lecture5

Lecture 5: Containers

What: Containers

What more: STL vector set stack queue map

---

1. What the heck is the STL? What are templates?

2025Fall-05-Containers, p.21

> STL: Standard Template Library

- **All STL containers are templates!**
- **STL include**: Containers, Iterators, Functions, Algorithms.

---

2. Sequence Containers

> Store a linear sequence of elements.

2025Fall-05-Containers, p.35

> Sequence containers store a linear sequence of elements

2.1 Vector

- **Initialization**:
  - `std::vector<int> v`
  - `std::vector<int> v(n)`
  - `std::vector<int> v(n, k)`
- **Functions**:
  - `v.push_back()` / `v.clear()`

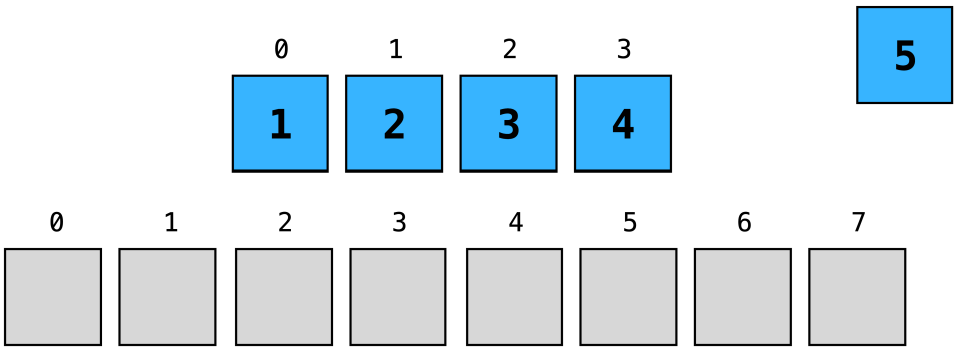
- `if(v.empty())` --> if v is empty
- `v[i]` and `v.at(i)`：区别在于 **Bounds checking**（边界检查）。`at()` 会报错 runtime error，但 `operator[]` 只有 undefined behavior。

Stanford vs. STL vector

What you want to do?	Stanford `Vector<int>`	`std::vector<int>`
Create an empty vector	`Vector<int> v;`	`std::vector<int> v;`
Create a vector with **n** copies of **0**	`Vector<int> v(n);`	`std::vector<int> v(n);`
Create a vector with **n** copies of value **k**	`Vector<int> v(n, k);`	`std::vector<int> v(n, k);`
Add **k** to the end of the vector	`v.add(k);`	`v.push_back(k);`
Clear vector	`v.clear();`	`v.clear();`
Check if **v** is empty	`if (v.isEmpty())`	`if (v.empty())`
Get the element at index **i**	`int v = v.get(i);` `int k = v[i];`	`int k = v.at(i);` `int k = v[i];`
Replace the element at index **i**	`v.get(i) = k;` `v[i] = k;`	`v.at(i) = k;` `v[i] = k;`

- **Implementation 建议**:

How is vector implemented?



C++

```
for(auto elem : vec){
   cout << elem << " ";
}
```

- **优化**: 如果是只读元素且数据类型占有内存很大：

```
for(const auto & elem : v)
```

这种写法不复制原数据，仅仅只读。

Tip: Use `const auto&` when possible

```
std::vector<MassiveType> vec { ... };
for (auto elem : vec) ...
```

```
for (const auto& elem : v)
```

- Applies for all iterable containers, not just `std::vector`
- Saves making a potentially expensive copy of each element

- **局限性**: 滑动窗口。让 vector 前面插入一个，后面删除一个，无法做到。

`std::vector` is not the best for all cases...

