UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño
y programación de Videojuegos

UNLVIRTUAL

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.

UNIDAD 5

Arrays y Structs

Introducción

En esta unidad aprenderemos a emplear dos importantes estructuras de datos en C++. Aplicaremos los conceptos estudiados en Fundamentos de Programación: los arreglos, empleando la sintaxis de C++. Estudiaremos como declarar y definir arreglos estáticos en C++, como se almacenan en memoria y como se acceden para ser modificados o utilizados. Analizaremos sus ventajas y limitaciones.

Además estudiaremos otra estructura de datos importante de C++: *struct,* y analizaremos las ventajas de su empleo.

Veremos que es posible combinar estructuras de datos de acuerdo a las necesidades del caso a resolver.

Hasta ahora hemos visto las estructuras de control que nos servían para hacer todo tipo de programas (en la unidad 3), luego vimos funciones (unidad 4) que nos servían para escribir el código una vez y usarlo muchas veces y para dividir un programa complejo en subprogramas más simples.

Ahora veremos una manera de agrupar muchos valores en una sola estructura y a trabajar con ellos. Esta estructura llamada arreglo nos permite manejar variables que si las tuviéramos que declarar una por una se nos haría imposible escribir el programa, si recuerdan el programa antiaereo, el manejar solo tres disparos a la vez generó un numero de variables bastante grande para manejar, pero si hubiéramos querido tener la capacidad de tener en el juego 10 disparos a la vez, hubiese sido muy complicado de manejar esas variables.

Definición de arreglo

Definimos como array (arreglo) a 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 entero 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.

Componentes de un arreglo

El nombre del arreglo sigue las mismas reglas que el de las variables, el índice se coloca entre corchetes a continuación de él.

Básicamente un arreglo es una lista de variables del mismo tipo agrupadas bajo un nombre una atrás de la otra, para obtener o modificar individualmente cada variable se utiliza el índice, que para la primer variable es 0, por ejemplo un array llamado a de números enteros, el primer valor sería a[0], el segundo sería a[1], y así sucesivamente. Si quisiéramos darle un valor al segundo podríamos hacer:

a[1]=9;

y si quisiéramos saber el valor que contiene el primer elemento:

cout<<a[0];

La declaración de un arreglo se hace colocando primero el tipo luego el nombre y luego entre corchetes la cantidad máxima de variables:

int a[5]; //un arreglo de enteros de 5 elementos, el índice estará entre 0 y 4

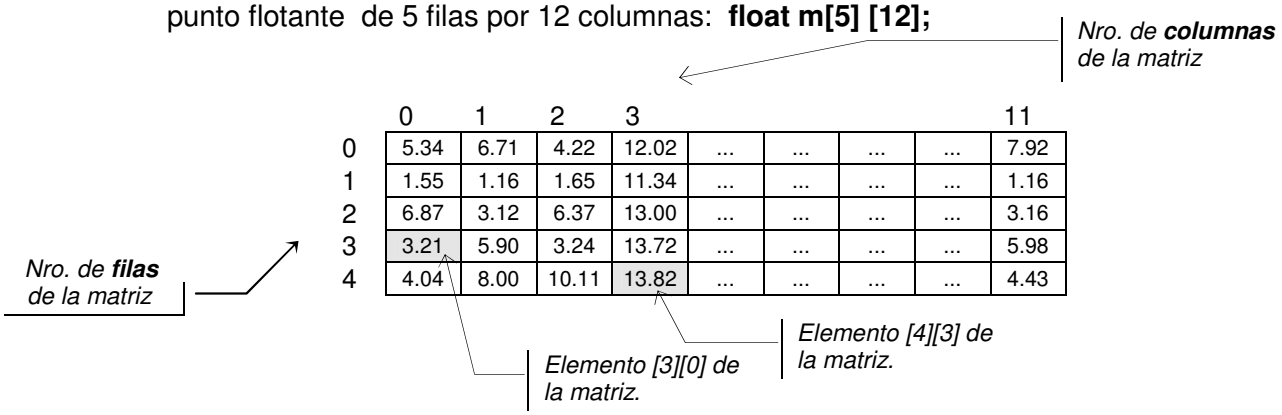
Clasificación de los Arreglos

En base al número de índices que determinan la posición de un elemento en el arreglo podemos definir:
Arreglos lineales o unidimensionales: la posición de un elemento la determina un único índice. También conocidos como listas o vectores
Ejemplo: en el caso de un arreglo lineal v declarado como: **int v[200];**

