

Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología
Sede: Oro Verde



FUNDAMENTOS DE PROGRAMACIÓN

RESUMEN DE CONTENIDOS N°10

Arrays

Introducción

En esta unidad se aprenderá a emplear una nueva estructura de datos en C++: Arrays.

En primer lugar, se procederá a definir el concepto “Estructura de datos”, y se conocerán los arreglos (arrays) unidimensionales y bidimensionales. Luego se aprenderá cómo declarar y definir arreglos estáticos en C++, cómo se almacenan en memoria y cómo se acceden para ser modificados o utilizados. Además, se analizarán sus ventajas y limitaciones.

En C++ es posible combinar estructuras de datos de acuerdo a las necesidades del caso a resolver. Uno de los objetivos de esta unidad, es aprender a elegir la mejor opción para la resolución de un problema determinado.

Los conceptos que introduciremos, son válidos y adaptables a cualquier lenguaje de programación. Pero en esta unidad, se hará hincapié en su forma de uso específicamente en C++.

Estructura de datos

Hasta ahora hemos visto el uso de datos simples o elementales, o sea variables en las cuales podíamos alojar un único valor por vez, independientemente del tipo que fuera.

Para poder diseñar ciertos algoritmos o programas, es necesario leer varias veces uno o más datos o bien reservar varios valores, con el consiguiente problema de uso reiterado de procesos de lectura/escritura que son las acciones más lentas de la algorítmica, en un caso, o bien el uso de demasiados nombres de variables, en el otro.

Veamos el siguiente ejemplo, codificado en C++:

Ejemplo 1: Una empresa recibe información mensual sobre las ventas de cada una de sus 10 sucursales y desea obtener un listado de aquellas cuyas ventas superan el promedio de las mismas.

Programa en C++:

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

int main() {
    int x=0;
    float tot;
    float venta, prom;

    tot=0;
    do
```

```
{
    cout<<"MONTO DE VENTA: $ ";
    cin>>venta;
    x++;
    tot=tot+venta;
}while (x!=10);

prom=tot/10;
cout<<"PROMEDIO DE VENTAS: $ "<<prom<<endl;
cout<<endl;
x=0;
do
{
    cout<<"MONTO DE VENTA: $ ";
    cin>>venta;
    x++;
    if (venta>prom)
        cout<<"Monto superior al promedio: $ "<<venta<<endl;
}while (x!=10);

