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

FUNDAMENTOS DE PROGRAMACIÓN

UNIDAD 9
ARRAYS Y STRUCTS

UNIDAD 9

Arrays y Structs

Introducción

En esta unidad se aprenderán a emplear dos importantes estructuras de datos en C++: Arrays y Structs.

En primer lugar, se aplicarán los conceptos vistos en Fundamentos de Programación cuando se estudiaron los arreglos, pero ahora empleando la sintaxis de C++. 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 segundo lugar, se aprenderá una nueva estructura de datos importante de C++: *struct,* y se analizarán las ventajas de su empleo.

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.

Definición de arreglo

*Se define **array** 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.*

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.

Clasificación de los Arreglos

En base al número de índices que determinan la posición de un elemento en el arreglo se pueden definir:

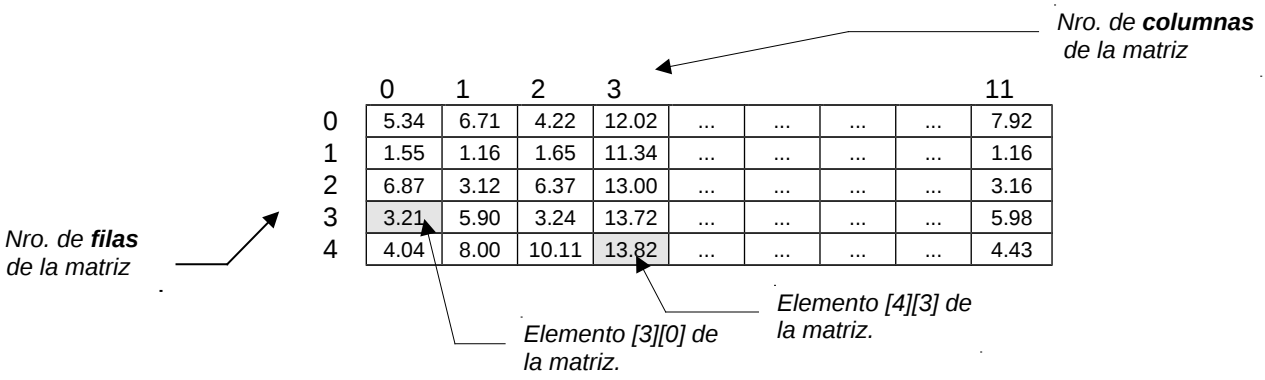
- **Arreglos lineales o unidimensionales**: la posición de un elemento la determina un único índice. Estos arreglos son también conocidos como listas o vectores

  Ejemplo: Supóngase el caso de un arreglo lineal **v** declarado como: **int v[200];**

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

- **Arreglos bidimensionales**: se requieren 2 índices para posicionar un elemento en la colección. También son conocidos como *tablas* o *matrices*.

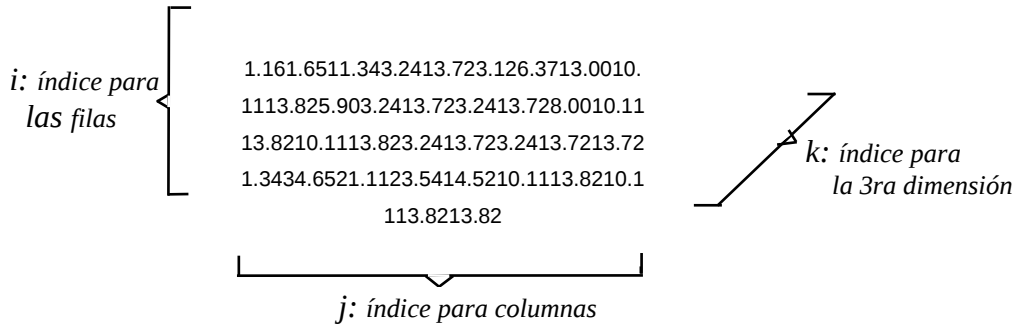
Ejemplo: Obsérvese gráficamente el caso de un arreglo **m** de números de punto flotante, compuesto por 5 filas y por 12 columnas: **float m[5] [12];**



	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

- **Arreglos multidimensionales**: se requieren 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.



