Laboratorio 04

Array y Matrices

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas usando array y matrices en el lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando array y matrices en el lenguaje de programación C++.

2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

3. Marco Teórico

3.1. Array, arreglos y vectores

Los vectores son un tipo de array (arreglos). Son, de hecho, un array de una sola dimensión y forman parte de la amplia variedad de estructuras de datos que nos ofrece C++, siendo además una de las principales y más útiles estructuras que podremos tener como herramienta de programación. Los vectores o arrays o arreglos de una dimensión (como los quieras llamar), son utilizados para almacenar múltiples valores en una única variable. En un aspecto más profundo, este tipo de arrays (vectores), permiten almacenar muchos valores en posiciones de memoria continuas, lo cual permite acceder a un valor u otro de manera rápida y sencilla. Estos valores pueden ser números, letras o cualquier tipo de variable que deseemos incluso tipos de datos complejos.

En múltiples ocasiones es necesario almacenar gran cantidad de información en una variable y a menudo sucede que no conocemos con exactitud la cantidad de datos que debemos almacenar, pero sabemos que sí sería más de uno, como por ejemplo almacenar las identificaciones de las personas ingresadas al sistema. Los vectores, arreglos (arrays) de una dimensión (ya sabes cómo es) son una estructura que nos permite solucionar este tipo de problemas.

3.1.1. Ejemplo de arrays de una dimensión o Vectores en C++

Imaginemos que queremos crear un programa con el cual podamos de algún modo almacenar los títulos y los autores de diferentes libros. El usuario es el encargado de suministrar la información de cada libro, así entonces, dado que es el usuario quien lo hace, nosotros no tenemos manera alguna de saber cuántos libros va querer él ingresar por medio de nuestro programa. El caso principal es que queremos almacenar en la memoria **el título** y **el autor** de TODOS y cada uno de los libros. Entonces ¿cómo crees que podrías hacer esto? Con lo que sabemos hasta hora, se nos podrían ocurrir un par de cosas.

Posible Solución: Usando vectores o matrices

Los arrays de una dimensión o los vectores han venido para ayudarnos en múltiples circunstancias similares a esta. Dado que un array, arreglo unidimensional o vector es capaz de almacenar múltiples valores en una misma variable, tenemos el elemento perfecto para almacenar la información de todos los libros, podremos crear un vector de un tamaño cualquiera capaz de contener en sí los nombres de los autores y otro con los títulos de los libros o alternativamente podríamos crear una matriz de dos columnas que contenga en la primera columna los autores y en la segunda los títulos; ambas soluciones son válidas y vamos a ver ambas, usando vectores en esta sección y usando matrices en la sección de matrices.

Nota: En C++, a diferencia de algunos otros lenguajes de programación, los vectores y las matrices presentan un "inconveniente" con el tamaño. Es decir, no es posible crear de una manera sencilla un vector capaz de almacenar una cantidad de información indefinida, es necesario ingresar con antelación la cantidad de datos (tamaño) que el vector o la matriz tendrá. Este problema se puede solucionar, con punteros en C++.

Muy bien, ahora que sabemos la gran utilidad de los vectores o arreglos (arrays) para nosotros, aprendamos más acerca de estos, veamos como declarar un vector, array o arreglo de una dimensión, como recorrer un vector y algunos ejemplos de esto en C++.

3.1.2. ¿Cómo declarar un Array de una dimensión o Vector en C++?

Para declarar un vector en C++, se deben seguir las mismas normas básicas que se siguen para declarar una variable cualquiera, con un pequeño cambio en la sintaxis. Para declarar un vector, arreglo o como lo quieras llamar, necesitaremos saber el tipo de los datos que irán al interior de este, es decir, serán número enteros, o números decimales o cadenas de texto, etc. necesitamos también, como siempre, un nombre para el vector y un tamaño máximo. La sintaxis para declarar un vector en C++ es la siguiente:

```
tipo_de_dato nombre_del_vector[tamaño];
```

Tenemos entonces, para declarar un vector en C++, debemos definirle un tipo de los datos, sea entero, float, string, etc., debemos darle un nombre y al interior de los corchetes "[]" debemos poner el tamaño máximo que tendrá el vector, es decir la cantidad máxima de datos que podrá contener (recuerda que en C++ esto es necesario hacerlo). Veamos un ejemplo en el cual pondré la declaración de varios vectores de diferentes tipos y tamaños en C++.