- Suppose we need to observe the last 10,000 prices of a stock
- What might be concerning about the code below?

```
void receivePrice(vector<double>& prices, double price)
{
  prices.push_front(price);
  if (prices.size() > 10000)
    prices.pop_back();   // Remove last price
                         // so we don't exceed 10k
}
```

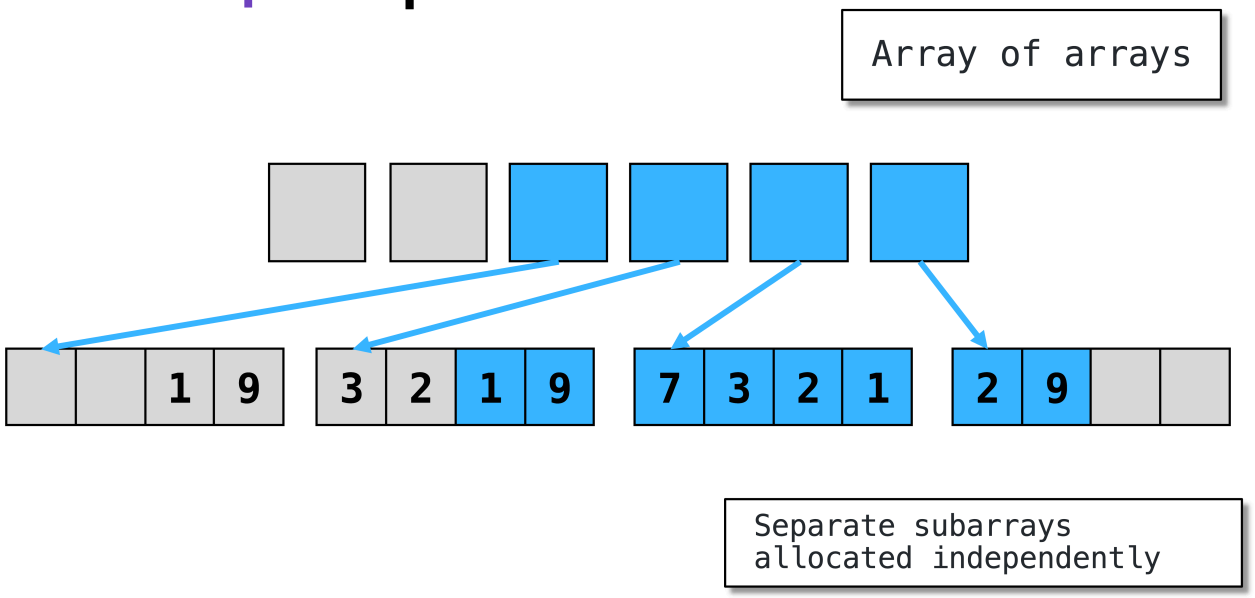
2.2 Deque

2025Fall-05-Containers, p.53

> A deque has the same interface as vector, except we can push_front / pop_front

- **特点**: 相比 vector，deque 可以前插可以后面删除。
- **Performance**: **Efficient insertion/removal**.
- **Implementation**: 分成小数组思想。前后各自留有位置，便于前后插入数据。

How is deque implemented?



---

3. Associative Containers

2025Fall-05-Containers, p.58

> Associative containers organize elements by unique keys

3.1 Map

- **Functions**:
  - `m.insert{{k, v}}` / `m[k] = v` / `m.erase(k)`
  - `if(m.count(k))` or `if(m.contains(k))`
  - `int i = m[k]` --> **retrieve/overwrite**

Stanford vs. STL map

What you want to do?	Stanford Map<char, int>	std::map<char, int>
Create an empty map	`Map<char, int> m;`	`std::map<char, int> m;`
Add key k with value v into the map	`m.put(k, v);` `m[k] = v;`	`m.insert({k, v});` `m[k] = v;`
Remove key k from the map	`m.remove(k);`	`m.erase(k);`
Check if k is in the map *(* C++20)*	`if (m.containsKey(k))`	`if (m.count(k))` `if (m.contains(k))` *(*)*
Check if the map is empty	`if (m.isEmpty())`	`if (m.empty())`
Retrieve or overwrite value associated with key k *(auto-insert default if doesn't exist)*	`int i = m[k];` `m[k] = i;`	`int i = m[k];` `m[k] = i;`

