CONTENIDO DE LA LECCIÓN 19

ARREGLOS MULTIDIMENSIONALES

LECCIÓN 19

ARREGLOS MULTIDIMENSIONALES

INTRODUCCIÓN

Un arreglo multidimensional es simplemente una extensión de un ***arreglo unidimensional***. Más que almacenar una sola lista de elementos, piense en un ***arreglo multidimensional*** como el almacenamiento de múltiples listas de elementos. Por ejemplo, un arreglo ***bidimensional*** almacena listas en un formato de tabla de dos dimensiones de ***filas*** y ***columnas***, en donde cada ***fila*** es una lista. Las filas proporcionan la dimensión vertical del ***arreglo***, y las columnas dan la dimensión horizontal. Un ***arreglo*** de tres dimensiones almacena listas en un formato de ***tres dimensiones*** de ***filas***, ***columnas*** y ***planos***, en donde cada ***plano*** es un ***arreglo bidimensional***. Las ***filas*** proporcionan la dimensión vertical; las ***columnas***, la dimensión horizontal; y los ***planos***, la dimensión de profundidad del arreglo.

- ***Declarar*** *y* ***manipular*** ***arreglos*** *con* ***múltiples*** ***índices***.

En esta lección, aprenderá acerca de los ***arreglos*** de ***dos*** y ***tres dimensiones*** (raramente se necesitan en programación arreglos de mayor dimensión) La lección concluirá con un extenso ejercicio de solución de problemas empleando ***arreglos*** de ***dos dimensiones*** para resolver una serie de ecuaciones simultáneas usando la regla de **Cramer**.

ARREGLOS DE DOS DIMENSIONES

El ***arreglo multidimensional*** más común es el denominado ***bidimensional*** que se muestra en la ***figura 19.1***. Aquí, es posible observar que un ***arreglo bidimensional*** contiene múltiples ***filas***. Es como si algunos ***arreglos*** de alguna dimensión se combinaran para formar una estructura de datos rectangular. Como resultado, considere esta estructura de datos rectangular como una ***tabla*** de elementos.

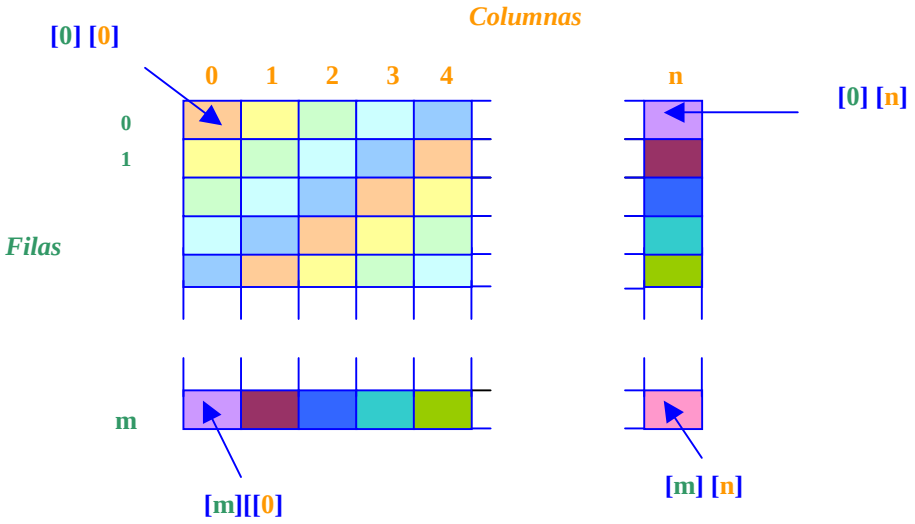


Figura 19.1*. La estructura de un* ***arreglo*** ***bidimensional****.*

Observe que el ***arreglo bidimensional*** de la ***figura 19.1*** se compone de elementos que se localizan mediante ***filas*** y ***columnas***. Las ***filas*** se etiquetan en el eje vertical en un intervalo de ***0*** a ***m***, mientras que las ***columnas***, en el eje horizontal en un intervalo de ***0*** a ***n***. **¿Cuántas *filas* y *columnas* hay?** Cada dimensión empieza con el índice ***[0]***, por tanto habrá ***m + 1 filas*** y ***n + 1 columnas***, **¿es correcto?** Como resultado, se dice que este ***arreglo bidimensional*** tiene una dimensión o tamaño de ***m + 1 filas*** por ***n + 1 columnas*** y se escribe ***(m + 1) × (n + l)***

¿Cuántos elementos están en el arreglo? Correcto: ***¡(m + 1) veces (n + 1) elementos!*** **¿Cómo supone que se localiza un *elemento* dado?** Correcto de nuevo: mediante la especificación de sus valores de ***índice*** de ***fila*** y ***columna***. Por ejemplo, el ***elemento*** en la esquina superior izquierda se localiza en la intersección de la ***fila 0*** y ***columna 0*** o ***índice [0][0]*** De igual manera, el ***elemento*** en la esquina inferior derecha se localiza donde la ***fila m*** se encuentra con la ***columna n*** o ***índice [m][n]*** Se dice que los ***arreglos*** de ***dos dimensiones*** en ***C*** y ***C++*** se ***ordenan con la fila principal.*** Esto significa que primero se lista el ***índice*** de la ***fila*** (por convención), seguido del ***índice*** de la ***columna*** (también por convención)

DEFINICIÓN DE ARREGLOS BIDIMENSIONALES EN C++

En ***C++*** se define un ***arreglo bidimensional*** casi igual como se define un ***arreglo unidimensional***. A continuación se muestra el formato general:

FORMATO PARA ARREGLO BIDIMENSIONAL

<clase de datos del elemento> <nombre del arreglo> [<número de filas>], [<número de columnas>];

La única diferencia entre esta definición y la que se requiere para un ***arreglo unidimensional*** se encuentra dentro de la especificación del tamaño. Se debe especificar el tamaño de ***filas*** y ***columnas***, como se muestra.

Ejemplo 19.1

Dadas las siguientes definiciones de ***arreglo bidimensional***, dibuje un diagrama de la estructura del ***arreglo*** mostrando los ***índices*** respectivos de ***filas*** y ***columnas***.

a.
```
float tabla[5][7];
```
b.
```
const int FILAS = 5;
const int COLS = 7;
float tabla[FILAS][COLS];
```
c.
```
const int CORRIENTE = 26;
const int RESISTENCIA = 1001;
int voltaje[CORRIENTE][RESISTENCIA];
```
d.
```
const int SEMANAS = 6;
const int DIAS = 7;
int mayo[SEMANAS][DIAS];
```
e.
```
const int FILAS = 57;
const int BUTACAS = 10;
int butaca0cupada[FILAS][BUTACAS]
;
```

Solución

a. *Observe la **figura 19.2(a)** Se trata de un **arreglo rectangular** o **tabla**, con un intervalo de **filas** de **0** a **4** y un intervalo de **columnas** de **0** a **6**. Recuerde que, debido a que los **índices** del*

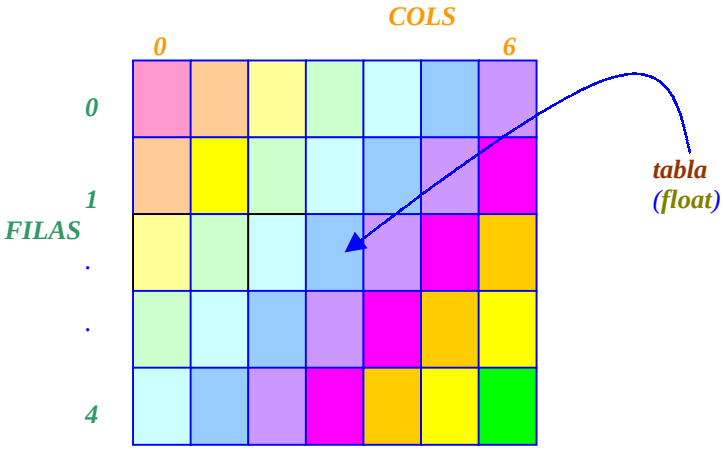
arreglo comienzan con [0], el último índice en una dimensión de arreglo dada es 1 menos que su tamaño. El nombre del arreglo es tabla, y almacenará valores de punto flotante.

b. *De nuevo observe la figura 19.2(a) Este arreglo es idéntico al primero. Aquí la única diferencia es la forma en que está definido. Observe que los índices de fila y columna se definen como [FILAS] [COLS], donde FILAS y COLS son constantes.*

c. *Observe la figura 19.2(b) Allí, las filas se llaman CORRIENTE y el intervalo es de 0 a 25. Las columnas se etiquetan como RESISTENCIA y el intervalo es de 0 a 1000. El nombre del arreglo es voltaje, y almacenará elementos enteros. Obviamente, el arreglo almacenará los valores de voltaje correspondientes a los de CORRIENTE de 0 a 25 y los valores de RESISTENCIA de 0 a 1000, usando la ley de Ohm.*

d. *Observe la figura 19.2(c) Este arreglo se construye para almacenar las fechas para el mes de mayo, como un calendario, observe un calendario común si tiene uno a la mano. ¿No es un mes sólo una tabla de enteros con valores en donde se localizan una semana y un día dentro de esa semana? Como puede observar en la figura 19.2(c), la estructura del arreglo parece un calendario mensual. Las filas se etiquetan de 0 a 5, representando las 6 posibles semanas en cualquier mes dado. Las columnas del arreglo se etiquetan de 0 a 6, representando los 7 días de la semana.*

e. *Observe la figura 19.2(d) Este último arreglo también tiene una aplicación práctica. ¿Es posible determinar lo que es a partir de la definición? Observe que se trata de un arreglo de elementos enteros. Las filas con un intervalo de 0 a 56 y las columnas con uno de 0 a 9, como se muestra. El nombre del arreglo es butacaOcupada. Suponga que usa este arreglo para almacenar valores enteros de 0 y 1, donde el 0 representa el valor booleano de falso y 1 representa el valor booleano de verdadero. Entonces, este arreglo se puede usar en un programa de reservación para un teatro o vuelo de línea aérea, para indicar si una butaca o un asiento dados están o no ocupados.*



(a)

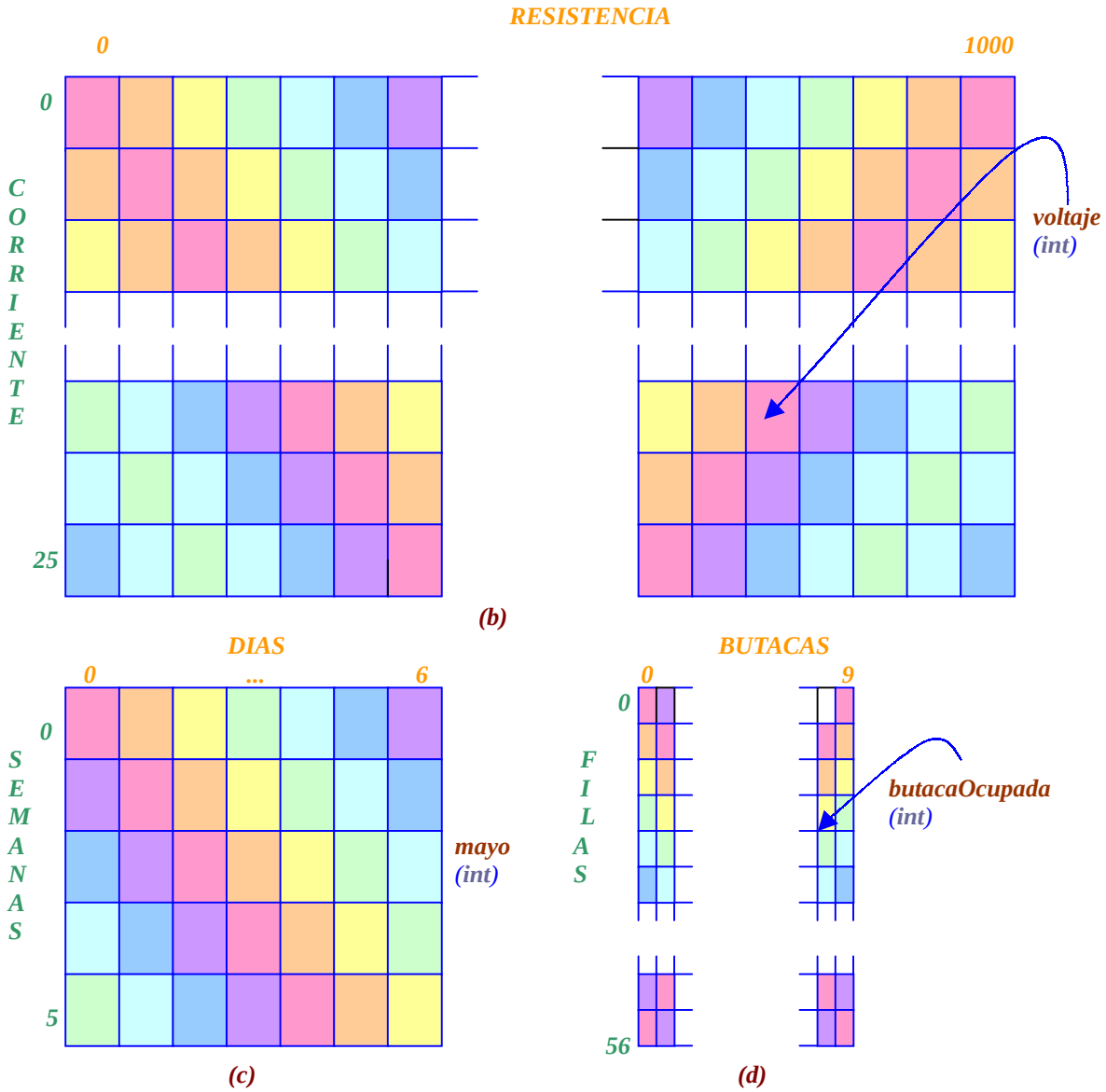


Figura 19.2. Cuatro arreglos bidimensionales para el ejemplo 19.1

Ejemplo 19.2

¿Cuántos y qué clase de elementos almacenará cada uno de los arreglos del ejemplo 19.1?

Solución

a. *El arreglo de la figura 19.2(a) almacenará 5 × 7 o 35 elementos. Los elementos serán valores de punto flotante.*

b. *El arreglo de la figura 19.2(b) tiene 26 filas, de 0 a 25 y 1001 columnas, de 0 a 1000. De esta manera, el arreglo almacenará 26 × 1001 = 26,026 elementos enteros.*

c. *El arreglo de la figura 19.2(c) tiene 6 filas y 7 columnas y, por lo tanto, almacenará 6 × 7 = 42 elementos enteros.*

d. *El arreglo de la figura 19.2(d) tiene 57 filas y 10 columnas y almacenará 57 x 10 = 570 elementos enteros que se manejarán como valores booleanos.*

Es posible verificar los valores anteriores con la función *sizeof()* La función *sizeof() regresa el número de bytes* que se requieren para almacenar una expresión o una clase de datos. Así, codificando los siguientes enunciados se mostrará el número de elementos en cada arreglo:

```
cout << sizeof(tabla)/sizeof(float) << endl;
     << sizeof(voltaje)/sizeof(int) << endl;
     << sizeof(mayo)/sizeof(int) << endl;
     << sizeof(butaca0cupada)/sizeof(int) << endl
```

Aquí, se llama dos veces a la función *sizeof()* para calcular la cantidad de elementos en cada arreglo. Observe que el tamaño del arreglo en bytes se divide entre el tamaño de la clase de datos del arreglo en bytes. Por tanto, el cociente será el número de elementos que el arreglo puede almacenar, ¿**correcto**? Tome en cuenta que el código anterior es totalmente portátil entre sistemas que deban representar tipos de datos usando un número diferente de bytes. Este cálculo siempre determinará el ***número de elementos*** en el arreglo respectivo, sin importar cuantos bytes se emplean para almacenar en una clase de datos dada.

Ejemplo 19.3

El siguiente programa, **TAMARRE.CPP**, utiliza el operador el operador *sizeof*() para determinar el número de bytes que utilizan diferentes clases de arreglos declarados.

```
/* El siguiente programa: TAMARRE.CPP, utiliza el operador sizeof() para determinar
   el número de bytes que utilizan diferentes clases de arreglos declarados.
*/

#include <iostream.h>              //Para cout y cin

void main(void)
        {
                int caja[3][3];
                float salarioAnual[52][5];
                char paginas[40][60][20];

                cout    << "Numero de bytes utilizado por caja[3][3]              = "
                        << sizeof(caja) << endl;
                cout    << "Numero de bytes utilizado por salarioAnual[52][5] = "
                        << sizeof(salarioAnual)<< endl;
                cout    << "Numero de bytes utilizado por paginas[40][60][20] = "
                        << sizeof(paginas)<< endl;
        }//Fin de main()
```

ACCESO A LOS ELEMENTOS DE UN ARREGLO BIDIMENSIONAL

Se accede a los **elementos** de un **arreglo bidimensional** en una forma muy parecida como a los **elementos** de un **arreglo unidimensional**. La diferencia es que para localizar los **elementos** en un **arreglo bidimensional**, se especificará un **índice** de **fila** y un **índice** de **columna**.

Es posible acceder a los **elementos** de **arreglo** usando ***asignación directa***, ***lectura/ escritura*** o ***ciclo***.