Declaración de un Array unidimensional o Vector en C++

```
int mi_vector1[10];
float mi_vector2[25];
string mi_vector3[500];
bool mi_vector4[1000];
char mi_vector5[2];
```

Veamos rápidamente que representa cada línea del código anterior.

Línea 1
Esta línea contiene la declaración de un vector llamado mi_vector1, el cual contendrá un máximo de 10 elementos de tipo entero.

Línea 2
Esta línea contiene la declaración de un vector llamado mi_vector2, el cual contendrá un máximo de 25 elementos de tipo float.

Línea 3
Esta línea contiene la declaración de un vector llamado mi_vector3, el cual contendrá un máximo de 500 elementos de tipo string.

Línea 4
Esta línea contiene la declaración de un vector llamado mi_vector4, el cual contendrá un máximo de 1000 elementos de tipo booleano.

Línea 5
Esta línea contiene la declaración de un vector llamado mi_vector5, el cual contendrá un máximo de 2 elementos de tipo char.

Ya que está claro cómo se declara un vector, vamos a ver cómo inicializarlo, es decir inicializar un vector en C++ o en otras palabras darle valores a un vector.

3.1.3. ¿Cómo inicializar un Array o Vector en C++?

En cuanto tenemos declarado un vector, es posible asignarle valores, evidentemente estos valores deben coincidir con el tipo de dato que le asignamos a dicho vector, no tendría sentido ingresar como valores de un vector cadenas de caracteres si el tipo de dato de dicho vector es numérico.

A continuación, se muestran formas distintas de inicializar un vector, todas son válidas, ya es cuestión de nuestras necesidades y conocimientos determinar cuál es útil y en qué momento.

Forma 1 de declarar un Array o Vector en C++

```
string vector[5] = {"5", "hola", "2.7", "8,9", "adios"};
                    0     1       2      3       4
```

Aquí hemos declarado un vector de tipo string tamaño 5 y lo hemos inicializado con diferentes valores, es necesario notar que cada valor va entre comillas dobles "" puesto que son strings. El valor inicial corresponde a la casilla o índice 0 y tiene el valor de "5", el índice 1 el valor es "hola" y el índice 4 el valor es "adiós", es importante notar que el primer índice de n array o vector no es el UNO sino que es el CERO.

Forma 2 de declarar un Array o Vector en C++

```
int vector2[] = {1,2,3,4,10,9,80,70,19};
```

Aquí hemos declarado un vector de tipo int y no especificamos su tamaño, si el tamaño no se especifica entre los corchetes, el vector tendrá como tamaño el número de elementos incluidos en la llave, para este caso es 9.

3.1.4. Particularidades de los Arrays, arreglos o Vectores en C++

Con C++, existen algunas particularidades, en cuanto a la declaración de vectores, que son importante destacar para que en momento de quizá caer en ellas comprender como podrían cambiar las cosas o básicamente en que consiste el error, veamos:

Particularidad 1 al momento de declarar o inicializar un Vector o Array en C++

```
int vector2[3];
vector2[3] = {1,5,10};
```

Dadas las características de C++, es fácil pensar que es factible crear o declarar un vector de un tamaño cualquiera y posteriormente inicializarlos de forma habitual como se muestra en este código, sin embargo, hacer esto es un error, si declaramos un vector y no lo inicializamos inmediatamente, no es posible inicializarlo de la forma que hemos visto, es decir entre llaves cada valor, como en la línea 2 del código anterior. La única forma de inicializar el vector, o, mejor dicho, darle valores a cada una de sus casillas, es hacerlo uno por uno, es decir darle un valor a la casilla cero (0) a la uno (1) y a la dos (2) (para un vector de tamaño 3). Por defecto, al declarar un vector sin ser inicializado, cada una de las casillas de este vector toma como valor el valor por defecto del tipo de variable, para el caso de los enteros (int) es -858993460. Así entonces para asignar valores a cada casilla lo hacemos así:

```
int vector2[3];
vector2[0] = 1;
vector2[1] = 3;
vector2[2] = 10;
```

Es importante notar en este código, que el número que va entre corchetes ya no indica tamaño (pues vector2 ya está declarado) sino que indica el índice o el numero de la casilla con la cual estaremos operando (recordemos que el primer índice es cero y no uno), en el código anterior, habíamos declarado un vector de tamaño 3, por lo cual debíamos asignar valores a los índices 0, 1 y 2.

