Matrices

Introducción

	Carro 1	Carro 2	Carro 3	Carro 4	Carro 5
Lunes					
Martes			2007,50		
Miercoles					
Jueves					
Viernes					
Sábado					

El dueño de una cadena de Carritos de venta de comida (5 en total) desea almacenar lo recaudado cada día de la semana en cada carrito. Se considera de lunes a sábado ,o sea 6 días. La planilla resultante se muestra en figura.

En cada casillero se almacena lo facturado en ese carro en ese día.

En nuestro ejemplo, el día Martes en el Carro 3 se facturaron 2007,50 pesos. El área coloreada, donde se van a almacenar los valores facturados es una matriz. En C++, para declarar una matriz haremos sustitución de días y carros por índices, manteniendo la consigna del arreglo que los índices comienzan en cero.

	0	1	2	3	4
0					
1			2007,50		
2					
3					
4					
5					

La declaración en C++ de esta matriz es así:

```
const int FIL = 6;
const int COL = 5;
typedef float Matriz [FIL][COL];
```

Luego, podemos declarar una variable de este tipo de la siguiente manera:

```
Matriz carros;
```

Para acceder al valor ubicado en la segunda fila y la tercera columna haremos:

```
carros[1][2]
```

Veamos a continuación un programa de ejemplo que realiza la declaración y la carga de la matriz del ejemplo anterior. Nótese que para realizar la carga es necesario utilizar una doble iteración. Es decir, una iteración con otra iteración anidada en su interior. La iteración ***exterior*** es para ir recorriendo, una a una, todas las ***filas*** de la matriz. A su vez, la iteración ***interior*** es para ir recorriendo, también una a una, todas las ***columnas***.

En este ejemplo concreto, se utilizan dos iteraciones `for` anidadas. Esto es así porque se pretende recorrer **completamente** la matriz. El `for` exterior irá recorriendo las filas una por una. A su vez, por **cada** entrada al `for` exterior, se ejecutará completamente el `for` interior. Esto tendrá como efecto que la matriz se recorrerá en su **totalidad** de la siguiente manera: primero se cargarán, uno por uno, todos los elementos de la primera fila. Enseguida, se cargarán, uno por uno, todos los elementos de la primera fila. Luego se cargarán, uno por uno, todos los elementos de la primera fila. Este proceso continúa hasta que la matriz haya sido cargada en su totalidad.

```
#include <stdio.h>

const int FIL = 6;
const int COL = 5;
typedef float Matriz [FIL][COL];

int main()
{
   Matriz carros;
   int i, j;

   for (i = 0; i < FIL; i++)
   {
      for (j = 0; j < COL; j++)
      {
          printf ("ingrese valor de fila %d y columna %d", i, j);
          scanf ("%f", &carros[i][j]);
      }
   }

   ...
   /* luego se realizan otras acciones sobre la matriz */
}
```

UNIVERSIDAD DE LA EMPRESA

Arreglos Bidimensionales

Una matriz es un ***arreglo bidimensional*** que se define de la siguiente manera:

```
typedef tipo_elemento ident_matriz [filas][columnas];
```

donde

`ident_matriz`	es el nombre del tipo de datos definido
`filas`	es un entero positivo que indica la cantidad de filas
`columnas`	es otro entero positivo que indica la cantidad de columnas
`tipo_elemento`	es un tipo cualquiera, correspondiente al tipo de los elementos a almacenar en las celdas de la matriz

Tanto las filas como las columnas se identifican mediante índices en el rango **0..(tamaño-1)**. En el caso de las filas esto es **0..(filas-1)** y en el caso de las columnas esto es **0..(columnas-1)**.

Al igual que en el caso de los arreglos comunes (arreglos unidimensionales) vistos anteriormente en el curso, es una buena práctica de programación utilizar ***constantes simbólicas*** para establecer la cantidad de filas y la cantidad de columnas de la matriz.

En el caso particular de que la cantidad de filas coincida con la cantidad de columnas, se dice que la matriz es una ***matriz cuadrada***.

Conjunto de operaciones

La operación de ***selección de un elemento*** se realiza de manera análoga a como se realiza en los arreglos unidimensionales. Esto es, dada una variable `ident_matriz var_matriz;` la operación de selección se realiza utilizando el operador [ ], de igual modo que en los arreglos de una dimensión. La diferencia consiste en que, en este caso, debe utilizarse dos veces seguidas, una vez para la fila y otra vez para la columna correspondientes a las coordenadas del elemento que es accedido.

Ejemplo:

```
const int FIL = 4;
const int COL = 5;
typedef int Matriz [FIL][COL];
...
Matriz mat; int fil = 1, col = 2;
...   /* imaginemos que aquí se hizo la carga de valores en la matriz */
printf (“%d”, mat[0][0]); /* se muestra por pantalla el valor de la celda (0,0) */
int num = mat[fil][col] + 2; /* se utiliza el valor de la celda (1,2) en una
                                expresión */
```

Observación: Al igual que en los arreglos unidimensionales **nunca** debemos acceder a una celda que esté fuera del rango de índices de una matriz. En caso de hacerlo, el compilador **no** lo detecta y ocurrirá un error ***en tiempo de ejecución***. Se debe programar con mucho cuidado a efectos de nunca acceder a celdas que no existen en la matriz.

La operación de ***reescritura de una celda*** se implementa con el operador de selección y con la asignación, de manera análoga a los arreglos unidimensionales. Para ello, dados una matriz `M` y dos índices válidos `i` y `j`, correspondientes a una fila y a una columna respectivamente, se requiere utilizar `M[i][j]` del lado izquierdo del operador de asignación. Así, la asignación `M[i][j] = expre;` expresa que se almacene en `M[i][j]` el resultado de `expre`. Como en cualquier asignación, el tipo de `expre` debe corresponder con el tipo de `M[i][j]`.

