medio de su nombre, que es único.

1.6.4 Combinaciones entre arreglos y registros

Los registros tienen varios campos. Cada uno de ellos puede ser de cualquier tipo de datos, simples o estructurados. Sin embargo, los componentes del nivel más bajo de un tipo estructurado siempre deben ser tipos simples de datos.

De acuerdo con esta condición, se infiere que un campo de un registro puede ser otro registro o bien un arreglo. Por otra parte, los componentes de un arreglo también pueden ser registros. Estos casos enunciados, además, se pueden presentar en forma anidada.

Arreglos de registros

En este caso, cada elemento del arreglo es un registro. Todos los componentes del arreglo tienen que ser del mismo tipo de registro, ya que es una estructura de datos homogénea. A continuación presentamos un ejemplo.

Ejemplo 1.15

Una empresa registra para cada uno de sus clientes los siguientes datos:

- Nombre (cadena de caracteres)
- Teléfono (cadena de caracteres)
- Saldo (real)
- Moroso (booleano)

Si la empresa tiene N clientes necesitará un arreglo de N elementos, en el cual cada uno de sus componentes es un registro como el descrito en el ejemplo 1.14. La figura 1.28 muestra la estructura de datos correcta para resolver este problema:

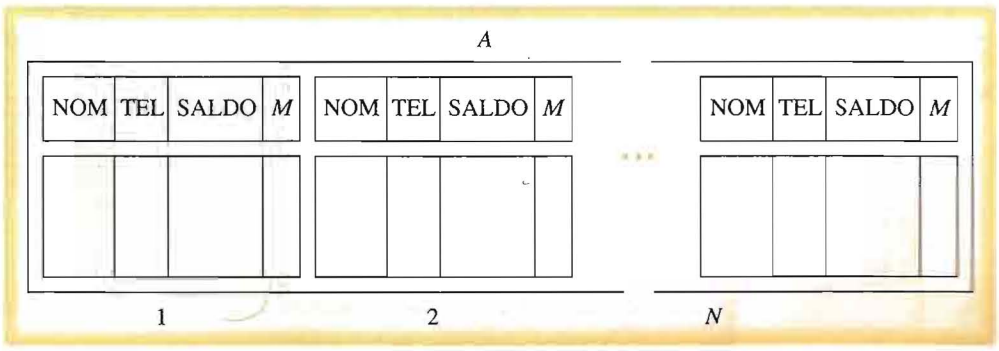
A = ARREGLO [1..100] DE CLIENTE

Cada elemento de A será un dato tipo CLIENTE. Por tanto, si se quiere, por ejemplo, leer el arreglo A, debe leerse por cada componente cada uno de los campos que forman al registro.

```
Repetir con I desde 1 hasta N
   Leer A[I].nombre
   Leer A[I].teléfono
   Leer A[I].saldo
   Leer A[I].moroso
```

Con $A[I]$ se hace referencia al elemento I del arreglo A, que es un registro; con .id_campo se especifica cuál de los campos del registro se leerá. De forma similar se procede para escritura, asignación, etcétera.

FIGURA 1.28
Arreglo de registros.



Registros anidados

En los registros anidados, al menos un campo del registro es del tipo registro. Observemos a continuación el siguiente ejemplo.

Ejemplo 1.16

Una empresa registra para cada uno de sus acreedores los siguientes datos:

- Nombre (cadena de caracteres)
- Dirección:
  - Calle (cadena de caracteres)
  - Número (entero)
  - Ciudad (cadena de caracteres)
  - País (cadena de caracteres)
- Saldo (real)

Para definir el tipo de dato del campo *dirección*, es necesario declarar previamente un registro formado por los cuatro componentes: calle, número, ciudad y país que se especifican. Se usará el registro del ejemplo 1.13, presentado anteriormente, para resolver este caso.

```
ACREEDOR = REGISTRO
    nombre: cadena_de_caracteres
    dirección: DOMICILIO
    saldo: real
{Fin de la declaración del registro ACREEDOR}
```

La figura 1.29 muestra la estructura de datos requerida.

En este caso, el registro tiene un campo —dirección— que es del tipo de datos DOMICILIO, el cual es un registro de cuatro campos. Para tener acceso a los campos que, a su vez, son registros, en la mayoría de los lenguajes se sigue la siguiente sintaxis:

variable_registro.id_campo$_1$.id_campo$_n$

Donde: variable_registro es una variable de tipo registro
id_campo$_1$ es el identificador de un campo del registro: el campo es de tipo registro
id_campo$_n$ representa el identificador de un campo

FIGURA 1.29
Registros anidados.

