

Taller de Estructuras Dinámicas - C++

Algoritmos y Estructuras de Datos

Tomás Assenza

"There are only two hard things in Computer Science:
cache invalidation and naming things."

Phil Karlton

"The best way to prepare (to be a programmer) is to
write programs, and to study great programs that
other people have written."

Bill Gates

Contenidos

1. ¿Por qué estudiar estructuras dinámicas?
2. Array
3. Vector
4. Map
5. Set (Ordered-Unordered Set)

¿Por qué estudiar estructuras dinámicas?

¿Por qué estudiar estructuras dinámicas?

- Competencias de programación.
- Entrevistas laborales (y trabajo en general).
- Útiles en I+D.
- Mayor facilidad de uso por funciones pre-armadas.

(Y por todo lo que deberíamos estudiar Algoritmos y Estructuras de Datos)

Array

Array

Conceptos generales:

- Arreglo similar a los tradicionales.
- Incorpora las funciones de la librería estándar de C++.
- Todo lo que hacemos con un arreglo normal, lo podemos hacer con Array. Pero, en algunos casos, las funciones facilitan la resolución de problemas.
- Es una estructura estática.

Array

```
#include <array>
```

Declaración

```
array<TipoDeDato,TamañoFisico> nombreDelArreglo;
```

Inicialización

```
array<TipoDeDato,TamañoFisico> nombreDelArreglo {dato1, dato2, dato3};
```

Acceso y modificación de elementos

```
nombreDelArreglo[indice] = nuevoValor;
```


Array - Funciones integradas

Imaginemos los arrays definidos como

```
array<int, 6> arreglo1 {1, 2, 3}; //Arreglo de tamaño físico 6, tamaño lógico 3
array<int, 6> arreglo2 {4, 5, 6}; //Arreglo de tamaño físico 6, tamaño lógico 3
```

Función front y back (acceso al primer o último elemento).

```
cout << arreglo1.front(); // Imprime 1
cout << arreglo1.back(); // Imprime 3
```

Funciones empty, size y max_size

```
cout << arreglo1.empty() //Imprime 0 (devuelve false) ya que contiene elementos.
cout << arreglo1.size() //Imprime 3. El tamaño lógico es 3.
cout << arreglo1.max_size() //Imprime 6. El tamaño físico es 6.
```

Función swap (Solo con arreglos de igual tamaño físico)

```
arreglo1.swap(arreglo2); //Intercambia el contenido de los arreglos arreglo1 = {1,
```

Array - Funciones complementarias

```
#include <array>
#include <algorithm>
```

Imaginemos el array definido como

```
array<int, 6> arreglo {3, 1, 2}; //Arreglo de tamaño físico 6, tamaño lógico 3
```

Función sort para ordenar de menor a mayor

```
sort(arreglo.begin(), arreglo.end()); //arreglo pasa a ser {1, 2, 3}
```

Función sort para ordenar de mayor a menor

```
sort(arreglo.begin(), arreglo.end(), greater<int>()); //arreglo pasa a ser {3, 2,
```

Array - Funciones complementarias

Definir nosotros mismos como ordenar

Imaginemos que tenemos

```
struct quimico{
    int num;
    int ganancia;
}

array<quimico, 3> quimicos {quimico1, quimico2, quimico3};
```

y queremos ordenarlos de menor a mayor con respecto a **num Y ganancia**

```
sort(arreglo.begin(), arreglo.end(), miComparacion);

bool miComparacion(const quimico a, const quimico b) {
    if(a.num!=b.num) return a.num<b.num;
    else return a.ganancia<b.ganancia;
}
```

Este tipo de funciones "auxiliares" son muy útiles si tenemos arrays de structs y queremos usar sort. Usamos más de un return ya que, en estos casos, es más visible para la resolución del problema (similar a lo que nos pasaba en funciones recursivas).

Comparación y recorrido de arreglos

Para comparar, podemos usar la comparación elemento a elemento, o mirar el total del arreglo.

```
if (arreglo1[0] == arreglo2[5]) //Compara el valor que está en la posición 0 del a  
if (arreglo1 == arreglo2) //Compara si arreglo 1 es igual a arreglo 2, verificando
```

Para recorrer un arreglo, en el taller usaremos dos formas

Recorrido clásico

```
for (int i = 0; i < arreglo.size(); i++)  
    cout << arreglo[i] << " ";
```

Recorrido por elementos

```
for (tipoDeDato elemento : arreglo)  
    cout << elemento << " "; //Si el elemento es de una estructura, debemos hacer
```

Ejemplo de resolución

"Leer dos arreglos de cinco elementos, ordenarlos de menor a mayor e intercambiarlos. Luego, indicar por pantalla si el elemento 3 del arreglo1 es igual al elemento 4 del arreglo2."

