

CS2013: Programación III

Teoría: Librería Estándar II

José Chávez

Agenda

1. Contenedores Secuenciales
2. Iteradores
3. ¿Qué contenedor secuencial usar?
4. Contenedores Asociativos
5. Contenedor **map**
6. Contenedor **set**

1.

Contenedores Secuenciales

Contenedores Secuenciales

- Un *contenedor* almacena objetos del **mismo tipo**.
- En un contenedor *secuencial* se puede controlar el **orden** en el que uno accede e inserta los elementos.
- El orden representa la **posición** de los elementos.
- Por el contrario, los contenedores *asociativos* almacenan sus elementos utilizando el par *key-value*.

Contenedores Secuenciales

Tipos:

Contenedores Secuenciales

Tipos:

- vector
 - Rápido acceso a cualquier elemento.
 - Insertar/Eliminar un elemento puede ser costoso si no es el último.

Contenedores Secuenciales

Tipos:

- **vector**
 - Rápido acceso a cualquier elemento.
 - Insertar/Eliminar un elemento puede ser costoso si no es el último.
- **list**
 - Acceso secuencial bidireccional.
 - Rápido al insertar/eliminar cualquier elemento.

Contenedores Secuenciales

Tipos:

- `vector`
 - Rápido acceso a cualquier elemento.
 - Insertar/Eliminar un elemento puede ser costoso si no es el último.
- `list`
 - Acceso secuencial bidireccional.
 - Rápido al insertar/eliminar cualquier elemento.
- `forward_list`
 - Acceso secuencial en una sola dirección.
 - Rápido al insertar/eliminar cualquier elemento.

Contenedores Secuenciales

Tipos:

- array
 - Rápido acceso a cualquier elemento.
 - No se puede añadir/eliminar elementos.

Contenedores Secuenciales

Tipos:

- array
 - Rápido acceso a cualquier elemento.
 - No se puede añadir/eliminar elementos.
- string
 - Rápido acceso a cualquier elemento.
 - Rápido al insertar/eliminar elementos al final.

2. Iteradores

Iteradores

```
string s = "laptop";  
if (s.begin() != s.end()){  
    auto it = s.begin();  
    *it = toupper(*it);  
}
```


Un iterador es un tipo de objeto que nos permite navegar a través de los elementos de un contenedor.

Iteradores

```

string s = "Laptop";
if (s.begin() != s.end()){
    auto it = s.begin();
    *it = toupper(*it);
}
  
```

Un iterador que apunta al principio del contenedor



Iteradores

```

string s = "laptop";
if (s.begin() != s.end()){
    auto it = s.begin();
    *it = toupper(*it);
}

```

Un iterador que apunta **a uno después** del final del contenedor

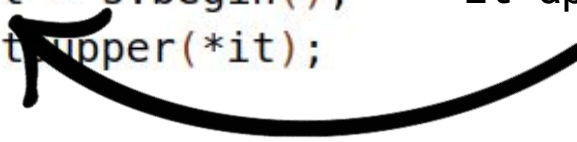
Iteradores

```

string s = "laptop";
if (s.begin() != s.end()){
    auto it = s.begin();
    *it = to_upper(*it);
}

```

it apuntará al primer elemento (la letra l)



Iteradores

```

string s = "laptop";
if (s.begin() != s.end()){
    auto it = s.begin();
    *it = to_upper(*it);
}

```

Podemos reemplazar esta línea por:

`string::iterator it = s.begin()`

Iteradores

```

string s = "laptop";
if (s.begin() != s.end()){
    auto it = s.begin();
    *it = toupper(*it);
}

```

Aquí capitalizamos la palabra laptop



Iteradores

Podemos imprimir
todos los elementos
de una lista STL



```
list<int> l = {1,2,4,5};
```

```
list<int>::iterator i;
for (i = l.begin(); i != l.end(); i++){
    cout << *i << endl;
}
```

Operaciones permitidas

<code>*it</code>	Referencia al elemento apuntado por it
<code>it->mem</code>	El miembro mem del elemento que apunta it
<code>++it</code>	Ahora it apuntará a siguiente elemento
<code>--it</code>	Ahora it apuntará a elemento previo
<code>it1 == it2</code>	Comparar dos iteradores. Dos iteradores serán iguales si apuntan al mismo elemento, o si ambos apuntan una posición más allá del último elemento (en el mismo contenedor).
<code>it1 != it2</code>	