ASIGNACIÓN DIRECTA DE ELEMENTOS DE UN ARREGLO BIDIMENSIONAL

El formato general para **asignar** en ***forma directa*** valores de **elementos** es como sigue:

FORMATO DE ASIGNACIÓN DIRECTA A UN ARREGLO BIDIMENSIONAL (*inserción de elementos*)

<nombre del arreglo> [índice de fila][índice de columna] = valor del elemento;

FORMATO DE ASIGNACIÓN DIRECTA A UN ARREGLO BIDIMENSIONAL (*extracción de elementos*)

<identificador variable> =<nombre del arreglo>[índice de fila][índice de columna];

Primero, observe el formato para insertar elementos en un **arreglo bidimensional**, seguido por el formato para obtener elementos del arreglo. Observe que en ambos casos se especificará un **índice** de ***fila*** y ***columna*** para acceder a la posición del elemento deseado. Primero se especifica el ***índice*** de ***fila***, seguido por el ***índice*** de ***columna***. Mediante el uso de los **arreglos** definidos en el ejemplo ***19.1***, las ***asignaciones directas*** posibles para la inserción son:

```
tabla[21[3]          = 0.5;
voltaje[2][10]       = 20;
mayo[1][3]           = 8;
butaca0cupada[5][0]  = 1;
```

En el **primer** caso, se coloca el valor de punto flotante ***0.5*** en la ***fila [2]***, ***columna [3]*** del arreglo ***tabla***. En el **segundo** caso, se inserta un valor de ***20*** en la ***fila [2]***, ***columna [10]*** del arreglo ***voltaje***. Observe que este valor de voltaje corresponde a un valor de corriente de ***2*** y un valor de resistencia de ***10*** cuando se usa la ley de **Ohm**. En el **tercer** caso, se coloca un valor de ***8*** en la ***fila [1]***, ***columna [3]*** del arreglo ***mayo***. Por último, el caso **final** asigna el valor entero ***1*** a la ***fila [5]***, ***columna [0]*** del arreglo ***butacaOcupada***. *Si* se interpreta como un valor booleano, esto indica que la **butaca está ocupada**.

Con los mismos arreglos, los enunciados de **asignación directa** para extraer elementos son:

```
ventas       = tabla[0][0];
volts        = voltaje[5][100];
hoy          = mayo[2][4];
manana       = mayo[2][5];
tomaButaca   = butaca0cupada[3][3];
```

En cada uno de estos enunciados, se asigna a un identificador variable el valor del elemento almacenado en la posición ***fila/columna*** dentro del arreglo respectivo. Desde luego, se definirá el identificador variable con la misma clase de datos que se asignó al elemento del arreglo.

Recuerde que las operaciones de extracción no tienen efecto en los elementos del arreglo. En otras palabras, realmente no se ***retiran*** los elementos del arreglo; tan sólo se ***copian*** sus valores a la variable de asignación.

Es posible inicializar un arreglo de múltiples índices en su declaración, de manera casi igual a la inicialización de un arreglo de un solo índice. Por ejemplo, el arreglo ***b[2][2]*** podría declararse e inicializarse de la siguiente manera:

```
int b[2][2] = { {1, 2}, {3, 4} };
```

Los valores se agrupan por ***filas*** entre corchetes. Por lo tanto, ***1*** y ***2*** inicializan a ***b[0][0]*** y ***b[0][1]*** y ***3*** y ***4*** inicializan a ***b[1][0]*** y ***b[1][1]***

Si no hay suficientes inicializadores para una ***fila*** determinada, los elementos se inicializan a ***0***. Por lo tanto, la declaración:

```
int b[2][2] = { {1}, {3, 4} };
```

inicializan a ***b[0][0]*** a ***1***, y ***b[0][1]*** a ***0***, ***b[1][0]*** a ***3***, y ***b[1][1]*** a ***4***.

LECTURA Y ESCRITURA DE ELEMENTOS EN ARREGLOS BIDIMENSIONALES

Es posible usar enunciados ***cin*** para insertar elementos en arreglos **bidimensionales** y utilizar enunciados ***cout*** para extraer elementos del arreglo, como los siguientes ejemplos:

```
cin >> tabla[1][1];
cout << tabla[1][1];
cin >> voltaje[5][20];
cout << voltaje[5][20];
cin >> mayo[1][3];
cout << mayo[1][3];
if (butacaOcupada[3][1])
   cout << "VERDADERO" << endl;
else
   cout << "FALSO" << endl;
```

De nuevo, observe que se deben especificar ambos índices de ***fila*** y ***columna***. Los enunciados ***cin*** insertarán elementos que se obtienen del teclado. Entonces los enuncia- dos ***cout*** extraerán el elemento del arreglo que se insertó y simplemente ***devuelven el carácter*** (***echo***) que escribió el usuario a la pantalla. Observe que el último enunciado ***cout*** está contenido dentro de un ***if/else*** para determinar si el elemento es un valor booleano verdadero o falso. Este enunciado producirá ***VERDADERO*** o ***FALSO*** en la pantalla, dependiendo del contenido de la localización ***[3][1]*** del arreglo ***butaca0cupada***.

USO DE CICLOS PARA ACCEDER ARREGLOS DE DOS DIMENSIONES

Como se sabe, los ***ciclos*** proporcionan una forma más eficiente para acceder arreglos, en especial cuando se trabaja con arreglos multidimensionales grandes. Recuerde que con los multidimensionales ***se necesita un ciclo separado para cada dimensión del arreglo***. Además, los ciclos deben ser anidados. Así, pues, un arreglo bidimensional requiere dos ciclos anidados.

Observe el calendario de la ***figura 19.3***. Como se ha visto, es posible almacenar este calendario en memoria usando un arreglo bidimensional. **¿Cómo supone que debe llenar el calendario con las fechas requeridas para un mes dado?** Un enfoque lógico será llenar todas las fechas de la semana ***0***, ***de Dom a Sab***, después llenar la semana ***1*** con sus fechas, entonces llenar la semana ***2*** con fechas y así sucesivamente.

Piense qué deberán hacer los índices del arreglo para realizar esta operación de llenado. El índice semana deberá empezar en ***[0]*** y luego el de días deberá iniciar en ***[0]*** e incrementarse a lo largo de los días de la semana a ***[6]*** una y otra vez. Esto llenará la primera semana. Para llenar

la segunda, el índice semana se incrementará a ***1***, con el índice días empezando a ***[0]*** y de nuevo incrementando hasta ***[6]*** Para llenar la tercera semana, el índice semana se incrementará a ***2*** y de nuevo el índice días se incrementa de ***[0]*** a ***[6]*** En otras palabras, se llenan las fechas semana a semana, una semana a la vez. A cada incremento del índice semana, el índice días comienza en ***[0]*** y se incrementa de uno en uno hasta ***[6]***, antes que se incremente el índice semana de la siguiente semana.

Mayo

	Dom [0]	Lun [1]	Mar [2]	Miér [3]	Jue [4]	Vie [5]	Sáb [6]
Semana [0]							1
Semana [1]	2	3	4	5	6	7	8
Semana [2]	9	10	11	12	13	14	15
Semana [3]	16	17	18	19	20	21	22
Semana [4]	23	24	25	26	27	28	29
Semana [5]	30	31					

*Figura 19.3. Calendario de **mayo**.*

¿Esto sugiere dos **ciclos**, **uno** para incrementar el **índice semana** y **otro** para incrementar el **índice días**? Más aún, **¿no es cierto que este proceso sugiere que el ciclo días se deberá anidar dentro del ciclo semana, porque los días deberán estar en su intervalo completo para cada semana?**

A continuación se muestra la estructura general del ciclo para acceder a los elementos en un arreglo bidimensional.

FORMATO DE CICLO PARA ACCEDER ELEMENTOS DE ARREGLOS BIDIMENSIONALES

```
for (int fila = 0; fila < tamanoFila; ++ fila)
   for (int columna = 0;  columna < tamanoColumna; ++ columna)
        <Procesa Arreglo [fila][columna]>;
```

Se observa que se anida el ciclo de índice de ***columna*** dentro del ciclo de índice de ***fila*.** De esta manera, el ciclo ***columna*** corre a través de todas sus iteraciones para cada iteración del ciclo ***fila*.** La inserción real se lleva a cabo dentro del ciclo ***columna***. Veamos algunos ejemplos para comprender muy bien la idea.

Ejemplo 19.4

El siguiente programa, ***MUESTRA.CPP***, visualiza los valores de un arreglo llamado ***tabla***.

```
/* El siguiente programa: MUESTRA.CPP, usa las variables renglón y columna para
   visualizar los valores del arreglo llamado tabla.
*/

#include <iostream.h>              //Para cout y cin

void main(void)
```

```
        {
                int renglon, columna;

                float tabla[3][5] = {   {1.0, 2.0, 3.0, 4.0, 5.0},
                                        {6.0, 7.0, 8.0, 9.0, 10.0},
                                        {11.0, 12.0, 13.0, 14.0, 15.0}};

                for (renglon = 0; renglon < 3; renglon++)
                        for (columna = 0; columna < 5; columna++)
                                cout    << "tabla[" << renglon << "][" << columna << "] = "
                                        << tabla[renglon][columna] << endl;
        }//Fin de main()
```

Ejemplo 19.5

El siguiente programa, ***FUNARRE.CPP***, utiliza la función ***mostrarArreglo***(), para visualizar el contenido de varios arreglos de dos dimensiones.

```
/* El siguiente programa: FUNARRE.CPP, utiliza la función mostrarArreglo()
   para visualizar el contenido de varios arreglos de dos dimensiones.
*/

#include <iostream.h>     //Para cout y cin

void mostrarArreglo(int arreglo[][10], int renglones)
        {
                int i, j;

                for (i = 0; i < renglones; i++)
                        for (j = 0; j < 10; j++)
                                cout    << "arreglo[" << i << "][" << j << "] = " << arreglo[i][j]
                                        << endl;
        }//Fin de mostrarArreglo()

void main(void)
        {
                int a[1][10] = {   {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}};
                int b[2][10] = {   {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
                                   {11, 12, 13, 14, 15, 16, 17, 18, 19, 20}};
                int c[3][10] = {   {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
                                   {11, 12, 13, 14, 15, 16, 17, 18, 19, 20},
                                   {21, 22, 23, 24, 25, 26, 27, 28, 29, 30}};

                mostrarArreglo(a, 1);
                cout << endl;

                mostrarArreglo(b, 2);
                cout << endl;

                mostrarArreglo(c, 3);
                cout << endl;
        }//Fin de main()
```

Observe que la definición de la función ***mostrarArreglo[]*** especifica que el parámetro del arreglo es ***int arreglo[][10]*** Cuando se recibe un arreglo con un solo índice como argumento de una función, los corchetes del arreglo van en blanco en la lista de parámetros de la función. Tampoco es necesario el tamaño del primer índice de un arreglo de múltiples índices, pero sí son indispensables los tamaños de todos los índices subsecuentes. El compilador emplea dichos tamaños para determinar las localidades de memoria de los elementos de los arreglos de múltiples índices. Todos los elementos del arreglo se almacenan consecutivamente en la memoria, sin importar la cantidad de índices. En los arreglos de doble índice, la primera fila se almacena en memoria seguida de la segunda.

Ejemplo 19.6

El siguiente programa, ***FUNARRE2.CPP***, utiliza la función ***mostrarArreglo***(), para visualizar el contenido de varios arreglos de dos dimensiones.

```
/* El siguiente programa: FUNARRE2.CPP, inicializa arreglos multidimensionales y los
   muestra mediante la función mostrarArreglo().
*/

#include <iostream.h>              //Para cout y cin

void mostrarArreglo(int [][3]);

void main(void)
        {
                int     arreglo1 [2][3]  =      {       {1, 2, 3}, {4, 5, 6}},
                        arreglo2 [2][3]  =      {        1, 2, 3, 4, 5},
                        arreglo3 [2][3]  =      {       {1, 2}, {4} };

                cout << "Los valores en arreglo1 por filas son:" << endl;
                mostrarArreglo(arreglo1);

                cout << "Los valores en arreglo2 por filas son:" << endl;
                mostrarArreglo(arreglo2);

                cout << "Los valores en arreglo3 por filas son:" << endl;
                mostrarArreglo(arreglo3);
        }//Fin de main()

void mostrarArreglo(int a[][3])
        {
                for(int i = 0; i < 2; i++)
                        {
                                for(int j = 0; j < 3; j++)
                                        cout << a[i][j] << ' ';

                                cout << endl;
                        }//Fin del for externo
        }//Fin de mostrar Arreglo()
```

Ejemplo 19.7

El siguiente programa, ***SUMARRE.CPP***, regresa la suma de los valores de un arreglo bidimensional

Nota: *Es importante que observe que se está pasando un* ***arreglo bidimensional*** *a una función que maneja arreglos de una dimensión.*

```
/* El siguiente programa: SUMARRE.CPP, regresa la suma de los valores de un
   arreglo bidimensional.
*/

#include <iostream.h>     //Para cout y cin

long sumarArreglo(int arreglo[], int elementos)
        {
        long suma = 0;
        int i;

        for (i = 0; i < elementos; i++)
                suma += arreglo[i];

        return(suma);
        }//Fin de sumarArreglo()

void main(void)
        {
        int a[10]          = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int b[2][10]       = {{1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
                              {11, 12, 13, 14, 15, 16, 17, 18, 19, 20}};

        int c[3][10]       = {{1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
                              {11, 12, 13, 14, 15, 16, 17, 18, 19, 20},
                              {21, 22, 23, 24, 25, 26, 27, 28, 29, 30}};

        cout << "La suma de los elementos del arreglo a[10] es    = " <<
                                sumarArreglo(a, 10)<< endl;
        cout << "La suma de los elementos del arreglo b[2][10] es = " <<
                                sumarArreglo((int *)b, 20)<< endl;
        cout << "La suma de los elementos del arreglo c[3][10] es = " <<
                                sumarArreglo((int *)c, 30)<< endl;
        }//Fin de
```

Ejemplo 19.8

El siguiente programa, ***MATRIZ1.CPP***, ilustra el uso de arreglos bidimensionales para calcular el promedio de cada columna en una matriz.

***Nota**: Conviene observar y analizar la forma en que se declara el arreglo **matriz**.*

```
/* El siguiente programa: MATRIZ1.CPP, calcula el promedio de cada columna en una matriz
   bidimensional.
*/

#include <iostream.h>              //Para cout y cin

const int COL_MAX = 3;
const int REN_MAX = 3;

void main(void)
        {
                double matriz[REN_MAX][COL_MAX] =  {     1, 2, 3,     // Renglón #1
                                                         4, 5, 6,     // Renglón #2
                                                         7, 8, 9      // Renglón #3
                                                   };
```

```
        double suma, promedio;
        int renglones = REN_MAX, columnas = COL_MAX;

        // Despliega los elementos de la matriz
        cout << "La matriz es:" << endl;
        for (int i = 0; i < renglones; i++)
                {
                        for (int j = 0; j < columnas; j++)
                                {
                                        cout.width(4);
                                        cout.precision(1);
                                        cout << matriz[i][j] << " ";
                                }//Fin del for interno

                        cout << endl;
                }//Fin del for externo

        cout << endl;

        // Obtiene la suma de cada columna
        for (int j = 0; j < columnas; j++)
                {
                        suma = 0.0;          // Inicializa suma
                        for (int i = 0; i < renglones; i++)
                                suma += matriz[i][j];

                        promedio = suma / renglones;
                        cout     << "El promedio de la columna " << j
                                 << " = " << promedio << endl;
                }//Fin del for
}//Fin de main()
```

Ejemplo 19.9

El siguiente programa, ***MATRIZ2.CPP***, ilustra el uso de arreglos bidimensionales para **calcular el promedio de cada columna en una matriz**. (Es una variante del problema anterior)

```
/* El siguiente programa: MATRIZ2.CPP, calcula el promedio de cada columna de una matriz. */

#include <iostream.h>                //Para cout y cin

const int COL_MIN = 1;
const int COL_MAX = 10;
const int REN_MIN = 2;
const int REN_MAX = 30;

// Pide al usuario el número de renglones y columnas
int ObtenNumPuntos(const char *tipoElemento, int minimo, int maximo)
        {
                int numPuntos;
```

```
                do
                        {
                                cout    << "Introduzca el número de "
                                        << tipoElemento << " ["
                                        << minimo << " a "
                                        << maximo << "]: ";
                                cin >> numPuntos;
                        } while (numPuntos < minimo || numPuntos > maximo);
                return numPuntos;
        }//Fin de obtenNumPuntos()

void main(void)
        {
                float matriz[REN_MAX][COL_MAX];
                float suma, promedio;
                int renglones, columnas;

                // Obtener el número de renglones y columnas
                renglones       = ObtenNumPuntos("renglones", REN_MIN, REN_MAX);
                columnas        = ObtenNumPuntos("columnas", COL_MIN, COL_MAX);

                cout << endl;
                // Obtener los elementos de la matriz
                for (int i = 0; i < renglones; i++)
                        {
                                for (int j = 0; j < columnas; j++)
                                        {
                                                cout << "matriz[" << i << "][" << j << "]: ";
                                                cin >> matriz [i][j];
                                        }//Fin del for interno
                                cout << endl;
                        }//Fin del for externo

                // Obtiene la suma de cada columna
                for (int j = 0; j < columnas; j++)
                        {
                                suma = 0.0;             // Inicializa suma
                                for (int i = 0; i < renglones; i++)
                                        suma += matriz[i][j];

                                promedio = suma / renglones;
                                cout    << "El promedio de la columna " << j
                                        << " = " << promedio << endl;
                        }//Fin del for externo
        }//Fin de main()
```