Particularidad 2 al momento de declarar o inicializar un Vector o Array en C++

```
float vector3[5] = {10.5};
```

En C++ a la hora de inicializar un array, arreglo o vector, estamos acostumbrados a que si inicializamos inmediatamente después de declarar el vector, debemos poner la misma cantidad de elementos al interior de las llaves de manera que corresponda con el tamaño

del vector, pues bien, estos es lo más recomendable, sin embargo si ponemos una cantidad de elementos menor a la del tamaño real del vector, estamos queriendo decir que estos elementos toman los valores puestos entre las llaves y los demás serian cero, para el caso del código anterior el primer elemento (el del índice cero) va a tener un valor de 10.5 y los otros 4 elementos van a valer cero.

3.1.5. Obtener el valor de una casilla específica en un array en C++

Es muy común el caso en el que tenemos un vector con una enorme cantidad de elementos, sin embargo, de todos estos, solo nos interesa uno en especial y corremos con la suerte de saber cuál es su índice, sabiendo el índice de un elemento en un array es bastante sencillo obtener el valor de este:

```
float vector4[5] = {10.5, 5.1, 8.9, 10, 95.2}; //Array con 5 elementos
float numero5 = vector4[4]; //Para acceder al elemento 5, se usa el índice 4
float primerNumero = vector4[0]; //Para el primer elemento se usa el índice 0
```

Como podemos ver, para acceder a un valor específico conociendo el índice del elemento, solo basta con escribir dicho índice entre los corchetes "[ ]", recuerda que el índice comienza desde cero, así por lo tanto en un vector de 5 elementos (como el del ejemplo), el último elemento está en el índice 4 y el primer elemento del array en el índice 0.

3.1.6. Recorrer un Array o Vector en C++

Para obtener todos los datos que se encuentran al interior de un vector, es necesario recorrer el array o vector, para recorrerlo, se usa casi siempre un ciclo for, en algunos casos más específicos un ciclo while, pero generalmente el ciclo for es el ideal para esto, dado que conocemos el tamaño del array. La lógica de este procedimiento es la siguiente, el ciclo for comenzara desde cero e ira hasta el tamaño del vector, de modo que la variable de control que generalmente llamamos "i", será la que va a ir variando entre cero y el tamaño del array, de esta forma al poner la i al interior de los corchetes, estaremos accediendo al valor de cada casilla del vector y podremos hacer lo que sea necesario con dicho valor.

Nota: A veces no es posible determinar con facilidad el tamaño exacto de un vector, pero en C++ existen varias formas de determinar el tamaño de un array o vector fácilmente, aquí explicare un método. Cabe notar que este tamaño es el que ira como tope del ciclo for y sería equivalente a que nosotros mismos, en caso de saber el tamaño del vector, lo pongamos allí, sin embargo, como veremos en otra sección no siempre es posible saber con certeza el tamaño de un vector, es por esto que explico cómo hacerlo.

```
#include "iostream"

using namespace std;

int main()
{
   int edades[] = {1,2,9,8,16,32,9,50,36,20,1,87};
   int limite = (sizeof(edades)/sizeof(edades[0])); // sizeof devuelve la memoria ocupada
   for (int i = 0; i < limite; i++)
   {
         cout<<edades[i]<<endl;
   }
}
```

Vamos a ver de forma resumida en qué consiste y que hace cada una de estas líneas (las que están al interior del main).

Línea 1:
Tenemos en la primera línea la declaración de un vector que contiene las edades de 12 personas, notemos que entre los corchetes no se puso ningún número, pues no es necesario, ya que el vector tendrá el tamaño según la cantidad de elementos que declaremos entre las llaves, evidentemente si pusiéramos un 12 entre los corchetes, no habría ningún problema.

Línea 2:
En la segunda línea, tenemos la declaración del límite del ciclo o en otras palabras el tamaño del array. El tamaño de un array se puede calcular de varias formas, aquí lo obtenemos calculando el tamaño (en bytes) del array entero, dividido por el tamaño del primer elemento de dicho array (también en bytes).