Operaciones permitidas en contenedores: vector y string

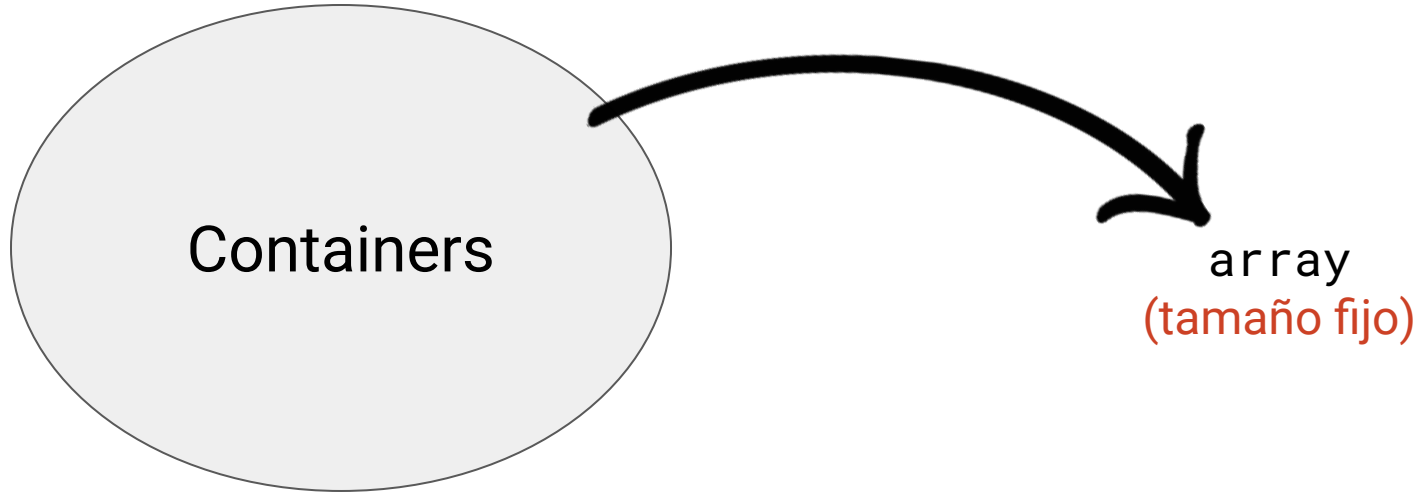
<code>it + n, it - n</code>	El resultado es un iterador que apunta a “n” elementos adelante o atrás del original
<code>it += n, it -= n</code>	La misma operación que la anterior, pero con diferente sintaxis
<code>it1 - it2</code>	El resultado es un iterador tal que si añadimos el iterador <code>it2</code> el resultado será el iterador <code>it1</code>
<code>>, >=, <, <=</code>	Un iterador será menor a otro si este apunta a un elemento en el contenedor que esté ubicado antes del que apunta el otro iterador.

3.