Para tener acceso a los campos de la variable *AC* de tipo ACREEDOR, la secuencia a seguir es la siguiente:

AC.nombre
AC.dirección.calle
AC.dirección.número
AC.dirección.ciudad
AC.dirección.país
AC.saldo

Registros con arreglos

Los registros con arreglos tienen, por lo menos, un campo que es de tipo arreglo. Analice cuidadosamente el siguiente ejemplo.

Ejemplo 1.17

Una empresa registra para cada uno de sus clientes los siguientes datos:

- Nombre (cadena de caracteres)
- Teléfono (cadena de caracteres)
- Saldo mensual del último año (arreglo de reales)
- Moroso (booleano)

La definición del registro correspondiente es:

```
CLIENTE = REGISTRO
    nombre: cadena_de_caracteres
    teléfono: cadena_de_caracteres
    saldos: ARREGLO [1..12] DE reales
    moroso: booleano
{Fin de la declaración del registro CLIENTE}
```

La figura 1.30 muestra la estructura requerida.

Para este caso el registro tiene un campo, saldos, que es un arreglo unidimensional de 12 elementos reales. Con el propósito de hacer referencia a ese campo, se procede de la siguiente manera:

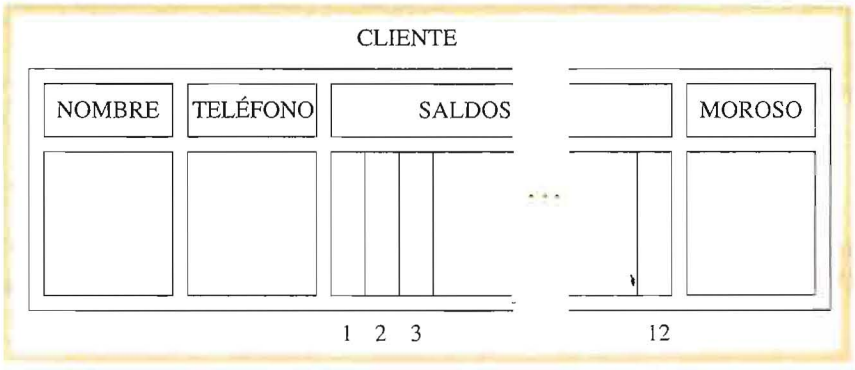


FIGURA 1.30
Registros con arreglos.

variable_registro.id_campo[índice]

Para tener acceso a los campos de la variable *CLI* de tipo CLIENTE se debe seguir la secuencia:

CLI.nombre
CLI.teléfono
Repetir con I desde 1 hasta 12
    CLI.saldos[*I*]
CLI.moroso

Las tres posibles combinaciones analizadas aquí: arreglos de registros, registros anidados y registros con arreglos, pueden presentarse de manera simultánea y en diferentes niveles en una misma estructura de datos. En estos casos, se recomienda que la estructura resultante sea comprensible y que no se complique demasiado el acceso a los datos individuales.

1.6.5 Arreglos paralelos

Por **arreglos paralelos** se entiende dos o más arreglos cuyos elementos se corresponden. Es decir, los componentes que ocupan una misma posición en diferentes arreglos tienen una estrecha relación semántica. Para ilustrar esta idea, a continuación se presentará un caso práctico y su solución, mediante arreglos paralelos.

Supongamos que se conoce el nombre del alumno y la calificación obtenida por éste en un examen que fue aplicado a un grupo de 30 alumnos. Si se quisiera usar estos datos para generar información, por ejemplo, promedio del grupo, calificación más alta, nombre de los alumnos con calificación inferior al promedio, etc., se tendrían dos alternativas principales en el diseño de la solución.

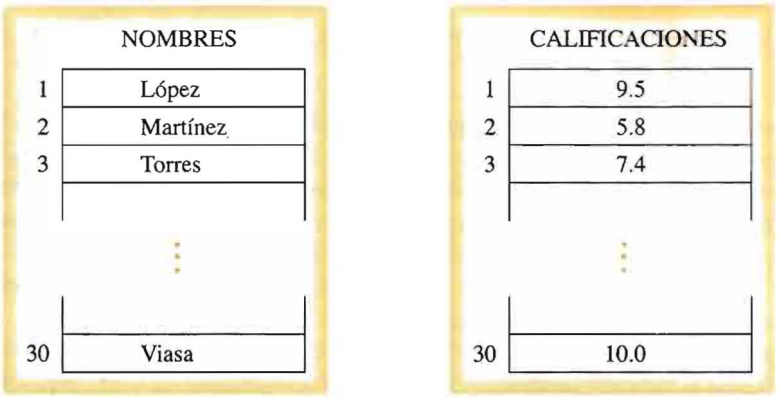
Uso de arreglos paralelos

Si se utilizan arreglos paralelos para resolver este problema, se requiere de dos arreglos unidimensionales; en uno se almacenará el nombre de los alumnos, y en otro la calificación obtenida por éste en el examen. Es decir, a cada elemento del arreglo NOMBRES le corresponderá entonces uno del arreglo CALIFICACIÓN. Así, si se quiere hacer referencia a la calificación de NOMBRES[*I*], se utilizará CALIFICACIÓN[*I*]. Observe la figura 1.31.