Ejemplo 19.10

El siguiente programa, ***MATRIZ3.CPP***, es una variante del programa anterior (utiliza funciones)

```
/* El siguiente programa: MATRIZ3.CPP, calcula el promedio de cada columna de una matriz. */

#include <iostream.h>                   //Para cout y cin

const int COL_MIN = 1;
const int COL_MAX = 10;
const int REN_MIN = 2;
const int REN  MAX = 30;
```

```
// Pide al usuario el número de renglones y columnas
int ObtenNumPuntos(const char *tipoElemento, int minimo, int maximo)
	{
		int numPuntos;
		do
			{
				cout	<< "Introduzca el número de "
					<< tipoElemento << " ["
					<< minimo << " a "
					<< maximo << "]: ";
				cin >> numPuntos;
			} while (numPuntos < minimo || numPuntos > maximo);
		return numPuntos;
	}//Fin de obtenNumPuntos()

// Obtener los elementos de la matriz
void elementosMatriz(float matriz[][COL_MAX], int renglones, int columnas)
	{
		cout << endl;
		for (int i = 0; i < renglones; i++)
			{
				for (int j = 0; j < columnas; j++)
					{
						cout << "matriz[" << i << "][" << j << "]: ";
						cin >> matriz [i][j];
					}//Fin del for interno
				cout << endl;
			}//Fin del for externo
	}//Fin de elementosMatriz()

// Obtiene la suma de cada columna
void mostrarPromedio(float matriz[][COL_MAX], int renglones, int columnas)
	{
		float suma, promedio;

		for (int j = 0; j < columnas; j++)
			{
				suma = 0.0;		// Inicializa suma
				for (int i = 0; i < renglones; i++)
				suma += matriz[i][j];

				promedio = suma / renglones;
				cout	<< "El promedio de la columna " << j
					<< " = " << promedio << endl;
			}//Fin del for externo
	}//Fin de mostrarPromedio()

void main(void)
	{
		float matriz[REN_MAX][COL_MAX];
		int renglones, columnas;

		// Obtener el número de renglones y columnas
		renglones	= ObtenNumPuntos("renglones", REN_MIN, REN_MAX);
		columnas	= ObtenNumPuntos("columnas", COL_MIN, COL_MAX);

		elementosMatriz(matriz, renglones, columnas);
		mostrarPromedio(matriz, renglones, columnas);
	}//Fin de main()
```

Ejemplo 19.11

El siguiente programa, ***FUNARRE3.CPP***, efectúa otras manipulaciones comunes de arreglos sobre un arreglo de **3 × 4** denominado **calificacionesAlumnos**. Cada fila del arreglo representa a un estudiante y cada columna representa la calificación de uno de los cuatro exámenes que realizaron dichos estudiantes durante el semestre. Las manipulaciones de arreglos las realizan cuatro funciones. La función **minimo()** determina la calificación más baja de todos los estudiantes durante el semestre; la función **maximo()** la calificación más alta. La función **promedio()** calcula el promedio semestral de un estudiante en particular. La función **mostrarArreglo()** envía la salida, en formato de tabla, el arreglo de doble índice.

```
/* El siguiente programa: FUNARRE3.CPP, ilustra el uso de los arreglos de doble índice. */

#include <iostream.h>    //Para cout y cin
#include <iomanip.h>     //Para setw()

const int ESTUDIANTES        = 3;     // Número de estudiantes
const int EXAMENES           = 4;     // Número de exámenes

int minimo(int[][EXAMENES], int, int);
int maximo(int[][EXAMENES], int, int);
float promedio(int[], int);
void mostrarArreglo(int[][EXAMENES], int, int);

void main(void)
        {
                int calificacionesAlumnos[ESTUDIANTES][EXAMENES] = {    {77, 68, 86, 73},
                                                                        {96, 87, 89, 78},{70, 90, 86, 81}};

                cout << "El arreglo es:\n";
                mostrarArreglo(calificacionesAlumnos, ESTUDIANTES, EXAMENES);

                cout    << "\n\nCalificación más baja: "
                        << minimo(calificacionesAlumnos, ESTUDIANTES, EXAMENES)
                        << "\nCalificación más alta: "
                        << maximo(calificacionesAlumnos, ESTUDIANTES, EXAMENES) << '\n';

                for(int persona = 0; persona < ESTUDIANTES; persona++)
                        cout    << "La calificación promedio para el estudiante " << persona << " es "
                                << setiosflags(ios:: fixed | ios:: showpoint)
                                        << setprecision(2)
                                        << promedio(calificacionesAlumnos[persona], EXAMENES) << endl;
        }//Fin de main

//Encuentra la calificación mínima
int minimo(int calificaciones[][EXAMENES], int alumnos, int pruebas)
        {
                int calificacionBaja = 100;

                for(int i = 0; i < alumnos; i++)
                        for(int j = 0; j < pruebas; j++)
                                if(calificaciones[i][j] < calificacionBaja)
                                        calificacionBaja = calificaciones[i][j];
                return calificacionBaja;
        }//Fin de minimo()
```

```
//Encuentra la calificación máxima
int maximo(int calificaciones[][EXAMENES], int alumnos, int pruebas)
        {
                int calificacionAlta = 0;

                for(int i = 0; i < alumnos; i++)
                        for(int j = 0; j < pruebas; j++)
                                if(calificaciones[i][j] > calificacionAlta)
                                        calificacionAlta = calificaciones[i][j];
                return calificacionAlta;
        }//Fin de máxima()

//Determina el promedio de un estudiante en particular
float promedio(int calificacionAlumno[], int pruebas)
        {
                int total = 0;

                for(int i = 0; i < pruebas; i++)
                        total += calificacionAlumno[i];

                return (float) total / pruebas;
        }//Fin de promedio()

//Imprime el arreglo
void mostrarArreglo(int calificaciones[][EXAMENES], int alumnos, int pruebas)
        {
                cout << "                    [0]  [1]  [2]  [3]";

                for(int i = 0; i < alumnos; i++)
                        {
                                cout << "\ncalificacionesEstudiantes[" << i << "] ";

                                for(int j = 0; j < pruebas; j++)
                                        cout     << setiosflags(ios::left) << setw(5)
                                                 << calificaciones[i][j];
                        }//Fin del for externo
        }//Fin de mostrarArreglo()
```

Ejemplo 19.12

Escriba una función que use ciclos para llenar un arreglo calendario para el mes de Mayo. Escriba otra función para mostrar el calendario Mayo.

Solución

Se empezará por definir el arreglo mes como antes.

```
const int SEMANAS       = 6;
const int DIAS          = 7;
int mayo[SEMANAS][DIAS];
enum diasSemana {Dom, Lun, Mar, Miér, Jue, Vie, Sab};
```

Además, como puede ver, se ha definido una clase de datos enumerados llamado ***diasSemana***. Se usará ésta para acceder a los días de la semana dentro del arreglo, como se verá en breve.

Después se escribirá una función para llenar el arreglo con las fechas de Mayo. Pero primero, se deberá considerar la interfaz de la función. La función deberá recibir al arreglo, obtener las fechas del usuario y

regresar el arreglo lleno al programa llamador. De esta manera, la descripción de interfaz de función se convierte en:

Función llenaMes():	***Obtiene** del usuario las fechas del mes y llena un arreglo de enteros de dos dimensiones.*
Acepta:	***Un arreglo bidimensional** de enteros de tamaño **SEMANAS** × **DÍAS***
Regresa:	***Un arreglo bidimensional** de enteros de tamaño **SEMANAS** × **DÍAS**.*

*Ahora se muestra la **función** que hará el trabajo:*

```
//ESTA FUNCIÓN LLENARÁ UN ARREGLO DE 2 DIMENSIONES PARA UN
//CALENDARIO MENSUAL

void llenaMes(int mes[SEMANAS][DIAS])
        {
                cout << "¿Cuál mes quiere llenar y mostrar?
                cin >> esteMes;
                cout << "Escriba las fechas del mes, empezando con Domingo de la primera\n"
                        "semana en el mes. Si no existe fecha para un día determinado\n"
                       "escriba un 0. Presione la tecla ENTER después de cada entrada."
                     << endl;

                for (int semana = 0; semana < SEMANAS; ++semana)
                        {
                                for (int dia = Dom; dia < Sab + 1; ++dia)
                                        cout << "\nEscriba la fecha para la Semana "
                                             << semana << " día " << dia << ": ";
                                        cin >> mes[semana][dia];
                        } // FINAL DEL CICLO DIA
        } // FINAL DE llenaMes()
```

Las primeras filas de la función proporcionan unas cuantas instrucciones sencillas al usuario y obtiene el mes que se va a llenar y a presentar. La operación llenado de arreglo se realiza dentro de los dos ciclos ***for***. Observe que ***semana*** es el ciclo externo y ***dia*** *es* el interno. A continuación se muestra cómo funciona: El contador ***semana*** empieza con ***0*** y el contador ***dia*** empieza con ***Dom***. Como resultado, la primera fecha que se inserta en ***mes[0][Dom]***, corresponde en el mes a ***Domingo*** de la semana ***0***. Observe que el nombre, del arreglo es ***mes***. ¿Cómo es posible, si se definió a ***mayo*** como el nombre del arreglo? Bueno, el arreglo ***mayo*** será el argumento real que se usa en la llamada de función. Sin embargo, ***mes*** es el parámetro formal listado en el encabezado de la función. De esta manera, la función recibirá del programa llamador el arreglo ***mayo***, la llenará y regresará al programa llamador. Por esta razón se usó un nombre de arreglo diferente ***(mes)*** en la función, para hacerla más general. Por ejemplo, es posible definir arreglos adicionales, como ***Junio***, ***Julio***, ***Agosto*** y la que guste para crear arreglos para estos meses. De nuevo, éstos actúan como argumentos reales cuando se llama a la función. Entonces, es posible llenar en forma separada los arreglos de meses respectivos ***(Junio***, ***Julio***, ***Agosto***, *etc*.) utilizando llamadas separadas para esta misma función. En cada caso, el parámetro de función ***mes*** tomará el argumento de arreglo real que se usó en la llamada de función. Ahora, de regreso a los ciclos. Después de que se lee un elemento en ***mes[0][Dom]***, el ciclo interno ***for*** incrementa el contador ***dia*** a ***Lun*** y se lee un elemento en ***mes[0][Lun]*** Observe que el contador ***semana*** permanece igual. **¿Cuál es la siguiente posición del arreglo que se va a llenar**? Correcto: ***mes[0][Mar]*** En resumen, el ciclo interno ***dia*** se incrementará de ***Dom*** a ***Sab*** para cada iteración del ciclo externo ***semana***. De esta manera, la primera iteración del ciclo externo ***semana*** se llenará como sigue:

mes[0][Dom]
mes[0][Lun]
mes[0][Mar]
mes[0][Mier]

mes[0][Jue]
mes[0][Vie]
mes[0][Sab]

La segunda iteración del ciclo externo fila llenará la segunda semana así:

mes[1][Dom]
mes[1][Lun]
mes[1][Mar]
mes[1][Mier]
mes[1][Jue] mes[1][Vie] mes[1][Sab]

Este proceso de llenado continuará para las semanas *2*, *3*, *4* y *5*. El ciclo se termina cuando se lee un valor en la última posición del arreglo, *mes[5][Sab]*

Ahora, veamos una función similar para presentar el calendario *mayo* una vez que se ha llenado. De nuevo, considere la siguiente descripción de interfaz de función:

Función muestraMes():	*Muestra al arreglo bidimensional del calendario del mes.*
Acepta:	*Un arreglo bidimensional de tamaño SEMANAS x DIAS.*
Regresa:	*Nada.*

A continuación se muestra la función completa:

```
//ESTA FUNCIÓN PRESENTARÁ EL ARREGLO DEL CALENDARIO MENSUAL
void muestraMes(int mes[SEMANAS][DIASI)
        {
                cout << "\n\n\t\tCALENDARIO PARA EL MES DE" <<  esteMes
                     << "\n\n\tDom\tLun\tMar\tMier\tJue\tVie\tSab" << endl;
                for (int semana = 0; semana < SEMANAS; ++semana)
                        {
                                cout << endl;
                                for (int dia = Dom; dia < Sab + 1; ++dia)
                                        cout << '\t' << mes[semana][dia];
                        } // FINAL DE CICLO SEMANA
        } // FINAL DE muestraMes()
```

De nuevo, *mes* es el parámetro formal definido en el encabezado de la función. La primera parte de la sección de enunciados de la función simplemente escribe la información del encabezado que se requiere para el calendario. Entonces, se ejecutan los ciclos anidados *for* para presentar el contenido del arreglo, Las estructuras básicas del ciclo son las mismas que se explicaron para la operación de llenado: Se incrementa el contador *dia* de *Dom* a *Sab* para cada iteración del ciclo *semana*. De esta manera, se presenta el contenido del arreglo de la forma fila a fila o semana a semana. Observe que se utiliza el enunciado *cout* para presentar los valores del elemento. El enunciado *cout* es el único dentro del ciclo interno *for*.

Ahora, se muestra el programa colocando todo junto:

```
/* Este programa: CALENDARIO.CPP, llenará y mostrará un arreglo de 2 dimensiones
    para un calendario mensual.
*/

#include <iostream.h>       //Para cin y cout

//Constantes globales y datos enumerados
const int SEMANAS = 6;
const int DIAS    = 7;
char esteMes[10];
enum diasSemana {Dom, Lun, Mar, Mier, Jue, Vie, Sab};
```

```
//Funciones prototipo
void llenaMes(int mayo[SEMANAS][DIAS]);
void muestraMes(int mayo[SEMANAS][DIAS]);

void main(void)
        {
                // Definición del arreglo
                int mayo[SEMANAS][DIAS];

                //Llamadas a las funciones
                llenaMes(mayo);
                muestraMes(mayo);
        } //Final de main()

/* Esta función llenará un arreglo de 2 dimensiones para un calendario mensual */
void llenaMes(int mes[SEMANAS][DIAS])
        {
                cout << "¿Qué mes quiere llenar y mostrar?: ";
                cin >> esteMes;
                cout    <<      "\nEscriba las fechas del mes, empezando con Domingo de la primera\n"
                                "semana en el mes. Si no existe fecha para un día determinado\n"
                                "escriba un 0. Presione la tecla ENTER después de cada entrada."
                        << endl << endl;

                for (int semana = 0; semana < SEMANAS; ++semana)
                        for (int dia = Dom; dia < Sab + 1; ++dia)
                                {
                                        cout    << "Escriba la fecha para la Semana "
                                                << semana << " día " << dia << ": ";
                                        cin >> mes[semana][dia];
                                }// Fin del ciclo día
        }// Fin de llenaMes()

//Esta función presentará el arreglo del calendario mensual
void muestraMes(int mes[SEMANAS][DIAS])
        {
                cout    << "\n\n\t\tCALENDARIO PARA El MES DE " << esteMes
                        << "\n\n\tDom\tLun\tMar\tMiér\tJue\tVie\tSáb";
                for (int semana = 0; semana < SEMANAS; ++semana)
                        {
                                cout << endl;
                                for (int dia = Dom; dia < Sab + 1; ++dia)
                                        cout << '\t' << mes[semana][dia];
                        }//Fin del ciclo semana
        } //Fin de muestraMes()
```

Primero, observe que se han enunciado en forma global ***SEMANAS***, ***DIAS***, ***esteMes*** y ***diasSemana*** para que todas las funciones tengan acceso a éstas. Observe que es corta la sección de enunciados ***main()***La función ***main()*** simplemente define el arreglo y llama las otras dos funciones. En cada llamada de función se emplea el argumento de arreglo real ***mayo***. Como se estableció al principio, también es posible definir otros arreglos mensuales para crear calendarios adicionales de mes. De hecho, es posible definir los ***12*** meses para crear un calendario anual. Para llenar o presentar un determinado mes, simplemente utilice este nombre de arreglo en la llamada de función respectiva.

Suponiendo que el usuario ejecuta este programa y escribe las fechas apropiadas para **Mayo**, el programa generará la siguiente presentación de calendario:

CALENDARIO PARA EL MES DE MAYO

Dom	Lun	Mar	Miér	Jue	Vie	Sáb
0	0	0	0	0	0	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	0	0	0	0	0

Desde luego, también es posible imprimir este calendario definiendo un objeto imprimir y utilizarlo para escribir el arreglo.

Ejemplo 19.13

Escriba un programa que utilice un arreglo para almacenar los nombres de todos los estudiantes de su grupo. Utilice una función para insertar dichos nombres dentro del arreglo y otra para presentar el contenido del arreglo una vez que se ha llenado. Suponga que no existen más que **25** caracteres en el nombre de cualquier estudiante y el tamaño máximo de la clase es 20 estudiantes.