- **Nature**: Map 由 pair 组成 (**Map as a collection of pair**)。

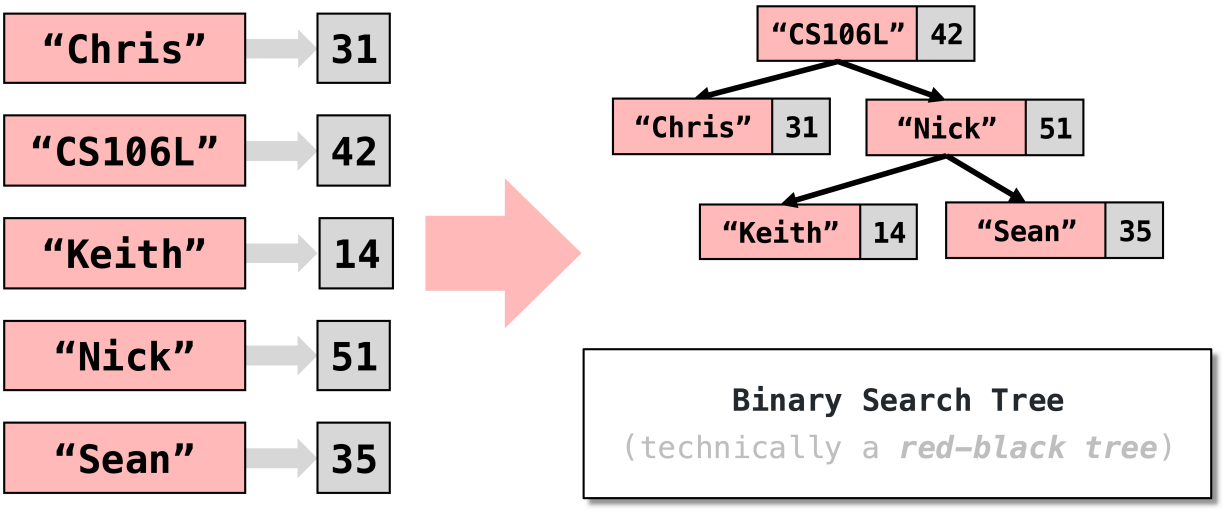
map as a collection of pair

We can iterate through the key-value pairs using a range based for loop

```
std::map<std::string, int> map;

for (auto kv : map) {
  // kv is a std::pair<const std::string, int>
  std::string key = kv.first;
  int value = kv.second;
}
```

- **Implementation**: **红黑树**（一种二叉搜索树）。

How is map implemented?



- **注意**: `map<K, V>` 的 K 必须支持 **"<"** 操作符（就是可以比较大小）。

3.2 Set

- **Functions**: `s.insert(k)`, `s.erase(k)`, `s.count(k)`, `s.empty()`.

Stanford vs. STL set

What you want to do?	Stanford Set<char>	std::set<char>
Create an empty set	`Set<char> s;`	`std::set<char> s;`
Add k to the set	`s.add(k);`	`s.insert(k);`
Remove k from the set	`s.remove(k);`	`s.erase(k);`
Check if k is in the set *(* C++20)*	`if (s.contains(k))`	`if (s.count(k))` `if (s.contains(k))` *(*)*
Check if the set is empty	`if (s.isEmpty())`	`if (s.empty())`

- **Implementation**: 同 map，基于红黑树。

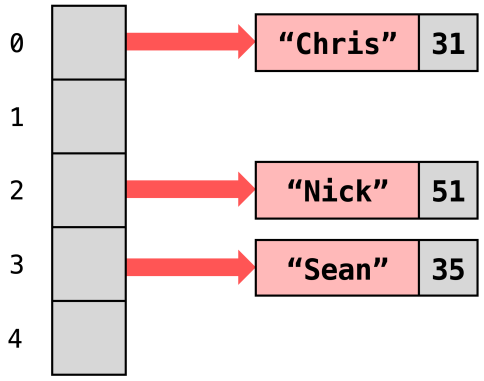
---

4. Unordered_map / set

- **特点**: 经过优化的 (**Optimized**)。
- **Implementation**: **Hash table** (哈希表)。

How is `unordered_map` implemented?

