Matrices

En Programación I se vio la utilidad de trabajar con vectores. El vector es una estructura de datos que nos permite agrupar bajo un mismo nombre un conjunto de datos de un mismo tipo, y acceder a cada elemento individual –para leer o escribir- utilizando el subíndice correspondiente. De esta manera, la programación resulta más sencilla y eficiente que si tuviéramos que trabajar con variables simples.

Un ejercicio clásico de trabajo con vectores, puede ser el siguiente:

Se dispone de un lote de registros con los gastos realizados por una persona durante un mes, en el que se indica el día del mes y el importe del gasto. El lote termina con un número de día igual a 0. Puede haber más de un registro para cada día. Se pide que se calcule e informe el gasto total diario.

Como ya hemos trabajado con vectores, sabemos que el ejercicio se resuelve utilizando un vector de 31 posiciones (1 para cada día), y sumando cada importe al importe existente para ese día. Como cada día está asociado a un elemento del vector, la función de carga de datos sería:

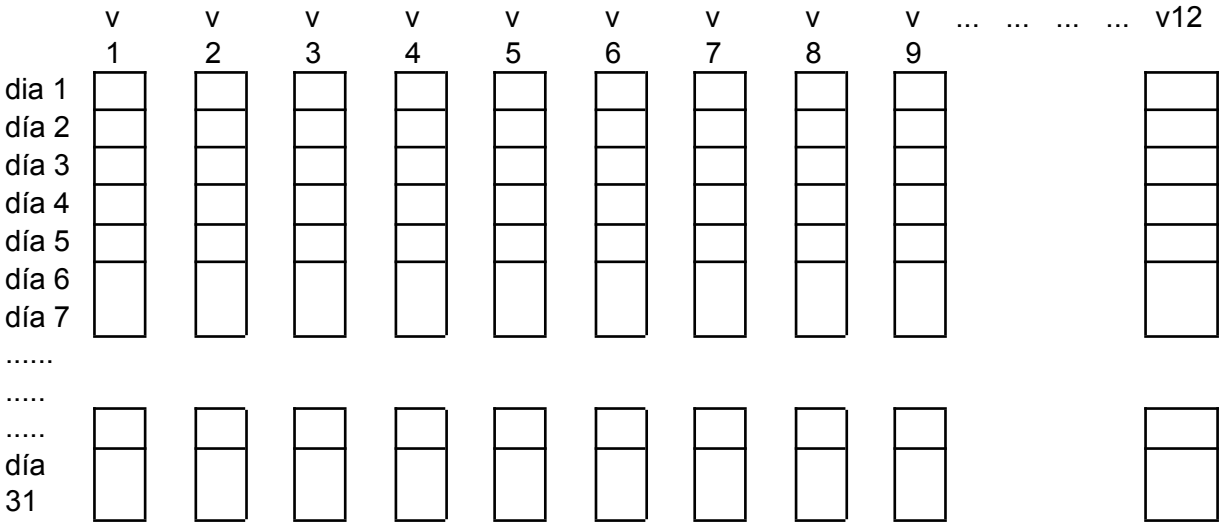
```
void cargar(float *v){
      int dia;
      float importe;
      cin>>dia;
      while(dia!=0){
            cin>>importe;
            v[dia-1]+=importe;
            cin>> día;
            }
      }
```

Previamente a usar esta función debería haberse hecho otra que ponga en 0 el vector, y finalmente otra que muestre cada elemento del vector.

Supongamos ahora que debemos resolver un problema similar, pero que en vez de limitarse a un mes se nos solicita que informemos día por día el gasto realizado durante todo el año, esto es, para cada día y cada mes cuál fue el gasto. Al lote de registros se le agregaría ahora el mes. ¿Cómo lo resolvemos?.

Una posibilidad sería declarar 12 vectores distintos de 31 elementos, un vector para cada mes, y preguntar para cada registro ingresado (usando un conjunto de if o un switch) a que mes corresponde para poder seleccionar el vector correcto. El problema se complica: habría que poner en 0, cargar y mostrar 12 vectores.

Se podría representar la situación mediante el siguiente esquema:



Una forma más sencilla sería utilizar una sola estructura de datos que bajo un mismo nombre permita acceder a cada una de las celdas, utilizando subíndices. Y esa estructura es una matriz.

Podemos definir entonces a una matriz como una estructura de datos compuesta de n filas y m columnas, donde en cada intersección de fila y columna se puede almacenar información de un tipo determinado (igual para toda la matriz). Al igual que en los vectores toda la estructura tendrá un único nombre, podrá almacenar un único tipo de datos, y para acceder a cada elemento de información o celda se accederá mediante subíndices. Como el vector tiene una sola dimensión, cada elemento se define por el nombre del vector y un subíndice; para las matrices habrá un subíndice para cada una de las dimensiones.