El operador sizeof en C++, retorna el tamaño en bytes del elemento que se indica. En este caso, como es un array, indica el tamaño total de ese array en bytes. Como el tipo de dato int tiene un tamaño de 4 bytes, un array de 12 elementos de tipo int tendrá 48 bytes. Luego, el tamaño del primer elemento (o cualquiera de ellos) será 4. Así, 48 dividido entre 4 es 12, que es el tamaño de nuestro array. Para más detalles de esto, verifica la información sobre el operador sizeof.

Línea 3 a 6:
Desde la tercera línea hasta la sexta, tenemos entonces un ciclo for que comienza en cero y termina en el límite (es importante notar que la condición usada es estrictamente menor "<" y no menor o igual "<="), al interior de este ciclo, es donde accedemos a cada uno de los elementos del vector por medio de la sintaxis explicada anteriormente

Línea 5:
La quinta línea es quizá la más vital, aunque sin las demás no tendríamos nada. En esta línea, estamos accediendo a cada uno de los elementos del array de edades, un elemento por cada vuelta que da el ciclo, accedemos a cada elemento poniendo entre los corchetes la variable i, que es la que está cambiando a medida que el ciclo va girando, así estaremos accediendo a todos los elementos e imprimiéndolos por pantalla

Muy bien, llego el momento de afianzar nuestros conocimientos viendo un ejemplo. Ahora que tenemos claro como declarar un vector en C++, como recorrerlo y como acceder a sus datos, vamos a ver un ejemplo basado en el problema que planteé al inicio de esta sección (el de los libros).

3.1.7. Ejemplo 1 de Arrays o Vectores en C++

El problema es simple, queremos crear un programa con el cual podamos guardar los títulos y los autores de diferentes libros sin perder ninguno de ellos. El usuario es el encargado de suministrar la información de cada libro. Vamos a suponer que el usuario solo podrá ingresar un máximo de 5 libros, para así tener un tamaño de vector fijo. Veamos entonces como se haría esto:

```
#include "iostream"

using namespace std;

int main()
{
   char titulos[5];
   char autores[5];
   cout << "Por favor ingrese la siguiente información de los Libros: \n";
   for(int i = 0; i < 5; i++)
   {
      cout << "\n******* Libro " << i + 1 <<"********:\n";
      cout << "Titulo: ";
      cin >> titulos[i];
      cout << "Autor: ";
      cin >> autores[i];
   }
}
```

Estoy seguro de que a estas alturas comprendes bien qué hace cada una de estas líneas. En caso de que no comprendas nada de esto, te recomiendo leer nuevamente esta sección, la sección de ciclos o la sección de entrada y salida de datos.

Hay que considerar que el tipo de dato que estamos usando para los vectores de títulos y autores es char por lo tanto debes ingresar un único carácter cuando pruebes el algoritmo, pues de lo contrario el comportamiento será un poco extraño (aunque tiene su explicación). A continuación, haremos este mismo ejemplo, pero usando cadenas de texto completas (strings) para poder ingresar sin problemas más de una letra.

3.1.8. Ejemplo 1 mejorado

Vamos a solucionar el mismo problema, pero esta vez lo haremos bien. Vamos a utilizar cadenas de texto completas (string) de modo que al ingresar un título o un autor podamos poner textos completos:

```
#include "iostream"
#include "string"
using namespace std;


int main()
{
   string titulos[5];
   string autores[5];
   cout << "Por favor ingrese la siguiente información de los Libros: \n";
   for(int i = 0; i < 5; i++)
   {
      cout << "\n******* Libro " << i + 1 << "********:\n";
      cout << "Titulo: ";
      cin >> titulos[i];
      cout << "Autor: ";
      cin >> autores[i];
   }
}
```

Muy bien, tal como dije en el ejemplo anterior ahora ya podemos ingresar más de un carácter para el título y los autores (tal y como debe ser) y nuestro algoritmo funciona aún mejor. Puedes ver que los únicos cambios necesarios fueron importar la librería string y poner los tipos de datos como string en vez de char y eso solucionó por completo nuestro problema. Ten en cuenta que en versiones antiguas de compiladores usar la función cin para leer strings genera un error así que asegúrate de usar una versión reciente o usa entonces la función getline

