Maps

CSE 232 - Dr. Josh Nahum

Reading:

Section 12.3-12.8

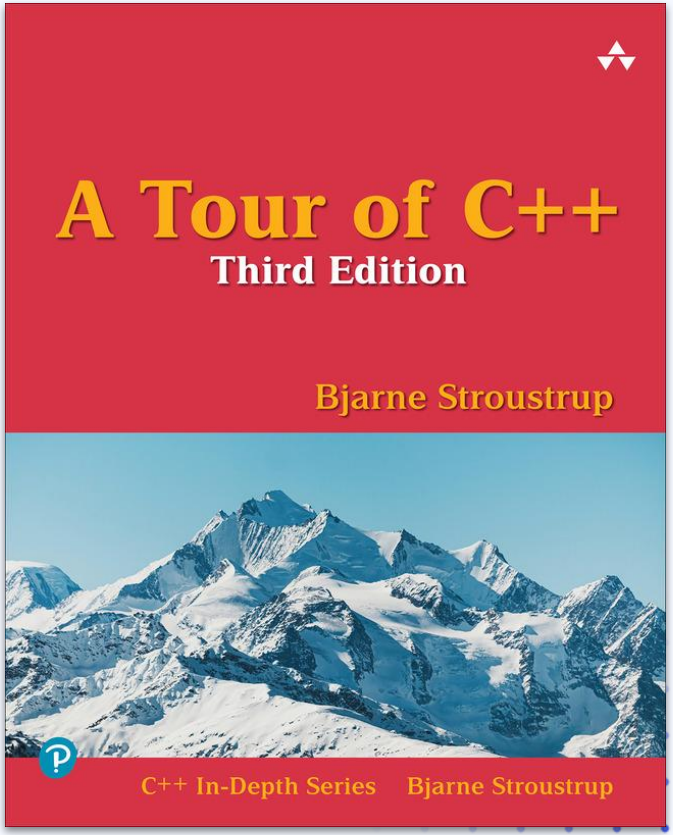

Table of contents

00

Pair and Iteration

std::pair

Hold two values

Those values don't have to have the same type

Example:

```
std::pair<string, int> word_count{
  "Mal", 100};
cout << word_count.first
     << word_count.second;
```

struct

pair is a struct and has two public data members

<utility>

pair is a templated type provided by the utility header

Range-based For Loop

```
map<string, int> phone_book {
  {"David Hume", 123456},
  {"Karl Popper", 234567},
  {"BAWR", 345678}
};
for (pair<string, int> entry : phone_book) {
  cout << entry.second << " -> "
       << entry.first << endl;
}
```

```
// Structure binding
for (auto [name, number] : phone_book) {
  cout << number << " -> "
       << name << endl;
}
```

01

Insert

Insert

No push_back

The order of a map is determined by its key (or hash of the key), you can't append an element to a map

Instead insert

```
map<string, int> m;
string word = “hello”;

m.insert(pair<string, int>{word, 1});
// or
m.insert({word, 1});
```

Insert's return value

pair<iterator, bool>

If inserted key is already in map, insert does nothing and the second attribute is false.
If key is not in the map, insert adds the pair and the second attribute is true.
The first element is an iterator pointing at the pair (either newly inserted or already there).

```
map<string, int> m {{"Mal", 100}};

// New Key
pair<map<string, int>::iterator, bool> result =
  m.insert({"Josh", 50});
cout << "Successful insert? " << result.second << endl;
pair<string, int> entry = *(result.first);
cout << entry.first << entry.second << endl;

// Old Key
result = m.insert({"Mal", 1000000});
cout << "Successful insert? " << result.second << endl;
entry = *(result.first);
cout << entry.first << entry.second << endl;
```

Auto

```
std::pair<std::map<std::string,
int>::iterator, bool> result =
m.insert({"Josh", 50});
```

Above is a initialization of the return value for the insert method of a map of strings to ints. It is quite long and it is reasonable to use auto to avoid typing it out as any developer comfortable with maps would know the type to expect.

```
auto result = m.insert({"Josh", 50});
```

The above does exactly the same thing, but uses type inference to avoid the need to spell out that long type declaration.

But beware, only use auto if you are certain of the type it will infer. Auto can be used to conceal problems that would be obvious if the types were made explicit.

02

Find

Lookup

```
int & number = phone_book["Josh"];
cout << number;
```

The above code will print "0" if the key Josh isn't in the map. But it has the side-effect of adding a new entry to the phone_book ({"Josh", 0}). Sometimes you want to lookup a key without adding it to the map.

```
map<string, int>::iterator iter =
    phone_book.find("David Hume");
if (iter != phone_book.end()) {
  cout << (*iter).second;
}
```

The find member function takes a key and returns an iterator to the entry matching that key. If the key isn't in the map, it returns an iterator that points at one past the last element (a.k.a. .end()).

03

Other Containers

Lists

Vector Alts

Many other sequence types (list, forward_list, deque) all have the same interface as vector.

Optimizations

They should not be your first choice. Only use them if you measure that their performance is better than vector for your application.

CSE 331

In Data Structures and Algorithms, you will learn the different performance characteristics of all these types.

Will it be on the test?

Yes

You need to be proficient with vector, map, and unordered_map.

No

The other types (and allocators) in Chapter 12 will not be involved in exams (but they may appear in labs and homework assignments).

Attribution

Please ask questions via Piazza

Dr. Joshua Nahum
www.nahum.us
EB 3504

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**