Solución

Primero, se deberá definir un arreglo que contenga los nombres de los estudiantes en su clase. Los nombres de los estudiantes son cadenas, así que se deberá definir un arreglo de caracteres de dos dimensiones para cada nombre de cadena que ocupe una fila en el arreglo. Considere lo siguiente:

```
const int MAX_ESTUDIANTES = 20;
const int MAX_CARACTERES = 26;
char lista[MAX_ESTUDIANTES][MAX_CARACTERES];
```

Aquí, el nombre del arreglo es ***lista*** y se define como un arreglo de caracteres con *20* **filas** y *26* **columnas**. Esto permite cadenas para *20* nombres con un máximo de *25* caracteres por cadena. (Deberá proporcionarse una columna adicional para el carácter terminador nulo)

Después, se trabajará con la función del usuario que obtiene los nombres de los estudiantes. Se llamará a esta función ***tomaEstudiantes()*** A continuación se muestra la descripción de interfaz de función:

Función tomaEstudiantes():	*El usuario escribe los nombres de estudiantes y llena un arreglo de caracteres de dos dimensiones.*
Acepta:	***Un lugar reservado** para el número de estudiantes y un arreglo bidimensional de caracteres con un tamaño de MAX_ESTUDIANTES x MAX_CARACTERES.*
Regresa:	***El número de estudiantes** y un arreglo bidimensional de caracteres con tamaño de MAX_ESTUDIANTES x MAX_CARACTERES.*

Esta interfaz requiere pasar a la función el arreglo bidimensional ***lista*** y después regresarlo lleno con los nombres de los estudiantes. Desde luego, pasar un arreglo a una función es fácil, no importa su tamaño, ya que sólo se utiliza el nombre del arreglo como el argumento real de función. A continuación la función completa:

```
// ESTA FUNCIÓN TOMA LOS NOMBRES DE LOS ESTUDIANTES
// DEL USUARIO Y LOS ESCRIBE EN EL ARREGLO
void tomaEstudiantes(int &n, char estudiantes[MAX_ESTUDIANTES][MAX_CARACTERES])
        {
                cout << "Escriba el número de estudiantes:
                cin >> n;
                for (int fila = 0; fila < n; ++fila)
                        {
                                cout << "\nEscriba el nombre del estudiante " << fila + 1
                                     << ": ";
                                cin >> ws;
                                cin.getline(&estudiantes[fila][0], MAX_ESTUDIANTES);
                        } // FINAL DE for
        } // FINAL DE tomaEstudiantes()
```

De nuevo, se observa el empleo de un nombre diferente para el arreglo en el encabezado de función. Esto hace que la función sea genérica, ya que es posible utilizarla para llenar otros arreglos de otras clases. Esta función empieza pidiendo al usuario que escriba el número de estudiantes en la clase y después lee este número y lo asigna a un parámetro de referencia llamado ***n***. El valor escrito proporcionará un valor máximo para el contador de fila en el ciclo ***for*****.** Después se observa un único ciclo ***for*****.** ¿**Existe un problema aquí por estar llenando un arreglo bidimensional**? **No**, todo lo que se requiere es el único ciclo, ya que se está llenando un arreglo con cadenas, no caracteres individuales, Observe que el contador de ciclo incrementa únicamente el índice **fila**. El índice de **columna** se ***fija*** a ***[0]*** Esto colocará la primera cadena en el arreglo empezando en ***[0][0]***, la segunda cadena en ***[1][0]***, la tercera cadena en ***[2][0],*** y así sucesivamente. Recuerde que ***cin*** >> terminará cuando encuentre un carácter de espacio en blanco, por lo tanto usaremos ***cin.getline()*** Se utiliza un símbolo ampersand para decir a ***getline()*** que coloque la cadena empezando en la *dirección* asociada con ***estudiantes[fila][0]*** Será necesario decir al compilador que inserte la cadena empezando en la dirección especificada cuando lea las cadenas en el arreglo bidimensional. Por tanto, se inserta la primera cadena en el arreglo empezando en la dirección de memoria asociada con el índice***[0] [0]***, la segunda cadena empieza en la dirección de memoria asociada con el índice***[1][0]*** y así sucesivamente.

Ahora se escribirá una función llamada ***muestraEstudiantes()*** para presentar el arreglo de los nombres de los estudiantes. A continuación la descripción de interfaz para esta función:

Función muestraEstudiantes():	***Presenta*** *las cadenas de un arreglo bidimensional.*
Acepta:	***El número de estudiantes*** *y un arreglo de caracteres de dos dimensiones con un tamaño de* ***MAX_ESTUDIANTES*** × ***MAX_CARACTERES****.*
Regresa:	***Nada****.*

La lista de parámetros para esta función será idéntica a la función anterior, ya que ésta acepta la misma estructura del arreglo. A continuación la función completa:

```
//ESTA FUNCIÓN MUESTRA EL CONTENIDO DEL ARREGLO
void muestraEstudiantes(int n, char estudiantes[MAX_ESTUDIANTES][MAX_CARACTERES])
        {
                cout << "Los estudiantes escritos en el arreglo son: " << endl;
                for(int fila = 0; fila < n; ++fila)
                        cout << "\nPosición del arreglo[" << fila << "][0] "
                             << &estudiantes[fila][0];
        } // FINAL DE muestraEstudiantes()
```

De nuevo se observa el uso de un solo ciclo ***for*****,** ya que se necesita hacer referencia únicamente al principio de la dirección de cada cadena. Se requiere el símbolo ampersand antes del nombre del arreglo en el

enunciado ***cout*** para especificar la dirección de la cadena. Sin el **ampersand**, sólo se deberá ver el primer carácter de cada cadena. (**¿Por qué?**)

SUGERENCIA DE PROGRAMACIÓN

Múltiples cadenas se almacenan en arreglos de dos dimensiones fila a fila. Se deberá especificar el principio de dirección de la cadena, cuando se acceda a las cadenas que están almacenadas en un arreglo bidimensional. Esto se hace con el símbolo ampersand, &, antes del nombre del arreglo y se especifica la fila de la cadena respectiva y se ajusta la columna a cero, como ésta: &arregloCadena[fila][0]

Además de presentar los nombres de los estudiantes, se le da formato al enunciado ***cout*** para presentar la posición en el arreglo de cada cadena de nombre. A continuación se presenta una muestra de lo que se verá en la pantalla del monitor:

```
Posición del arreglo [0][0] Juan Pérez
Posición del arreglo [1][0] Alberto Ramírez
Posición del arreglo [2][0] Elías Cárdenas
Posición del arreglo [3][0] Silvestre Carranza
```

Por último, el programa completo es como sigue:

```
/* El siguiente programa: LISTAS.CPP, llenará un arreglo de caracteres de 2 dimensiones
   con cadenas escritas por el usuario y después mostrará el arreglo lleno.
*/

#include <iostream.h>       // PARA cin Y cout

//Constantes globales
const int MAX_ESTUDIANTES   = 20;
const int MAX_CARACTERES    = 26;

//Prototipos de funciones
void tomaEstudiantes(int &n, char estudiantes[MAX_ESTUDIANTES][MAX_CARACTERES]);
void muestraEstudiantes(int n, char estudiantes[MAX_ESTUDIANTES][MAX_CARACTERES]);

void main(void)
        {
                int numero = 0;
                char lista[MAX_ESTUDIANTES][MAX_CARACTERES];

                tomaEstudiantes(numero, lista);   // Llenar el arreglo
                muestraEstudiantes(numero, lista); // Muestra el arreglo
        } //FINAL DE main()

/* Esta función toma los nombres de los estudiantes escritos por el usuario y los
   escribe en el arreglo.
*/
void tomaEstudiantes(int &n, char estudiantes[MAX_ESTUDIANTES][MAX_CARACTERES])
        {
                cout << "Escriba el número de estudiantes: ";
                cin >> n;

                for(int fila = 0; fila < n; ++fila)
                        {
                                cout << "Escriba el nombre del estudiante " << (fila + 1) << ": ";
                                cin >> ws;
                                cin.getline(&estudiantes[fila][0], MAX_ESTUDIANTES);
                        } // FINAL DE for
        } // FINAL DE tomaEstudiantes()
```

```
//Esta función muestra el contenido del arreglo
void muestraEstudiantes(int n, char estudiantes[MAX_ESTUDIANTES][MAX_CARACTERES])
        {
                cout << "\nLos estudiantes escritos en el arreglo son:" << endl;
                for(int fila = 0; fila < n;++fila)
                        cout    << "\Posición del arreglo[" << fila << "][0]: " << &estudiantes[fila][0]
                                << endl;
        } // FINAL DE muestraEstudiantes()
```

EXAMEN BREVE 39

ARREGLOS DE MÁS DE DOS DIMENSIONES

Para algunas aplicaciones se requieren arreglos de más de dos dimensiones. En esta lección se consideran sólo arreglos de tres dimensiones, ya que sólo unas cuantas aplicaciones especializadas requieren arreglos de mayor dimensión. La forma más fácil de imaginar un arreglo de tres dimensiones es dibujar un cubo como el que se muestra en la ***figura 19.4***. Piense en un arreglo de tres dimensiones como la combinación de algunos arreglos de dos dimensiones para formar una tercera dimensión, profundidad. El cubo se hace de **filas** (dimensión vertical), **columnas** (dimensión horizontal) y **planos** (dimensión de profundidad) De esta manera, se localiza un elemento dado dentro de un arreglo de cubo especificando su **plano**, **fila** y **columna**. Verifique en la ***figura 19.4*** las posiciones del elemento especificado.

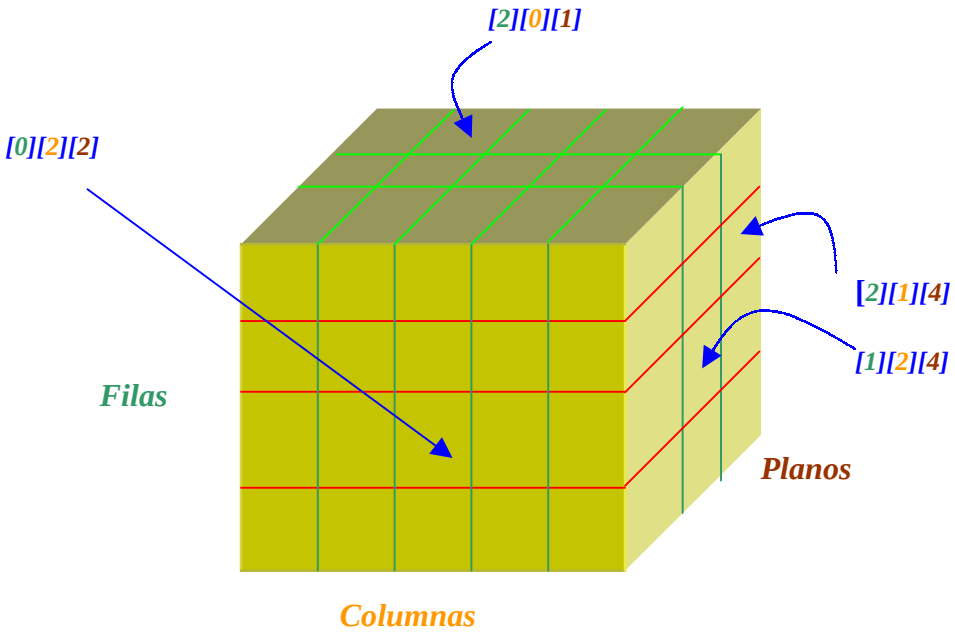


Figura 19.4*. Arreglo de tres dimensiones 3 × 4 ×5.*

Ahora, veamos un ejemplo práctico de un arreglo de tres dimensiones para que sea posible observar cómo se define y se acceda en ***C++***. Piense en estas lecciones como un arreglo en tres dimensiones, donde cada página de las lecciones es un arreglo bidimensional compuesto de filas y columnas. Entonces las páginas combinadas forman los planos dentro de un arreglo de tres

dimensiones que conforman las lecciones. Después suponga que existen *45* líneas en cada página que forman las **filas** para el arreglo y *80* caracteres por **fila** que forman las **columnas** del arreglo. Si existen *750* páginas en las lecciones, existen *750* **planos** en el arreglo. De esta manera, este arreglo de las lecciones es un arreglo de $45 \times 80 \times 750$. **¿Cuáles son los elementos del arreglo y cuántos hay**? Bueno, los elementos del arreglo serán caracteres, ya que éstos forman las palabras en una página. Además existirán $45 \times 80 \times 750 = 2{,}700{,}000$ caracteres, incluyendo espacios en blanco, porque éste es el tamaño de las lecciones en términos de **filas**, **columnas** y **páginas**.

¿Cómo se definirá el *arreglo lecciones* en *C++*? **¿Qué opina de esta forma**?

```
const int PAGINAS      = 750;
const int FILAS        = 45;
const int COLUMNAS     = 80;
char lecciones[PAGINAS][FILASI[COLUMNASI;
```

Es posible comprender esta definición a partir de su trabajo con arreglos de una y dos dimensiones. Existen tres dimensiones *[PAGINAS], [FILAS] y [COLUMNAS] que* definen el tamaño del arreglo *lecciones*. Un arreglo de tres dimensiones en *C* y *C++* se *ordena con el plano principal*. Esto es porque primero se especifica el tamaño del plano, seguido por el tamaño del renglón, seguido por el tamaño de columna. La clase de datos del arreglo es *char*, ya que los elementos son caracteres.

Después, **¿cómo supone que se acceda a la información del** *arreglo lecciones*? La forma más fácil es utilizar ciclos anidados. ¿Cómo deberán anidarse los ciclos? Ya que el arreglo está ordenado por el **plano** principal, el ciclo **página** deberá ser el ciclo externo, y el ciclo **columna**, el ciclo interno. Esto deja que el ciclo **fila** se inserte entre los ciclos **página** y **columna**. Traduciendo esto al *arreglo lecciones*, se obtiene:

```
for (int pagina = 0; pagina < PAGINAS; ++pagina)
        for (int fila = 0; fila < FILAS; ++fila)
                for (int columna = 0; columna < COLUMNAS; ++columna)
                        <proceso lecciones[pagina][fila][columna]>
```

Con este enfoque de anidamiento, se ejecuta *80* veces el ciclo *columna* por cada iteración del ciclo *fila*, que se ejecuta *45* veces por cada iteración del ciclo *pagina*. Desde luego, el ciclo *pagina* se ejecuta *750* veces. Esta estructura *for* procesará elementos una fila a la vez para una página dada. Observe la utilización de las variables *pagina*, *fila*, *y columna* como los contadores de ciclo. Aquéllas deberán ser diferentes de las constantes *(PAGINAS, FILAS, COLUMNAS)* que se utilizan para definir el arreglo, ya que éstas son locales para los ciclos *for*.

PRECAUCIÓN

Cuando se define un arreglo, C++ reserva suficiente memoria primaria para almacenarlo. Ésta es exclusiva para el arreglo definido y no es posible utilizarlo para otras tareas de programación o del sistema. En otras palabras, un arreglo grande ***consume*** *mucha memoria. Por ejemplo, el arreglo lección, contiene* ***2,700,000*** *caracteres elemento. Cada uno de éstos requiere un byte de memoria para almacenarse, por lo tanto C++ asignará casi* ***2,637 Kbytes*** *de memoria de usuario para el arreglo lecciones. Esto es mucho más de lo que está disponible en algunos sistemas PC y podrá enviar un mensaje de error* ***array size too big*** *durante la compilación. Por lo tanto, cerciórese de que sus arreglos no sean demasiado grandes para almacenarlos en su sistema. Existen otras formas más eficientes del uso de la memoria para almacenar grandes cantidades de datos -una lista vinculada dinámica.*

Ejemplo 19.14

Dada la definición del *arreglo* ***lecciones*** anterior:

a. ***Escriba*** *un segmento de programa que pueda utilizarse para* ***llenar el arreglo lecciones****.*
b. ***Escriba*** *un segmento de programa que pueda utilizarse para* ***imprimir el arreglo lecciones****.*
c. ***Escriba*** *un segmento de programa que pueda utilizarse para* ***imprimir la página dos del arreglo lecciones****.*

.

Solución

a. *Utilizando los tres ciclos anidados anteriores, se podrá llenar el arreglo lecciones de esta manera:*

```
for (int pagina = 0; pagina < PAGINAS; ++pagina)
        for (int fila = 0; fila < FILAS; ++fila)
                for (int columna = 0; columna < COLUMNAS; ++columna)
                        cin.get(lecciones[pagina][fila][columna]);
```

El ciclo interno emplea una función ***get()*** *para leer un carácter a la vez y lo coloca en la posición indexada. El operador de extracción* ***cin*** *(>>) no funciona aquí, ya que >> ignora los espacios en blanco, que obviamente forman parte del material de las lecciones.*

b. *Se emplea un objeto imprimir en el ciclo interno para imprimir las lecciones:*

```
for (int pagina = 0; pagina < PAGINAS; ++pagina)
    for (int fila = 0; fila < FILAS; ++fila)
            {
                    imprimir << endl;
                    for (int columna = 0; columna < COLUMNAS; ++columna)
                            imprimir << lecciones[pagina][fila][columna];
            } / /Fin del  for MEDIO
```

Observe el único enunciado ***imprimir*** utilizado en el ciclo intermedio para proporcionar un ***CRLF*** después de que se ha impreso una fila determinada. Desde luego, se deberá definir el objeto ***imprimir*** como ya antes se explicó.

c. *Solamente se necesitan dos ciclos para imprimir cierto número de páginas, como se muestra a continuación:*

```
for (int fila = 0; fila < FILAS; ++fila)
    {
            imprimir << endl;
            for (int columna = 0; columna < COLUMNAS; ++columna)
                    imprimir << lecciones[1][fila][columna];
    } // Fin del for externo
```

Observe que el índice pagina se ajusta a ***[1]*** *dentro del enunciado imprimir para imprimir la* ***página 2*** *de las lecciones. (Recuerde que la primera página de las lecciones realmente está en el índice* ***[0]*** *de la página)* ***¿Cómo se podrá modificar este segmento para imprimir cualquier página que desee el usuario? ¡Piénselo!*** *Esto se deja como ejercicio.*

EXAMEN BREVE 40

SOLUCIÓN DE PROBLEMAS EN ACCIÓN: Solución de ecuaciones simultáneas

Problemas

Recuerde su clase de álgebra en la que una serie de ecuaciones simultáneas existe cuando se tienen dos o más ecuaciones con dos o más incógnitas. Por ejemplo, considere lo siguiente:

$$7x - 5y = 20$$
$$-5x + 8y = -10$$

Aquí se tienen dos ecuaciones y dos incógnitas. Para resolver las ecuaciones, se deberá encontrar x y y. Esto es imposible utilizando sólo una de las ecuaciones, pero no representa un problema cuando ambas ecuaciones se solucionan *en forma simultánea* o juntas. Algo que debe recordar de la clase de álgebra es que para resolver ecuaciones simultáneas, deberá haber al menos tantas ecuaciones como incógnitas existan. Esto significa que no es posible resolver dos incógnitas utilizando una sola ecuación. Sin embargo, dos incógnitas se pueden resolver utilizando dos o más ecuaciones.