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 ~~de~~ 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.

Declaración e inicialización de un arreglo

Arreglos lineales

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];
double tabla[4] [12];
float lista[200];
```

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]`.

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.

```
/* declaración e inicialización  del arreglo x formado por 100
enteros al azar */
int  x[100];
for (int i=0; i<99; i++)
 {
   x[i]= rand( );
 }
```

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

```
// 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 a cero */
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}
```

Arreglos bidimensionales

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 */

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];
 }
```

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

```
//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:

```
m[0][0]=12      m[0][1]=34      m[0][2]=56
m[1][0]=78      m[1][1]=90      m[1][2]=100
```

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

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. 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**.

Estructuras. El tipo struct.

Un arreglo es una estructura de datos homogénea, es decir sólo admite una colección de elementos de igual tipo. A menudo se requiere organizar los datos de una entidad en una estructura, pero admitiendo información de diferente naturaleza (tipo). C++ dispone para este caso del tipo **struct.**

*Un **struct** en C++ es una colección de componentes, los cuales pueden ser de diferente tipo. Cada componente o miembro debe declararse individualmente.*

Su sintaxis general es la siguiente:

```
struct compuesta{
          miembro 1;
          miembro 2;
          miembro 3;
          .....
          miembro n;
                };
```

En la definición es obligatorio explicitar el tipo **`struct`** y a continuación el identificador de la estructura (en el ejemplo **`compuesta`**). Luego, entre llaves deben definirse cada uno de los componentes o miembros.

Observar el siguiente ejemplo:

ficha
apellido
nombres
Dni
Edad
Cant_materias

```
struct ficha {
      char apellido[20];
      char nombres[20];
      long dni;
      int   edad;
      int   cant_materias;
            };
```

Los miembros individuales de una estructura pueden ser de tipos simples, arrays, punteros e inclusive **`struct`**. En cuanto a los nombres de estos miembros o componentes deben ser diferentes, pero pueden coincidir con el identificador de alguna otra variable definida fuera de dicha estructura.

Al definir una estructura se está planteando el esquema de la composición pero sin definir ninguna variable en particular. El tipo de dato `struct`, es justamente eso: un tipo de dato. Con esto quiere significarse, que al declarar a una variable de tipo struct, se está creando un tipo de dato propio. Así como se tiene int x, dónde x es una variable simple de tipo entero, ahora se tendrá struct x {….}, que será una variable compuesta, conformada por diferentes miembros o campos.
Es por eso que los miembros de un struct no se pueden inicializar en la definición de la estructura, ya que al definirla planteamos sólo el esquema.

Para declarar variables de tipo `struct` se debe indicar lo siguiente:

```
struct ficha x,y;  // x e y se declaran de tipo ficha
```

Se puede combinar la definición de la composición de la estructura con la declaración de las variables y con su inicialización:

```
struct ficha{
        char apellido[20];
        char nombres[20];
        long dni;
        int edad;
        int cant_materias;
             } x = {“Lopez”, “Gerardo”, 24567890, 21, 11};
```

En este ejemplo, `ficha` es el nombre de la estructura, `x` la variable de tipo ficha y los datos especificados entre { }, los valores iniciales de los miembros `apellido, nombres, dni, edad, cant_materias.`

Arreglo de structs

Es posible emplear arreglos como miembros de una composición **`struct`**. Pero también podemos definir un arreglo cuyos elementos sean estructuras (**`struct`**).