**¿Qué contenedor
secuencial usar?**

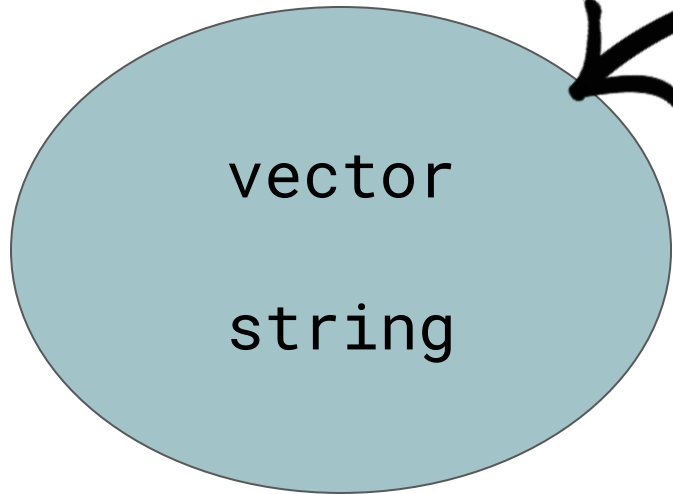
Similitudes y Diferencias

Aumentar y encoger el tamaño



Similitudes y Diferencias

Almacenan sus elementos de
forma contigua

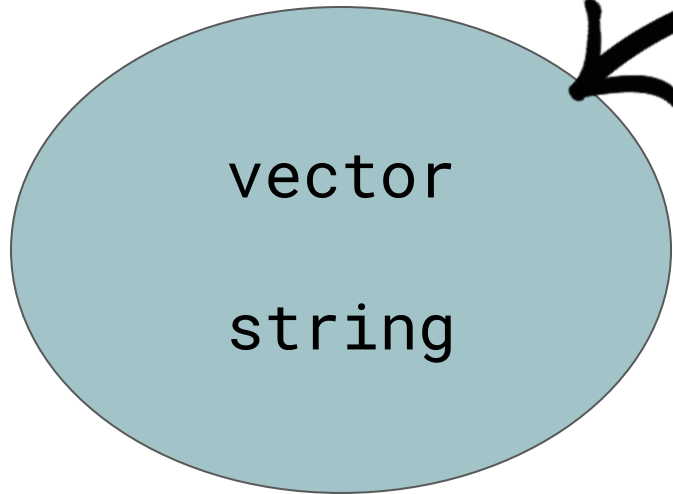


De fácil y rápido acceso

```
vector<int> v{1,2,4,5};  
cout << v[2] << endl;
```

Similitudes y Diferencias

Almacenan sus elementos de
forma contigua



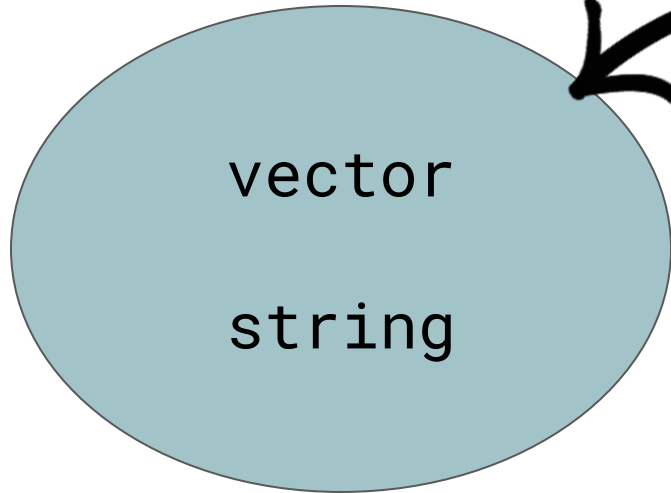
Insertar o remover elementos
del medio tiene un costo
computacional

```
vector<int> v{1,2,4,5};
```

```
v.insert(v.begin()+2,3);
```


Similitudes y Diferencias

Almacenan sus elementos de
forma contigua



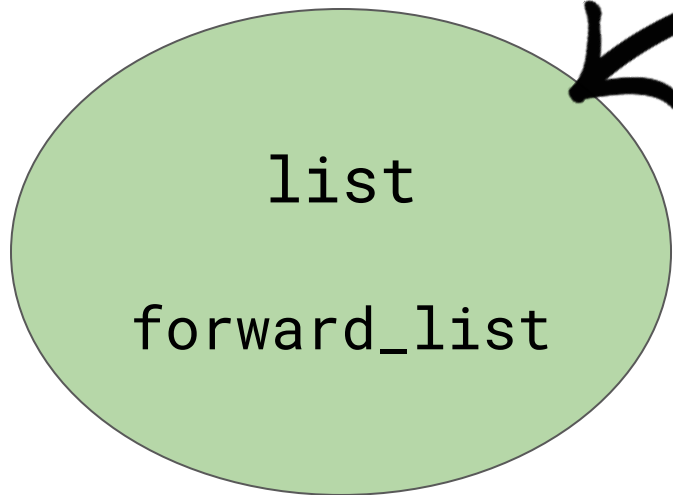
Insertar o remover elementos
del medio tiene un costo
computacional

Listas

```
vector<int> v{1,2,4,5};  
v.insert(v.begin()+2,3);
```

Similitudes y Diferencias

Insertar/Eliminar Elementos en
cualquier posición



Optimizados para insertar y
remover elementos

Operaciones permitidas en forward_list

```
forward_list<int> lista = {1,4,5};  
lista.insert_after(lista.begin(),2);
```

```
for (int e: lista)  
    cout << e << " ";  
cout<< endl;
```

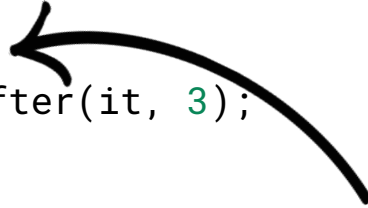
Salida:

1 2 4 5

Operaciones permitidas en `forward_list`

```

forward_list<int>::iterator it = lista.begin();
it = next(it);
lista.insert_after(it, 3);
  
