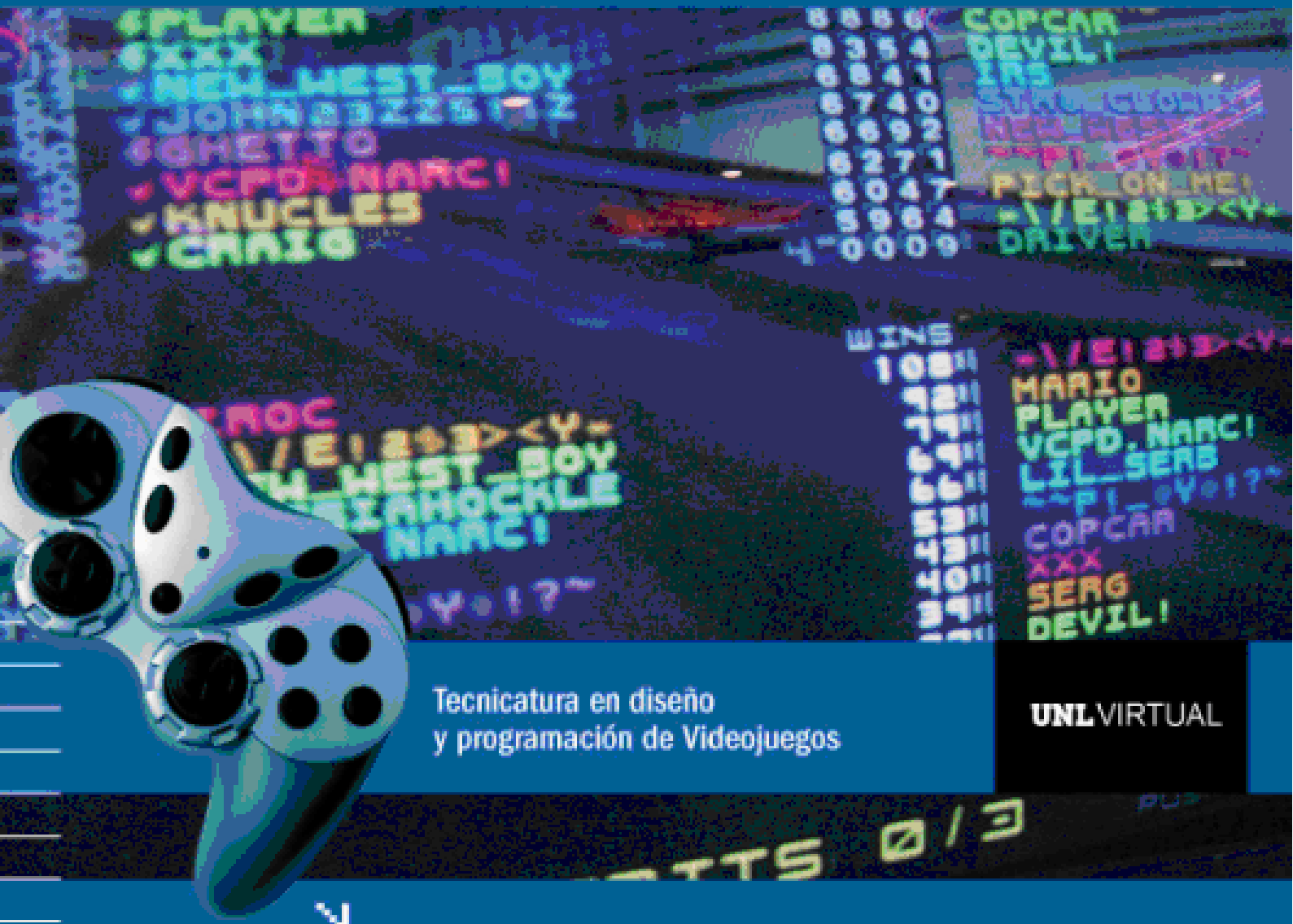
UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS



Introducción a la programación

Unidad Temática Número 5

Arrays y Structs

Objetivo: introducir la estructura de datos arreglo para comprender las técnicas básicas para la solución de problemas como para almacenar, ordenar y buscar datos.

Temas: Arreglos. Declaración de arreglos. Como pasar arreglos a funciones. Arreglos con múltiples subíndices.