López	obtuvo una calificación de 9.5
Martínez	obtuvo una calificación de 5.8
Torres	obtuvo una calificación de 7.4
...	...
Viasa	obtuvo una calificación de 10.0

A continuación se incluye un algoritmo que calcula el promedio del grupo e imprime el nombre de los alumnos que tengan calificación menor al promedio.

FIGURA 1.31
Arreglos paralelos.



Algoritmo 1.11 Arreglos_paralelos

Arreglos_paralelos

{Este algoritmo calcula el promedio del grupo e imprime el nombre de los alumnos con calificación menor al promedio}
{NOMBRE y CALIFICACIÓN son variables de tipo arreglo. *I* es una variable de tipo entero. PROM y *AC* son variables de tipo real}

1. Hacer $AC \leftarrow 0$
2. *Repetir* con *I* desde 1 hasta 30
   Leer NOMBRE[*I*] y CALIFICACIÓN[*I*]
   Hacer $AC \leftarrow AC$ + CALIFICACIÓN[*I*]
3. {Fin del ciclo del paso 2}
   {Se calcula el promedio del grupo}
4. Hacer PROM $\leftarrow AC/30$
5. Escribir "El promedio del grupo es": PROM

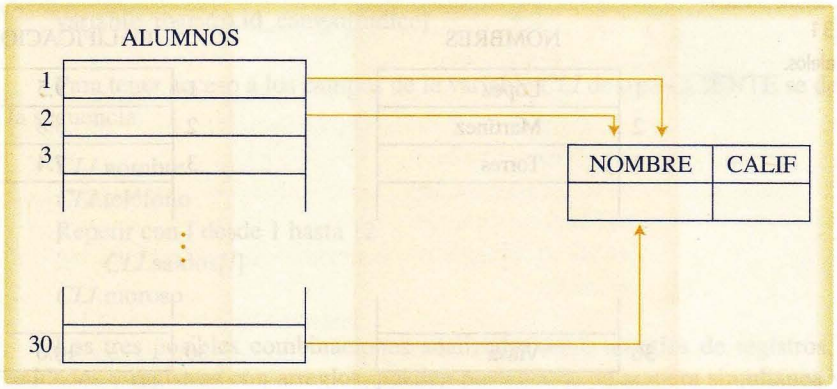
   {Búsqueda e impresión de los nombres de los alumnos con calificación inferior al promedio}

6. *Repetir* con *I* desde 1 hasta 30
   6.1 *Si* (CALIFICACIÓN[*I*] < PROM) *entonces*
   Escribir NOMBRE[*I*]
   6.2 {Fin del condicional del paso 6.1}
7. {Fin del ciclo del paso 6}

Uso de arreglos de registros

Otra solución al problema sería utilizar un arreglo de registros. En este caso, cada componente del arreglo ALUMNO es un registro que tiene dos campos: NOMBRE y CALIF. Observe la figura 1.32.

FIGURA 1.32



Así: ALUMNOS[*I*].NOMBRE hará referencia al nombre del alumno *I*
ALUMNOS[*I*].CALIF hará referencia a la calificación obtenida por el alumno *I*

El siguiente algoritmo presenta la solución al problema anterior mediante un arreglo de registros.

Algoritmo 1.12 Arreglo_de_registros

Arreglo_de_registros

{Este algoritmo calcula el promedio del grupo e imprime el nombre de los alumnos con calificación menor al promedio}
{ALUMNOS es un arreglo de registros. *I* es una variable de tipo entero. *AC* y PROM son variables de tipo real}

1. Hacer *AC* ← 0
2. *Repetir* con *I* desde 1 hasta 30
   Leer ALUMNOS[*I*].NOMBRE y ALUMNOS[*I*].CALIF
   Hacer *AC* ← *AC* + ALUMNOS[*I*].CALIF
3. {Fin del ciclo del paso 2}
4. Hacer PROM ← *AC*/30
5. Escribir "El promedio del grupo es": PROM

   {Búsqueda e impresión de los alumnos con calificación inferior al promedio}

6. *Repetir* con *I* desde 1 hasta 30
   6.1 *Si* (ALUMNOS[*I*].CALIF < PROM) *entonces*
   Escribir ALUMNOS[*I*].NOMBRE
   6.2 {Fin del condicional del paso 6.1}
7. {Fin del ciclo del paso 6}

1.7 REGISTROS Y CLASES

Los registros son las estructuras de datos que más se parecen al concepto de clase presentado. En la sección anterior se dijo que un registro almacena las principales características de un conjunto de objetos. Cada una de esas características constituye un campo del registro. Al establecer la relación con las clases, los campos representan los atributos. Por tanto, sólo se agregan los métodos —operaciones que pueden aplicarse sobre los campos— para completar la definición de una clase.