3.1.9. Ejemplo perfeccionado

En la versión del ejemplo anterior tenemos un problema un poco delicado, y es que cuando ingresamos el título del libro o el autor de este con espacios, es decir, más de una palabra (habitualmente es así) el objeto cin interpreta esto como un fin de línea y no nos solicita el siguiente valor. Para solucionar esto haremos uso de la función getline (la que mencioné hace un momento) que nos permite solucionar este tipo de problemas. Vamos a solucionar el mismo problema, pero esta vez lo haremos bien. Vamos a utilizar cadenas de texto completas (string) de modo que al ingresar un título o un autor podamos poner textos completos:

```
#include "iostream"
#include "string"
using namespace std;

int main()
{
   string titulos[5];
   string autores[5];
   cout << "Por favor ingrese la siguiente información de los Libros: \n";
   for(int i = 0; i < 5; i++)
   {
      cout << "\n******* Libro " << i + 1 << "********:\n";
      cout << "Titulo: ";
      //cin >> titulos[i]; //No funciona con espacios
      getline(cin, titulos[i]);
      cout << "Autor: ";
      //cin >> autores[i]; //No funciona con espacios
      getline(cin, autores[i]);
   }
}
```

Como puedes apreciar, hemos reemplazado las líneas que usaban cin para leer los datos por la función getline(...) que recibe como primer argumento el flujo de entrada de cin y como segundo argumento la variable en la que queremos poner el valor.

3.2. Matrices en C++. Uso, declaración, y sintaxis de las matrices en C++

Las matrices o como algunos las llaman "arreglos multidimensionales" son una estructura de datos bastante similar a los vectores o arreglos. De hecho, una matriz no es más que una serie de vectores contenidos uno en el otro (u otros), es decir, una matriz es un vector cuyas posiciones son otros vectores. Hablemos con más detalle de esto para quedar más claros.

Primero, dejemos claro qué es una matriz. En términos generales, una matriz es una estructura conformada por filas y columnas, idealmente más de dos filas y columnas, de hecho, podemos decir que, si una "matriz" tiene una única fila o una única columna, entonces estamos hablando de un vector y no una matriz como tal.

La intersección de una fila y una columna de la matriz son las casillas y cada una de ellas podrá poseer información, simple o compleja (ya dependerá de nuestras necesidades).

Ahora, un vector posee una única fila (o columna, como lo quieras ver) y de este modo un grupo de vectores unidos conforman una matriz, es por esto que al comienzo se dijo que una matriz es un vector conformado por otra serie de vectores.

Viéndolo desde el punto de vista de la programación, una matriz es un vector cuyas posiciones (de la cero a la n) son, cada una de ellas, otro vector

3.2.1. Matrices en C++ un buen ejemplo

Vamos a retomar el ejemplo de la sección anterior donde teníamos la necesidad de almacenar los títulos y los autores de una serie de libros dada.

Una solución alternativa al problema, además de correcta y adecuada; es crear una matriz que almacenará en la primera columna los títulos, y en la segunda columna los autores; cada fila será entonces la información completa de un libro.

Para solucionar este problema, aprendamos primero algunas normas básicas para poder crear y usar matrices en C++.

3.2.2. ¿Cómo se crea una Matriz en C++?

Declarar una matriz en C++ es muy similar a la de un vector, se deben seguir las mismas normas para declarar una variable, pero una vez más con un pequeño cambio en la

sintaxis. Primero necesitaremos saber el tipo de los datos que irán al interior de este (números, decimales o cadenas de texto, etc.) necesitamos también, como siempre, un nombre para la matriz y un tamaño máximo tanto para las filas como para las columnas. La sintaxis para declarar una matriz en C++ es la siguiente:

```
tipoDato nombreMatriz[filas][columnas];
```

Nota: Recuerda que, en C++, no es posible crear de una manera sencilla un vector (y por ende una matriz) capaz de almacenar una cantidad de información indefinida, es necesario ingresar con antelación la cantidad de datos (filas y columnas) que la matriz tendrá.

Tenemos entonces, como podrás ver, que la sintaxis es casi la misma excepto que hemos añadido un par de corchetes "[]" más esta vez y al interior de éstos debemos poner el número de filas y columnas máximas de la matriz, respectivamente. Veamos un ejemplo en el cual pondré la declaración de varias matrices de diferentes tipos y tamaños en C++.

Declaración de una matriz en C++

```
int miMatriz1[10][5];
float miMatriz2[5][10];
string miMatriz3[15][15];
bool miMatriz4[1000][3];
```

Veamos rápidamente que representa cada línea del código anterior.

Línea 1
Esta línea contiene la declaración de una matriz llamada miMatriz1 que tendrá 10 filas y 5 columnas y cada una de las 50 casillas tendrá datos de tipo entero.