INTRODUCCIÓN

Ya vimos que si quisiéramos obtener el promedio de un conjunto de valores es posible si los ingresamos y los procesamos al mismo tiempo (recordemos el ejemplo de 10 valores):

```
    int suma = 0;
    for(int i=0; i<10; i++){
        int valor;
        cin>>valor;
        suma+=valor;
    }
    float promedio = suma/10;
    cout<<promedio;
```

¿Qué ocurre si quisiéramos seguir trabajando con los valores que ingresamos? ¿Cómo podríamos hacer si quisiéramos, por ejemplo, mostrar todos aquellos valores que superan el promedio? El promedio lo obtenemos recién cuando terminamos de ingresar los valores, así que no podemos hacer nada mientras los estamos cargando. Podríamos hacerlo con 10 variables: *valor1*, *valor2*, *valor3*… *valor10*. En tal casi no podríamos utilizar la estructura FOR (ni ninguna otra) ya que las variables se acceden una por una:

```
     int suma = 0;
    int valor1, valor2, valor3, valor4, valor5, valor6, valor7, valor8,
valor9, valor10;
        cin>>valor1;
        cin>>valor2;
        cin>>valor3;
        cin>>valor4;
        cin>>valor5;
        cin>>valor6;
        cin>>valor7;
        cin>>valor8;
        cin>>valor9;
        cin>>valor10;
        suma = valor1 + valor2 + valor3 + valor4 + valor5 + valor6 +
valor7 + valor8 + valor9 + valor10;
    float promedio = suma/10;
    cout<<promedio;
```

Esta forma de solucionar el problema es muy poco general, además de ser bastante larga de escribir. Con los 10 valores independizados sí podríamos entonces preguntar uno por uno cual supera el promedio y entonces mostrarlo. ¿Pero qué pasa si tenemos 100 valores? La solución se hace impracticable.
Una solución a este problema es utilizar alguna estructura de dato que nos permita trabajar con varios valores. Existe una estructura que nos permite integrar un conjunto de valores y poder identificarlos individualmente mediante un índice, esta estructura se denomina Arreglo (Array en inglés).

ARREGLO

Como dijimos, el arreglo es una estructura de datos que nos permite almacenar varios valores relacionados y poder recuperarlos. Podemos ver a un arreglo como una cajonera, donde en cada cajón podemos almacenar un valor. Luego para saber qué valor se encuentra en ese cajón deberíamos buscar el número del cajón.

En C++, un arreglo de 10 elementos enteros se declara de la siguiente manera:

```
int miArreglo[10];
```

Al igual que cualquier variable, primero debemos poner el tipo de dato, luego el nombre de la variable (en este caso miArreglo) y luego, para identificar que es una arreglo, ponemos entre corchetes la cantidad de valores que queremos que el arreglo pueda almacenar. Aquí la variable es todo el arreglo, es decir que nuestra variable se llama

miArreglo, la diferencia es que dentro de sí tiene la capacidad de almacenar 10 valores independientes.
La pregunta es ¿Cómo almacenamos y recuperamos valores del arreglo? Debemos hacerlo mediante un índice que identifica la posición de la variable. Esto lo hacemos, también, con los corchetes:

miArreglo[0] : tiene el valor de la posición 0
miArreglo[1] : tiene el valor de la posición 1
miArreglo[2] : tiene el valor de la posición 2
…
miArreglo[9] : tiene el valor de la posición 9

IMPORTANTE: Los valores que se encuentran entre corchetes se llaman "índices" del arreglo y comienzan en la posición 0, siempre (para C++). Entonces, un arreglo de 10 elementos va de la posición 0 a la 9.

La escritura y lectura de los arreglos se realiza como cualquier variable. Si le queremos asignar el valor de 100 al 5to elemento del arreglo (índice 4) lo hacemos de la siguiente manera:

```
miArreglo[4] = 100;
```

De la misma manera lo podemos leer.

```
cout<<miArreglo[4];
```

Si tenemos los valores iniciales, podemos también asignarle valores en la misma declaración. En ese caso el conjunto de valores va entre llaves, separados por coma (,).

```
int miArreglo[10] = {100,15,20,46,32,1,72,42,23,7};

// o bien

int miArreglo[] = {100,15,20,46,32,1,72,42,23,7};
```

En la segunda declaración podemos omitir colocar el tamaño del arreglo, ya que lo estamos asignando con la misma declaración. Si en cambio decidimos colocar el número, la cantidad de elementos que coloquemos deberá coincidir con el número de elementos reservados. Es decir, no podemos decir que declaramos un arreglo de 10 elementos y colocar entre llaves menos o más elementos.