La **clase *Registro*** como tal no se declara, porque lo que se requiere es una clase por cada registro. Es decir, si se desea representar a los clientes de una empresa, según el ejemplo visto en la sección anterior, desde el punto de vista de la programación orientada a objetos, se deberá definir una clase que contendrá tanto los atributos —lo que en registros se llaman campos— como todas las operaciones válidas para un cliente, por ejemplo, actualizar el saldo, cambiar el número telefónico, etcétera. Gráficamente la clase ***Cliente*** puede verse como se muestra en la figura 1.33.

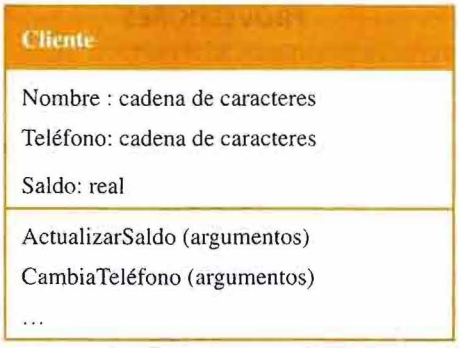
Un objeto de la clase ***Cliente*** es una instancia de la misma. Es decir, está representando a un cliente con un nombre, un número telefónico y un saldo específico.

La **notación de puntos** utilizada en los registros —<variable_registro>.<campo>— es similar a la usada en los lenguajes orientados a objetos para tener acceso a los miembros no privados de un objeto —<objeto>.<miembro>—. Al asumir que la variable CLI es un objeto de la clase *Cliente* previamente definida, se puede tener acceso a los miembros no privados de dicho objeto por medio de las instrucciones:

a) CLI.ActualizarSaldo (NuevoSaldo)
En este ejemplo se está invocando al método que actualiza el saldo del cliente. El método tiene un argumento que indica el nuevo valor que se asignará al atributo Saldo.

b) CLI.CambiaTeléfono (NuevoTel)
En este ejemplo se está invocando al método que actualiza el número telefónico del cliente. El método tiene un argumento que indica el nuevo valor que se asignará al atributo *Teléfono*.

FIGURA 1.33
Clase Cliente.



EJERCICIOS

Arreglos de una dimensión y arreglos paralelos

1. En un arreglo unidimensional se ha almacenado el número total de toneladas de cereales cosechadas durante cada mes del año anterior. Escriba un programa que obtenga e imprima la siguiente información:

a) El promedio anual de toneladas cosechadas.
b) ¿Cuántos meses tuvieron cosecha superior al promedio anual?
c) ¿Cuántos meses tuvieron cosecha inferior al promedio anual?

2. En un arreglo unidimensional se almacenan las calificaciones finales de N alumnos de un curso universitario. Escriba un programa que calcule e imprima:

a) El promedio general del grupo.
b) Número de alumnos aprobados y reprobados.
c) Porcentaje de alumnos aprobados y reprobados.
d) Número de alumnos cuya calificación fue mayor o igual a 8.

3. Dada una cadena de caracteres como dato, se desea saber el número de veces que aparecen las letras '*a*', '*b*',...,'*z*' y '*A*', '*B*',...,'*Z*' en dicha cadena. Escriba un programa que resuelva el problema.

a) Si usó arreglos, ¿cuántos necesitó? ¿Por qué?
b) ¿Existe otra forma de resolverlo?

4. Dado un arreglo unidimensional de números enteros, ordenados crecientemente, escriba un programa que elimine todos los elementos repetidos. Considere que de haber valores repetidos, éstos se encontrarán en posiciones consecutivas del arreglo.

5. Una compañía almacena la información relacionada con sus proveedores en los siguientes arreglos:

PROVEEDORES

p_1	p_2	p_3	...	p_n

Cada p_i es el nombre del proveedor i. Este arreglo está ordenado alfabéticamente.

CIUDAD

c_1	c_2	c_3	...	c_n

Cada c_i representa el nombre de la ciudad en la que reside el proveedor i.

NÚMERO DE ARTÍCULOS

a_1	a_2	a_3	...	a_n

Cada a_i es el número de artículos diferentes que provee el proveedor i.