```



El siguiente iterador

Salida:

1 2 3 4 5

Operaciones permitidas en `forward_list`

```
lista.insert_after(next(lista.begin()), 3);
```



Salida:

1 2 3 4 5

Se insertará después de la
segunda posición

Operaciones permitidas en `forward_list`

```
lista.push_front(0);
```

```
for (int e: lista)  
    cout << e << endl;
```

Salida:

0 1 2 3 4 5

Operaciones permitidas en `forward_list`

```
lista.pop_front();
```

```
for (int e: lista)  
    cout << e << endl;
```

Salida:

1 2 3 4 5

Operaciones permitidas en forward_list

```
lista.push_front(3);
```

```
lista.remove(3);
```

Salida:

3 1 2 3 4 5

```
for (int e: lista)
```

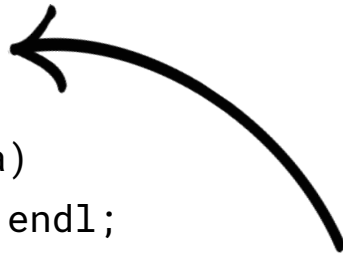
```
    cout << e << endl;
```

Salida:

1 2 4 5

Operaciones permitidas en `forward_list`

```
lista.push_front(3);
lista.remove(3);
for (int e: lista)
    cout << e << endl;
```



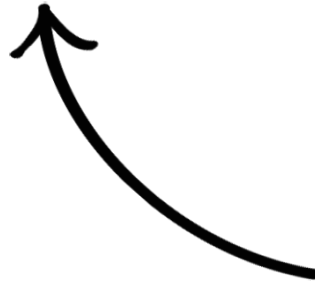
Aquí eliminamos todos los 3

Salida:
1 2 4 5

Operaciones permitidas en `forward_list`

```
lista.remove_if([](int x){return x % 2 == 0;});
```

```
for (int e: lista)
    cout << e << endl;
```



Función Lambda para calcular los números pares

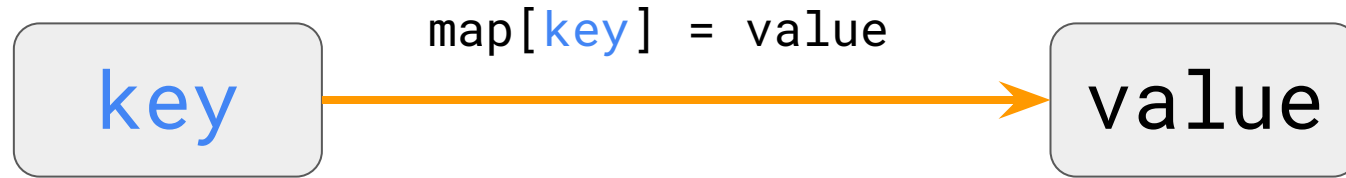
Salida:

1 5

4●

Contenedores Asociativos

Contenedores Asociativos



Los elementos en un contenedor asociativo se almacenan y recuperan mediante una clave

5. Contenedor map

Contenedor `map`

Ejemplo:

Implementar un contador de palabras utilizando el contenedor `map`.

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

using namespace std;

int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}
```

```

#include <iostream>
#include <map>

```

```
using namespace std;
```

```

int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

Un contenedor **map** con **keys** de tipo string y con **values** de tipo int


```

#include <iostream>
#include <map>

using namespace std;

int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin, word)){
        if (word.begin() == word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

Aquí almacenaremos una palabra

```

#include <iostream>
#include <map>

using namespace std;

int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin, word)){
        if (word.begin() == word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

Aquí el contador

```

#include <iostream>
#include <map>

using namespace std;

int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

obtenemos una palabra por línea

```

#include <iostream>
#include <map>

using namespace std;

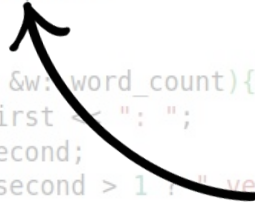
int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces" : " una") << endl;
    }

    return 0;
}

```



Si la palabra no se encuentra en el contenedor se creará un nuevo elemento con **key** igual a `word` y un **value** igual 0. Luego este valor se incrementa en 1.

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

using namespace std;


int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces" : " una") << endl;
    }

    return 0;
}
```

Si la palabra existe en el contenedor, su valor se incrementa en 1.



```

#include <iostream>
#include <map>

using namespace std;

int main(){
    map<string, int> word_count;
    string word;


    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

Si no se ingresa ningún símbolo/palabra,
salimos del bucle