Ejemplo:

```
Matriz mat;
int fil = 1, col = 2;
mat[fil][col] = 5;      /* se asigna el valor 5 a la celda (1,2) */
```

Recorridas en matrices

Al igual que para los arreglos unidimensionales, siempre que se desee realizar una ***recorrida completa*** de las celdas de la matriz, la estructura de control adecuada es `for`. Corresponde utilizar dicha estructura porque, al conocerse de antemano la cantidad de celdas a recorrer, se conoce entonces la cantidad de veces que se debe repetir la acción a realizar.

Sin embargo, a diferencia de los arreglos unidimensionales, puede suceder que dicha estructura se utilice una sola vez, o bien anidada dentro de otra estructura de iteración. Esto es inherente a la naturaleza bidimensional de la matriz. Si se desea recorrer la matriz en su ***totalidad***, se deben anidar dos estructuras de iteración `for`. Por el contrario, si se desea recorrer únicamente una porción unidimensional de la matriz (como ser, por ejemplo, una fila determinada), entonces se debe utilizar una sola vez dicha estructura, como si se tratarse de un arreglo unidimensional.

Ejemplo: Desplegamos por pantalla la totalidad de celdas de la matriz

```
Matriz mat;
int i, j;
...   /* imaginemos que aquí se hizo la carga de valores en la matriz */
for (i = 0; i < FIL; i++)
{
   for (j = 0; j < COL; i++)
      printf (“%d ”, mat[i][j]);
   printf (“\n”);
}
```

Ejemplo: Calculamos la suma de los valores almacenados únicamente en la fila 0 de la matriz.

```
Matriz mat;
int j, sum;
...   /* imaginemos que aquí se hizo la carga de valores en la matriz */
sum = 0;
for (j = 0; j < COL; j++)
   sum = sum + mat[0][j];
printf (“La suma de los valores de la fila 0 es: %d ”, sum);
```

Búsquedas en matrices

Al igual que para los arreglos unidimensionales, siempre que se desee realizar una ***búsqueda*** en las celdas de la matriz, corresponde utilizar una estructura de control de ***repetición condicional***. Dado que se requiere consultar al menos una celda, la estructura ideal para hacerlo es `do-while`. Como en los arreglos de una sola dimensión, se utilizará una **doble condición**, donde la primera condición verificará que no nos pasemos del rango de índices válidos (recordar que, de hacerlo, ocurrirá un error en tiempo de ejecución), mientras que la segunda condición verificará si hemos encontrado o no el elemento que cumpla con la condición buscada.

Sin embargo, al igual que en el caso de las recorridas completas vistas en la página anterior, y nuevamente a diferencia de los arreglos unidimensionales, puede suceder que dicha estructura de repetición condicional se utilice una sola vez, o bien anidada dentro de otra estructura de iteración, lo cual es nuevamente inherente a la naturaleza bidimensional de la matriz. Veamos algunos ejemplos:

Ejemplo: Buscamos si un valor determinado pertenece o no a la matriz

```
Matriz mat;
int i, j, num;
boolean existe;
...   /* imaginemos que aquí se hizo la carga de valores en la matriz */
printf (“Ingrese un valor a buscar en la matriz: ”);
scanf (“%d”, &num);
existe = false;
i = 0;
do
{
   j = 0;
   do
   {
      if (mat[i][j] == num)
         existe = true;
      else
         j++;
   }
   while (j < COL && !existe);
   i++;
}
while (i < FIL && !existe);
```

Ejemplo: Buscamos únicamente en la última columna de la matriz si existe o no un valor negativo.

```
Matriz mat;
int i;
boolean existe;
...   /* imaginemos que aquí se hizo la carga de valores en la matriz */
existe = false;
i = 0;
/* continúa en la próxima página */
```

```
do
{
   if (mat[i][COL-1] < 0)
      existe = true;
   else
      i++;
}
while (i < FIL && !existe);
```

Combinando estructuras repetitivas en problemas con matrices

Al igual que con los arreglos unidimensionales, a la hora de trabajar con matrices existen problemas en los que **no** alcanza con utilizar una sola estructura repetitiva, sino que requieren combinar varias estructuras repetitivas. Dependiendo del problema, se pueden combinar dos o más estructuras de recorrida completa (`for`), dos o más estructuras de búsqueda (repetición condicional con doble condición) o incluso combinar recorridas completas con búsquedas. En cada problema concreto, se debe analizar cuidadosamente las estructuras repetitivas que se debe combinar para resolverlo.

Ejemplo: Contar cuántas filas de la matriz contienen un valor determinado. Este problema requiere realizar una recorrida completa de las filas de la matriz, mientras que implica realizar una búsqueda dentro de cada fila. Por lo tanto, se resolverá utilizando una estructura `for` junto con una estructura `do-while` (con doble condición) anidada dentro de ella.

```
Matriz mat;
int i, j, num, cont;
boolean existe;
...   /* imaginemos que aquí se hizo la carga de valores en la matriz */
printf (“Ingrese un valor a buscar en la matriz: ”);
scanf (“%d”, &num);
cont = 0;
for (i = 0; i < FIL; i++)
{
   j = 0;
   existe = false;
   do
   {
      if (mat[i][j] == num)
         existe = true;
      else
         j++;
   }
   while (j < COL && !existe);
   if (existe)
       cont = cont + 1;
}
printf (“\n El valor %d aparece en %d filas de la matriz”, num, cont);
```