Escriba un programa que pueda llevar a cabo las siguientes transacciones:

a) Dado el nombre de un proveedor, informar el nombre de la ciudad en la que reside y el número de artículos que provee.
b) Actualizar el nombre de la ciudad, en caso de que un proveedor cambie de domicilio. Los datos serán el nombre del proveedor y el nombre de la ciudad a la cual se mudó.
c) Actualizar el número de artículos, manejados por un proveedor para el caso de que éste aumente o disminuya. Los datos serán el nombre del proveedor y la cantidad en la que aumenta (+) o disminuye (−) el total de artículos que provee.
d) La compañía incorpora a un nuevo proveedor. Actualizar los arreglos sin alterar el orden de PROVEEDORES. Los datos serán el nombre del proveedor, el nombre de la ciudad y el total de artículos que provee.
e) La compañía da de baja a un proveedor. Actualizar los arreglos. El dato será el nombre del proveedor.

6. Una inmobiliaria tiene información sobre departamentos en renta almacenada en dos arreglos:

EXTENSIÓN

e_1	e_2	e_3	...	e_n

El arreglo EXTENSIÓN almacena la superficie, en metros cuadrados, de cada uno de los N departamentos.

PRECIO

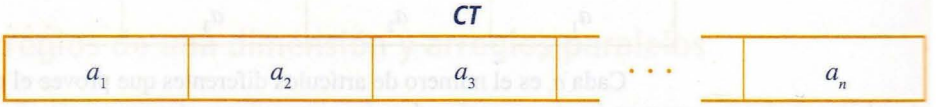
P_1	P_2	P_3	...	P_N

El arreglo PRECIO almacena los precios de alquiler de los N departamentos. Este arreglo está ordenado de manera creciente. Considere que no existen departamentos con igual superficie y distintos precios.

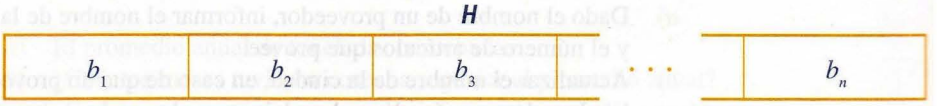
Escriba un programa que pueda llevar a cabo las siguientes operaciones:

a) Llega un cliente a la inmobiliaria y solicita rentar un departamento. Si existe alguno con superficie mayor o igual a la buscada y precio menor o igual al buscado, se dará de baja al departamento seleccionado.
b) Se vence un contrato y el cliente no desea renovarlo. Se deben actualizar los arreglos.

7. Se tiene la siguiente información:

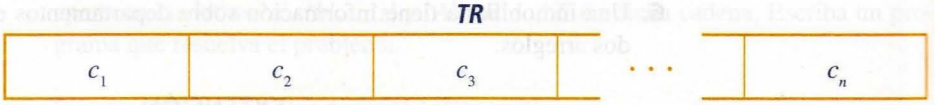


En el arreglo *CT* se almacenan los nombres de *N* centros turísticos del país.



En el arreglo *H* se almacena el número de habitaciones de cada tipo, sencilla o doble, de cada centro turístico.

Por ejemplo:
H[1] guarda el número de habitaciones sencillas del centro 1.
H[2] guarda el número de habitaciones dobles del centro 1.
H[3] guarda el número de habitaciones sencillas del centro 2.
H[4] guarda el número de habitaciones dobles del centro 2.
etcétera.



En el arreglo *TR* se almacena el número total de restaurantes por centro turístico.
Deberá desarrollar un programa que proporcione la siguiente información:

a) El nombre del centro turístico que cuenta con más restaurantes.
b) El nombre del centro turístico que cuenta con más habitaciones, teniendo en cuenta las sencillas y las dobles.
c) Dado el nombre de un centro turístico como dato, informar cuántas habitaciones tiene: sencillas, dobles y el total.
d) El nombre del centro turístico que más restaurantes tiene en relación con el número de habitaciones.

8. Se tienen tres arreglos: SUR, CENTRO y NORTE que almacenan los nombres de los países de Sur, Centro y Norteamérica, respectivamente. Los tres arreglos están ordenados alfabéticamente.

Escriba un programa que mezcle los tres arreglos anteriores, formando un cuarto arreglo, AMÉRICA, en el cual aparezcan los nombres de todos los países del continente ordenados alfabéticamente.

