Manual 4 - 2do Torneo de Programación Competitiva

Lions R.C.

Julio 2019

Contents

1 Arreglos multidimensionales

A veces es conveniente manejar datos como si fueran a estar en una matriz de más de una dimensión, asi que C++ te permite crear arreglos multidimension-

ales para facilitar este proceso. Casi nunca se requieren más de tres dimensiones para resolver un problema asi que el usuario debe definir previamente cuantas dimensiones tiene su arreglo, además la memoria que se requiere para el arreglo incrementa exponencialmente con cada dimensión.

Para definir un arreglo de dimensión N, se debe escribir el tipo de dato, el nombre del arreglo y N corchetes []. Si queremos un arreglo de 7 x 3 x 3 enteros, podemos definirlo con **int miArreglo[7][3][3];**. También se pueden definir los datos iniciales de este arreglo utilizando multiples llaves anidados:

Listing 1: Asignando valores

```
#include <iostream>

using namespace std;

int main() {
    int cuboide[2][3][3] = {{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
    {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}}};
}
```

2 Structs

A veces es frustrante tener que manejar grupos de datos que deben ir juntos debido a que se tienen que crear pares de pares o multiples arreglos. Esto se puede solucionar con los structs, que son parte de la programación orientado a objetos.

Los structs son estructuras que un usuario puede definir para guardar multiples variables bajo un solo "objeto".

Por ejemplo, digamos que trabajas para un banco y quisieras guardar los datos importantes de tus clientes: su nombre, su apellido, su número de tarjeta y la cantidad de dinero que tiene. Si quisieramos guardar estos valores convencionalmente, tendriamos que usar cuatro arreglos o cuatro pares de pares anidados.

Usando structs, podemos definir un struct por cada cliente con estos tipos de datos y crear un solo arreglo o vector de clientes. No se requiere ninguna librería para definir un struct y se puede crear de la siguiente manera:

Listing 2: Definición de un struct

```
#include <iostream>

using namespace std;

struct Cliente {
    string nombre;
    string apellido;
    int tarjeta[16];
```

```
    float dinero;
};

int main() {

}
```

Como se puede ver, los structs siempre deben ir antes de nuestra función main y deben tener un punto y coma despues de su llave de cierre. Luego dentro de las llaves debe tener una lista de todas las variables que se desean agrupar.

Para crear una instancia de un struct, se debe poner el nombre del struct como el tipo de dato seguido por el nombre especifico de esa instancia:

Listing 3: Instanciamiento

```
#include <iostream>

using namespace std;

struct Cliente {
    string nombre;
    string apellido;
    int tarjeta[16];
    float dinero;
};

int main() {
    Cliente jorge;
    Cliente pablo;
}
```

Como se puede observar, se crearon dos clientes, **jorge** y **pablo**. Podemos modificar sus datos escribiendo el nombre de cada variable despues de un punto:

Listing 4: Modificando valores

```
#include <iostream>

using namespace std;

struct Cliente {
    string nombre;
    string apellido;
    int tarjeta[16];
    float dinero;
};

int main() {
```

```
    Cliente jorge;
    jorge.nombre = "Jorge";
    jorge.apellido = "Velazquez";
    jorge.tarjeta = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5};
    jorge.dinero = 50726.35;
    Cliente pablo;
    pablo.nombre = "Pablo";
    pablo.apellido = "Cesar"
    pablo.tarjeta = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 2};
    pablo.dinero = 999999999.9999;
}
```

Para simplificar este proceso, es más facil guardar las estructuras en un arreglo o vector:

Listing 5: Clientes bancarios

```
#include <iostream>
#include <vector>

using namespace std;

struct Cliente {
    string nombre;
    string apellido;
    int tarjeta[16];
    float dinero;
};

int main() {
    int numeroDeClientes = 3;
    vector<Cliente> clientes;
    for(int i = 0; i < numeroDeClientes; i++) {
        Cliente nuevo;
        cout << "Nombre del cliente: " << endl;
        cin >> nuevo.nombre;
        cout << "Apellido del cliente: " << endl;
        cin >> nuevo.apellido;
        string tarjeta;
        cout << "Tarjeta del cliente: " << endl;
        cin >> tarjeta;
        for(int i = 0; i < 16; i++) {
            nuevo.tarjeta[i] = tarjeta[i] - '0';
        }
        cout << "Dinero: " << endl;
        cin >> tarjeta;
        clientes.push_back(nuevo);
```

```
        cout << "Cliente " << nuevo.nombre << " guardado con exito" << endl;
    }
    cout << clientes.size() << " clientes guardados" << endl;
    for(int i = 0; i < clientes.size(); i++) {
        cout << clientes[i].nombre << endl;
    }
}
```

Liga al código

El último código guarda 5 clientes en un vector y pide sus datos al usuario. Después, se imprimen los nombres de estos clientes.

3 Grafos

3.1 Nodos, ramas, hojas y raíces

3.2 Grafos dirigidos y no dirigidos

3.3 Grafos cíclicos y acíclicos

4 Algoritmos de busqueda

4.1 Busqueda en anchura

4.2 Busqueda en profundidad

4.3 Algoritmo de Dijsktra

4.4 Algoritmos heuristicos

4.5 A*