La bibliografía suele referirse genéricamente tanto a vectores como a matrices como array, o arreglos. Para el caso de vectores, se los denomina arrays unidimensionales, y para las matrices arrays bidimensionales (o multidimensionales ya que una matriz puede tener más de 2 dimensiones).

Una matriz de enteros de nombre m de 5 filas por 5 columnas se declararía de la siguiente manera:

```
int m[5][5];
```

tipo de dato nombre[cant. de filas][cant. de columnas]

Su representación gráfica puede ser:

columna 0	columna 1	columna 2	columna 3	columna 4	
m[0][0]	m[0][1]	m[0][2]	m[0][3]	m[0][4]	fila 0
m[1][0]	m[1][1]	m[1][2]	m[1][3]	m[1][4]	fila 1
m[2][0]	m[2][1]	m[2][2]	m[2][3]	m[2][4]	fila 2
m[3][0]	m[3][1]	m[3][2]	m[3][3]	m[3][4]	fila 3
m[4][0]	m[4][1]	m[4][2]	m[4][2]	m[4][4]	fila 4

Para el ejercicio mencionado la resolución utilizando matrices sería la siguiente:

```
#include <iostream>

using namespace std;

void ponerCeroMatriz(float m[12][31], int , int);
void cargarDatos(float m[12][31]);
void mostrarMatriz(float m[12][31], int , int);

int main(){
      float m[12][31];
      ponerCeroMatriz(m, 12 , 31);
      cargarDatos(m);
      mostrarMatriz(m, 12, 31);
      system(“pause”);
      return 0;
      }

void ponerCeroMatriz(float m[12][31], int FILA, int COL)
      {
      int i, j;
      for(i=0;i<FILA;i++)
            for(j=0;j<COL;j++)
                  m[i][j]=0;
      }


void cargarDatos(float m[][31]){
      int dia, mes;
      float importe;
      cin>>dia;
      while(dia!=0){
            cin>>mes;
            cin>>importe;
            m[mes-1][dia-1]+=importe;
            cin>>día;
            }
      }


void mostrarMatriz(float (*m)[31], int FILA, int COL){
      int i, j;
      for(i=0;i<FILA;i++)
            {
            cout<<”Mes: “<<i+1<<endl;
            for(j=0;j<COL;j++)
                  cout<< “Dia: “<<j+1<<m[i][j]<<endl;
            }
      }
```

En el ejercicio se utilizó una matriz de 12 filas (una para cada mes), y 31 columnas (una para cada día) de tipo float, ya que los importes a almacenar son números reales. Nótese que se podría haber definido invirtiendo las filas y las columnas (31 filas y 12 columnas). Para este ejemplo (y para la mayoría de los casos) es exactamente lo mismo: sólo hay que mantener el criterio durante todo el ejercicio.

Luego se llama a la función ponerCeroMatriz(), a cargarDatos() y a mostrarMatriz(), a las cuales se les pasa como parámetro el nombre de la matriz, o sea la dirección de inicio de la matriz (a la primera y la última se les envió además la cantidad de filas y columnas).

Las funciones reciben la dirección de inicio en la variable puntero m, que es un puntero a una matriz. Cada función recibe la dirección de manera distinta:

```
float m[12][31]
float m[][31]
float (*m)[31]
```

Cualquiera de las 3 notaciones es correcta, y puede utilizarse indistintamente. La primera es la notación explícita, ya que se indica la cantidad de filas y columnas. La segunda y la tercera sólo hacen mención a la cantidad de columnas; la segunda deja vacío el primer par de corchetes, y la tercera pone el nombre entre paréntesis antecedido por el *, y luego la cantidad de columnas. Como se verá más adelante, la variable m de la función es en realidad un puntero a una fila de 31 elementos float, por eso en todas las notaciones es obligatorio indicar la cantidad de columnas.
Se recomienda que se opte por alguna de ellas, para evitar confusiones.

En la función ponerCeroMatriz() y en mostrarMatriz(), la matriz se recorre utilizando 2 ciclos exactos anidados. En este caso se ha optado por dejar fija la fila, por medio del primer for, y recorrer luego con el segundo for cada una de las columnas de esa fila. Eso podría haberse hecho al revés, es decir, dejar fija la columna y recorrer cada una de las filas.