```
struct ficha {
        char apellido[20];
        char nombres[20];
```

```
        long dni;
        int edad;
        int cant_materias;
                    };

struct ficha z[5] ={ {“Lopez”,“Gerardo”,24567890, 21, 11},
                     {"Giménez","Ana",25689901,20),
                     {"Zapata","Andrés", 26701231,19,8},
                     {"Farías","Marina",23199870,22,15},
                     {"Martino","Manuel",24500654,21,10}
                    }
```

En el caso anterior se declara e inicializa un arreglo **z** de 5 elementos de tipo ficha. Para mostrar por pantalla el miembro `dni` del tercer elemento del arreglo, deberá escribirse:-

```
cout<<z[2].dni;
```

Para cambiar el miembro edad a 22 en el cuarto elemento del arreglo **z**, deberá escribirse:-

```
z[3].edad=22;
```

Para leer los apellidos, nombres y dni en modo consola, deberá escribirse:

```
for (int i=0; i<5; i++)
    { cin.getline( z[i].apellido,20);
      cin.getline( z[i].nombres,20 );
      cin>>z[i].dni;
    }
```

Nota: cin.getline() es una función que permite el correcto ingreso de una cadena de caracteres. Esta función necesita como mínimo 2 parámetros: el nombre del string y la cantidad de caracteres del mismo.

Para mostrar un listado con los miembros apellido y dni del arreglo **z**, se codifica de la siguiente forma:

```
for (int i=0; i<5; i++)
    { cout<<z[i].apellido<<" "<<z[i].dni<<endl; }
```

Procesamiento de una variable struct

Para procesar la información relacionada a una estructura se puede operar con sus miembros individualmente o en ocasiones con la estructura completa. Para acceder a un miembro individual se debe utilizar el identificador de la variable **`struct`**, un punto de separación y el nombre del miembro componente.

```
variable.miembro
```

Considérese el siguiente código de ejemplo.

```
1. #include <iostream>
2. using namespace std;
3. //-------------------------------------------------------
4. struct registro{
5.      char ape[15];
6.      char nom[20];
7.      long dni;
8.                 };
9. //-------------------------------------------------------
10. int main( )
11. {
12.  registro f,z;
13.  cin.getline(f.ape,15);
14.  cin.getline(f.nom,20);
15.  cin>>f.dni;
16.  z=f;
17.  cout<<z.ape<<" "<<z.nom<<"---DNI:"<<z.dni<<endl;
18.  return 0;
19. }
```

Obsérvese en la línea 12 la declaración de las variables tipo struct **f** y **z**. Las líneas 13, 14 y 15 permiten asignar datos ingresando los valores de los miembros en modo consola. En la línea 16 se asigna la variable struct completa **f** a una variable de igual tipo **z**. En la línea 17 se muestran los miembros de **z**.

Otros ejemplos con struct

```
/* Declaración de un struct Direccion con el esquema para contener
calle y número y otro struct Persona que, entre otros miembros,
contiene uno del tipo Direccion */

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

int main() {
     struct Direccion {
          char calle[25];
          int numero;
     };
     struct Persona {
          char nombre[35];
          Direccion domi;
          } candidato;
```

```
/* Se define una variable candidato del tipo Persona. Para el
ingreso de calle y número, se debe referenciar de la variable
candidato el elemento domi (tipo Direccion) y del mismo, calle y/o
numero */

      cout<< "Ingrese apellido y nombre: ";
      cin.getline(candidato.nombre, 35);
      cout<< "Ingrese Calle donde vive: ";
      cin.getline(candidato.domi.calle, 25);
      cout<< "Ingrese numeración: ";
      cin>> candidato.domi.numero;
      cout<< "Candidato " << candidato.nombre << endl;
      cout<< "Domicilio " << candidato.domi.calle << ' ' <<
candidato.domi.numero;
      return 0;
}


/* Declaración de un struct Libro con el esquema para contener datos
de un libro y de la variable colección que es un array de Libro */
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
      struct Libro {
            char titulo[25];
            char editorial[10];
            int anio_edicion;
      };