return 0;
}
```

Este programa es ineficiente por el hecho de leer dos veces el mismo conjunto de datos. Una forma alternativa de resolver el problema es definiendo 10 variables distintas cuyos valores corresponden a las ventas de cada sucursal. Pero ¿cómo plantear el algoritmo para 100 sucursales?

Estos problemas plantean la necesidad de extender el concepto de dato a un conjunto o grupo de ellos. Surge así la idea de **Estructura de Datos**.



Una **Estructura de Datos** es un conjunto de datos homogéneos que se tratan como una sola unidad y bajo un mismo nombre colectivo.

Existen distintas estructuras de datos que se diferencian por la forma en que se relacionan los datos primitivos, por el tipo de los mismos y por la manera de referenciar cada dato.

Al tratar la solución de un problema en particular, debemos analizar la organización de sus datos y las relaciones entre ellos, que definen distintas estructuras de datos.

La primera estructura de datos que conoceremos es la llamada: **Arreglo o Array**

Definición de arreglo



Se define **arreglo** como la estructura de datos formada por una secuencia de elementos homogéneos (de igual tipo). Cada elemento tiene una posición relativa -que puede ser establecida por uno o más índices- dentro de la secuencia.

Análisis de la definición

- Decimos que es una **estructura de datos**, lo cual significa que se trata de un conjunto de datos o valores, donde cada uno de ellos se almacena en un lugar de memoria diferente, uno a continuación del otro, pero todo el conjunto tiene un único nombre. Cada dato del arreglo se llama **elemento**.
- **Homogéneos (de igual tipo)**: todos los datos del arreglo deben ser del mismo tipo: numérico, carácter o lógico. En C++, los elementos deberán ser int, float, char, etc.

Todo arreglo tiene asociado:

- **Nombre (identificador)**: para formar el nombre se siguen las mismas reglas que en los nombres de variables
- **Tipo**: está determinado por el tipo de los elementos que los componen (numérico, carácter o lógico)
- **Dimensión**: valor o valores numéricos que indican la cantidad máxima de elementos que componen el arreglo.
- **Índice**: variable, constante o expresión numérica que hace referencia a una posición del arreglo, y que toma todos los valores dentro del rango de la dimensión.



Dimensión y longitud de un arreglo

Los arreglos son estructuras estáticas que requieren establecer la cantidad de elementos que pueden almacenar. Pero en la práctica no siempre se conoce la cantidad exacta de elementos a asignar al arreglo; entonces se debe dimensionar por exceso.

El valor empleado para declarar el arreglo se denomina dimensión **D** y la cantidad real de elementos utilizados es la longitud **L** del arreglo.

Puede suceder entonces que:

D = L, donde coincide el tamaño asignado con la cantidad de elementos utilizados.

D > L, donde el tamaño asignado (dimensión por exceso) es mayor a la cantidad de elementos utilizados.

Lo que no se permite aquí, es que **L > D**, es decir, que no se haya reservado lugar suficiente para almacenar todos los datos.

Características de los arreglos estáticos en C++

- Un arreglo es una colección de datos relacionados, de igual tipo e identificados bajo un nombre genérico único.
- La estructura completa ocupa un segmento de memoria único y sus elementos se almacenan en forma contigua.
- Para procesar un elemento del arreglo, se debe especificar el nombre de la estructura y uno o más índices que determinan la posición del elemento.
- Se debe establecer en la declaración, la cantidad de elementos (dimensión) que puede tener como máximo el arreglo en el programa.
- Se puede emplear como índice cualquier expresión que arroje un número entero positivo dentro del rango establecido (dimensión) para dicho índice.

- El índice que determina la posición de los elementos de un arreglo comienza siempre con cero.
- C++ admite operar fuera del rango preestablecido por la dimensión, pero con resultados impredecibles ($L > D$)

ARREGLO UNIDIMENSIONAL, LINEAL O VECTOR.

Un **arreglo es unidimensional** cuando la referencia a uno de sus elementos se realiza a través de un único valor llamado **índice**, que determina la posición dentro del arreglo. A cada elemento del arreglo se puede acceder en forma directa indicando el nombre del vector seguido del índice entre corchetes

En forma general diremos que un valor del arreglo se indica de la siguiente forma:

A[i]

Donde:

- **A** nombre del arreglo
- **i** índice del arreglo. El mismo puede ser una: constante, variable o expresión, de tipo numérica entera positiva. Además, debe tomar valores dentro del rango de la dimensión.

Representación gráfica

Consideremos un vector **V** de 5 elementos: 3, 6, 9, 12, 15.

La representación gráfica del mismo, puede variar dependiendo del lenguaje de programación utilizado. El concepto será el mismo, lo que deberá considerarse, es el valor que se otorga a la primera posición del arreglo.

En C++, a diferencia de otros lenguajes, la primera posición del arreglo es la 0, y la última, D-1, siendo D la dimensión del mismo.

Gráficamente en C++, el vector V mencionado, se representaría de la siguiente manera:



- **Dimensión del vector V= 5.**
- **V[0]** guarda el valor 3, es decir el elemento ubicado en la posición 1 del vector.
- **V[2]** almacena el número 9, es decir el elemento ubicado en la posición 3 del vector.
- Si en un algoritmo indicamos **V[k]** estaremos referenciando al elemento que ocupa la posición k dentro del vector y para ello la variable k debe tener valor.

Ejemplo: En C++, supóngase el caso de un arreglo lineal **m** declarado como: **int m [200];**

m[0]	m [1]	m [2]	m [3]	m [4]	m [5]	m [6]	m [199]
23	56	71	19	33	90	48	74

Declaración e inicialización de un arreglo lineal

Los arreglos estáticos en C++ se declaran y definen como cualquier otra variable. Usualmente se establece la dimensión de la estructura, es decir, la cantidad máxima de elementos que puede contener en el programa.

