Definición de una tabla y acceso a los datos

Una tabla, vector, matriz o array (que algunos autores traducen por "arreglo") es un conjunto de elementos, todos los cuales son del mismo tipo. Estos elementos tendrán todos el mismo nombre, y ocupan un espacio contiguo en la memoria. Por ejemplo, si queremos definir un grupo de 4 números enteros, usamos

```
int ejemplo[4];
```

Podemos acceder a cada uno de los valores individuales indicando su nombre (ejemplo) y el número de elemento que nos interesa, pero con una precaución: **se empieza a numerar desde 0,** así que en el caso anterior tendríamos 4 elementos, que serían **`ejemplo[0], ejemplo[1], ejemplo[2], ejemplo[3]`**.

Como ejemplo, vamos a definir un grupo de 5 números enteros y hallar su suma:

```
// Introducción a C++, Nacho Cabanes
// Ejemplo 05.01:
// Primer ejemplo de tablas

#include <iostream>
using namespace std;

int main()
{
    int numero[5]; // Un array de 5 números enteros
    int suma; // Un entero que guardará la suma

    numero[0] = 200; // Les damos valores
    numero[1] = 150;
    numero[2] = 100;
    numero[3] = -50;
    numero[4] = 300;
    suma = numero[0] + // Y hallamos la suma
        numero[1] + numero[2] + numero[3] + numero[4];
    cout << "Su suma es " << suma;
    /* Nota: esta es la forma más ineficiente e incómoda...
    Ya lo iremos mejorando */
    return 0;
}
```

Valor inicial de una tabla

Al igual que ocurría con las variables o datos simples, podemos dar valor a los elementos de una tabla al principio del programa (inicializar). Será más cómodo dar los valores uno por uno, como hemos hecho antes. Esta vez los indicaremos todos entre llaves, separados por comas:

```
// Introducción a C++, Nacho Cabanes
// Ejemplo 05.02:
// Segundo ejemplo de tablas

#include <iostream>
using namespace std;

int main()
{
    int numero[5] = // Un array de 5 números enteros
        {200, 150, 100, -50, 300};
    int suma; // Un entero que guardará la suma

    suma = numero[0] + // Y hallamos la suma
        numero[1] + numero[2] + numero[3] + numero[4];
    cout << "Su suma es " << suma;
```

```
    /* Nota: esta forma es algo menos engorrosa, pero todavía no
    */ /* está bien hecho. Lo seguiremos mejorando */

    return 0;
}
```

Recorriendo los elementos de una tabla

Es de esperar que exista una forma más cómoda de acceder a varios elementos de un array, sin tener siempre que repetirlos todos, como hemos hecho en **`suma = numero[0] + numero[1] + numero[2] + numero[3] + numero[4];`**

El "truco" consistirá en emplear cualquiera de las estructuras repetitivas que ya hemos visto (while, do..while, for). Lo más habitual, ya que sabemos cuántos elementos tiene un array, es usar un "for" para contar, así:

```
// Introducción a C++, Nacho Cabanes
// Tercer ejemplo de tablas (for)

#include <iostream>
using namespace std;

int main()
{
    int numero[5] = // Un array de 5 números enteros
        {200, 150, 100, -50, 300};
    int suma; // Un entero que guardará la suma
    int i; // Para recorrer el array

    suma = 0; // Valor inicial
    for (i=0; i<=4; i++) // Y hallamos la suma
        suma += numero[i];

    cout << "Su suma es " << suma;
    return 0;
}
```

En este caso, que sólo sumamos 5 números, no hemos escrito mucho menos, pero si trabajaremos con 100, 500 o 1000 números, la ganancia en comodidad sí que está clara.

En ***física*** se usan mucho los "***vectores***", para ***representar magnitudes*** que no se pueden expresar sólo con un número. Por ejemplo, para una temperatura basta con decir cosas como **"tengo 39 grados"**, pero para una fuerza no basta con decir que es de 200 N, sino que también es importante en qué dirección (y sentido) se aplica esa fuerza: no tendrá el mismo efecto si se aplica en el centro de un cuerpo o en un extremo, ni si comprime que si estira. **Un vector se suele representar a partir de sus "componentes"**, que son las coordenadas de su extremo (suponiendo que comienza en el punto (0,0). ***Un vector en el plano tendrá dos componentes (x,y) y un vector en el espacio tendrá 3 componentes (x,y,z).como un conjunto de varias coordenadas.*** Así, un programa que pida al usuario las coordenadas de 2 vectores en el espacio y halle su vector suma sería:

```
// Introducción a C++, Nacho Cabanes
// Ejemplo 05.04:
// Vectores, primer contacto

#include <iostream>
using namespace std;

int main()
{
    float vector1[3];
    float vector2[3];
```

```
        float vectorSuma[3];
        int i;

        // Pedimos los datos del primer vector
        for (i=0; i<3; i++)
        {
            cout << "Introduce la componente " << i
                << " del primer vector: ";
            cin >> vector1[i];
        }

        // Pedimos los datos del segundo vector
        for (i=0; i<3; i++)
        {
            cout << "Introduce la componente " << i
                << " del segundo vector: ";
            cin >> vector2[i];
        }
         // Calculamos la suma
        for (i=0; i<3; i++)
            vectorSuma[i] = vector1[i] + vector2[i];

        // Y mostramos el resultado
        cout << "El vector suma es ";
        for (i=0; i<3; i++)
            cout << vectorSuma[i] << " ";

        return 0;
    }
```