Determinantes

Una forma común para resolver ecuaciones simultáneas es mediante *determinantes*. Recuerde del álgebra, que un *determinante* es simplemente un *arreglo cuadrado*. Por un *arreglo cuadrado*, se entiende un arreglo que tiene el mismo número de *filas* y *columnas*. Aquí está uno sencillo de 2×2, llamado un *determinante* de *orden 2:*

$$\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix}$$

Los elementos son A_1, B_1, A_2 y B_2 (*Nota:* Estos elementos serán valores numéricos cuando realmente se utilizan *determinantes* para resolver ecuaciones simultáneas. Observe las *barras verticales* a los lados izquierdo y derecho del arreglo. Estas barras se utilizan para indicar que el arreglo es un *determinante*. Este *determinante* se dice que es de *orden dos*, porque tiene dos *filas* y dos *columnas*. Existen también otros de *orden tres*, *orden cuatro*, *orden cinco* y así sucesivamente. En cada caso, el *determinante* es un *arreglo cuadrado*. A continuación se muestra un *determinante* de *orden tres*:

$$\begin{vmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{vmatrix}$$

Desarrollo de un determinante

Se dice que se desarrolla un *determinante* cuando se reemplaza el arreglo con un solo valor. El desarrollo de un *determinante* de *orden 2 es* bastante simple. Para tener una idea, observe lo siguiente:

$$\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix} = A_1 B_2 - A_2 B_1$$

Imagine las dos diagonales en el *determinante*. Existe una diagonal de la parte superior izquierda a la parte inferior derecha. Se llamará a ésta la *diagonal descendente*, que forma el producto A_1 y B_2 La segunda diagonal va de la parte inferior izquierda a la superior derecha. Se llamará a ésta la *diagonal ascendente*, que forma el producto A_2 B_1 En el desarrollo del lado derecho del signo igual, se observa que el producto de la *diagonal ascendente* se resta del producto de la *descendente*.

Ejemplo 19.15

Desarrolle los siguientes ***determinantes***:

a. $\begin{vmatrix} 20 & -5 \\ -10 & 8 \end{vmatrix}$

b. $\begin{vmatrix} 7 & 20 \\ -5 & -10 \end{vmatrix}$

c. . $\begin{vmatrix} 7 & 5 \\ -5 & 8 \end{vmatrix}$

Solución

a. $\begin{vmatrix} 20 & -5 \\ -10 & 8 \end{vmatrix} = (20)(8) - (-10)(-5) = 160 - 50 = 110$

b. $\begin{vmatrix} 7 & 20 \\ -5 & -10 \end{vmatrix} = (7)(-10) - (-5)(20) = -70 - (-100) = -70 + 100 = 30$

c. . $\begin{vmatrix} 7 & 5 \\ -5 & 8 \end{vmatrix} = (7)(8) - (-5)(5) = 56 - (-25) = 56 * 25 = 81$

El desarrollo de un ***determinante*** de ***orden 3*** es un poco más complejo. A continuación se muestra nuevamente un ***determinante*** general de ***orden 3***:

$$\begin{vmatrix} A_1 & B_1 & C_1. \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{vmatrix}$$

Para desarrollar en forma manual este ***determinante***, se deberán rescribir las primeras dos ***columnas*** a la derecha del ***determinante***, y después usar el método de ***diagonales***, como sigue:

$$\begin{vmatrix} A_1 & B_1 & C_1 & A_1 & B_1 \\ A_2 & B_2 & C_2 & A_2 & B_2 \\ A_3 & B_3 & C_3 & A_3 & B_3 \end{vmatrix} = A_1 B_2 C_3 + B_1 C_2 A_3 + C_1 A_2 B_3 - A_3 B_2 C_1 - B_3 C_2 A_1 - C_3 A_2 B_1$$

Aquí, se crean **tres** ***diagonales descendentes*** que forman los productos $A_1 B_2 C_3$, $B_1 C_2 A_3$, $C_1 A_2 B_3$ y **tres** ***diagonales ascendentes*** que forman los productos $A_3 B_2 C_1$, $B_3 C_2 A_1$, $C_3 A_2 B_1$ .Los tres productos de ***diagonales ascendentes*** se restan de la suma de los tres productos de ***diagonales descendentes***. Esto se conoce como el ***método de diagonales*** para desarrollar ***determinantes*** de ***orden 3***. Sin embargo, cabe aquí una advertencia: El ***método de diagonales***

no funciona para ***determinantes*** de ***orden*** mayor a ***3***. En estos casos deberá usarse el ***método de cofactores***, el cual es un proceso recursivo y se dejará como ejercicio investigar este método al final de la lección.

Ejemplo 19.16

Desarrolle el siguiente ***determinante*** de ***orden 3***:

$$\begin{vmatrix} 6 & -2 & -4 \\ 15 & -2 & -5 \\ -4 & -5 & 12 \end{vmatrix}$$

Solución

Rescribiendo las primeras dos ***columnas*** a la derecha de ***determinante***, se obtiene:

$$\begin{array}{ccccc} 6 & -2 & -4 & 6 & -2 \\ 15 & -2 & -5 & 15 & -2 \\ -4 & -5 & 12 & -4 & -5 \end{array}$$

Ahora, la multiplicación de los elementos diagonales y la suma y resta de los productos diagonales, da como resultado:

$$+(6)(-2)(12) + (-2)(-5)(-4) + (-4)(15)(-5) -(-4)(-2)(-4) - (-5)(-5)(6) - (12)(15)(-2)$$

Por último, realizando la aritmética requerida se obtiene:

$$\begin{aligned} &+(-144) + (-40) + (300) \\ &-(-32) - (150) - (-360) \\ &= -144 - 40 + 300 + 32 - 150 + 360 \\ &= 358 \end{aligned}$$

Como se puede ver en el ejemplo anterior, el desarrollo de un ***determinante*** de ***orden 3*** puede tener algo de dificultad. Se debe prestar especial atención a los signos. Un solo error de signo durante su aritmética originará un desarrollo incorrecto. ¿**No sería agradable escribir un programa de computadora para realizar el desarrollo**? Ésta es una aplicación ideal para una función ***C++***, ya que la operación a desarrollar regresa un solo valor.

Función para el desarrollo de un determinante de orden 2

Escribamos una función ***C++*** para desarrollar un ***determinante*** de ***orden 2***. Se supone que los elementos en el ***determinante*** se almacenan en un arreglo de 2×2. Se deberá pasar este arreglo a la función, y después ésta evaluará al arreglo y regresará un solo valor como resultado. A continuación la descripción de interfaz de función:

Función desarrollo():	*Desarrolla un determinante de orden 2.*
Acepta:	***Un arreglo de 2 × 2.***
Regresa:	***El valor resultante.***

Con esta descripción de interfaz, el prototipo de función se convierte en:

```
float desarrollo(float determinante[2][2]);
```

El nombre de la función es ***desarrollo()*** El parámetro formal es ***determinante[2][2]***, dado que éste es el tamaño del arreglo que se desarrollará. La clase de datos del resultado que se regresa es ***float***. Ahora, el arreglo ***determinante*** tiene índices de ***filas***, con intervalo de ***[0]*** a ***[1]*** e índices de ***columna***, con intervalo de ***[0]*** a ***[1]*** A continuación podrá analizar el arreglo que muestra el acomodo del índice de ***filas*** y ***columnas***:

```
[0][0]   [0][1]
[1][0]   [1][1]
```

Recuerde que éstos son únicamente los índices del arreglo y no los elementos almacenados en él. Recuerde que para desarrollar el ***determinante***, debe restarse la ***diagonal ascendente*** de la ***descendente***. De esta manera, el producto de los índices ***[1][0]*** y ***[0] [1]*** se resta del producto de ***[0] [0] y [1][1]*** Utilizando esta idea en la función, se obtiene un sencillo enunciado ***return*****,** como sigue:

```
return determinante[0][0] * determinante[1][1] - determinante[1][0] * determinante[0][1]
```

¡Eso es todo! A continuación se muestra la función completa:

```
//ESTA FUNCIÓN DESARROLLA UN DETERMINANTE 2 × 2
float desarrollo(float determinante[2][2])
        {
                return determinante[0][0] * determinante[1][1]
                -determinante[1][0] * determinante[0][1];
        } // FINAL DE desarrollo()
```

La escritura de una función para desarrollar un ***determinante*** de ***orden 3*** se deja como ejercicio.

Regla de Cramer

La regla de **Cramer** permite resolver ecuaciones simultáneas con ***determinantes***. Empecemos con dos ecuaciones y dos incógnitas.

Se dice que **una ecuación** está *en* ***forma estándar*** cuando todas las variables están del lado izquierdo del signo igual y el término constante aparece del lado derecho de éste.

A continuación una ecuación general que contiene dos variables en forma estándar:

$$Ax + By = C$$

Esta ecuación tiene dos variables, ***x*** y ***y***. El coeficiente de ***x*** es ***A*** y el coeficiente de ***y*** es ***B***. El término constante es ***C***.

La siguiente es una serie de **dos ecuaciones generales simultáneas** en ***forma estándar***:

$$A_1 x + B_1 y = C_1$$
$$A_2 x + B_2 y = C_2$$

Las variables comunes entre las dos ecuaciones son *x* y *y*. Los subíndices ***1*** y ***2*** denotan los coeficientes y constantes de las ecuaciones ***1*** y ***2***, respectivamente. La regla de **Cramer** permite solucionar a *x* y *y* con ***determinantes***, de la siguiente manera:

$$x = \frac{\begin{vmatrix} C_1 & B_1 \\ C_2 & B_2 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} A_1 & C_1 \\ A_2 & C_2 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix}}$$

Como se ve, los determinantes se forman utilizando los coeficientes y constantes de las dos ecuaciones. ¿**Descubre un patrón**? Primero, observe que los determinantes del denominador son idénticos y se forman utilizando los coeficientes *x* y *y* directamente de las dos ecuaciones. Sin embargo, los determinantes del numerador son diferentes. Cuando se resuelve *x*, el determinante del numerador se forma por el reemplazo de los coeficientes de *x* con los términos constantes. Cuando se resuelve *y*, el determinante del numerador se forma reemplazando los coeficientes de *y* con los términos constantes.

Ejemplo 19.17

Resuelva la siguiente serie de **ecuaciones simultáneas** utilizando la regla de **Cramer**:

$$x + 2y = 3$$
$$3x + 4y = 5$$

Solución

Al formar los determinantes requeridos, se obtiene:

$$x = \frac{\begin{vmatrix} 3 & 2 \\ 5 & 4 \end{vmatrix}}{\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} 1 & 3 \\ 3 & 5 \end{vmatrix}}{\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}}$$

Desarrollando los determinantes y dividiendo se obtiene *x* y *y*:

$$x = \frac{(3)(4) - (5)(2)}{(1)(4) - (3)(2)} = \frac{2}{-2} = -1$$

$$y = \frac{(1)(5) - (3)(3)}{(1)(4) - (3)(2)} = \frac{-4}{-2} = 2$$

Instrumentación de la regla de Cramer en C++

Piense en las funciones que tuvo que realizar utilizando la regla de **Cramer** para resolver la serie de ecuaciones simultáneas anteriores. Existen tres tareas principales que se deben realizar:

- *Tarea 1: **Obtener** los **coeficientes** y **constantes** de las ecuaciones.*
- *Tarea 2: **Formar** los **determinantes**, **numerador** y **denominador**.*
- *Tarea 3: **Desarrollar** los **determinantes**.*

Ya se tiene desarrollada una función para realizar la ***tarea 3***. Ahora se deberán desarrollar las funciones ***C++*** para realizar las primeras dos tareas.

Para realizar la ***tarea 1***, se deberá escribir una función que obtenga los **coeficientes** y **constantes** de las ecuaciones que se van a resolver. Existen dos ecuaciones y tres elementos (dos **coeficientes** y una **constante**) que deberá obtener para cada ecuación.

¿**Esto sugiere alguna estructura de datos en particular**? Desde luego, **¡un arreglo de *2* × *3*!** Así, se escribe una función para llenar un arreglo de ***2 × 3*** de **coeficientes** y términos **constante** de las dos ecuaciones que escribe el usuario. Aquí están:

```
//ESTA FUNCIÓN LLENARA UN ARREGLO CON LA ECUACIÓN DE COEFICIENTES
void llena(float ecuaciones[2][3])
        {
                for(int fila = 0; fila < 2; ++fila)
                        {
                                cout << "Escriba los coeficientes de variable y constantes"
                                     " para la ecuación " << fila + 1 <<
                                     "\nObserve que la ecuación deberá estar en forma"
                                     " estándar" << endl << endl;

                                for(int col = 0; col < 3; ++col)
                                        {
                                                if(col == 2)
                                                        cout << "Escriba el término constante"
                                                else
                                                        cout << "Escriba el coeficiente para la"
                                                             " variable" << col + 1 << ": ";
                                                cin >> ecuaciones[fila][col];
                                        } // Fin del ciclo col
                        } // Fin del ciclo fila
        } // Fin de llena()
```

Una función como ésta no deberá ser novedosa, dado que simplemente emplea ciclos anidados ***for*** para llenar un arreglo. El nombre de la función es ***llena()*** Esta función llena un arreglo llamado ***ecuaciones*** que se deberá definir como un arreglo de ***2 × 3*** en el programa llamador. El arreglo pasará a la función por referencia usando el nombre del arreglo en la llamada de la función. Una vez que la función llena el arreglo, ésta regresa al programa llamador.

Dadas las siguientes **ecuaciones generales** en **forma estándar**:

$$A_1x + B_1y = C_1$$
$$A_2x + B_2y = C_2$$

La función ***llena()*** llenará el arreglo ***2 × 3***, como sigue:

$$\begin{matrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \end{matrix}$$

Como se puede ver, los primeros coeficientes y el término constante de la ecuación se insertan en la primera fila del arreglo. La segunda fila del arreglo almacena los coeficientes y el término constante de la segunda ecuación. Desde luego, la función supone que el usuario escribirá los coeficientes y constantes en el orden apropiado.

Para llevar a cabo la ***tarea 2***, se desarrollará una función para formar los determinantes a partir de los coeficientes y constantes de la ecuación **¿Cómo se obtienen los coeficientes y constantes**? Correcto: **¡del arreglo *2 × 3* que se acaba de llenar!** Ahora, **¿cuántos determinantes únicos requiere la regla de Cramer para resolver dos ecuaciones y dos incógnitas**? **Tres**: Un determinante numerador para la incógnita ***x***, otro más para la incógnita ***y***, y un determinante denominador para ambas incógnitas ***x*** y ***y***. Todos los determinantes deben ser de **orden *2***, **¿correcto**?

Por lo tanto, la función deberá obtener el único arreglo ***2 × 3*** que se llenó con los coeficientes y constantes en la función ***llena()*** y genera tres arreglos ***2 × 2*** que formarán los determinantes de dos numeradores y un denominador que requiere la regla de **Cramer**. A continuación la función:

```
// Esta función formará los determinantes
void formaDet(float ecuaciones[2][3], float x[2][2], float y[2][2], float d[2][2])
        {
                for(int fila = 0; fila < 2; ++fila)
                        for(int col = 0; col < 2; ++col)
                                {
                                        x[fila][col] = ecuaciones[fila][col];
                                        y[fila][col] = ecuaciones[fila][col];
                                        d[fila][col] = ecuaciones[fila][col];
                                } // Fin del ciclo col
                        x[0][0] = ecuaciones[0][2];
                        x[1][0] = ecuaciones[1][2];
                        y[0][1] = ecuaciones[0][2];
                        y[1][1] = ecuaciones[1][2];
        } // Fin de formaDet()
```

Primero, observe el encabezado de la función. El nombre de la función es ***formaDet()*** Esta función requiere *4* parámetros: ***ecuaciones[2][3]***, ***x[2][2]***, ***y[2][2]*** y ***d[2][2]***. ***ecuaciones[2][3]*** es el arreglo ***2 × 3*** que contiene los coeficientes y constantes de la función ***llena()***. El parámetro ***x[2][2]*** representa el determinante del numerador para la incógnita ***x***. El parámetro ***y[2][2]*** representa el determinante del numerador para la incógnita ***y***. El parámetro ***d[2][2]*** representa al determinante del denominador para ambas incógnitas ***x*** y ***y***.