9. Se tienen dos arreglos: CINES y TEATROS. El primero almacena los nombres de todos los cines de la ciudad. Está ordenado alfabéticamente de manera ascendente:

$$CINES[1] \leq CINES[2] \leq ... \leq CINES[N]$$

El segundo arreglo guarda los nombres de todos los teatros de la ciudad. Está ordenado alfabéticamente de manera descendente:

$$TEATROS[1] \geq TEATROS[2] \geq ... \geq TEATROS[K]$$

Escriba un programa que mezcle estos arreglos formando un tercero, ENTRETENIMIENTOS, que quede ordenado alfabéticamente de manera ascendente.

10. Se tienen registradas las calificaciones obtenidas en un examen a 50 alumnos. Los datos son cal_1, cal_2, ..., cal_{50}, donde cal_i es un número entero comprendido entre los valores 0 y 10 ($0 \leq cal_i \leq 10$).

Escriba un programa que calcule e imprima la frecuencia de cada uno de los posibles valores.

La salida del programa se muestra a continuación:

Calificación	Frecuencia
0	1 ALUMNO
1	—
2	—
3	4 ALUMNOS
4	2 ALUMNOS
...	...
10	3 ALUMNOS

11. Escriba sus propios algoritmos para insertar, eliminar o modificar un elemento de un arreglo:

a) Si el arreglo está desordenado.
b) Si el arreglo está ordenado.

12. Dado un arreglo unidimensional de tipo entero que contiene calificaciones de exámenes de alumnos, construya un programa que calcule lo siguiente:

a) *Media aritmética*. Se calcula como la suma de los elementos entre el número de elementos.
b) *Varianza*. Se calcula como la suma de los cuadrados de las desviaciones de la media, entre el número de elementos.
c) *Desviación estándar*. Se calcula como la raíz cuadrada de la varianza.
d) *Moda*. Se calcula al obtener el número con mayor frecuencia.

13. Escriba un programa que almacene en un arreglo unidimensional los primeros 30 números perfectos. Un número se considera perfecto, si la suma de los divisores excepto el mismo es igual al propio número. El 6, por ejemplo, es un número perfecto.

Arreglos multidimensionales

14. Sean los arreglos bidimensionales $A(M \times N)$ y $B(M \times N)$
Donde: $1 \leq M \leq 10$,
$1 \leq N \leq 20$,
a_{ij} y b_{ij} son reales.
Escriba un programa que calcule $C(M \times N) = A(M \times N) + B(M \times N)$.

15. Sean los arreglos bidimensionales $A(M \times N)$ y $B(N \times P)$
Donde: $1 \leq M \leq 10$,
$1 \leq N \leq 10$,
$1 \leq P \leq 5$,
a_{ij} y b_{ij} son reales.
Escriba un programa que calcule $C(M \times P) = A(M \times N) * B(N \times P)$

16. Escriba un programa que llene de ceros una matriz $A(N \times N)$ excepto en la diagonal principal donde debe asignar 1. Si $N = 4$, la matriz debe quedar:

	1	2	3	4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

17. Escriba un programa que intercambie por renglón los elementos de un arreglo bidimensional. Los elementos del renglón 1 deben intercambiarse con los del renglón N, los del renglón 2 con los del $N - 1$, y así sucesivamente.
Por ejemplo, si A es:

	1	2	3	4
1	1	4	5	−5
2	0	87	1	0
3	2	4	10	3
4	9	5	7	5

El resultado de la operación debe ser:

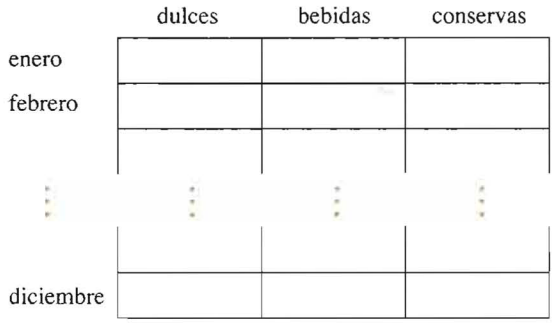
	1	2	3	4
1	9	5	7	5
2	2	4	10	3
3	0	87	1	0
4	1	4	5	−5

18. Dado como dato el arreglo bidimensional $A(M \times N)$, que almacena números reales.

Donde: $1 \leq M \leq 20$,
$1 \leq N \leq 20$,

Escriba un programa que encuentre e imprima el valor más grande almacenado en cada una de las columnas y en cada uno de los renglones del arreglo. Su programa debe imprimir, junto al valor encontrado, la columna o renglón en la cual se encontró.

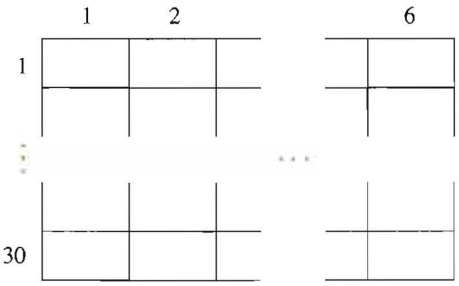
19. Se tienen los costos de producción de tres departamentos: dulces, bebidas y conservas, correspondientes a los 12 meses del año anterior.