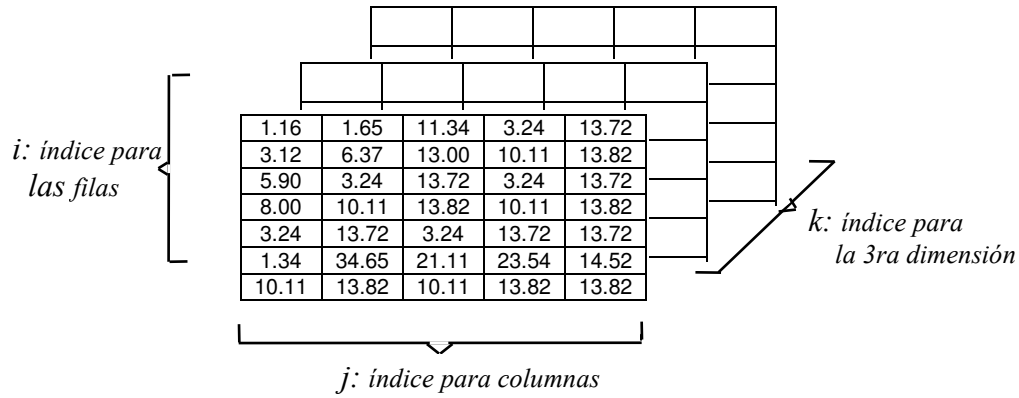
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	.....	v[199]
23	56	71	19	33	90	48	.....	74

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

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



Arreglos multidimensionales: se requieren más de 2 índices para posicionar un elemento en la colección. También conocidos como *tablas* o *matrices multidimensionales*.
Ejemplo: arreglo de reales tri-dimensional. Analice algunos casos donde se requiera la necesidad de plantear estructuras como la de la figura.



Organización en memoria de los arreglos

C/C++ como la mayoría de los lenguajes de programación, reserva memoria para almacenar las variables, lo mismo ocurre con los arreglos, para ello reserva una cantidad suficiente para el total de todas las variables dentro del arreglo, luego ubica a cada elemento en forma contigua uno detrás de otro. La memoria reservada se basa en la declaración del arreglo, y constituye un segmento estático esto significa que durante el programa no puede ni agrandarse ni achicarse, por lo que hay que tener en cuenta que el tamaño del mismo sea suficiente para todo lo que dure la ejecución del programa.
También cabe aclarar que los arreglos siguen las mismas reglas que las variables, pueden ser locales o globales, y si son declarados dentro de una función, el mismo se destruye cuando se sale de esta.

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];
static float lista[200];
```

Tener en cuenta que C++ considera al primer elemento de un array en la posición 0, por tanto una declaración **`int x[3]`** implica que podremos referenciar en el programa a los elementos **`x[0], x[1], x[2].`**

Para inicializar un arreglo en el programa podemos 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.

```
// definición 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.

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

/* definición 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 */
int  m[]={34, 56, 20, 41, 72}
```

Arreglos bidimensionales

Para operar con arreglos bidimensionales (matrices) debemos declarar las dos dimensiones de la tabla.

```
/*Definición  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];
 }
```

Como ven, para dos dimensiones se dos índices que se separan por corchetes, como abran adivinado por cada dimensión agregada a un arreglo se agrega un índice extra entre corchetes, por ejemplo una declaración de arreglo de tres dimensiones sería int a[3][4][2];
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<iomanip>
#include<conio.h>

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 quisiéramos asignarlos por columnas solo deberíamos intercambiar 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 problema, el problema se presentaría si quisiéramos usar mas posiciones del arreglo que lo que reservamos al declararlo.
Obviamente, debemos tener cuidado en recorrer el arreglo en toda su longitud L y no en toda su dimensión D. Para ello lo más conveniente es mantener en una variable la longitud L, es decir, cuantas posiciones del arreglo estamos usando en cada momento del programa.

Arreglos como parámetros de funciones

Es posible utilizar arreglos como parámetros de funciones. En C++ no es posible pasar por valor un bloque de memoria completo como parámetro a una función, aún si está ordenado como un arreglo, pero está permitido pasar su dirección de memoria, lo cuál es mucho más rápido y eficiente. Al pasar una dirección de memoria estamos efectuando un pasaje por referencia.
Para admitir arreglos lineales como parámetros lo único que se debe hacer al declarar la función es especificar en el argumento el tipo de dato que contiene el arreglo, un identificador y un par de corchetes vacíos **[ ]**. Por ejemplo, observemos la siguiente función:

```
void  leer_arreglo(int arg[])
```

Vemos que la función `leer_arreglo( )` admite un parámetro llamado **arg** que es un arreglo de enteros (int). ¿Donde está el pasaje por referencia o la dirección de memoria del inicio del arreglo? Respuesta: en el nombre del arreglo.

> El nombre del arreglo identifica a una variable que contiene la dirección de memoria donde comienza el bloque donde se aloja el arreglo. En otras palabras: el nombre del arreglo es una variable que contiene la dirección de memoria del elemento 0 del arreglo.

Para llamar a la función **`leer_arreglo()`** del ejemplo anterior debemos utilizar como parámetro actual un arreglo; para ello es suficiente utilizar el identificador del arreglo:

```
int miarreglo [30];
leer_arreglo(miarreglo);
```

En el siguiente ejemplo se invoca 2 veces a la función `muestra_arreglo( )` pasándole en cada caso un arreglo y su longitud.