Ahora observe en la sección de enunciados de la función. Las primeras dos columnas del arreglo de la ecuación se copian en cada uno de los arreglos determinantes utilizando ciclos ***for***. Después, con el uso de asignación directa se insertan los términos constantes en los arreglos determinantes ***x*** y ***y*** en las posiciones requeridas. El patrón de formación resulta de la regla de **Cramer**. Observe que en todos los ejemplos, los determinantes se forman usando los elementos del arreglo ***2 × 3*** de ***ecuaciones*** de los coeficientes y constantes que se generan con la función ***llena()***

Ahora se tienen todos los ingredientes para un programa ***C++*** que solucionará dos ecuaciones con dos incógnitas usando la regla de **Cramer**. Combinando las funciones ***llena()***, ***formaDet()*** *y* ***desarrollo()*** en un solo programa, se obtiene lo siguiente:

```
/* El siguiente programa: CRAMER.CPP, solucionará dos ecuaciones con dos incógnitas
   utilizando la regla de Cramer.
*/

#include <iostream.h>              //Para cout y cin

// Prototipo de las funciones.
void llena(float ecuaciones[2][3]);
void formaDet(float ecuaciones[2][3], float x[2][2], float y[2][2], float d[2][2]);
float desarrollo(float determinante[2][2]);

void main(void)
        {
                // Definición de arreglos
                float ecuaciones[2][3];
                float x[2][2];
                float y[2][2];
                float d[2][2];

                // Llamada de funciones para llenar el arreglo de la ecuación y formación
                // de determinantes.
                llena(ecuaciones);
                formaDet(ecuaciones, x, y, d);

                // Si el denominador = 0, escribe un mensaje de error. Si no calcula x y y.
                if(!desarrollo(d))
                        cout << "\nDenominador = 0. Ecuaciones sin solución." << endl;
                else
                        cout << "\nEl valor de la primera variable es: "
                             << desarrollo(x) / desarrollo(d)
                             << "\n\nEl valor de la segunda variable es: "
                             << desarrollo(y) / desarrollo(d) << endl;
        } // Fin de main()

// Esta función llenará un arreglo con la ecuación de coeficientes.
void llena(float ecuaciones[2][3])
        {
                for(int fila = 0; fila < 2; ++fila)
                        {
                                cout << "\nEscriba los coeficientes de las variables y las constantes para la ecuación "
                                     << (fila + 1) << "\nObserve que la ecuación deberá estar en forma estándar."
                                     << endl << endl;

                                for(int col = 0; col < 3; ++col)
                                        {
                                                if(col == 2)
                                                        cout << "Escriba el coeficiente constante: ";
                                                else
                                                        cout << "Escriba el coeficiente para la variable "
                                                             << (col + 1) << ": ";
                                                cin >> ecuaciones[fila][col];
                                        } // Final del ciclo col
                        } // fin del ciclo fila
        } // Fin de llena()
```

```
// Esta función formará los determinantes
void formaDet(float ecuaciones[2][3], float x[2][2], float y[2][2], float d[2][2])
        {
                for(int fila = 0; fila < 2; ++fila)
                        for(int col = 0; col < 2; ++col)
                                {
                                        x[fila][col] = ecuaciones[fila][col];
                                        y[fila][col] = ecuaciones[fila][col];
                                        d[fila][col] = ecuaciones[fila][col];
                                } // Fin del ciclo col

                x[0][0] = ecuaciones[0][2];
                x[1][0] = ecuaciones[1][2];
                y[0][1] = ecuaciones[0][2];
                y[1][1] = ecuaciones[1][2];
        } // Fin de formaDet()

// Esta función desarrollará un determinante de orden 2 × 2.
float desarrollo(float determinante[2][2])
        {
                return determinante[0][0] * determinante[1][1]
                        - determinante[1][0] * determinante[0][1];
        } // Final de desarrollo
```

Observe que todos los arreglos se definen en forma local para ***main()***. ***ecuaciones*** es un arreglo ***2 × 3*** que almacenará los coeficientes y constantes de las dos ecuaciones. Esto se infiere de las definiciones para los tres arreglos determinantes ***2 × 2***. Los prototipos para las tres funciones se dan antes que ***main()*** y tales funciones se listan después de ***main()***

Ahora, observe la sección de enunciación de ***main()***, se llama primero a la función ***llena()*** para tener los coeficientes y términos constantes de las dos ecuaciones. El argumento real que se usa para la llamada de función es el nombre del arreglo de ecuaciones, ***ecuaciones***. Después, se llama a la función ***formaDet()*** para formar los determinantes requeridos. Los argumentos reales que se utilizan en esta llamada de función son ***ecuaciones***, ***x***, ***y*** y ***d***. Se requiere el argumento ***ecuaciones*** para pasar el arreglo ***2 × 3*** a la función. Se necesitan los argumentos ***x***, ***y*** y ***d*** para pasar los tres arreglos determinantes a la función.

Por último, observe cómo se invoca a la función ***desarrollo()*** Esto es lo primero que se invoca como parte de un enunciado ***if/else*** para ver si el valor del determinante denominador es ***0***. Si así es, no se pueden resolver las ecuaciones utilizando la regla de **Cramer** porque la división entre ***0*** es indefinida. Si el determinante del denominador no es ***0***, se invoca la función ***desarrollo()*** dos veces para calcular la primera incógnita **(*x*)** dentro de un enunciado ***cout*** como este: ***desarrollo(x) / desarrollo(d)*** Esto desarrolla el determinante para ***x***, desarrolla los determinantes comunes del denominador y divide los dos para obtener el valor de la primera incógnita **(*x*)** De nuevo se llama a la función dos veces para encontrar la segunda incógnita **(*y*)**

Dadas las siguientes dos ecuaciones:

$$x + 2y = 3$$
$$3x + 4y = 5$$

a continuación se muestra lo que **se verá al ejecutar el programa**:

Escriba los coeficientes de las variables y las constantes para la ecuación 1
Observe que la ecuación deberá estar en forma estándar.

Escriba el coeficiente para la variable 1: 1↵
Escriba el coeficiente para la variable 2: 2↵
Escriba el coeficiente constante: 3↵

Escriba los coeficientes de las variables y las constantes para la ecuación 2
Observe que la ecuación deberá estar en forma estándar.

Escriba el coeficiente para la variable 1: 3↵
Escriba el coeficiente para la variable 2: 4↵
Escriba el coeficiente constante: 5↵

El valor para la primera variable es: -1
El valor para la segunda variable es: 2

¿Piensa que es posible desarrollar un programa similar para resolver una serie de tres ecuaciones simultáneas? ¡Ahora tiene todos los conocimientos necesarios!

PENSANDO EN OBJETOS: *Identificación de los comportamientos de una clase*

En lecciones anteriores ***pensando en objetos*** realizamos las primeras dos fases de un diseño orientado a objetos para nuestro simulador del elevador: la identificación de las clases necesarias para implementar el simulador y la identificación de los atributos de dichas clases.

Ahora nos concentraremos en determinar los comportamientos de las clases que se necesitan para implementar el simulador del elevador. En la próxima lección nos dedicaremos a las iteraciones entre los objetos de estas clases.

Consideremos los comportamientos de algunos objetos reales. Los comportamientos de un radio incluyen sintonizar una estación y ajustar el volumen. Los comportamientos de un automóvil abarcan la aceleración (al presionar el pedal del acelerador) y la desaceleración (al presionar el pedal de freno)

Como veremos, por lo general los objetos no realizan sus comportamientos de manera espontánea. En cambio, normalmente se llama a un comportamiento específico cuando se le envía un ***mensaje*** al objeto, solicitándole que lo realice. Esto suena como una llamada de función; precisamente es la manera en que se envían mensajes a los objetos de ***C++***.

Tarea de laboratorio del elevador 3

1. Continúe trabajando con el archivo de hechos que elaboró en lección anterior. Dividió en dos grupos los hechos relacionados con cada clase. Etiquetó al primer grupo como ***Atributos*** y al segundo como ***Otros hechos***.

2. A cada clase agregue un tercer grupo llamado ***comportamientos***. Ponga en este grupo todos los comportamientos de una clase que pueda ser llamada diciéndole a un objeto de dicha clase que haga algo, es decir, enviándole un mensaje al objeto. Por ejemplo, un botón puede ser oprimido (por alguna persona), así que liste ***oprimirBoton*** como comportamiento de la clase botón. La función ***oprimirBoton*** y los demás comportamientos de la clase botón se llaman ***funciones miembro (o métodos)*** de la clase botón. Los

atributos de la clase (como que el botón esté ***encendido*** o ***apagado***) se llaman ***datos miembro*** de la clase botón. Las funciones miembro de una clase por lo general manipulan los datos miembro de la clase (como que ***oprimirBoton*** cambie uno de los atributos del botón a ***encendido***) Las funciones miembro comúnmente también envían mensajes a los objetos de otras clases (como que un objeto botón envíe un mensaje ***venPorMi*** para llamar al elevador) Suponga que el elevador tendrá un botón que se ilumine al ser oprimido. Cuando llega al piso, el elevador deberá enviar un mensaje ***restablecerBoton*** para apagar la luz del botón. El elevador necesitará determinar si se ha oprimido un botón particular, así que podemos dar otro comportamiento llamado ***obtenerBoton***, que simplemente examina cierto botón y devuelve ***1*** o ***0***, indicando si actualmente está ***encendido*** o ***apagado***. Probablemente esperará que la puerta del elevador responda a los mensajes ***abrirPuerta*** y ***cerrarPuerta***, etcétera.

3. Por cada comportamiento que asigne a una clase, dé una breve descripción de lo que deberá hacer. Liste cualquier cambio de atributos que provoque el comportamiento, así como todos los mensajes que dicho comportamiento les envía a los objetos de otras clases.

Notas

1. Comience por **listar** los comportamientos de clase que se mencionan explícitamente en el planteamiento del problema. Después liste los comportamientos implicados directamente en el planteamiento.

2. **Agregue** comportamientos pertinentes a medida que sea evidente que son necesarios.

3. Nuevamente, el diseño de sistemas no es un proceso perfecto ni completo, así que haga lo mejor que pueda por ahora y prepárese para **modificar** su diseño a medida que siga desarrollando el ejercicio en las siguientes lecciones.

4. Como veremos, es difícil recopilar todos los comportamientos posibles en esta etapa del diseño. Probablemente **agregará** comportamientos a sus clases cuando prosiga este ejercicio en lecciones posteriores.

TEMAS ESPECIALES

ERRORES COMUNES DE PROGRAMACIÓN

1. Es importante notar la diferencia entre el ***séptimo elemento del arreglo*** y el ***elemento siete del arreglo***. Debido a que los índices inician en **0**, el ***séptimo elemento del arreglo*** tiene un índice de **6**, mientras que el ***elemento siete del arreglo*** tiene un índice de ***7*** y, de hecho, es su octavo elemento. Desgraciadamente ésta es una fuente de errores ***por diferencia de uno***.

2. **Olvidar inicializar los elementos** de un arreglo que deberían ser inicializados es un **error de lógica**.

3. Si en una lista de iniciación de un arreglo **indica más inicializadores** que el número de elementos que hay en dicho arreglo, sucederá un **error de sintaxis**.

4. La **asignación de un valor a una variable constante** en una instrucción ejecutable es un **error de sintaxis**.

5. **Sólo se pueden utilizar constantes** para declarar arreglos automáticos y estáticos. No emplear una constante con este fin es un **error de sintaxis**.

6. **Hacer referencia a un elemento que está fuera de los límites de un arreglo** es un **error de lógica** en tiempo de ejecución. **No** es un **error de sintaxis**.

7. Aunque es posible **utilizar la misma variable de contador** en un ciclo ***for*** y en otro ciclo ***for*** anidado dentro del primero, lo normal es que esto sea un **error de lógica**.

8. Si no le proporciona ***cin >> un arreglo de caracteres*** lo bastante grande como para guardar una cadena introducida desde el teclado, puede dar como resultado una **pérdida de información** en el programa y otros **errores graves** en tiempo de ejecución.

9. Suponer que los **elementos de un arreglo** ***static*** local se inicializan a cero cada vez que se llama a la función puede ser causa de **errores de lógica** en los programas.

10. **Olvidar que los arreglos** se pasan por referencia y, que por lo tanto es posible modificarlos, puede causar un **error de lógica**.

11. **Hacer referencia a un elemento de un arreglo** con doble índice ***a[x][y]*** como ***a[x, y]*** es un **error**. De hecho, ***a[x, y]*** se trata como ***a[y]***, pues *C++* evalúa la expresión ***x, y*** (que contiene el operador de coma) simplemente como ***y*** (que es la última de las expresiones separadas por comas)

BUENAS PRÁCTICAS DE PROGRAMACIÓN

1. **La definición del tamaño de un arreglo** como ***"variable constante"*** en lugar de cómo constante hace que los programas sean más claros. Esta técnica sirve para eliminar los llamados números mágicos; por ejemplo, la mención repetida del tamaño ***10*** en el código de procesamiento de un arreglo de ***10*** elementos da a da a dicho número un significado artificial que, desgraciadamente, puede confundir al lector cuando el programa contiene otros ***10*** que no tienen nada que ver con el tamaño del arreglo.

2. **Busque la claridad de los programas**. A veces vale la pena perder un poco en el uso eficiente de la memoria o del tiempo de procesador a favor de escribir programas más claros.

3. Algunos programadores **incluyen los nombres de variables** en los prototipos de función con la finalidad de hacer más claros los programas. **El compilador los ignora**.

PROPUESTAS DE DESEMPEÑO

1. Si en lugar de **inicializar un arreglo con instrucciones** de asignación en tiempo de ejecución, lo **inicializa en tiempo de compilación** mediante una lista de inicialización, **el programa se ejecutará con mayor rapidez**.

2. A veces las **consideraciones de desempeño** tienen mucho más peso que las de **claridad**.

3. **Podemos aplicar** ***static*** a las declaraciones de arreglos para no crear e inicializar los arreglos cada vez que se llame a las funciones en que residen, además de que el arreglo no se destruye cada vez que se sale del programa. **Con esto se mejora el desempeño**.

4. **Pasar arreglos simulando llamadas por referencia** tiene sentido por cuestiones de desempeño. Si se pasaran los arreglos mediante llamada por valor, se pasaría una copia de cada elemento. En el caso de arreglos grandes pasados con frecuencia, esto consumiría tiempo y un espacio de almacenamiento considerable para las copias de los arreglos.

5. **A veces los algoritmos más sencillos tienen un mal desempeño**. Su virtud es que son fáciles de escribir, probar y depurar. A veces son necesarios **algoritmos más complejos** para **lograr un máximo desempeño**.

6. **Las enormes ganancias en desempeño** que tiene la **búsqueda binaria** sobre la **búsqueda lineal** no están exentas de costos. El ordenamiento de un arreglo es una operación costosa en comparación con la búsqueda de un elemento en un arreglo completo. **La sobrecarga de ordenar un arreglo vale la pena cuando es necesario hacer muchas búsquedas en él a alta velocidad**.

SUGERENCIAS DE PORTABILIDAD

1. Los **efectos** (normalmente graves) provocados por **referencia a elementos** fuera de los límites de los arreglos dependen del sistema.

OBSERVACIONES DE INGENIERÍA DE *SOFTWARE*

1. **La definición del tamaño de cada arreglo** como variable constante en lugar de cómo constante hace que los programas sean más escalables.

2. Es posible **pasar un arreglo por valor** aunque pocas veces se hace.

3. El **calificador de clase *const*** puede aplicarse a un parámetro de un arreglo de una definición de función para evitar que el arreglo original sea modificado en el cuerpo de la función. Este es otro ejemplo del principio de menor privilegio. A las funciones no debe dárseles la capacidad de modificar arreglos, a menos que sea absolutamente necesario.

INDICACIONES DE PRUEBA Y DEPURACIÓN

1. **Cuando recorremos un arreglo utilizando un ciclo**, el índice nunca debe ser menor que ***0*** y siempre deberá ser menor que el número total de elementos que tenga el arreglo (uno menos que el tamaño del mismo) Asegúrese de que la condición de terminación del ciclo evite acceder a elementos que estén fuera de este rango.

2. **Los programas deberían validar que todos los valores de entrada sean correctos**, a fin de evitar que los cálculos del programa sean afectados por información errónea.

3. Mediante el **concepto de clases** es posible implementar un ***arreglo inteligente***, que en tiempo de ejecución revise automáticamente que todas las referencias a índices estén dentro de los límites. Dichos tipos de datos inteligentes ayudan a eliminar errores.

4. Aunque es **posible modificar un contador de ciclo** en el cuerpo de un ***for***, evite hacerlo, pues es común que esto genere fallas sutiles.

LO QUE NECESITA SABER

Antes de continuar con la siguiente lección, asegúrese de haber comprendido los siguientes conceptos:

- Un **arreglo bidimensional** o **tabla** es una combinación de dos o más **filas** o **listas de elementos**. Tiene una dimensión $m \times n$, en donde m es el número de **filas** en el arreglo y n el de **columnas** del arreglo.
- Es posible utilizar los **arreglos** para representar **tablas de valores** que consisten en información dispuesta en **filas** y **columnas**. Para identificar un **elemento** particular de una **tabla**, se especifican dos índices: el **primero** (por convención) identifica la **fila** en la que está contenido el **elemento** y el **segundo** (por convención) identifica su **columna**. Las **tablas** o **arreglos** que requieren dos índices para identificar un elemento particular se llaman **arreglos de doble índice**.
- **Cuando se recibe un arreglo de un solo índice** como argumento de una función, **se dejan vacíos los corchetes del arreglo** en la lista de parámetros de la función. **Tampoco es necesario el tamaño del primer índice** de un arreglo de múltiples índices, pero sí son necesarios los tamaños de todos los índices subsiguientes. El compilador se vale de ellos para determinar las localidades en memoria de los elementos de los arreglos de múltiples índices.
- **Para pasar una fila de un arreglo de doble índice** a una función que recibe un arreglo de un solo índice, simplemente hay que **pasar el nombre del arreglo seguido por el primer índice**.
- **Un arreglo de tres dimensiones** es la combinación de dos o más arreglos de dos dimensiones y se compone de **filas**, **columnas** y **planos**. De esta manera, un arreglo de tres dimensiones tiene dimensión $p \times m \times n$, en donde p es el número de **planos** en el arreglo, m es el número de **filas** del arreglo y n es el número de **columnas**.
- En ***C++***, los **arreglos de dos dimensiones se ordenan por filas** principales, y los de **tres dimensiones se ordenan por planos** principales.
- Se requiere un ciclo separado ***for*** para acceder a cada dimensión del arreglo. Además, los ciclos deben estar anidados cuando se accedan arreglos multidimensionales. De esta manera, para acceder un **arreglo de tres dimensiones**, el **ciclo columna** se anida en el **ciclo fila**, el cual se anida en el **ciclo plano**.

- **Existen muchas aplicaciones técnicas para arreglos**. Un uso común de un arreglo es almacenar determinantes que se usan para resolver sistemas de ecuaciones simultáneas usando la regla de **Cramer**.