```
const m = 100;
.....
int z[m];
char mensaje[30];
float lista[200];
```

Para inicializar un arreglo en el programa, se puede “recorrer” cada elemento del arreglo para asignar los valores correspondientes, o bien, pueden inicializarse los elementos en la misma declaración, enumerando la lista de datos a asignar. Esto último, puede efectuarse de la siguiente manera:

```
// declaración e inicialización del array de caracteres z
char z[7] = {'J', 'M', 'L', 'P', 'Q', 'W', 'H'}

/* declaración de un array t con algunos valores iniciales; el resto
de los elementos se inicializan con valor incierto*/
float t[5] = {7.1,8.4}

/* El siguiente arreglo no tiene dimensión. Por defecto C++ asume
como tamaño la cantidad de valores iniciales. En este caso,
considerará 5, como la dimensión del arreglo */
int m[]={34, 56, 20, 41, 72}
```



Observación: Deberá recordarse que NO SE PERMITE DEFINIR Y DECLARAR UN ARREGLO CON UNA VARIABLE. SIEMPRE DEBE HACERSE CON CONSTANTES NUMÉRICAS (un número).



Observación: Deberá recordarse que C++ considera que el primer elemento de un array, se ubica en la posición 0, por tanto una declaración `int x[3]` implica que se podrá referenciar en el programa a los elementos `x[0]`, `x[1]`, `x[2]`.

Operaciones con arreglos unidimensionales

Las operaciones que podemos realizar con arreglos son:

- **Asignación**
- **Lectura**
- **Escritura**
- **En expresiones**

Asignación

En general podemos asignar valores a los elementos de un vector siempre que indiquemos el nombre y la posición del elemento al cual queremos asignar el valor.

En general:

$$H[j] = n$$

Donde:

- **H** es el nombre del arreglo
- **j** indica la posición
- **n** indica el valor que deseamos asignar

Consideremos un vector **H** de dimensión 5 al que queremos asignar el número 15 en la posición 3; lo haremos: **H[2] = 15**

Lectura

Si queremos asignar un valor leído como elemento de un vector, podemos hacerlo:

$$\text{cin} >> H[i]$$

Donde la variable **i** deberá tener valor entero positivo. Se asigna el valor leído al elemento del vector en la posición **i**.



Observación: la asignación de valores a un vector se hace elemento por elemento. Así por ejemplo, en C++, para guardar en un vector 3 números enteros leídos, necesitamos un vector de 3 posiciones; sea por ejemplo el vector **vec** donde:

- en **vec** [0] colocaremos el 1er valor leído
- en **vec** [1] colocaremos el 2do valor leído
- en **vec** [2] colocaremos el 3er valor leído

FORMA 1

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

int main() {
    int vec[10], i, dat;
    int n=5;

    for (i=0; i<n; i++)
    {
        cout<<"Dato: ";
        cin>>dat;
        vec[i]=dat;
    }

    return 0;
}
```

FORMA 2

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

int main() {
    int vec[10], i;
    int n=5;

    for (i=0; i<n; i++)
    {
        cout<<"Dato: ";
        cin>>vec[i];
    }

    return 0;
}
```

Si los valores leídos son 150, 320, 30, 84 y 15 el resultado final en ambos casos será:

150	320	30	84	15
0	1	2	3	4

En adelante, usaremos directamente la **FORMA 2**.

Escritura

Si queremos escribir los valores de los elementos de un vector, debemos hacerlo uno a uno, indicando el nombre del arreglo y la posición del elemento que queremos mostrar.

cout<<H[i]

Para mostrar los valores de cada uno de los elementos del vector vec del ejemplo anterior, haríamos:

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

int main() {
    int vec[10], i;
    int n=5;
    //Carga del vector vec
    for (i=0; i<n; i++)
    {
        cout<<"Dato: ";
        cin>>vec[i];
    }

    //Mostrar contenido del vector
    cout<<endl;
}
```



```
    for (i=0; i<n; i++)  
    {  
        cout<<"Posicion "<<i<<": "<<vec[i]<<endl;  
    }  
    return 0;  
}
```



Observación: Deberá recordarse que en C++, en la sentencia **cout**, si se desea que la los elementos del vector se muestren uno debajo del otro, debe forzarse la ubicación del cursor a la siguiente línea (endl o '\n'). Caso contrario, los elementos del vector, se mostrarán uno a continuación del otro, en la misma línea.