```
// Ejemplo: arreglos como parámetros
#include <iostream.h>
#include <iomanip.h>

void muestra_arreglo (int lista[], int largo)

int main ()
{ int v1[] = {35, 41, 22};
  int v2[] = {12, 4, 6, 15, 10};
  muestra_arreglo(v1,3);
  muestra_arreglo(v2,5);
  return 0;
}

void muestra_arreglo (int lista[], int largo)
{ for (int i=0; i<largo; i++)
      cout<<setw(4)<<lista[i];
  cout<<endl;}
```

La salida del programa será:

```
35  41  22
12   4   6  15  10
```

Obsérvese que en el ejemplo se invoca dos veces a la función `muestra_arreglo()` empleando como argumentos arreglos de diferente dimensión.
Obsérvese también que además del arreglo pasamos el largo del mismo, ya que de por si solo no hay forma de saber cuantas posiciones usamos en los mismos.

Arreglos multidimesionales

Si se trata de pasar como parámetro un arreglo de más de una dimensión, el parámetro formal debe tener los corchetes correspondientes al primer índice vacíos, pero establecer explícitamente las otras dimensiones.

```
void  leer_tabla(int t[][10], int num_filas);
```

El modificador *const* y parámetros de tipo arreglo

Hemos visto que al pasar un arreglo como parámetro, se pasa la dirección de memoria del inicio del arreglo y por lo tanto estamos efectuando un pasaje de parámetros por referencia. Esto implica que cualquier modificación efectuada en uno o más elementos del arreglo durante la ejecución de una función, se producirá la automática modificación del arreglo utilizado como parámetro actual o de llamada (se trata en realidad de un único arreglo pues se está trabajando sobre esa única dirección de memoria).
Para evitar que un arreglo sea modificado al pasarlo como parámetro, debe utilizarse la etiqueta *const* precediendo al parámetro formal en el prototipo de la función. Observemos las dos funciones siguientes:

```
float permite_cambiar(int lista[], int n);
{ ......
  lista[3]= 255; //modifica el parámetro de llamada
  .....
}

float no_permite_cambiar(const int lista[], int n);
{ ......
  lista[10]= 320;  //error de compilación
  .....
}
```

En la primer función, el cambio efectuado en `lista[3]` se reflejará en el arreglo que se utilice como parámetro para llamar a esta función. La segunda función producirá un error de compilación, pues se establece con const que la posición de memoria a la que se quiere acceder es de solo lectura y no puede modificarse el valor allí alojado.

Estructuras. El tipo *struct.*

Un arreglo es una estructura de datos homogénea, es decir solo 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 (**`compuesta`** en el ejemplo) de la estructura. Luego, entre paréntesis deben definirse cada uno de los componentes o miembros).
Tomemos el siguiente ejemplo:

ficha
apellido
nombres
Dni
Edad
Cant_materias

```
struct ficha(
        char apellido[15];
        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. No se puede inicializar un miembro al definir la estructura.

Al definir una estructura estamos planteando el esquema de la composición pero sin definir ninguna variable en particular. 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[15];
        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 ficha {
        char apellido[15];
        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 el miembro dni del tercer elemento del arreglo debemos escribir:

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

Para cambiar el miembro edad a 22 en el 4to elemento del arreglo z debemos escribir:

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

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

```
for (int i=0; i<5; i++)
   { cin.getline( z[i].apellido,15);
     cin.getline( z[i].nombres,20 );
     cin>>z[i].dni;
   }
```

Para mostrar un listado con los miembros apelido 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 podemos operar con sus miembros individualmente o en ocasiones con la estructura completa. Para acceder a un miembro individual debemos utilizar el identificador de la variable **`struct`**, un punto de separación y el nombre del miembro componente.

variable.miembro
Considere el siguiente código de ejemplo.

```
1. #include <iostream.h>
2. //-----------------------------------------------------
3. typedef struct{
4.    char ape[15];
5.    char nom[20];
6.    long dni;
7. } registro;
```

```
  //-------------------------------------------------------
8. int main( )
9. { registro f,z;
10.  cin.getline(f.ape,15);
11.  cin.getline(f.nom,20);
12.  cin>>f.dni;
13.  z=f;
14.  cout<<z.ape<<" "<<z.nom<<"---DNI:"<<z.dni<<endl;
15.  return 0;
16. }
```

Observe en la línea 9 la declaración de las variables struct f y z. Las líneas 10, 11 y12 permiten asignar datos ingresando los valores de los miembros en modo consola. En la línea 13 se asigna la variable struct completa f a una variable de igual tipo z. En 14 se muestran los miembros de z.

Tipos definidos por el usuario: *typedef*

Es posible en C++ 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 tiene la importancia que revisten en otros lenguajes (Pascal por ejemplo). Para asignar un nombre correspondiente a una definición de tipo en C/C++ debemos empelar la palabra reservada **`typedef`**.
Veamos 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;
      char *nombres;
      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 el último 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.

Bibliografía

Ing. Horacio Loyarte. Arrays y Strucs [UNL-FICH] [2008]