PREGUNTAS Y PROBLEMAS

PREGUNTAS

*Utilice las siguientes definiciones de **arreglo** para contestar las preguntas **1**-**11**:*

```
float registrosSemestre[10];
const int MULTIPLICADOR = 12;
const int MULTIPLICANDO = 20;
int producto[MULTIPLICADOR][MULTIPLICANDO];
bool cubo[3][7][4];
enum colores{Café, Negro, Rojo, Naranja, Amarillo, Verde, Azul, Violeta, Gris, Blanco};
float codigoColor[10][10][10];
```

1. **Dibuje** un diagrama que muestre cada **estructura** de arreglo y sus **índices**.
2. **Liste** los **identificadores** que se deban usar para **acceder** a cada arreglo.
3. ¿**Cuáles** son las **dimensiones** de cada arreglo?
4. ¿**Cuántos** elementos **almacenará** cada arreglo?
5. **Liste** todos los **valores** posibles de **elementos** para el arreglo ***cubo***.
6. **Escriba** un **enunciado** ***C++*** que muestre el **elemento** de la **cuarta fila** y **segunda columna** del arreglo ***producto***.
7. **Escriba** un **enunciado** ***C++*** que asigne cualquier **elemento legal** a la **segunda fila** y **última columna** del arreglo ***producto***.
8. **Escriba** un **enunciado** ***C++*** que muestre los **valores del elemento** en la **tercera fila**, **segunda columna**, y **tercer plano** de los arreglos ***cubo*** *y* ***codigoColor***. Suponga que se usarán los elementos de clase de datos enumerados como índices para acceder al arreglo ***codigoColor***.
9. **Escriba** **enunciados** ***C++*** que **inserten valores** en el *arreglo* ***codigoColor*** utilizando las siguientes combinaciones de código de color y valores asociados:

   a. ***Café, Negro, Rojo = 1000***
   b. ***Café, Negro, Verde = 1000000***
   c. ***Amarillo, Violeta, Rojo = 4700***
   d. ***Rojo, Rojo, Rojo = 2200***

10. **Escriba** el **código** ***C++*** que se requiere para **llenar** cada arreglo desde el teclado.
11. **Escriba** el **código** ***C++*** que se requiere para **mostrar** cada arreglo e incluya los **encabezados** de tabla apropiados.
12. **Indique** si la siguiente oración es **verdadera** o **falsa**; en el caso de ser **falsa**, explique por qué.
    a. *Un programa **C++** que totalice los **elementos** de un **arreglo** de **doble índice** debe contener instrucciones **for anidadas**.*
13. **Escriba** instrucciones en ***C++*** que lleven a cabo lo siguiente:

    a. *Copie el arreglo **a** a la primera parte del arreglo **b**. Suponga que están declarados como **flota a[11], b[34]***

14. **Considere** un arreglo de enteros ***t*** de ***2 × 3***.

    a) ***Escriba** la **declaración** de **t**.*
    b) *¿Cuántas **filas** tiene **t**?*
    c) *¿Cuántas **columnas** tiene **t**?*
    d) *¿Cuántos **elementos** tiene **t**?*
    e) ***Escriba** los **nombres** de los elementos de la **segunda fila** de **t**.*

f) *Escriba los nombres de los elementos de la tercera columna de t.*
g) *Escriba una instrucción que establezca a cero el elemento de t que está en la fila 1 y la columna 2.*
h) *Escriba una serie de instrucciones que inicialicen a cero todos los elementos de t. No utilice una estructura de repetición.*
i) *Escriba una estructura for anidada que inicialice a cero los elementos de t.*
j) *Escriba una instrucción que acepte como entrada, desde la terminal, los valores de los elementos de t.*
k) *Escriba una serie de instrucciones que determinen e impriman el menor valor del arreglo t.*
l) *Escriba una instrucción que presente los elementos de la primera fila de t.*
m) *Escriba una instrucción que totalice los elementos de la cuarta columna de t.*
n) *Escriba una serie de instrucciones que imprima el arreglo t de manera tabular. Liste los índices de columna como encabezado y los índices de fila a la izquierda de cada fila.*

15. **Llene** los espacios en blanco.

a) *Un arreglo que tiene dos índices se conoce como arreglo de _____.*
b) *En los arreglos de doble índice, el primero (por convención) identifica a la _____ de un elemento y el segundo (también por convención) identifica su _____.*
c) *Un arreglo de m × n contiene _____ filas, _____ columnas y _____elementos.*
d) *El nombre del elemento de la fila 3 y la columna 5 del arreglo d es _____.*

16. **Conteste** las siguientes preguntas relacionadas con un **arreglo** llamado *tabla*.

a) *Declare el arreglo para que sea de enteros y tenga 3 filas y 3 columnas. Suponga que la variable constante tamanoArreglo se ha definido como 3.*
b) *¿Cuántos elementos contiene el arreglo?*
c) *Por medio de una estructura de repetición for inicialice cada elemento del arreglo a la suma de sus índices. Suponga que las variables enteras x y y se declaran como variables de control.*
d) *Escriba un segmento de programa que imprima el valor de cada elemento de la tabla en forma tabular, con tres filas y tres columnas. Suponga que el arreglo se inicializó con la declaración*

```
int tabla[tamanoArreglo][tamanoArreglo] = {{1,8}, {2, 4, 6}, {5}};
```

y que las variables enteras *x* y *y* se han declarado como **variables de control**. Muestre la salida.

17. **Encuentre** el **error** en el siguiente segmento de programa y **corríjalo**.

a) Suponga que
```
int a[2][2] = {{1,2}, {3,4}};
a[1, 1] = 5;
```

PROBLEMAS

1. **Escriba** un **programa** para leer **15** elementos enteros desde el teclado y almacenarlos en un arreglo *3 × 5*. Una vez que se han leído los **elementos**, muéstrelos como un *arreglo 5 × 3*. (***Sugerencia:*** Invierta las **filas** y **columnas**.)

2. **Escriba** una **función** que muestre **cualquier página** dada del arreglo ***lecciones*** que se utilizó en el ***ejemplo 19.6***. Suponga que el usuario escribirá el número de **página** que se mostrará.

3. **Escriba** un **programa** que almacene la **tabla de estado** para un contador **decena** de **4** bits (***BCD***) Escriba las funciones para llenar y mostrar la **tabla de estado**. Esto se muestra a continuación:

Estado	Contador
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

4. **Escriba** un **programa** que emplee **dos funciones** para **llenar** e **imprimir** un **calendario** para el **mes** actual.

Utilice *el* ***programa*** *desarrollado en el estudio de caso de esta lección para resolver los problemas* ***5-7****.* ***Modifique*** *el* ***programa*** *para encontrar la aplicación dada.*

5. El diagrama del circuito de la ***figura 19.9*** muestra dos corrientes incógnitas, I_1, e I_2. Un ingeniero escribe dos ecuaciones que describen el circuito, como sigue:

$$300I_1 + 500(I_1 - I_2) - 20 = 0$$
$$2001_2 + 500(1_2 - I_1) + 10 = 0$$

Coloque *estas ecuaciones en* ***forma estándar*** *y* ***resuelva*** *las dos ecuaciones utilizando el software desarrollado en el estudio de caso de esta lección.*

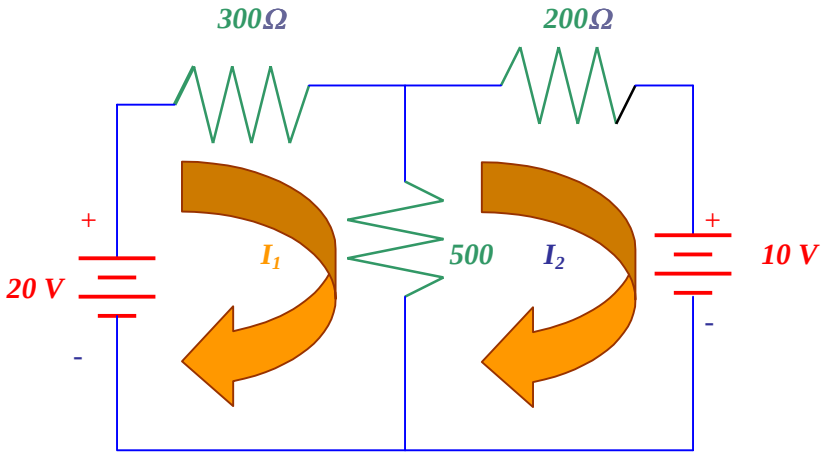


Figura 19.9*. Circuito de dos ciclos para el problema* ***5***

6. Vea la palanca de la ***figura 19.10***. Si conoce uno de los pesos y todas las distancias de los pesos desde el punto de apoyo, es posible calcular los otros dos pesos utilizando dos ecuaciones simultáneas. Las dos ecuaciones tienen la siguiente forma general,

$$w_1 d_1 + w_2 d_2 = w_3 d_3$$

en donde:

w_1, w_2, *y* w_3 *son los tres* ***pesos****.*
d_1, d_2, *y* d_3 *son las* ***distancias*** *de los tres pesos que se localizan desde el punto de apoyo, respectivamente.*

Utilizando este formato de ecuación general, se obtienen dos ecuaciones conociendo dos puntos de equilibrio. Suponga que el peso w_3 *es de* **5** *libras, y que se obtiene un estado de equilibrio para los siguientes valores de distancia:*

Punto de equilibrio 1:

d_1 = **3 pulg**
d_2 = **6 pulg**
d_3 = **36 pulg**

Punto de equilibrio 2:

d_1 = **5 pulg**
d_2 = **4 pulg**
d_3 = **30 pulg**

*Encuentre los dos **pesos** desconocidos, w_1, y w_2.*

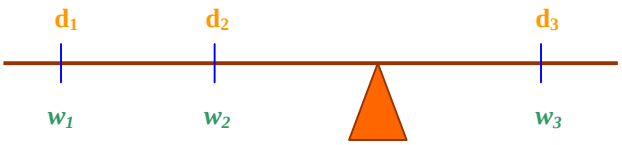


*Figura 19.10. Acomodo de **palanca** y **punto de apoyo** para el problema **6***

7. Las siguientes ecuaciones describen la **tensión**, en libras, de los dos **cables** que soportan un **objeto**. **Encuentre** la cantidad de **tensión** de cada **cable** *(T_1 y T_2.)*

$$0.5T_2 + 0.93T_1, - 120 = 0$$
$$0.42T_1 - 0.54T_2 = 0$$

8. **Escriba** una **función** para llenar un arreglo 3×4 con los **coeficientes** y términos constante de tres **ecuaciones simultáneas** expresadas en forma estándar. Suponga que el usuario escribirá los **elementos** del arreglo en el orden solicitado.

9. **Escriba** una **función** para presentar el arreglo de la **ecuación** en el problema **8**.

10. Utilizando la regla de **Cramer**, **escriba** una **función** para formar determinantes de 3×3 del arreglo de **ecuación** de 3×4 que se llenó en el problema **8**.

11. **Escriba** una **función** para desarrollar un **determinante** de orden **3**.

12. **Utilice** las **funciones** que desarrolló en los problemas 8 a 11 para **escribir** un programa que resuelva una serie de tres **ecuaciones simultáneas**.

13. **Utilice** el **programa** en el problema **12** para resolver las tres corrientes *(I_1, I_2, I_3,)* en el circuito del puente **Wheatstone** mostrado en la ***figura 19.11***. A continuación las **ecuaciones** que escribió un ingeniero para describir el circuito:

$$2000(I_1 - I_2) + 4000(I_1 - I_3) - 10 = 0$$
$$2000(I_2 - I_1) + 8000I_2 + 5000(I_2 - I_3) = 0$$
$$5000(I_3 - I_2) + 3000I_3 + 4000(I_3 - I_1) = 0$$

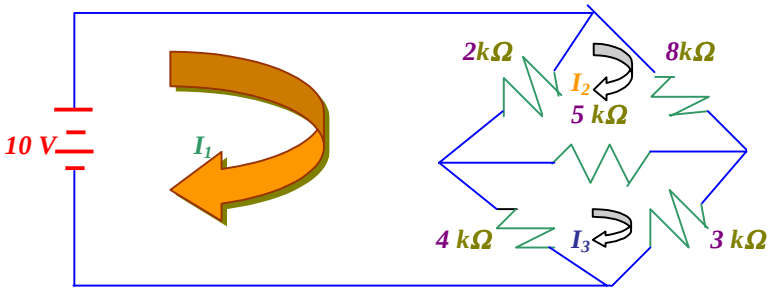


*Figura 19.11. **Tres ciclos del circuito** del puente de **Wheatstone** para el problema **13***

14. Suponga que el **perímetro** de un **triángulo** es *14* pulgadas. El **lado más corto** es la **mitad** del **más largo** y **2** pulgadas más que la **diferencia** de los dos **lados más largos**. **Encuentre** la **longitud** de cada **lado** utilizando el programa que se desarrolló en el problema **12**.

15. Suponga que la siguiente **tabla** representa el **precio** de renta mensual de **6 cabañas** de descanso durante un periodo de **5 años**.

		AÑO				
		PRIMERO	SEGUNDO	TERCERO	CUARTO	QUINTO
	1	200	210	225	300	235
	2	250	275	300	350	400
CABAÑA	3	300	325	375	400	450
	4	215	225	250	250	275
	5	355	380	400	404	415
	6	375	400	425	440	500

Escriba un programa que emplee las funciones para realizar las siguientes tareas:

- *Llene un arreglo de 2 dimensiones con la tabla.*
- *Calcule el ingreso de renta total por año para cada cabaña y almacene los totales anuales en un segundo arreglo.*
- *Calcule el porcentaje de incremento y decremento en el precio entre los años adyacentes para cada cabaña y almacene el porcentaje en un tercer arreglo.*
- *Genere un informe que muestre los tres arreglos en una tabla con encabezados apropiados en fila y columna.*

16. El método de *cofactores* que se utilizó para encontrar un valor de un determinante es un **proceso recursivo**. **Escriba** una función que encuentre el valor de un determinante de orden **n**, en donde el usuario escriba el orden del determinante *n*. Incorpore esta función a un programa que encuentre la solución para **n** ecuaciones de ***n*** incógnitas. (Deberá consultar un buen libro de texto de álgebra lineal para información sobre el uso del *método de cofactores* para resolver un determinante.)

17. Por medio de un arreglo de **doble índice**, **resuelva** el siguiente problema. Una compañía tiene **4 vendedores** (**1** a **4**) que venden **cinco productos** (**1** a **5**) Una vez por día cada **vendedor** entrega una hoja por cada tipo de **producto** vendido. Las hojas contienen:

a) *El número del vendedor*
b) *El número del producto*
c) *El importe de las ventas de dicho producto en el día*

Por lo tanto, cada **vendedor** entrega entre **0** y **5** hojas de **ventas** por día. Suponga que están a la mano las hojas de información de las ventas del último mes. Escriba un programa que tome como entrada esta información y resuma las **ventas totales** por **vendedor** y **producto**. Todos los totales deben almacenarse en el arreglo de doble índice ***ventas***. Tras procesar la información del último mes, imprima el resultado en formato de tabla; que cada **columna** represente un **vendedor** y cada **fila** represente un **producto**. Totalice cada **fila** para obtener las **ventas totales** por **producto**; totalice cada **columna** para determinar las **ventas totales** por **vendedor**. La impresión tabular deberá incluir estos **totales** a la derecha de las **filas** y al final de las **columnas**.

18. (***Gráficos por tortuga***) El lenguaje **Logo**, muy popular entre los usuarios de computadoras personales, hizo famoso el concepto de ***gráficos por tortuga***. Imagine una **tortuga** mecánica que camina por la habitación bajo el control de un programa en *C++*. La **tortuga** sostiene una **pluma** en cualquiera de dos posiciones, **arriba** o **abajo**. Mientras la **pluma** está hacia **abajo**, la **tortuga** traza formas mientras se mueve; cuando está hacia **arriba**, se mueve libremente sin dibujar nada. En este problema, usted **simulará** la operación de la **tortuga** y también **creará** un **pizarrón computarizado**.

Utilice un arreglo ***piso*** de *20* × *20*, inicializado a **cero**. Lea los comandos de un arreglo que los contenga. Tenga el registro de la **posición** de la **tortuga** en todo momento y si la **pluma** está hacia **arriba** o hacia **abajo**. Suponga que la **tortuga** siempre inicia en la posición *0*,*0* del piso con la **pluma** hacia **arriba**. El conjunto de **comandos** de la **tortuga** que deberá procesar el programa es el siguiente:

COMANDO	*SIGNIFICADO*
1	**Pluma hacia arriba**
2	**Pluma hacia abajo**
3	**Gira a la derecha**
4	**Gira a la izquierda**
5, 10	**Avanza 10 espacios (o algún número distinto de *10*)**
6	**Imprime el arreglo de *20* por *20***
7	**Fin de datos (centinela)**

Suponga que la **tortuga** está en algún punto cerca del centro del piso. El siguiente ***programa*** dibujará un cuadrado de *12* por *12*, dejando la **pluma** en la posición hacia **arriba**:

```
2
5,12
3
5,12
3
5,12
```

3
5,12
1,
6
9

A medida que la **tortuga** avanza con la **pluma** hacia **abajo**, establezca a **1** los elementos apropiados del arreglo ***piso***. Cuando se da el comando *6* (imprimir), haga que cualquier lugar que tenga un **1** en el arreglo imprima un asterisco u otro carácter. Donde haya **cero**, que presente un espacio. **Escriba** un programa que implemente las capacidades de gráficos por **tortuga** explicados aquí. **Escriba** varios programas de gráficos por **tortuga** que presenten formas interesantes. **Añada** otros comandos que aumente el poderío de su lenguaje de gráficos por **tortuga**.

19. (***Circuito del caballo***) Uno de los problemas más interesantes para los aficionados al ajedrez es el ***circuito del caballo***, propuesto inicialmente por el matemático ***Euler***. La pregunta es la siguiente: **¿Puede el caballo del ajedrez moverse por un tablero vació y tocar los 64 cuadros una sola vez?** Estudiaremos profundamente este interesante problema.