Línea 2
Esta línea contiene la declaración de una matriz llamada miMatriz2 que tendrá 5 filas y 10 columnas y cada una de las 50 casillas tendrá datos de tipo flotante.

Línea 3
Esta línea contiene la declaración de una matriz llamada miMatriz3 que tendrá 15 filas y 15 columnas (una matriz cuadrada) y cada una de las 225 casillas tendrá datos de tipo string.

Línea 4
Esta línea contiene la declaración de una matriz llamada miMatriz4 que tendrá 1000 filas (sí, leíste bien) y 3 columnas y cada una de las 3000 casillas (también leíste bien, tres mil casillas) tendrá datos de tipo booleano.

Ya que está claro cómo se declara una matriz, vamos a inicializarla, es decir darle un valor a cada casilla, según su tipo de dato.

3.2.3. ¿Cómo inicializar una matriz en C++?

En cuanto tenemos declarado una matriz, es posible asignarle valores a cada una de sus casillas, evidentemente estos valores deben coincidir con el tipo de dato que le asignamos a dicha matriz

A continuación, se muestran como inicializar una matriz:

```
int miMatriz1[2][2] = {{1,2},{3,4}};
```

Aquí hemos declarado una matriz de tipo int de dos filas y dos columnas y la hemos inicializado con diferentes valores. El valor inicial corresponde a la casilla 0,0 (fila cero, columna cero) y tiene el valor de 1, en la fila cero columna uno tenemos el valor de 2, en la fila uno columna cero el valor de 3 y finalmente en la fila uno columna uno el valor de 4. Es importante notar que el primer tanto la fila como la columna comienzan desde cero y no desde uno, por esto la primera casilla corresponde a la fila y columna cero.

3.2.4. Obtener el valor de una casilla específica

Para acceder al valor de una casilla nuevamente haremos uso de los corchetes, pero esta vez no para declarar tamaños (porque eso ya lo hicimos) sino para indicar posiciones (fila y columna).

```
int miMatriz1[2][2] = {{1,2},{1,1}}; //Matriz con 4 elementos
int fila1Casilla1 = myMatriz[1][1]; //Para acceder a la casilla 1,1 se usan dichos índices
int primerNumero = myMatriz[0][0]; //La primer casilla siempre será la de la fila 0 columna 0
```

Para acceder a un valor específico conociendo el índice de la casilla, solo basta con escribir dicho índice entre los corchetes "[][]", recuerda que el índice comienza desde cero, así por lo tanto en una matriz de vector de 2 por 2 (como el ejemplo), el último elemento está en el índice 1 y el primer elemento en el índice 0.

3.2.5. Recorrer una matriz en C++

Para obtener todos los datos que se encuentran al interior de una matriz, debemos acceder a cada posición y esto se hace fácilmente con dos ciclos for (anidados). La lógica de este procedimiento es la siguiente, el primer ciclo for comenzará desde cero e ira hasta el número de filas, de modo que la variable de control que generalmente llamamos "i", será la que va a ir variando entre cero y el tamaño del array multidimensional (matriz), de esta forma al poner la i al interior de los corchetes, estaremos accediendo al valor de cada fila y el segundo ciclo irá de cero al número de columnas y normalmente se usa la variable llamada j para acceder a cada columna:

Nota: En el siguiente código uso una forma sencilla y rápida de obtener la cantidad o número de filas de una matriz y también cómo obtener el número o cantidad de columnas de una matriz. Ten en cuenta que esto es importante, pues a veces no tenemos la certeza del tamaño de la matriz.

```
#include <iostream>

using namespace std;

int main()
{
   int edades[3][2] = {{1,2},{9,8},{14,21}};
   int filas = (sizeof(edades)/sizeof(edades[0]));
   int columnas = (sizeof(edades[0])/sizeof(edades[0][0]));
   for (int i = 0; i < filas; i++)
   {
      for (int j = 0; j < columnas; j++)
      {
         cout<<edades[i][j]<<endl;
      }
   }
}
```

Vamos a ver de forma resumida en qué consiste y que hace cada una de estas líneas

Línea 1:
Tenemos en la primera línea la declaración de una matriz que contiene las edades de tres parejas de personas y asignamos cada uno de los valores.

Líneas 2 y 3:
En estas líneas, tenemos la declaración del número de filas y columnas de la matriz, que serán el límite del primer y segundo ciclo, respectivamente.

