Vector Details

CSE 232 - Dr. Josh Nahum

Reading:

Section 12.1 and Section 12.2

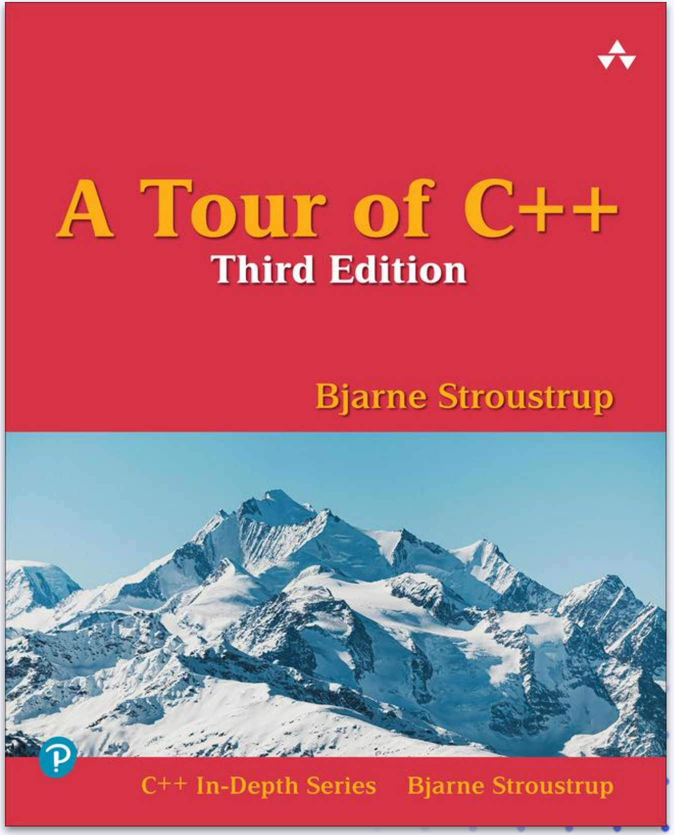


Table of contents

00

Iterators

01

Capacity

02

Invalidation

03

Front/Back

00

Iterators

Pointers and Arrays

An arrays is a set of contiguous objects allocated in memory. The name of the array can be used like a pointer to the first element. Adding an integer to that pointer yields a pointer to that index.

A pointer can be used to access elements in the array, and can be incremented to point at the next element.

```
int const size = 4;
char array[size] = {'a', 'b', 'c', 'd'};
char * beginning = array;
char * one_past_end = array + size;

for (char const * current_element = beginning;
    current_element != one_past_end;
    ++current_element) {
  std::cout << *current_element;
}
std::cout << std::endl;
```

.begin() and .end()

All the std containers (e.g. vector, initializer_list, string, e.t.c.) have member functions that return a pointer-like object called an iterator.

Iterators are used to access elements within the container. begin() returns an iterator that points at the first element. end() returns an iterator that points at one past the last element.

```
std::vector<char> vec = {'a', 'b', 'c', 'd'};
for (std::vector<char>::iterator iter{vec.begin()};
    iter != vec.end(); ++iter) {
  std::cout << *iter;
}
std::cout << std::endl;
```

Why "one past the last"

At first glance it might seem strange that end returns a pointer that points to "one past the last" instead of just pointing at the last element.

But this way makes it easier to write loops like the one on the right and also makes such loops automatically also work on containers of size 0.

Test it out!

```
std::vector<char> vec = {'a', 'b', 'c', 'd'};
for (std::vector<char>::iterator iter{vec.begin()};
    iter != vec.end(); ++iter) {
  std::cout << *iter;
}
std::cout << std::endl;
```

Other Iterators

Reverse

rbegin() and rend() return iterators to the last element and one before the first element. Incrementing a reverse iterator moves it toward the first element. Any loops that you write with regular begin and end can be made to loop in the reverse order by just adding those 'r's.

Const

cbegin() and cend() return iterators to const (similar to pointers to const). They are used when you don't need to change elements in the container (the previous examples could have used these functions).

Both

crbegin() and crend() also exist when you want a reverse iterator to const.

> “Iterators are a topic that will become much more useful next week with generic algorithms!”