El **caballo** hace movimientos en forma de ***L*** (dos **casillas** en una dirección y una en sentido perpendicular) Por lo tanto, desde una **casilla** del centro de un tablero vació, el **caballo** puede hacer ocho movimientos distintos (***0*** a ***7***), como se muestra en la ***figura 19.12***.

a. *Dibuje en una hoja de papel un tablero de 8 por 8 e intente el circuito del caballo a mano. Ponga un 1 en el primer cuadro al que se mueva, un 2 en el segundo, un 3 en el tercero, etc. Antes de iniciar el circuito estime hasta dónde piensa que llegará, recordando que un circuito completo consiste de 64 movimientos ¿Qué tan lejos llegó? ¿Cerca de su estimación?*

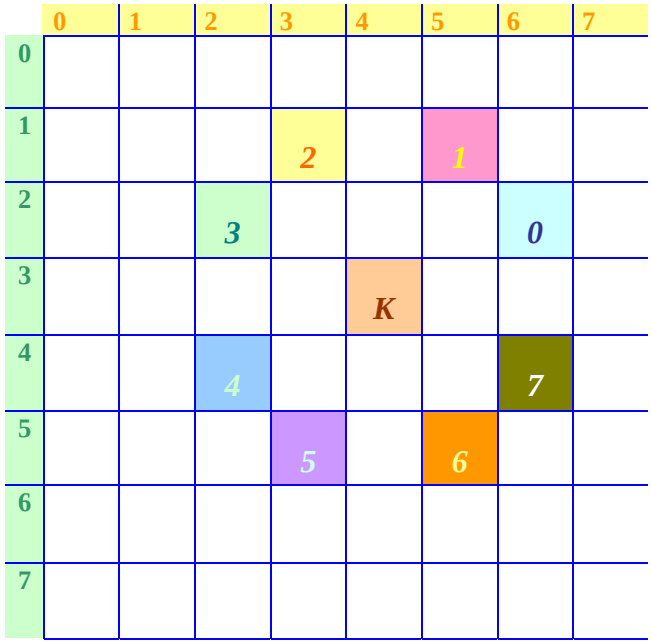


Figura 19.12**. Los ocho movimientos posibles del **caballo

b. *Ahora desarrollemos un programa que mueva el caballo por el tablero. El tablero se representa por medio de un arreglo **tablero**, de doble índice de 8 × 8. Cada uno de los cuadros se inicializa a **cero**. Describimos cada uno de los **ocho** movimientos en términos de sus componentes horizontales y verticales. Por ejemplo, un **movimiento** del tipo 0, como el indicado en la **figura 19.12**, consiste del **movimiento horizontal** de dos cuadros a la **derecha** y de un cuadro **vertical**, hacia **arriba**. El **movimiento** 2 consiste del **movimiento horizontal** de un cuadro a la **izquierda** y de dos cuadros **verticales**, hacia **arriba**. Los **movimientos horizontales** a la **izquierda** y los **movimientos verticales** hacia **arriba** se indican mediante números negativos. Los **ocho** movimientos se pueden describir por medio de dos arreglos de un solo índice, **horizontal**, **vertical**:*

```
horizontal[0] = 2
horizontal[1] = 1
horizontal[2] = -1
horizontal[3] = -2
horizontal[4] = -2
horizontal[5] = -1
horizontal[6] = 1
horizontal[7] = 2

vertical[0] = -1
vertical[1] = -2
vertical[2] = -2
vertical[3] = -1
vertical[4] = 1
vertical[5] = 2
vertical[6] = 2
vertical[7] = 1
```

*Haga que las variables **filaActual** y **columnaActual** indiquen la **fila** y **columna** de la posición actual del **caballo**, respectivamente. Para hacer un **movimiento** del tipo **numeroMovimiento**, donde **numeroMovimiento** está entre **0** y **7**, el programa se vale de las instrucciones*

```
filaActual + = vertical[numeroMovimiento];
columnaActual += horizontal[numeroMovimiento];
```

*Lleve un contador que cuente del **1** al **64**. Registre la última cuenta en cada cuadro al que se mueva el **caballo** y, claro está, pruebe cada **movimiento** potencial para asegurarse de que el **caballo** no cae fuera del tablero. Ahora escriba un programa para mover el **caballo** por todo el **tablero**. Ejecute dicho programa. ¿**Cuántos movimientos hizo el caballo**?*

c. *Tras su intento por escribir y ejecutar el programa del circuito del **caballo**, probablemente habrá desarrollado algunas habilidades valiosas. Nos valdremos de ellas para desarrollar una **heurística** (o estrategia) de **movimiento** del **caballo**. Esta no garantiza el éxito. Tal vez haya observado que los cuadros exteriores son más problemáticos que los que están más cerca del centro del tablero. De hecho, los cuadros más difíciles o inaccesibles son los de las esquinas.*

*La intuición podría sugerir que lo que se debe hacer es mover primero el **caballo** a los puntos más complicados y dejar abiertos los de más fácil acceso, de modo que cuando se congestione el tablero hacia el final del circuito haya mayor posibilidad de éxito.*

*Podemos desarrollar una **heurística** de accesibilidad clasificando cada uno de los cuadros de acuerdo con su accesibilidad y luego moviendo el **caballo** al cuadro (dentro de sus **movimientos** en forma de **L**, claro está) que sea más inaccesible. Se etiqueta un arreglo de doble índice **accesibilidad** con números que indiquen la cantidad de cuadros **accesibles** a partir de cada cuadro. En un tablero en blanco, cada cuadro central tiene una **calificación** de **8** y cada cuadro de esquina una **calificación** de **2**; los demás cuadros tienen **calificaciones** de **3**, **4** o **6**, según la siguiente tabla:*

2	3	4	4	4	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

*Ahora escriba una versión del programa del circuito del **caballo** aplicando la **heurística** de **accesibilidad**. En cualquier momento el **caballo** debe moverse al cuadro con la menor **calificación** de acceso. En caso de empate, se podrá mover a cualquiera de los cuadros empatados. Por lo tanto, el*

*circuito puede comenzar en cualquiera de las cuatro esquinas. (**Nota**: a medida que el caballo se mueva por el tablero, el programa deberá reducir las **calificaciones** de **accesibilidad**, dado que más y más cuadros quedan ocupados. De esta manera, en cualquier momento dado durante el circuito, las cifras de **accesibilidad** a los cuadros permanecerán iguales a la cantidad de cuadros desde donde puede llegar a ellos.) Ejecute esta versión del programa. ¿**Hizo el circuito complejo**? Modifíquelo ahora para que ejecute **64** circuitos, comenzando desde cada cuadro del tablero. ¿**Cuántos circuitos completos logró**?*

d. *Escriba una versión del programa del circuito del **caballo** que, al tener un empate entre dos o más cuadros, decida el cuadro a seleccionar analizando hacia delante los cuadros a los que se podrá llegar desde los cuadros empatados. El programa deberá moverse al cuadro en el que el siguiente movimiento llevará al cuadro con la menor **calificación** de **accesibilidad**.*

20. (***Circuito del caballo: enfoque de fuerza bruta***) En el ejercicio **19** se desarrolló una solución para el problema del circuito del **caballo**. El enfoque que se empleó, llamado ***heurística*** *de* ***accesibilidad*****,** genera muchas soluciones y se ejecuta con eficiencia.

A medida que aumente el poderío de las computadoras, podremos resolver más problemas a base de puro poder de cómputo y algoritmos relativamente sencillos. Llamaremos a este enfoque solución de problemas por ***fuerza bruta.***

a. *Mediante generación de **números aleatorios** permita que el **caballo** camine por el tablero (describiendo sus movimientos legales, en forma de **L**, por supuesto) al **azar**. El programa deberá hacer un circuito e imprimir el tablero final. ¿**Qué tan lejos llegó el caballo**?*

b. *Lo más probable es que el programa previo haya producido un circuito relativamente corto. Ahora **modifique** su programa para que intente **1000** circuitos. Con un arreglo de un solo índice, registre la cantidad de circuitos de cada longitud.. Cuando el programa termine de intentar los **1000** circuitos, deberá **imprimir** esta información en forma de tabla ¿**Cuál fue el mejor resultado**?*

c. *Con mucha probabilidad el programa previo le dio algunos circuitos respetables, pero ninguno completo. Ahora elimine todos los obstáculos y simplemente deje que el programa se ejecute hasta que efectúe un circuito completo. (**Precaución**: esta versión del programa podría ejecutarse durante horas en una computadora poderosa.) Nuevamente lleve una tabla de la cantidad de circuitos de cada longitud e imprímala cuando se haga el primer circuito completo. ¿**Cuántos circuitos intentó el programa antes de hacer uno completo**? ¿**Cuánto tiempo tardó**?*

d. *Compare la versión de **fuerza bruta** del circuito del **caballo** con la versión **heurística** de **accesibilidad**. ¿**Con cuál tuvo que hacer un estudio más cuidadoso del problema**? ¿**Qué algoritmo fue más difícil de desarrollar**? ¿**Cuál requirió de más poder de cómputo**? ¿**Podríamos tener la seguridad** (por adelantado) **de lograr un circuito completo con el enfoque de heurística de accesibilidad**? ¿**Podríamos tener la certeza** (por adelantado) **de lograr un circuito completo con el enfoque de fuerza bruta**? Indique los **pros** y **contras** de la solución de problemas en general por **fuerza bruta**.*

21. (***Ocho reinas***) Otro problema para los fanáticos del ajedrez es el de las **ocho reinas**. En pocas palabras: ¿es posible poner **ocho reinas** sobre un tablero vacío, de manera que no se ataquen entre sí, es decir, que no haya **reinas** en la misma **fila**, **columna** o **diagonal**? Válgase del pensamiento desarrollado en el ejercicio **19** para formular una **heurística** que solucione el problema de las **ocho reinas**. Ejecute el programa. (***Sugerencia**: es posible asignar un valor a cada cuadro del tablero que indique cuántos cuadros se **eliminan** de un tablero vació si se coloca una reina en él. A cada una de las esquinas se asignaría el valor **22**, como en la **figura 19.13**.)* Una vez puestos estos ***números de eliminación*** en los *64* cuadros, una **heurística** apropiada podría ser: poner la siguiente reina en el cuadro con el número de **eliminación menor**. ¿**Por qué es atractiva esta estrategia desde el punto de vista intuitivo**?

```
********
**
* *
*  *
*   *
*    *
*     *
*      *
```

*Figura 19.13. Los 22 cuadros **eliminados** al poner a una **reina** en la **esquina** superior izquierda.*

22. (***Ocho reinas: enfoque de fuerza bruta***) En este ejercicio se desarrollarán varios enfoques de **fuerza bruta** para la solución del problema de las **ocho reinas** presentado en el problema ***21***.

a) ***Resuelva** el problema de las **ocho reinas** mediante la técnica **aleatoria** de **fuerza bruta** desarrollada en el problema **21**.*

b) ***Utilice** una técnica **exhaustiva**, es decir, intente todas las combinaciones posibles de **ocho reinas**.*

c) *¿Por qué supone que el enfoque **exhaustivo** de **fuerza bruta** tal vez no sea adecuado para resolver el problema del circuito de **caballo**?*

d) ***Compare** y **contraste** en genera el enfoque **aleatorio** de **fuerza bruta** y el **exhaustivo** de **fuerza bruta**.*

23. (***Circuito del caballo: prueba de circuito cerrado***) En el circuito del **caballo**, sucede un circuito completo cuando el **caballo** hace **64 movimientos**, tocando cada uno de los cuadros del tablera una sola vez. Ocurre un circuito cerrado cuando el **64º movimiento** está a un **movimiento** de la localidad en que el **caballo** inició el circuito. **Modifique** el programa del circuito del **caballo** del problema ***19*** para que determine si ha ocurrido un **circuito cerrado** al realizarse un circuito completo.

24. (***Ordenamiento en cubetas***) El **ordenamiento en cubetas** inicia con un arreglo de un solo índice de enteros positivos a ordenar y un arreglo de **doble índice** de enteros con **filas** que contienen índices del ***0*** al ***9*** y **columnas** con índices del ***0*** al ***n-1***, donde ***n*** es el número de valores en el arreglo a ordenar. Cada **fila** del arreglo de **doble índice** es una **cubeta**. Escriba una función ***clasificacionCubeta()*** que tome como argumentos un arreglo de enteros y su tamaño y lleve a cabo lo siguiente:

a) ***Ponga** cada valor del arreglo de un solo índice en una **fila** del arreglo de **cubetas**, basándose en el dígito de unidades del valor. Por ejemplo, **97**, se coloca en la **fila 7**; **3** en la **fila 3** y **100** en la **fila 0**. A esto se le llama **pasada de distribución**.*

b) ***Recorra** el arreglo de **cubetas** por medio de un ciclo, **fila** por **fila**, y vuelva a copiar los valores al arreglo original de un solo índice. A esto se le llama **pasada de recopilación**. El nuevo orden de los valores previos en el arreglo de un solo índice es **100**, **3** y **97**.*

c) ***Repita** este proceso para cada posición subsecuente (**decenas**, **centenas**, **millares**, etc.)*

En la **segunda pasada**, ***100*** se coloca en la ***fila 0***; ***3*** en la ***fila 0*** (pues ***3*** no tiene dígito de **decenas**) y ***97*** en la ***fila 9***. Tras el pase de **recopilación**, el orden de los valores en el arreglo de un solo índice es ***100***, ***3*** y **97**. En la **tercera pasada**, ***100*** se coloca en la ***fila 1***; ***3*** en la ***fila 0*** y ***97*** en la ***fila 0*** (después del ***3***) Tras la **última pasada** de **recopilación**, queda ordenado el arreglo de un solo índice.

Observe que el arreglo de **cubetas** de **doble índice** es diez veces mayor que el arreglo de enteros que se está ordenando. Esta técnica da un mejor desempeño que el ordenamiento de **burbuja**, pero necesita mucha memoria. El ordenamiento de **burbuja** sólo necesita espacio para un elemento más de datos. Este es un ejemplo de la concesión **espacio-tiempo**: el ordenamiento en **cubetas** utiliza más memoria que el ordenamiento de **burbuja**, pero tiene un mejor desempeño. Esta versión del ordenamiento en **cubetas** requiere volver a copiar todos los elementos al arreglo original después de cada pasada. Otra posibilidad es crear un segundo arreglo de doble índice e intercambiar la información entre ambos.

25. **Escriba** un **programa** que emplee **apuntadores** para encontrar los elementos **máximo** y **mínimo** en un arreglo de enteros de **dos dimensiones**. **Inicialice** el arreglo desde el teclado utilizando las entradas del usuario.

Problemas de recursividad

1. (***Ocho reinas***) **Modifique** el **programa** de las **ocho reinas** del problema ***21*** para que resuelva **recursivamente** el problema.

EXAMEN BREVE 39

1. Dada la siguiente definición de un **arreglo bidimensional**,

   ```
   float muestra[10][15];
   ```

   ¿**Cuál** es el **índice máximo de fila**?, ¿Cuál es el **índice máximo de columna**?
2. ¿**Qué** presentará el siguiente enunciado cuando se aplique al **arreglo** definido en la pregunta **1**?

   ```
   cout << sizeof(muestra) / sizeof(float) << endl;
   ```
3. **Escriba** un **enunciado** que lea un valor del teclado y lo coloque en la **primera fila** y **última columna** del **arreglo** definido en la pregunta **1**.
4. **Escriba** un **enunciado** que muestre el valor almacenado en la **segunda fila** y **tercera columna** del **arreglo** definido en la pregunta **1**.
5. **Escriba** el **código**, utilizando ciclos ***for*,** que muestre los elementos del **arreglo** definido en la pregunta **1** en el formato ***fila*** y ***columna***.
6. **Escriba** un **prototipo** para una **función** llamada ***presenta()*** que muestre el contenido del **arreglo** definido en la pregunta **1**.
7. **Escriba** un **enunciado** para llamar a la **función** en la pregunta **6**.
8. Un **arreglo bidimensional** en ***C++*** se **ordena** con la _____ **principal**.

EXAMEN BREVE 40

1. ¿**Qué** problema se encuentra cuando se definen **arreglos multidimensionales grandes**?
2. Un **arreglo de tres dimensiones** en ***C++*** está **ordenado** por _____.
3. **Defina** un **arreglo de tres dimensiones** de enteros que tenga **diez planos**, **quince filas** y **tres columnas**.
4. ¿**Cuántos** bytes de almacenamiento ocupa el **arreglo** que se definió en la pregunta ***3***?
5. **Escriba** el **código** necesario para presentar el **contenido del arreglo** que se definió en la pregunta ***3***, un plano a la vez.

RESPUESTA EXAMEN BREVE 39

1. Dada la definición de arreglo bidimensional ***float muestra[10][15];***, el índice máximo en la fila es **9** y el índice máximo en la columna es **14**.
2. El enunciado ***cout << sizeof(muestra) / sizeof(float) << endl;*** mostrará el número de posiciones del arreglo ***muestra[]***, o sea ***150***.
3. Un enunciado que leerá un valor desde el teclado y lo colocará en la primera fila y en la última columna del arreglo definido en la pregunta **1** es:

   ```
   cin >> muestra[0][4];
   ```
4. Un **enunciado** que mostrará el valor almacenado en la segunda fila y en la tercera columna del arreglo definido en la pregunta **1** es:

   ```
   cout << muestra[1][2];
   ```
5. El **código**, usando los ciclos ***for***, que mostrará los elementos del arreglo definido en la pregunta ***1*** es:

   ```
   for(int fila = 0; fila < 10; ++fila)
   {
       for(int col = 0; col < 15; ++col)
   ```

```
            cout << muestra[fila][col];
        cout << endl;;
    } // Fin del for
```

6. Un prototipo para una función llamada *presenta()* que mostrará el contenido del arreglo definido en la pregunta *1* es:

```
void presenta(float muestra[10][15]);
```

7. Un enunciado para llamar a la función *presenta()* en la pregunta 6 es:

```
presenta(muestra);
```

8. Un arreglo bidimensional en ***C++*** se ordena con la ***fila*** principal.

RESPUESTA EXAMEN BREVE 40

1. Un **problema** que se podría encontrar cuando definimos arreglos multidimensionales grandes es el mensaje de error: ***Array size too big*** (arreglo de tamaño muy grande). Esto significa que usted está intentando reservar más memoria para el arreglo de la que un sistema de cómputo en particular puede asignar.
2. Un **arreglo** de tres dimensiones en ***C++*** está ordenado por ***planos***.
3. Una **definición** para un arreglo tridimensional de enteros que tiene ***10*** **planos**, ***15*** **filas** y ***3*** **columnas** es:

```
int enteros [10][15][3];
```