Escriba un programa que pueda proporcionar la siguiente información:

a) ¿En qué mes se registró el mayor costo de producción de dulces?
b) Promedio anual de los costos de producción de bebidas.
c) ¿En qué mes se registró el mayor costo de producción en bebidas, y en qué mes el menor costo?
d) ¿Cuál fue el rubro que tuvo el menor costo de producción en diciembre?

20. Se tiene una tabla con las calificaciones obtenidas por 30 alumnos en seis exámenes diferentes:



Escriba un programa que calcule:

a) El promedio general de calificaciones de los 30 alumnos, considerando los seis exámenes.
b) El alumno que obtuvo la mayor calificación en el tercer examen.
c) El alumno, si lo hubiera, que obtuvo la mayor calificación en el primero y en el sexto exámenes.
d) Dado el número que identifica a un alumno, informar en qué examen logró la menor calificación.
e) ¿En cuál examen fue más alto el promedio de los 30 alumnos?

21. Escriba un programa que genere e imprima un cuadrado mágico de dimensión N. Observe que N es entero, positivo e impar. Un cuadrado mágico es una matriz cuadrada de orden N, que contiene a los números naturales del 1 al $N * N$, y donde la suma de cualquiera de los renglones, columnas o diagonales principales es siempre la misma. Puede utilizar los siguientes pasos para generar un cuadrado mágico:

a) El número 1 se coloca en la casilla central del primer renglón.
b) El siguiente número se coloca en la casilla correspondiente al renglón anterior y columna posterior.
c) El renglón anterior al primero es el último, y la columna posterior a la última es la primera.
d) Si el número es un sucesor de un múltiplo de N, no se aplica la regla 2, sino que se coloca en la casilla del renglón posterior y en la misma columna.

Si $N = 5$, el cuadrado generado debe quedar:

	1	2	3	4	5
1	17	24	1	8	15
2	23	5	7	14	16
3	4	6	13	20	22
4	10	12	19	21	3
5	11	18	25	2	9

22. Sean $A(M \times N)$ y $B(N)$ arreglos de dos y una dimensión, respectivamente. Escriba un programa que asigne valores a B, a partir de A, teniendo en cuenta los siguientes criterios:

a) $b_i = \sum_{j=1}^{n} a_{i,j}$ Si i es impar

b) $b_i = \sum_{j-1}^{n} a_{i,j} * a_{i-1,j} * a_{i-1,j}$ Si i es par

23. Sean $A(M \times N)$ y $B(N)$ dos arreglos de dos y una dimensión, respectivamente.

Escriba un programa que asigne valores a A, a partir de B, teniendo en cuenta los siguientes criterios:

a) $a_{ij} = bi$ Si $i \leq j$
b) $a_{ij} = 0$ Si $i > j$

Combinaciones entre arreglos y registros

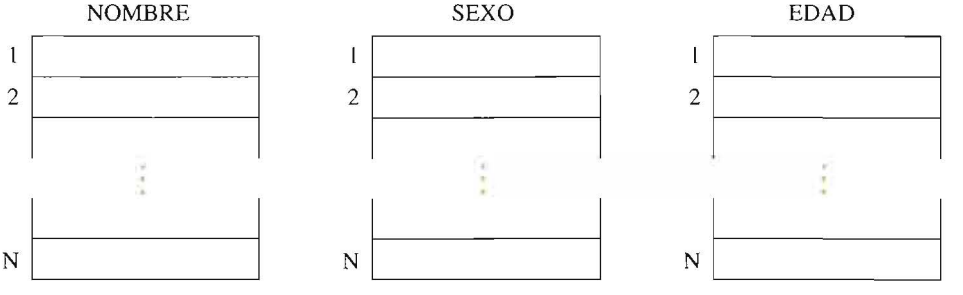
24. El departamento de personal de una escuela tiene registros del nombre, sexo y edad de cada uno de los profesores adscritos ahí.

NOMBRE	SEXO	EDAD

Escriba un programa que calcule e imprima los siguientes datos:

a) Edad promedio del grupo de profesores.
b) Nombre del profesor más joven del grupo.
c) Nombre del profesor de más edad.
d) Número de profesoras con edad mayor al promedio.
e) Número de profesores con edad menor al promedio.