      Libro coleccion[10];

      for (int i=0; i< 10; i++)
      {
            cout<< "Datos del libro : " << i << endl;
            cout<< "Ingrese titulo: ";
            cin.getline(coleccion[i].titulo, 25);
            cout<< "Ingrese editorial:  ";
            cin.getline(coleccion[i].editorial, 25);
            cout<< "Ingrese año edición: ";
            cin>> coleccion[i].anio_edicion;
      }
      cout<< "Listado de Libros " << endl;
      cout<< " Titulo           Editorial       Año edición " <<
endl;
      for (int i=0; i< 10; i++)
      {
        cout<< coleccion[i].titulo<<"    "<<
coleccion[i].editorial<<"    "<< coleccion[i].anio_edicion<< endl;
      }
      return 0;
}
```

Tipos definidos por el usuario: typedef

En C++ es posible identificar a tipos existentes o estructuras de datos con nombres y tratarlos como si fueran nuevos tipos de datos. Entonces, las variables o funciones podrán declararse en base a estos nuevos nombres que representan tipos de datos de C++. Estos tipos de datos definidos por el usuario no tienen la importancia que revisten en otros lenguajes (Pascal por ejemplo).

Para asignar un nombre correspondiente a una definición de tipo en C++ deberá emplearse la palabra reservada **`typedef`**.

A continuación, algunos ejemplos:

```
typedef unsigned char byte;
/*se le da el alias byte a los char sin signo */

typedef int tabla[50][12];
/*definición del nombre tabla como matriz de 50x12 enteros */


typedef struct {
        char apellido[20];
        char nombres[20];
        long dni;
        int edad;
        int cant_materias;
            } ficha // definición del tipo ficha como estructura
```

En base a las definiciones de tipo anteriores, son válidas las siguientes declaraciones:

```
byte b;
ficha x,y[200];
tabla m,t;
```

En este ejemplo, **b** se declara de tipo byte; **`x`** es una variable individual que puede almacenar la información de una entidad de acuerdo a los miembros definidos en el tipo **`struct ficha;`** la variable **`y`** es un **`array`** donde cada **`elemento`** es del tipo **`struct ficha`**.; las variables **m** y **t** se declaran como arreglos bidimensionales de 50x12 elementos enteros.

Síntesis

1. Un arreglo es una colección de datos relacionados de igual tipo e identificados bajo un nombre genérico único.
2. La estructura completa de un arreglo ocupa un segmento de memoria único y sus elementos se almacenan en forma contigua.
3. Para procesar un elemento del arreglo, se debe especificar el nombre del arreglo y uno o más índices que determinan la posición relativa del elemento.
4. El índice que determina la posición de los elementos de un arreglo comienza siempre con cero.

5. En la declaración se debe establecer la cantidad de elementos (dimensión) que puede tener como máximo el arreglo en el programa o definir los elementos que lo componen.
6. La dimensión de un arreglo es la cantidad de elementos que un arreglo puede almacenar y debe ser siempre mayor o igual a su longitud.
7. Si se requieren dos índices para establecer la posición de un elemento, el arreglo es una tabla o matriz. Con 3 índices es una matriz tridimensional y con más índices, una matriz multidimensional.
8. El nombre de un arreglo representa la dirección de memoria del inicio del arreglo (de su primer elemento).
9. Un struct es una estructura de datos de C++ que permite organizar un conjunto de datos de diferentes tipos relacionados. Estos datos que conforman el struct se denominan miembros.
10. Los miembros de un struct pueden ser de tipo simple o también otras estructuras de datos: arreglos, structs, etc.
11. También los elementos de un arreglo pueden ser de tipo struct.
12. Los objetos cin y cout para flujos de entrada y salida no permiten operar estructuras de datos completas. Se deben utilizar los componentes de una estructura, como el elemento de un arreglo o un miembro de un struct.
13. C++ admite la identificación de tipos a través de la palabra clave `typedef`. Estas definiciones suelen ser globales y se realizan usualmente fuera de la función principal de programa `main()`.