Ejemplo de resolución

"Leer dos arreglos de cinco elementos, ordenarlos de menor a mayor e intercambiarlos. Luego, indicar por pantalla si el elemento 3 del arreglo1 es igual al elemento 4 del arreglo2."

```
#include <iostream>
#include <array>
#include <algorithm>

using namespace std;

int main() {
    array<int, 5> arreglo1;
    array<int, 5> arreglo2;

    for (int i = 0; i < 5; i++)
        cin >> arreglo1[i];

    for (int i = 0; i < 5; i++)
        cin >> arreglo2[i];

    sort(arreglo1.begin(), arreglo2.end());
    arreglo1.swap(arreglo2);

    if (arreglo1[3] == arreglo2[4]) cout << "Son iguales" << endl;
    else cout << "No son iguales" << endl;
}
```

Vector

Array

Vector

Conceptos generales:

- Lista, o arreglo dinámico.
- El tamaño físico es igual al tamaño lógico y puede variar.
- Cuenta con más funciones que array.

Vector

```
#include <vector>
```

Declaración

```
vector<tipoDeDato> nombreDelVector; //vector de tamaño 0  
vector<tipoDeDato> nombreDelVector2(tamaño, valor); //vector de tamaño <tamaño>, c
```

Inicialización

```
vector<tipoDeDato> nombreDelVector {dato1, dato2, dato3}; //vector {dato1, dato2,
```

Acceso y modificación de elementos

```
nombreDelVector[indice] = nuevoValor;
```

Vector - Funciones integradas

Imaginemos los vectores definidos como

```
vector<int> vector1 {1, 2, 3}; //Vector de tamaño 3  
vector<int> vector2 {4, 5, 6}; //Vector de tamaño 3
```

Función front y back (acceso al primer o último elemento).

```
cout << vector1.front(); // Imprime 1  
cout << vector1.back(); // Imprime 3
```

Funciones empty y size

```
cout << vector1.empty() //Imprime 0 (devuelve false) ya que contiene elementos.  
cout << vector1.size() //Imprime 3 ya que el tamaño es 3.
```

Función swap

```
vector1.swap(vector2); //Intercambia el contenido de los vectores vector1 = {1,2,3
```

Vector - Funciones complementarias

```
#include <vector>
#include <algorithm>
```

Imaginemos el vector definido como

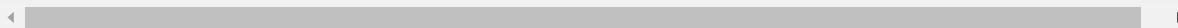
```
vector<int> vector1 {3, 1, 2}; //Vector de tamaño 3
```

Función sort para ordenar de menor a mayor

```
sort(vector1.begin(), vector1.end()); //vector1 pasa a ser {1, 2, 3}
```

Función sort para ordenar de mayor a menor

```
sort(vector1.begin(), vector1.end(), greater<int>()); //vector1 pasa a ser {3, 2,
```



Aclaración: la estrategia del criterio mencionada en array también es válida para vectores

Vector - Funciones complementarias

Función push_back para insertar elementos

```
vector1.push_back(elemento); //Aumenta uno el tamaño de vector1, y agrega el eleme
```

Función pop_back para eliminar el último elemento

```
vector1.pop_back(); //Elimina el último elemento y reduce en uno el tamaño.
```

Función resize para modificar el tamaño del vector

```
vector1.resize(nuevoTamaño); //el vector pasa a tener como tamaño nuevoTamaño. Los
```

Función insert

```
vector1.insert(vector1.begin()+indice, valor); //Inserta el valor en el índice def
```

Función clear

```
vector1.clear(); //elimina todos los elementos y establece el tamaño del vector co
```

Comparación y recorrido de vectores

Para comparar, podemos usar la comparación elemento a elemento, o mirar el total del vector.

```
if (vector1[0] == vector2[5]) //Compara el valor que está en la posición 0 del vec  
if (vector1 == vector2) //Compara si vector 1 es igual a vector 2, verificando si
```

Para recorrer un arreglo, en el taller usaremos dos formas

Recorrido clásico

```
for (int i = 0; i < vector1.size(); i++)  
    cout << vector1[i] << " ";
```

Recorrido por elementos

```
for (tipoDeDato elemento : vector1)  
    cout << elemento << " "; //Si el elemento es de una estructura, debemos hacer
```

Encontrar máximo y mínimo elemento

```
int maximo = *max_element(vector1.begin(), vector1.end());  
int minimo = *min_element(vector1.begin(), vector1.end());
```