25. Resuelva el problema anterior con tres arreglos paralelos. Compare sus soluciones.



26. En una escuela por cada alumno se tienen los siguientes datos:

- Nombre
- Matrícula
- Número de semestres cursados
- Calificación promedio por semestre

Escriba un programa que, dada la información de N alumnos, pueda realizar las siguientes operaciones:

a) Listar nombre y matrícula de estudiantes con promedios generales mayores o iguales a 8.
b) Actualizar los campos que correspondan cuando un estudiante ha concluido un semestre.
c) Listar nombre y matrícula de estudiantes que hayan obtenido 9 o más de calificación en todos los semestres cursados hasta el momento.

27. Una compañía distribuye *N* productos a distintos comercios de la ciudad. Para ello almacena en un arreglo toda la información relacionada con su mercancía:

- Clave
- Descripción
- Existencia
- Mínimo a mantener de existencia
- Precio unitario

Escriba un programa que efectúe las siguientes operaciones:

a) *Venta de un producto*: se deben actualizar los campos que correspondan y verificar que la nueva existencia no esté por debajo del mínimo. (Datos: clave, cantidad vendida.)
b) *Reabastecimiento de un producto*: se deben actualizar los campos que correspondan. (Datos: clave, cantidad comprada.)
c) *Actualizar el precio de un producto*. (Datos: clave, porcentaje de aumento.)
d) *Informar sobre un producto*: se deben proporcionar todos los datos relacionados con un producto. (Dato: clave.)

28. Al momento de su ingreso al hospital, a un paciente se le solicitan los siguientes datos:

- Nombre
- Edad
- Sexo
- Domicilio:
  - Calle
  - Número
  - Ciudad
- Teléfono
- Seguro (este campo tendrá el valor VERDADERO si el paciente tiene seguro médico y FALSO en otro caso)

Escriba un programa que pueda llevar a cabo las siguientes operaciones:

a) Listar los nombres de todos los pacientes hospitalizados.
b) Obtener el porcentaje de pacientes hospitalizados en las siguientes categorías (dadas por la edad):

Niños: hasta 13 años.
Jóvenes: mayores de 13 años y menores de 30.
Adultos: mayores de 30 años.

c) Obtener el porcentaje de hombres y de mujeres hospitalizados.
d) Dado el nombre de un paciente, listar todos los datos relacionados con dicho paciente.
e) Calcular el porcentaje de pacientes que poseen seguro médico.

29. Una inmobiliaria tiene información sobre departamentos en renta. De cada departamento se conoce:

- *Clave*: es un entero que identifica al inmueble.
- *Extensión*: superficie del departamento, en metros cuadrados.
- *Ubicación*: (excelente, buena, regular, mala).
- *Precio*: es un real.
- *Disponible*: VERDADERO si está disponible para la renta y FALSO si ya está rentado.

Diariamente acuden muchos clientes a la inmobiliaria solicitando información.

Escriba un programa capaz de realizar las siguientes operaciones sobre la información disponible:

a) Liste los datos de todos los departamentos disponibles que tengan un precio inferior o igual a cierto valor *P*.
b) Liste los datos de los departamentos disponibles que tengan una superficie mayor o igual a un cierto valor dado *E* y una ubicación excelente.
c) Liste el monto de la renta de todos los departamentos alquilados.
d) Llega un cliente solicitando rentar un departamento. Si existe alguno con una superficie mayor o igual a la deseada, con precio y ubicación que se ajustan a las necesidades del cliente, el departamento se rentará. Actualizar los datos que correspondan.
e) Se vence un contrato si no se renueva, actualizar los datos que correspondan.
f) Se ha decidido aumentar las rentas en un X%. Actualizar los precios de las rentas de los departamentos no alquilados.

Problemas interesantes

(Decida el lector qué estructura de datos debe utilizar para resolverlos.)

30. Escriba un programa que lea un número romano e imprima su equivalente en arábigo.

Recuerde que:

I = 1
V = 5

X = 10
L = 50
C = 100
D = 500
M = 1 000

31. Escriba un programa que calcule e imprima los números perfectos comprendidos entre dos números *A* y *B*. Un número es perfecto si la suma de sus divisores, excepto él mismo, es igual al propio número.

32. Escriba un subprograma que reciba como datos el nombre de un día de la semana y un número entero *N*, positivo o negativo, e imprima el día de la semana correspondiente a *N* días después —positivo— o *N* días antes —negativo— del día dado.

33. Lo mismo que en el problema 35, pero ahora con respecto a un mes.

34. Escriba un programa que calcule e imprima los números primos menores que cierto número dado *N*.

35. Escriba un programa que calcule e imprima los números primos gemelos menores que cierto número dado *N*. Dos números son primos gemelos si son números primos con una diferencia entre ellos de exactamente 2. Por ejemplo, 3 y 5 son primos gemelos.