Ahora sí se hace fundamental la utilización de una estructura de control iterativa, particularmente el FOR, que ya tiene a disposición un índice. Veamos entonces cómo hacemos el ejercicio que nos habíamos planteado en un principio:

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

int main (int argc, char *argv[]) {

    int miArreglo[10];
    int suma = 0;
    for(int i=0; i<10; i++){
        cout<<"Ingrese el valor "<<i<<" :"<<endl;
        cin>>miArreglo[i];
        suma+=miArreglo[i];
    }
    float promedio = suma/10;

    cout<<"el promedio es "<<promedio<<endl;

    // mostramos solo aquellos que superan el promedio
```

```
    for(int i=0; i<10; i++){
        if (miArreglo[i] > promedio){
            cout<<miArreglo[i]<<endl;
        }
    }


    return 0;
}
```

ARREGLOS MULTIDIMENSIONALES

Los arreglos que vimos antes se consideran arreglos unidimensionales porque sólo tienen una dimensión que varía. Existen también arreglos de dos, tres y N dimensiones. Vamos a ver el caso para dos dimensiones, el resto se puede extender de este uso.

Habíamos dicho que un arreglo unidimensional se declara de la siguiente manera:

```
int miArreglo[10];
```

Agregando una dimensión más, tenemos un arreglo de dos dimensiones. Vamos a ver como declaramos un arreglo de 10x5 elementos:

```
int miArreglo[10][5];
```

Si quisiéramos completar con valores al azar deberíamos utilizar dos ciclos FOR.

```
    int miArreglo[10][5];

    for (int i=0;i<10;i++){
        for (int j=0;j<5;j++){
            miArreglo[i][j] = rand()%100 + 1;
        }
    }
```

Al igual que los arreglos de una dimensión, podemos asignarle valores en la misma declaración. También hacerlo de varias maneras:

```
int miArreglo[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

 // o bien

int miArreglo[3][4] = {{0,1,2,3}, {4,5,6,7}, {8,9,10,11}};

// o bien

int miArreglo[][4] = {{0,1,2,3}, {4,5,6,7}, {8,9,10,11}};
```

En la primer manera estamos definiendo todos los elementos de manera lineal.
La segunda manera es un poco más explícita y posiblemente la más fácil de leer. En la tercer manera estamos obviando el número de elementos de la primer dimensión. Esto es análogo a la declaración de arreglos de una dimensión.

Es muy importante mantenerse dentro del rango de los índices de los arreglos, caso contrario el compilador dará errores de memoria en tiempo de ejecución (muy difíciles de encontrar).
En la práctica no es tan común encontrar arreglos de más de dimensiones, aunque en caso de ser necesario, es fácil extender el concepto.

REPRESENTACION GRAFICA DE LOS ARREGLOS

Los arreglos de una y dos dimensiones tienen representaciones gráficas particulares que se relacionan con objetos matemáticos. Cada una de las dimensiones de los arreglos se entiende como un eje cartesiano {x, y, z,…}.

De este mismo concepto podemos decir que los arreglos de una dimensión se representan como **vectores**. Sin entrar en detalles de su representación matemática, podemos mostrar un vector de la siguiente manera.

0	1	2	3	4	5	…	N

Donde 0, 1, 2, …. N son los índices de ese arreglo. Es decir, un arreglo con los valores: 4, 7, 3, 10, 5. Será.

0: 4	1: 7	2: 3	3: 10	4: 5

Por su parte, los arreglos de dos dimensiones se representan como **matrices**. Donde cada elemento posee un par de coordenadas:

0,0	0,1	0,2	0,3	0,4	0,5	…	0,N
1,0	1,1	1,2	1,3	1,4	1,5	…	1,N
2,0	2,1	2,2	2,3	2,4	2,5	…	2,N
3,0	3,1	3,2	3,3	3,4	3,5	…	3,N
…	…	…	…	…	…	…	…
M,0	M,1	M,2	M,3	M,4	M,5	…	M,N

Donde cada celda se identifica con un par de coordenadas. La primera se puede entender como al fila y la segunda como al columna. En videojuegos tienen muchas aplicaciones, la pantalla, una imagen, un mapa de tile, entre otras, cosas se definen como matrices.

Por ejemplo, una matriz de 3 x 5 (con elementos al azar) se puede escribir de la siguiente manera:

0,0: 1	0,1: 10	0,2: 6	0,3: 9	0,4: 2
1,0: 4	1,1: 5	1,2: 2	1,3: 10	1,4: 19
2,0: 6	2,1: 8	2,2: 11	2,3: 15	2,4: 4

STRINGS COMO ARREGLOS

Cuando vimos tipos de datos vimos que los strings servían para guardar palabras y frases, aunque dijimos que no son tipos de datos naturales sino que son tipos de datos especiales llamados objetos. El objeto string tiene, entre otras cosas, la posibilidad de escribirse como un arreglo de letras. Es decir, un string con el valor “manzana” puede accederse de manera individual a cada carácter referenciando la posición de cada letra:

0: m	1: a	2: n	3: z	4: a	5: n	6: a

```
string manz = "manzana";

for (int i=0;i<manz.length();i++){
    cout<<manz[i]<<endl;
}
```

La salida en consola será:

Como habrán notado en la condición del FOR escribimos *manz.length()* eso tiene que ver con más propiedades del objeto string. Ya las veremos más adelante. Lo que va luego del punto (.) de un objeto son los métodos, es decir, las funciones que éste tiene. El método length permite obtener la longitud de un string.