—Dr. Nahum

01

Capacity

Conditional Operator

```
\\ In section 12.2
  reserve(size()==0?8:2*size());
\\ ...
cond ? if_true_value : if_false_value
```

The conditional operator is the only operator with 3 operands (arguments). It is sometimes called ternary operator for that reason. It is similar to a simple if statement, but it returns a value (because it is an expression).

The argument in the call to reserve means, "if size() is equal to 0, the vale returned is 8, else the value returned is twice the size()".

Reading Review

`size()`

The number of elements in the vector

`capacity()`

The max size the vector can have before a memory allocation is required

`reserve(int)`

Expand the capacity to the given argument allocate memory if needed

`push_back(elem)`

Add elem to the end of the vector, increasing the size by one. Increase capacity if required.

Who cares?

Memory allocation is sometimes an expensive operation, so making that occur when at the developer's control is one type of possible optimization.

However, the more important reason is the next topic, **invalidation**!

02

Invalidation

Breaking Iterators!

Hardware

Iterators are basically wrappers around pointers to add safety and other features.

Arrays

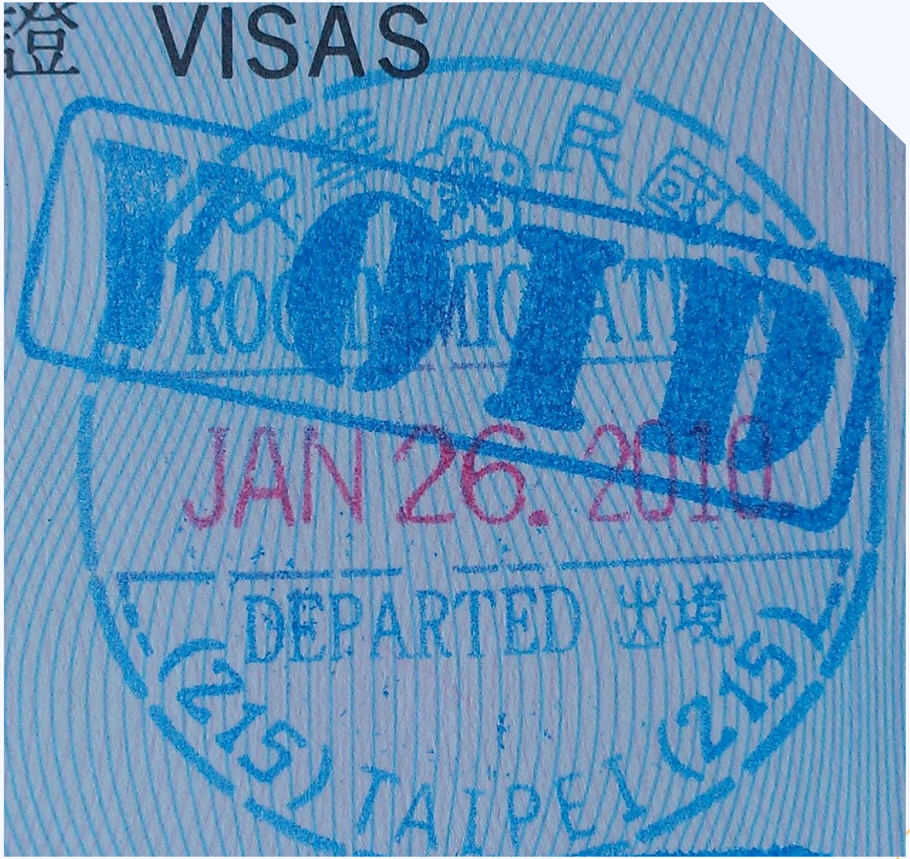
And just like if you copy an array, the pointers to the old array won't work with the new copy.

Memory Allocation

When a vector does a memory allocation (e.g. when push_back causes the size to exceed its capacity), all the iterators are invalidated.

Danger!

Do not use iterators to data structures while you are changing their size. Because memory allocations will invalidate iterators, and using an invalid iterator is undefined behavior! Best case scenario you get a segfault.



03

Front/Back

Reference to the ends

front()

Returns a reference to the first element.

back()

Returns a reference to the last element.

Consult the C++ docs to see what happens if called on an empty container.

Other Useful Members

empty

Returns true if size is 0

clear

Removes all elements

insert

Adds an element into a specified position

pop_back

Removes the last element

resize

Removes elements or adds elements to achieve desired size

swap

Exchanges contents/data with another container

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**