Arrays bidimensionales

Hasta el momento aprendimos a usar el tipo `vector`, que nos permite declarar una secuencia ordenada de variables del mismo tipo, lo que constituye un array unidimensional. Luego de declarado el array, podemos referirnos a cualquier elemento del mismo con un índice a través de la notación de corchetes.

Nuestro objetivo ahora es declarar un array bidimensional, es decir una secuencia ordenada de variables del mismo tipo a la que podamos referirnos utilizando dos índices, preferentemente con una notación de dos corchetes. Es decir que si `matriz` es una variable del tipo que buscamos, de $N \times N$ elementos, queremos podemos asignar valores a los elementos primero y último de la siguiente forma:

```
matriz[0][0] = 1.0;
matriz[N-1][N-1] = 1.0;
```

donde en este caso en particular estamos asignándoles el valor 1.0.

vector 2D

Existen varias alternativas para declarar arrays bidimensionales dinámicos (ver la ayuda sobre arrays unidimensionales para una explicación de por qué no queremos arrays estáticos). La que vamos a usar nosotros es probablemente la más sencilla en cuanto a notación, y consiste en declarar un vector de vectores (abusando del hecho de que podemos declarar un vector de cualquier tipo de variable). La notación para esto es:

```
vector< vector<double> > miArray;
```

en nuestro caso de interés, en el cual queremos guardar valores reales en nuestro array. A diferencia de la declaración que hacíamos para arrays unidimensionales, no vamos a poder determinar *ab initio* sus dimensiones. Por el momento nuestra variable `miArray` tiene tamaño nulo. En las siguientes líneas nos ocupamos de dimensionarla en $M \times N$ elementos:

```
miArray.resize(M);
for(int i=0; i<M; i++) miArray[i].resize(N);
```

Lo que estamos haciendo es declarar M elementos de tipo `vector<double>`, y luego recorremos cada uno de ellos dimensionándolo hasta N elementos de tipo `double`. De esta forma, si queremos escribir el último elemento de nuestro array bidimensional, podemos hacerlo con la línea

```
cout << miArray[M-1][N-1];
```

Ejemplo

En el siguiente ejemplo declaramos un array bidimensional de 3×3 elementos de tipo `double`, y le asignamos los valores correspondientes a la matriz identidad.

```
#include <vector>

using namespace std;

int main() {
        miMatriz.resize(3);
        for(int i=0; i<3; i++) miMatriz[i].resize(3);

        for(int i=0; i<3; i++)
                for(int j=0; j<3; j++)
                        if (i == j) miMatriz[i][j] = 1.0;
                                else miMatriz[i][j] = 0.0;
        return 0;
}
```

Shortcuts

Nuestra declaración de arrays bidimensionales se puede volver bastante tediosa (y poco elegante!) si la tenemos que usar muchas veces. Para compactarla, es una práctica común generar un alias a través del comando `typedef`. Para ello, podemos escribir al principio de nuestro código (después de la declaración del `namespace`):

```
        typedef vector< vector<double> > matriz;
```

Siguiendo la definición del alias, podemos sustituir todas las apariciones de `vector< vector<double> >` en nuestro código por simplemente `matriz`.

También nos va a resultar útil, para compactar nuestro código, encapsular las líneas que dimensionan nuestros arrays bidimensionales. En el siguiente ejemplo, gracias al correcto encapsulamiento, hemos reducido la declaración y dimensionamiento de un array 2D de 3×3 a una sola línea de código:

```
#include <vector>

using namespace std;

typedef vector< vector<double> > matriz;

matriz init_matrix(int M, int N) {
        matriz tmp;
        tmp.resize(M);
        for (int i=0; i<M; i++) tmp[i].resize(N);
        return tmp;
}

int main() {
        matriz grilla = init_matrix(3, 3);

        return 0;
}
```

No debe confundirnos en este ejemplo el uso que hacíamos en la parte precedente del nombre `miMatriz`. `matriz` es ahora el nombre de un tipo de variable, como lo son `double` e `int`, y no el nombre de una variable (instancia de un tipo).