Extra: matrices dinámicas

Declaración

```
vector<vector<tipoDeDato>> nombreDeMatriz; //matriz de tamaño 0
```

¿Como podemos ingresar todos sus elementos?

```
nombreDeMatriz.resize(cantFilas);  
int numeroIngreso;  
for (int i = 0; i < cantFilas; i++) {  
    for (int j = 0; j < cantColumnas; j++) {  
        cin >> numeroIngreso;  
        nombreDeMatriz[i].push_back(numeroIngreso);  
    }  
}
```

Recorridos

Recorrido clásico

```
for (unsigned i = 0; i < nombreDeMatriz.size(); i++) {  
    for (unsigned j = 0; j < nombreDeMatriz[i].size(); j++)  
        cout << nombreDeMatriz[i][j] << " ";  
    cout << endl;  
}
```

Recorrido por elementos

```
for (vector<int> fila : nombreDeMatriz) {  
    for (int elemento : fila)  
        cout << elemento << " ";  
    cout << endl;  
}
```

Ejemplo de aplicación

Problema

"Definimos a la mediana como el valor en la posición central de un conjunto de datos **ordenados**.

Escribir un programa que reciba un numero N que representa una cantidad de mediciones, luego, leer N mediciones e indicar cual es la mediana. Finalmente, imprimir los valores ordenados y su promedio.

Aclaración: si N es par, la mediana es el promedio entre los dos valores centrales."

Antes de resolver, simplificamos el problema

"Para un conjunto de números N , calcular su mediana y mostrarlos ordenados."

Resolución

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

using namespace std;

int main() {
    vector<int> mediciones;
    int n, medicion;

    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> medicion;
        mediciones.push_back(medicion);
    }

    sort(mediciones.begin(), mediciones.end());

    cout << "Mediana: ";
    int mitad = mediciones.size()/2;
    if (n % 2 != 0) cout << mediciones[mitad] << endl;
    else cout << (mediciones[mitad - 1] + mediciones[mitad])/2.0 << endl;

    cout << "Arreglo: ";
    for (int elemento : vector1) cout << elemento << " ";
    cout << endl;
}
```

Map

Array

Vector

Map

Conceptos generales:

- El acceso sigue siendo mediante [] pero se cambia el concepto de índice por clave.
- A cada clave existente en un mapa se le corresponde un valor.
- Las claves **no necesariamente deben ser una secuencia ordenada de valores consecutivos**.
- Las claves pueden ser declaradas como de **cualquier tipo**.
- El mapa **automáticamente** está ordenado por sus claves lexicográficamente.

Map

```
#include <map>
```

Declaración

```
map<tipoDeDatoClave, tipoDeDatoValor> nombreDelMapa; //Mapa con 0 elementos.
```

Inicialización

```
map<tipoDeDatoClave, tipoDeDatoValor> nombreDelMapa { {clave1, dato1}, {clave2, da
```

Acceso y modificación de elementos

```
nombreDelMapa[clave] = nuevoValor; //si la clave no tenía ningún valor, el mapa au
```

Map - Funciones integradas

Imaginemos mapas definidos como

```
map<int, int> mapa1 { {1, 1}, {2, 2}, {3, 3} }; //Mapa de tamaño 3
map<int, int> mapa2 { {4, 4}, {5, 5}, {6, 6} }; //Mapa de tamaño 3
```

Funciones empty y size

```
cout << mapa1.empty() //Imprime 0 (devuelve false) ya que contiene elementos.
cout << mapa1.size() //Imprime 3 ya que el tamaño es 3.
```

Función swap

```
mapa1.swap(mapa2); //Intercambia el contenido de los mapas mapa1 = { {1, 1}, {2, 2}
```

Función count

```
mapa1.count(clave) //Devuelve 1 si la clave tiene algún valor asociado, y 0 en cas
```

Comparación y recorrido de arreglos

Para comparar, podemos usar la comparación elemento a elemento, o mirar el total del mapa.