En expresiones

El tipo de arreglo y en consecuencia el tipo de sus elementos determina qué operaciones se pueden realizar con ellos.

Las operaciones que se pueden realizar son las mismas que con variables vistas hasta el momento, teniendo en cuenta su tipo.

Siguiendo el ejemplo anterior con el vector vec, supongamos que ahora queremos hallar la suma de los elementos mayores que 100 e informarla:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int vec[10], i;  
    int n=5, sum=0;  
    //Carga del vector vec  
    for (i=0; i<n; i++)  
    {  
        cout<<"Dato: ";  
        cin>>vec[i];  
    }  
  
    //Determinar los valores mayores que 100 y sumarlos  
    cout<<endl;  
    for (i=0; i<n; i++)  
    {  
        if (vec[i]>100)  
            sum=sum+vec[i];  
    }  
    cout<<"La suma de los valores mayores a 100 es: "<<sum;  
    return 0;  
}
```

A continuación, retomaremos el Ejemplo 1 planteado en la página 1, y se lo resolverá con el uso de vectores.

Ejemplo 1: Una empresa recibe información mensual sobre las ventas de cada una de sus 10 sucursales y desea obtener un listado de aquellas cuyas ventas superan el promedio de las mismas.

```
#include <iostream>
using namespace std;
int main() {
    int i;
    float tot;
    float venta[10], prom;

    tot=0;
    for (i=0; i<10; i++)
    {
        cout<<"MONTO DE VENTA: $ ";
        cin>>venta[i];
        tot=tot+venta[i];
    }

    prom=tot/10;
    cout<<"PROMEDIO DE VENTAS: $ "<<prom<<endl;
    cout<<endl;
    cout<<"VENTAS MAYORES AL PROMEDIO"<<prom<<endl;
    for (i=0; i<10; i++)
    {
        if (venta[i]>prom)
            cout<<"Monto superior al promedio: $ "<<venta[i]<<endl;
    }
    return 0;
}
```



Observación: Si en un programa $L < D$ habrá posiciones vacías en la estructura, lo cual no representa un inconveniente. Obviamente, para la realización de un cálculo o simplemente para visualizar el contenido del arreglo, se deberá tener cuidado en recorrer el arreglo en toda su longitud L y no en toda su dimensión D .

Inicialización de un arreglo lineal utilizando la función RAND()

La función `rand()` pertenece a la librería `stdlib.h` y es una rutina matemática para generar números pseudoaleatorios enteros entre 0 y `RAND_MAX`.

Para utilizarla, deberá incluirse en el encabezado la librería correspondiente: `#include <cstdlib>`.

El siguiente ejemplo, muestra la aplicabilidad de la función. Se generará un vector de 100 posiciones, con valores desde 0 a 50. Este límite lo da el agregado de `%51`, como límite superior. Si se omite esta acción, el límite máximo lo otorga `RAND_MAX`.

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

int main() {
    int x[100];
    for (int i=0; i<99; i++)
    {
        x[i]= rand( )%51;
    }
    //mostrar el contenido del vector
    cout<<"Datos generados"<<endl;
    for (int i=0; i<99; i++)
    {
        cout<<x[i]<<endl;
    }
    return 0;
}
```

ARREGLO BIDIMENSIONAL, TABLA O MATRIZ.



Un **arreglo es bidimensional** cuando sus elementos están dispuestos en filas y columnas y la referencia a cada uno de sus elementos se realiza a través de 2 (dos) índices, que determinan la posición del mismo dentro del arreglo.