Líneas 4 a 7:
Aquí, tenemos entonces un ciclo for que comienza en cero y termina en el número de filas y luego tenemos otro ciclo for (anidado) que irá de cero hasta el número de columnas (es importante notar que la condición usada en ambos ciclos es estrictamente menor "<" y no menor o igual "<="), al interior del segundo ciclo, es donde accedemos a cada una de las casillas de la matriz usando los corchetes.

Línea 8:
La octava línea es quizá la más vital, aunque sin las demás no tendríamos nada. En esta línea, estamos accediendo a cada una de las casillas de la matriz, fila por fila y columna por columna. Accedemos a cada elemento poniendo entre los corchetes la variable i y j, que son las que están cambiando a medida que los ciclos van "girando", así estaremos accediendo a todos los elementos e imprimiéndolos por pantalla por medio de cout.

3.2.6. Ejemplo de Matrices en C++

El problema es simple, queremos crear un programa con el cual podamos guardar los títulos y los autores de diferentes libros sin perder ninguno de ellos. El usuario es el encargado de suministrar la información de cada libro. Vamos a suponer que el usuario solo podrá ingresar un máximo de 5 libros, para así tener un tamaño de vector fijo. Veamos entonces cómo se haría esto usando matrices:

```
#include "iostream"
#include "stdio.h"
#include "string"

using namespace std;

int main()
{
   string libros[5][2];
   cout << "Por favor ingrese la siguiente información de los Libros: \n";
   string titulo ,autor;
   for(int i = 0; i < 5; i++)
   {
      cout << "\n******* Libro " << i + 1 << "********:\n";
      cout << "Titulo: ";
```

```
        getline(cin,titulo);
        cout << "Autor: ";
        getline(cin,autor);
        libros[i][0] = titulo;
        libros[i][1] = autor;
    }

    system("pause");

    return 0;
}
```

Notar que, en el código anterior, debido a que tenemos la completa certeza de sólo usar dos columnas, no es necesario usar otro ciclo for (de hecho, eso complicaría todo) basta con poner de manera explícita que el valor del título va en la columna cero y el del autor en la columna uno.

En este caso tenemos una matriz con una cantidad de fila que dependen de la cantidad de libros y dos columnas. Para el ejemplo, decidimos tener 5 posibles libros. Donde cada uno tendrá su respectivo título y autor. Así entonces, en la columna cero (0) iría el título y en la columna uno (1) el autor.

Por cada libro que queremos agregar, debemos especificar su título y su autor. Por ello, la segunda posición que se especifica en esta matriz de libros siempre será 0 o 1, según sea el caso. Mientras que en la primera posición usamos la variable i del ciclo, que va a variar de 0 a 4, para un total de cinco libros.

4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Escribir un programa donde se pueda ingresar los datos de tres personas, como el nombre, apellido, edad y DNI y luego lo muestre por pantalla.
2. Hacer un array unidimensional que acepte ocho números enteros. Luego le pregunte al usuario que ingrese un número a buscar, implementar una función que busque ese número, si lo encuentra, debe retornar un valor de verdaderos, en caso contrario, retornar falso.
3. Hacer un array 5x3 que acepte números enteros ingresados por el usuario. Al final, debe mostrar la suma de todos los números que estén en una fila par.
4. Implementar un programa que rellene un array con los números primos comprendidos entre 1 y 100 y los muestre en pantalla en orden descendente.

5. Implemente un programa que gestione los datos de stock de una tienda de abarrotes, la información a recoger será: nombre del producto, precio, cantidad en stock. La tienda dispone de 10 productos distintos. El programa debe ser capaz de:
    a. Dar de alta un producto nuevo.
    b. Buscar un producto por su nombre.
    c. Modificar el stock y precio de un producto dado.
6. Escribe un programa que pida nueve números enteros y los almacene en una matriz 3x3. Calcula la suma de los números de cada fila y mostrar por pantalla el número de fila con mayor suma.
7. Escribe un programa, que trabajando mediante funciones, presente un menú al usuario, mueva las columnas a la derecha o izquierda, asimismo mueva las filas para arriba o para abajo.

5. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.

2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.

3. El nombre del archivo (comprimido como el documento PDF), será su LAB04_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB04_GRUPO_A _2022123_PEDRO_VASQUEZ).

4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB04_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB04_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.