- Remember, `map` is a collection of `std::pair`
- `unordered_map` stores a collection of n "buckets" of pairs

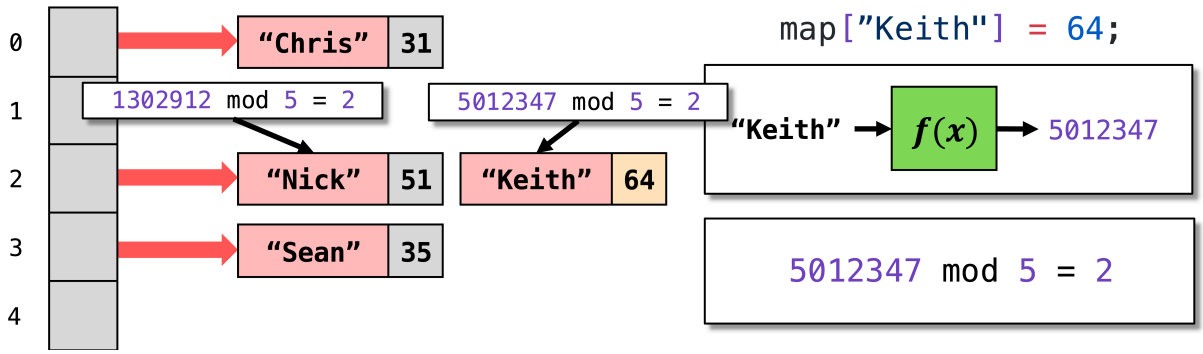



```
std::unordered_map
<std::string, int> map {
   { "Chris", 31 },
   { "Nick", 51 },
   { "Sean", 35 },
};
```

- **步骤**:
  1. 先对 key 做 hash 函数变化 --> 。
  2. 取 ，决定放在哪个桶（Bucket)。
  3. 若多个 key 分到一个桶，共存一个桶 (**Hash collision**)。

How is `unordered_map` implemented?

- If two keys hash to the same bucket, we get a **hash collision**
- During lookup, we loop through bucket and check key equality
  - Two keys with the same hash are not necessarily equal!



- **Hash Function 要求**:
  1. 打乱，随机。
  2. **雪崩效应**：Small changes produce large changes.
  3. `std::unordered_map<K, V>` requires K to have a hash function (and equality).

4.1 Load Factor (负载因子)

- **Definition**: (Average number items per bucket).

- **Optimization**: `unordered_map` allows super fast lookup by keeping load factor small.
- **Possion 分布**: 默认 load factor = 1 时，桶内元素个数：
  - 0 个: 36.8%
  - 1 个: 36.8%
  - 2 个: 18.4%
- **Rehash**: If load factor gets too large (above 1.0 by default), we **rehash**. 你可以手动调整这个 `max_load_factor`。

Fun C++ Trivia: `max_load_factor`

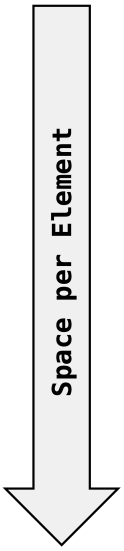
You can control the max load factor before rehashing

```cpp
std::unordered_map<std::string, int> map;

double lf = map.load_factor(); // Get current load factor
map.max_load_factor(2.0); // Set the max load factor

// Now the map will not rehash until load factor exceeds 2.0
// You should almost never need to do this,
// but it's a fun fact (good for parties!)
```

5. Summary & When to use

- **A good hash function**: 均匀分布、雪崩效应、确定性、高效计算。
- **Selection**: `unordered_map` 比 `map` 更快（一个，一个），但花大内存。若 key_type 不用比较大小，可以用。
- **补充**: `multiset` 和 `multimap` 可以包含可重复元素和键值对。

Summary of Data Structures



	i^{th} element	Search	Insertion	Erase
std::vector	Very Fast	Slow	Slow	Slow
std::deque	Fast	Slow	Fast (front/back) Slow (all others)	Fast (front/back) Slow (all others)
std::set	Slow	Fast	Fast	Fast
std::map	Slow	Fast	Fast	Fast
std::unordered_set	N/A	Very Fast	Very Fast	Very Fast
std::unordered_map	N/A	Very Fast	Very Fast	Very Fast

Recap

- What the heck is the STL? What are templates?
  - "The Standard Template Library"
- Sequence Containers
  - A linear sequence of elements
  - `std::vector, std::deque`
- Associative Containers
  - A set of elements organized by unique keys
  - `std::map, std::set, std::unordered_map, std::unordered_set`