```
if (mapa1["clave1"] == mapa2[5]) //Compara el valor que está en la clave "clave1"  
if (mapa1 == mapa2) //Compara si mapa1 es igual a mapa2, verificando si todos los
```

Aclaración: solo se comparan los valores, no las claves.

Para recorrer un mapa, en el taller usaremos un único tipo de recorrido.

Recorrido por elementos

```
for (pair<tipoDatoClave, tipoDatoValor> elemento : mapa1) {  
    cout << "clave: " << elemento.first << " valor: " << elemento.second << " "; /  
}
```

Ejemplo de aplicación

Problema

"Twitter es una red social que permite a los usuarios enviar mensajes públicos de hasta 240 caracteres. Generalmente dentro de la red se generan "tendencias" gracias a un análisis que hace la plataforma sobre la cantidad de ocurrencias de una palabra durante un período determinado.

Como sos un ingeniero en sistemas de la UTN - FRSF (y, obviamente, un apasionado de los Algoritmos y Estructuras de Datos), decidiste trabajar para Twitter ya que la considerás una gran red social. Para iniciar, te solicitaron que generes un sistema que reciba un número N de palabras (con repetidas entre ellas), y un número M de consultas. Para cada consulta, deberás indicar cuantas apariciones tiene cada palabra."

Antes de resolver, simplificamos el problema

"Para un conjunto de palabras N , contar la frecuencia de cada una. Luego, responder cada consulta con la frecuencia de la palabra correspondiente."

Resolución

```
#include <iostream>
#include <map>
#include <algorithm>

using namespace std;

int main() {
    map<string, int> frecuencias;
    int n, m;
    string palabra;

    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> palabra;
        frecuencias[palabra]++;
    }

    cin >> m;
    for (int i = 0; i < m; i++) {
        cin >> palabra;
        cout << frecuencias[palabra] << endl;
    }
}
```


Set

Array

Vector

Map

Set

Conceptos generales:

- No podemos acceder de forma aleatoria (está prohibido el uso de []).
- Conjunto ordenado (automáticamente) de elementos de un mismo tipo.
- Todos los elementos deben ser distintos.
- Existe una variante llamada "unordered_set" que funciona igual, pero los elementos no están ordenados.

Set

```
#include <set>
```

Declaración

```
set<TipoDeDato> nombreDelSet; //set ordenado  
unordered_set<TipoDeDato> nombreDelUnorderedSet; //set no ordenado
```

Inicialización

```
set<TipoDeDato> nombreDelSet {dato1, dato2, dato3}; //los elementos se ordenan aut
```

Agregar un elemento

```
miSet.insert(elemento); //inserta elemento y re-ordena el set.
```

Comparación y recorrido de set

Podemos comparar mirando el set completo.

```
if (set1 == set2) //Compara si set1 es igual a set2, verificando si todos los elem
```

Para recorrer un set, vamos a usar, al igual que en map, una única forma.

Recorrido por elementos

```
for (tipoDeDato elemento : set)
    cout << elemento << " "; //Si el elemento es de una estructura, debemos hacer
```

Ejemplo de aplicación

Problema

Valentina es una mujer muy dedicada que trabaja hasta tarde todos los días. Para ahorrar tiempo, ella escribe todos los ítems al mismo tiempo que los recuerda. Ella usa una aplicación móvil para esta tarea.

El problema es que esta aplicación no elimina los ítems duplicados y, como Valentina es distraída, ella frecuentemente anota el mismo ítem más de una vez y la lista se hace demasiado larga. Tu trabajo consta en mejorar esa aplicación, por lo que tendrás que escribir un programa que excluya ítems duplicados de la lista de compras y los ordene en orden alfabético.

La entrada consta de un número N (cantidad de ítems), seguida de N ítems separados por un espacio.

Versión simplificada del problema

<https://www.beecrowd.com.br/judge/es/problems/view/2729>

Antes de resolver, simplificamos el problema

Leer un número N de palabras y mostrarlos en pantalla sin repetidos y en orden alfabético.

Resolución

```
#include <iostream>
#include <set>

using namespace std;

int main() {
    set<string> items;
    int n;
    string palabra;

    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> palabra;
        items.insert(palabra);
    }

    for (string elemento : items) cout << elemento << " ";
    cout << endl;
}
```

Aclaración general

- En todas las estructuras vistas, el elemento por default si no hay una asignación es **0**. Los elementos no contienen basura por defecto.

Algunos temas útiles que no se dictan en el taller

- Iteradores
- Deque, queue, priority queue, stack
- Macros

Todos estos temas están contenidos dentro del libro "Competitive Programmer's Handbook"

<https://cses.fi/book/book.pdf>

Formas de seguir aprendiendo

- Resolviendo problemas de jueces en línea (codeforces, leetcode, uri, etc.)
- Participando de competencias de programación.
- Leyendo bibliografía.
- Participando de la comunidad de programación competitiva de la UTN - FRSF. <https://discord.gg/VEsMc9qU>

Fin

Contacto: tassenza@frsf.utn.edu.ar (o, por mensaje privado de Teams).

¡Gracias por participar de parte del Equipo de Algoritmos y Estructuras de Datos!