Los arreglos bidimensionales cumplen las mismas características que las vistas para vectores. Así, todo arreglo bidimensional o matriz tiene un **nombre**, un **tipo** (carácter, numérico o lógico según el tipo de sus datos), una **dimensión**.

En un arreglo bidimensional, cada elemento se identifica con el nombre del arreglo seguido entre corchetes de los 2 índices. El primer índice indica la fila y el segundo la columna.

En forma general, un elemento de la matriz se define como:

$A[i][j]$

Donde:

- **A** nombre del arreglo
- **i** índice del arreglo que referencia a la **fila**. El mismo puede ser una: constante, variable o expresión, de tipo numérica entera positiva. Además, debe tomar valores dentro del rango de la dimensión.
- **j** índice del arreglo que referencia a la **columna**. El mismo puede ser una: constante, variable o expresión, de tipo numérica entera positiva. Además, debe tomar valores dentro del rango de la dimensión.

Representación gráfica

Consideremos una matriz MAT compuesta de 5 filas y 3 columnas, es decir que contiene 15 datos: 2, 6, 9, 15, 27, 7, -2, 13, 3, 35, 3, 52, 10, 4, 85. Gráficamente la podemos representar como:

2	6	9
15	27	7
-2	13	3
35	3	52
10	4	85

FILAS

COLUMNAS

- **MAT[2,2]** es el elemento ubicado en la intersección de la fila 3 y la columna 3 y su contenido es el valor 3.
- **MAT[4,0]** es el elemento ubicado en la intersección de la fila 4 y la columna 0 y su contenido es el valor 10.
- En general: **MAT[f,c]** indica el elemento ubicado en la intersección de la fila f y la columna c; por lo tanto f y c deben representar un valor determinado.

Dimensión de una Matriz

La definición dada para el caso de un vector es válida para un arreglo bidimensional: es la máxima cantidad de elementos que puede tener.

Esta dimensión se calcula como el producto entre el número máximo de filas y el número máximo de columnas.

En el ejemplo anterior, la dimensión de la matriz MAT es 15.

Ejemplo: En C++, obsérvese gráficamente el caso de un arreglo **m** de números de punto flotante, compuesto por 5 filas y por 12 columnas: **float m[5][12];**

	0	1	2	3	11
0	5.34	6.71	4.22	12.02	7.92
1	1.55	1.16	1.65	11.34	1.16
2	6.87	3.12	6.37	13.00	3.16
3	3.21	5.90	3.24	13.72	5.98
4	4.04	8.00	10.11	13.82	4.43

Nro. de **columnas** de la matriz

Nro. de **filas** de la matriz

Elemento [3][0] de la matriz.

Elemento [4][3] de la matriz.

Declaración e inicialización de un arreglo bidimensional

Para operar con arreglos bidimensionales (matrices) deberán declararse las dos dimensiones de la tabla.

```
/*Declaración de una matriz mat de 10x6 elementos enteros */
int mat[10][6] // declaración de la matriz de enteros mat
```

```
/* Inicialización de la matriz mat recorriéndola por filas con datos
ingresados por consola */
```

```
#include <iostream>
using namespace std;
int main() {

int mat[10][6];
for (int i=0; i<10; i++)
    for (int j=0; j<6; j++)
    { cout<<"dato de fila "<<i<<" columna "<<j<<":";
      cin >> mat[i][j];
    }
return 0;
}
```

El ejemplo siguiente contiene el código C++ que permite mostrar una matriz dispuesta en filas y columnas en la pantalla.

Para inicializar un arreglo en el programa, se puede “recorrer” cada elemento del arreglo para asignar los valores correspondientes, o bien, pueden inicializarse los elementos en la misma declaración, enumerando la lista de datos a asignar.

```
//Ejemplo: mostrar una matriz de 2x 3 elementos
//en forma de tabla
#include <iostream>
#include<iomanip>
#include<conio.h>
using namespace std;

int main()
{
    int m[2][3]={12,34,56,78,90,100};
    int i,j;
    for (i=0;i<2;i++)
        {for (j=0; j<3; j++)
            {cout<<setw(4)<<m[i][j]; //escribe elementos de una fila
            }
            cout<<endl;    //avanza a la próxima línea
        }
    return 0;
}
```

En el ejemplo anterior se ha inicializado una matriz *m* de 6 elementos enteros donde se asignan los datos por filas.

