Arreglos

- Un vector (arreglo de una dimensión) se declara en C++ usando [ ]

```
int v[10]; /* declara un vector de enteros, de 10
              elementos */
```

- Los subíndices en C++ comienzan en 0 por lo tanto si quiero mostrar los 10 valores del vector (supongamos que en algún momento ya lo cargamos) podemos hacerlo así

```
for (i = 0; i < 10 ; ++i)
        cout << i << ": " << v[i] << endl;
```

Arreglos

- Un vector también puede iniciarlizarse al momento de declararlo

```
int v[5] {1, 6, 8, 3, 9};
int w[] {3, 12, 7};
int x[10] {0};
int z[10] {};
```

- Notar que en el caso de **w** no se indicó el tamaño explícitamente, pero al iniciarlo con 3 elementos el compilador lo genera de ese tamaño.
- También podemos indicar el tamaño e inicializar una cantidad de valores menor al tamaño, en cuyo caso los valores no inicializados explícitamente, se inicializan, implícitamente, en cero. Por ejemplo el vector **x** tendrá sus 10 elementos inicializados en cero. Lo mismo ocurre con z.

Ejemplo básico

```
#include <iostream>
using namespace std;

int main()
{
        const int dim = 5;
        int v[dim];
        for (int i = 0; i < dim; ++i) {
                cout << "Ingrese el elemento " << i << ": ";
                cin >> v[i];
        }
        cout << "--------" << endl
             << "Los elementos ingresados son:" << endl;
        //Muestra desde v[0] hasta v[4]
        for (int i = 0; i < dim; ++i)
                cout << "v[" << i << "]= " << v[i] << endl;
        return 0;
}
```

Salida

Ingrese el elemento 0: 5
Ingrese el elemento 1: 7
Ingrese el elemento 2: 150
Ingrese el elemento 3: 22
Ingrese el elemento 4: 3

Los elementos ingresados son:
v[0]= 5
v[1]= 7
v[2]= 150
v[3]= 22
v[4]= 3

Arreglos y Funciones

- Los arreglos pueden pasarse como parámetros a funciones, pero tienen particularidades con respecto a otros tipos de variables
  - No pueden pasarse por valor
  - Al pasar un arreglo “es como si” se lo pasara por referencia. Es decir que la función modificará el arreglo que se le pasa
  - No se puede devolver un arreglo (más sobre esto en el apunte sobre punteros)
- La función que recibe el arreglo desconoce su dimensión. Es necesario pasar un parámetro adicional para informarla

Ejemplo de función con arreglo

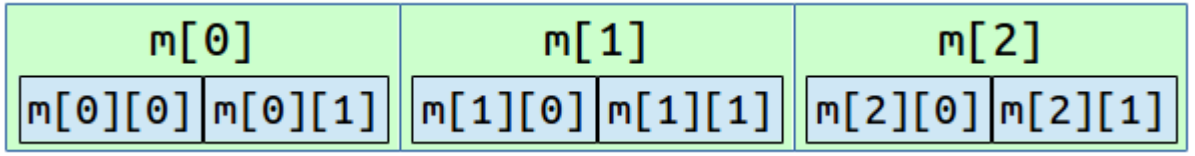
```
#include <iostream>
using namespace std;

void doble(int vec[], int dim)
{
        for (int i = 0; i < dim; ++i)
                vec[i] *= 2;
}

int main()
{
        const int dimv = 5;
        int v[dimv] {1, 2, 3, 4, 5};
        doble(v, dimv);
        cout << "Valores duplicados:" << endl;
        for (int i = 0; i < dimv; ++i)
                cout << "v[" << i << "]= " << v[i] << endl;
        return 0;
}
```

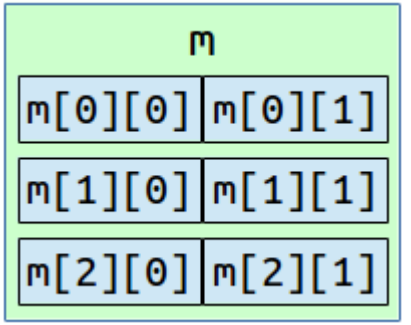
Salida

```
Valores duplicados:
v[0]= 2
v[1]= 4
v[2]= 6
v[3]= 8
v[4]= 10
```

Matrices

- No hay matrices como tales, solo vector de vector. Por ejemplo si declaramos
  - `double mat[3][2];`
- Entonces `mat` es un vector de 3 elementos, donde cada uno de ellos es a su vez un vector de 2 doubles.
- Como cada elemento es un vector de 2 doubles implica que en memoria se guardará la “matriz” por filas (cada fila es un elemento del “vector” `mat`)





Matrices - Uso

```
#include <iostream>

using namespace std;

int main()
{
        const int dfil {3};
        const int dcol {4};
        //Laves internas no necesarias, pero conceptualmente más claras
        int mat[dfil][dcol] { {0 ,  1,  2,  3},
                              {10, 11, 12, 13},
                              {20, 21, 22, 23} };

        for (int i = 0; i < dfil; ++i) {
                for (int j = 0; j < dcol; ++j)
                        cout << "mat[" << i << "][" << j << "] = "
                             << mat[i][j] << endl;
                cout << endl;
        }
        return 0;
}
```

Salida

```
mat[0][0] = 0
mat[0][1] = 1
mat[0][2] = 2
mat[0][3] = 3

mat[1][0] = 10
mat[1][1] = 11
mat[1][2] = 12
mat[1][3] = 13

mat[2][0] = 20
mat[2][1] = 21
mat[2][2] = 22
mat[2][3] = 23
```

Matrices - Funciones

- Con un vector no paso la dimensión, esto no afecta a la función porque sabe el tipo de dato, y por tanto el tamaño del mismo, que es lo que necesita para calcular donde se almacena el elemento siguiente.
- Con una matriz, en realidad arreglo de arreglo, necesito pasar la dimensión de las columnas. El compilador lo necesita para saber donde se comienza a almacenar la fila siguiente, que viene siendo el elemento siguiente del “primer arreglo”

Matrices - Funciones

```
//Solo la llave más interna sin dimensión, todas las demás deben levar si o si
void mostrar(int m[][4], int filas) //Válido pero menos flexible: m[3][4]
{
        const int cols {4}; //Solo por prolijidad
        for (int i = 0; i < filas; ++i) {
                for (int j = 0; j < cols; ++j)
                        cout << "m[" << i << "][" << j << "] = "
                             << m[i][j] << endl;
                cout << endl;
        }
}

int main()
{
        const int dfil {3};
        const int dcol {4};
        //Sin las llaves asigna en el mismo orden que llena la memoria
        int mat[dfil][dcol] { 0 , 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23 };

        mostrar(mat, dfil);
        cout << "=========================================" << endl << endl;
        mostrar(mat, dfil - 1); //Si no quiero todas las filas
        return 0;
}
```

Funciones - Salida

```
m[0][0] = 0
m[0][1] = 1
m[0][2] = 2
m[0][3] = 3

m[1][0] = 10
m[1][1] = 11
m[1][2] = 12
m[1][3] = 13

m[2][0] = 20
m[2][1] = 21
m[2][2] = 22
m[2][3] = 23

=======================================

m[0][0] = 0
m[0][1] = 1
m[0][2] = 2
m[0][3] = 3

m[1][0] = 10
m[1][1] = 11
m[1][2] = 12
m[1][3] = 13
```

Licencia