Veamos como sería la función poner_cero():

```
void ponerCeroMatriz(float m[12][31], int FILA, int COL)
	{
	int i, j;
	for(i=0;i<COL;i++)
		for(j=0;j<FIL;j++)
			m[j][i]=0;
	}
```

En este caso se empieza con m[0][0], luego m[1][0], m[2][0], etc.
De acuerdo al tipo de ejercicio que estemos resolviendo, podemos elegir recorrer una matriz por filas, o por columnas. Para ponerCeroMatriz() es exactamente lo mismo, pero para mostrarMatriz() depende de si lo que deseamos es mostrar dentro de cada mes lo gastado en cada uno de los días, o si deseamos que se muestre para todos los meses, lo gastado en un día en particular.

De manera similar, habrá situaciones en las cuales es necesario buscar un máximo o un mínimo por filas o por columnas. Se deja como ejercicio los siguientes puntos:

- Informar para cada día el mes que más gasto se realizó.
- Informar para cada mes, el día en que más gasto se realizó.

Nota: Al igual que con los vectores, C no controla si en la programación se pretende hacer asignaciones fuera de los límites para los que está definida la matriz (p.e. m[50][50]=0). Esto debe ser verificado por el programador, para evitar errores.

La utilidad de las matrices es que nos permiten clasificar información por 2 ó más características, limitando la cantidad de variables independientes necesarias y simplificando la programación.

Representación de una matriz en la memoria

Para nuestro trabajo, podemos pensar en una matriz como una tabla de 2 dimensiones, con n filas y m columnas, tal como la representamos gráficamente más arriba. Esta representación nos permite, además, aplicar las propiedades matemáticas de las matrices, y crear modelos matemáticos basados en matrices. Sin embargo, cuando declaramos una matriz en un programa fuente, el compilador reserva un conjunto de posiciones contiguas en la memoria. Supongamos que declaramos una matriz de 3x3 de tipo entero, y que la dirección de inicio es la número 1000; en este caso se reservarán para la matriz 36 bytes a partir de la posición inicial (3x3x4 bytes=36 bytes)

1000	1004	1008	1012	1016	1020	1024	1028	101032
m[0][0]	m[0][1]	m[0][2]	m[1][0]	m[1][1]	m[1][2]	m[2][0]	m[2][1]	m[2][2]
fila 0			fila 1			fila 2		

Sabemos que en los vectores, el nombre sin subíndice es la dirección de inicio del vector. Para las matrices el nombre también es la dirección de inicio, pero no del primer elemento, sino de la primera fila. Entonces:

m=&m[0] dirección de inicio de la primera fila de la matriz.

Si aplicamos * a ambos miembros

m=&m[0] como * y & se anulan

*m=m[0] dirección de inicio del primer elemento de la primera fila, o sea &m[0][0]

*m=&m[0][0] aplicando nuevamente * a ambos miembros

**m=m[0][0] contenido del primer elemento de la primera fila de la matriz.

Con vectores vimos que al anteponer el * al nombre del vector nos referíamos al contenido del primer elemento del vector. Para una matriz de 2 dimensiones, necesitamos 2 * (uno por cada dimensión) para llegar al contenido del elemento [0][0].

Veamos ahora cómo recorrer la matriz usando punteros.
Ya se ha dicho que m es un puntero a la primera fila de la matriz. Por lo tanto si lo incrementamos en 1 apuntará a la segunda:

m+0=&m[0]

m+1=&m[1]

m+2=&m[2]

Generalizando el número de fila con nf tenemos:

m+nf=&m[nf]

Aplicando * a ambos miembros, y de acuerdo a lo visto en el párrafo anterior:

*(m+nf)=&m[nf][0]

Como *(m+nf) apunta a la dirección del primer elemento de la fila nf, si lo incrementamos en 1 apuntará a la posición siguiente dentro de esa fila, o sea la columna siguiente:

*(m+nf)+1=&m[nf][1]

*(m+nf)+2=&m[nf][2]

Aplicando nuevamente * a ambos miembros, y generalizando el número de columna con nc:

((m+nf)+nc)=m[nf][nc]

Al igual que con los vectores, también podemos trabajar con matrices utilizando la notación de punteros. La función ponerCeroMatriz() del ejercicio visto, podría plantearse de la siguiente manera:

```
void ponerCeroMatriz(float m[12][31], int FILA, int COL)
      {
      int i, j;
      for(i=0;i<FILA;i++)
            for(j=0;j<COL;j++)
                  *(*(m+i)+j)=0;
      }
```

Si usamos punteros para manejar los for, el ejercicio quedaría:

```
void ponerCeroMatriz(float m[12][31], int FILA, int COL){
      int (*i)[31], *j;
      for(i=m;i<m+FILA;i++)
            for(j=*i;j<(*i)+COL;j++)
                  *j=0;
      }
```