No veremos más detalles sobre vectores, porque este no es un curso de física. Pero sí propondemos algunos ejercicios para que pueda aplicar sus conocimientos quien ya ha estudiado con anterioridad lo que es un vector... **Tablas bidimensionales**

Podemos declarar tablas de dos o más dimensiones. Por ejemplo, si queremos guardar datos de dos grupos de alumnos, cada uno de los cuales tiene 20 alumnos, tenemos dos opciones:

- Podemos usar **`int datosAlumnos[40]`** y entonces debemos recordar que los 20 primeros datos corresponden realmente a un grupo de alumnos y los 20 siguientes a otro grupo.
- O bien podemos emplear **`int datosAlumnos[2][20]`** y entonces sabemos que los datos de la forma **`datosAlumnos[0][i]`** son los del primer grupo, y los **`datosAlumnos[1][i]`** son los del segundo. En cualquier caso, si queremos indicar valores iniciales, lo haremos entre llaves, igual que si fuera una tabla de una única dimensión.

Vamos a verlo con un ejemplo de su uso:

```
    // Array de dos dimensiones

    #include <iostream>
    using namespace std;

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

        cout << "La nota del tercer alumno del grupo 1 es "
```

```
            << notas[0][2];

        return 0;
    }
```

Este tipo de tablas son las que se usan también para guardar matrices, cuando hay que resolver problemas matemáticos más complejos. Por ejemplo, un programa que pida al usuario los datos de dos matrices de 3x3 y luego muestre su suma podría ser así:

```
// Matrices, primer contacto

#include <iostream>
using namespace std;

int main()
{
    float matriz[2][3][3];
    float matrizSuma[3][3];
    int m, fila, columna;

    // Pedimos los datos de las dos matrices
    for (m=0; m<2; m++)
        for (fila=0; fila<3; fila++)
            for (columna=0; columna<3; columna++)
            {
                cout << "En la matriz " << m+1
                    << ", dime el dato de la fila " << fila+1
                    << " y la columna " << columna+1 << ": ";
                cin >> matriz[m][fila][columna];
            }

    // Calculamos la suma
    for (fila=0; fila<3; fila++)
        for (columna=0; columna<3; columna++)
            matrizSuma[fila][columna] = matriz[0][fila][columna]
                + matriz[1][fila][columna];


    // Y mostramos el resultado (puede salir un poco descolocado)
    cout << "La matriz suma es " << endl;
    for (fila=0; fila<3; fila++)
    {
        for (columna=0; columna<3; columna++)
            cout << matrizSuma[fila][columna] << " ";
        cout << endl;
    }

    return 0;
}
```

Cuidado: no podemos dar por sentado que los datos de un array tengan valor inicial 0, sino que pueden contener "basura" (lo que hubiera en cada posición de memoria anteriormente), de modo que este programa es incorrecto y puede mostrar resultados incorrectos:

```
// Matrices, lógica errónea (sin dar valor inicial)

#include <iostream>
using namespace std;

int main()
{
    float matriz[2][3][3];
```

```
    float matrizSuma[3][3];
    int m, fila, columna;

    // Pedimos los datos de las dos matrices
    for (m=0; m<2; m++)
        for (fila=0; fila<3; fila++)
            for (columna=0; columna<3; columna++)
            {
                cout << "En la matriz " << m+1
                    << ", dime el dato de la fila " << fila+1
                    << " y la columna " << columna+1 << ": ";
                cin >> matriz[m][fila][columna];
            }

    // Calculamos la suma
    for (m=0; m<2; m++)
        for (fila=0; fila<3; fila++)
            for (columna=0; columna<3; columna++)
                matrizSuma[fila][columna] += matriz[m][fila][columna];
    // La línea anterior es errónea: estamos dando por sentado
    // que la matriz suma contiene ceros, y quizá no sea así


    // Y mostramos el resultado (puede salir un poco descolocado)
    cout << "La suma es ";
    for (fila=0; fila<3; fila++)
    {
        for (columna=0; columna<3; columna++)
            cout << matrizSuma[fila][columna] << " ";
        cout << endl;
    }

    return 0;
}
```

Arrays indeterminados.

Si damos un valor inicial a un array, no será necesario que indiquemos su tamaño, porque el compilador lo puede saber contando cuantos valores hemos detallado, así:

```
// Arrays en los que no se indica el tamaño

#include <iostream>
using namespace std;

int main()
{
    int diasMes[] = // Días de cada mes
        {31, 28, 31, 30, 31, 30,
         31, 31, 30, 31, 30, 31};

    cout << "Los días de noviembre son: " << diasMes[10];
    return 0;
}
```