```

#include <iostream>
#include <map>

using namespace std;

int main(){
    map<string, int> word_count;
    string word;

    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

Iteramos a través del contenedor map



```

#include <iostream>
#include <map>

using namespace std;

int main(){
    map<string, int> word_count;
    string word;

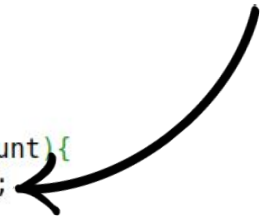
    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << ": ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

Imprimimos el **key**




```

#include <iostream>
#include <map>

using namespace std;

int main(){
    map<string, int> word_count;
    string word;

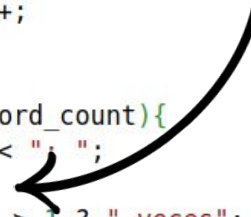
    while (getline(cin,word)){
        if (word.begin()==word.end())
            break;
        word_count[word]++;
    }

    for (const auto &w: word_count){
        cout << w.first << " ";
        cout << w.second;
        cout << (w.second > 1 ? " veces": " vez") << endl;
    }

    return 0;
}

```

Imprimimos el **value**



Contenedor map

```
map<string, int> word_count;
```

(Inicializar un map vacío)

Contenedor map

```
map<string, int> word_count;
```

(Inicializar un map vacío)

```
map<string, int> word_count = {{ "Hola", 1 }, { "Mundo", 1 } };
```

(Inicializar un map con elementos)

6. Contenedor set

Contenedor `set`

Ejemplo:

Implementar un contador de palabras sin contar conectores.

Contenedor set

```
#include <set>

int main(){
    map<string, int> word_count;
    string word;

    set<string> stop_words = {"en", "y"};

    while (getline(cin, word)){
        if (word.begin() == word.end())
            break;
        if (stop_words.find(word) == stop_words.end())
            word_count[word]++;
    }
}
```

Contenedor set

```

#include <set>

int main(){
    map<string, int> word_count;
    string word;

    set<string> stop_words = {"en", "y"};

    while (getline(cin, word)){
        if (word.begin() == word.end())
            break;
        if (stop_words.find(word) == stop_words.end())
            word_count[word]++;
    }
  
```

Un `set` con dos elementos



Contenedor set

```
#include <set>

int main(){
    map<string, int> word_count;
    string word;

    set<string> stop_words = {"en", "y"};

    while (getline(cin, word)){
        if (word.begin() == word.end())
            break;
        if (stop_words.find(word) == stop_words.end())
            word_count[word]++;
    }
```

Buscamos la palabra
dentro del conjunto



Contenedor set

```

#include <set>

int main(){
    map<string, int> word_count;
    string word;

    set<string> stop_words = {"en", "y"};

    while (getline(cin, word)){
        if (word.begin() == word.end())
            break;
        if (stop_words.find(word) == stop_words.end())
            word_count[word]++;
    }
  
```

Buscamos la palabra dentro del conjunto

Si no se encuentra la palabra dentro de `stop_words`, entonces se cuenta

Contenedor `set`

```
set<int> s;
```

(Inicializar un `set` vacío)

```
set<int> s = {3,4,5,6};
```

(Inicializar un `set` con elementos)

Contenedor set

```
vector<int> v;

for (int i = 0; i < 4; i++){
    v.push_back(i);
    v.push_back(i);
}

set<int> s;
for (const int& e: v)
    s.insert(e);

cout << v.size() << endl; // Salida: 8
cout << s.size() << endl; // Salida: 4
```

Contenedor set

```
vector<int> v;
```

```
for (int i = 0; i < 4; i++){
    v.push_back(i);
    v.push_back(i);
}
```

```
set<int> s;
for (const int& e: v)
    s.insert(e);
```

```
cout << v.size() << endl; // Salida: 8
cout << s.size() << endl; // Salida: 4
```

No cuenta los
elementos repetidos



Contenedor set

```
vector<int> v;
```

```
for (int i = 0; i < 4; i++){
    v.push_back(i);
    v.push_back(i);
}
```

Puedo crear los elementos directamente
con los elementos del vector

```
set<int> s(v.begin(), v.end());
```

```
cout << v.size() << endl; // Salida: 8
```

```
cout << s.size() << endl; // Salida: 4
```



Resumen

En esta sesión se practicaron los tópicos siguientes:

- Contenedores secuenciales
- Iteradores
- Contenedores asociativos