Es decir que: `int m[2][3]={12,34,56,78,90,100};` ha permitido asignar los datos de la manera siguiente:

<code>m[0][0]=12</code>	<code>m[0][1]=34</code>	<code>m[0][2]=56</code>
<code>m[1][0]=78</code>	<code>m[1][1]=90</code>	<code>m[1][2]=100</code>

Si se deseara mostrar los datos por columnas, sólo deberían intercambiarse los ciclos `for` del ejemplo.

Operaciones con arreglos bidimensionales

Las operaciones que podemos realizar con estos arreglos, son las mismas vistas para arreglos unidimensionales. La única salvedad, es que en lugar de utilizar un índice, deberán utilizarse dos.

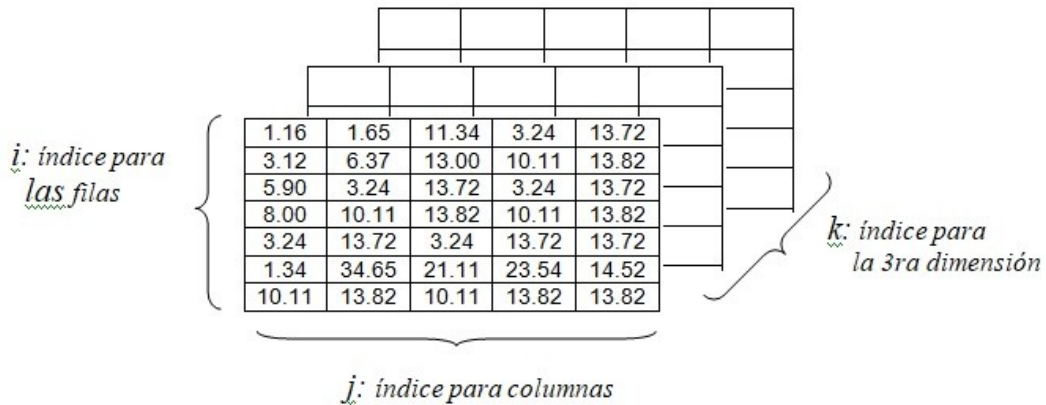
Arreglos multidimensionales

Los arreglos multidimensionales cumplen las mismas características que las vistas para los demás arreglos.

Así, todo arreglo multidimensional tiene un **nombre**, un **tipo** (carácter, numérico o lógico según el tipo de sus datos), una **dimensión**.

En este caso en particular, se requerirán más de 2 índices para posicionar un elemento en la colección. Son también conocidos como tablas o matrices multidimensionales.

Ejemplo: A continuación, se muestra gráficamente un arreglo tri-dimensional, compuesto por números de punto flotante (reales). Trate de encontrar algunos casos donde se requiera la necesidad de plantear estructuras como la de la figura.



i: índice para las filas

1.16	1.65	11.34	3.24	13.72
3.12	6.37	13.00	10.11	13.82
5.90	3.24	13.72	3.24	13.72
8.00	10.11	13.82	10.11	13.82
3.24	13.72	3.24	13.72	13.72
1.34	34.65	21.11	23.54	14.52
10.11	13.82	10.11	13.82	13.82

j: índice para columnas

k: índice para la 3ra dimensión

Organización en memoria de los arreglos

C++, como la mayoría de los lenguajes de programación, reserva segmentos de memoria para almacenar los arreglos, ubicando a cada elemento en forma contigua, uno al lado del otro. La memoria reservada se basa en la declaración del arreglo, y constituye un segmento estático; esto es: si el arreglo se declara en `main{ }`, durante toda la vida o ejecución del programa, estará ocupando la porción de memoria necesaria y ese recurso no podrá ser empleado en otra cosa.

Si se lo declara dentro de una función o bloque, ocupará memoria mientras el control de ejecución opere dentro de la función o del bloque.