Containers and algorithms — session 2

Tristan Brindle

Feedback

- We'd love to hear from you!
- The easiest way is via the *cpplang* channel on Slack — we have our own chatroom, *#cpplondonuni*
- Go to https://cpplang.now.sh/ for an "invitation"

Advent of Code

- For those who like programming challenges, Advent of Code is a daily series of problems running from 1st December to Christmas Day

- Join up at adventofcode.com

- If you like, you can join the C++ London Uni leaderboard! See the Slack channel for the code.

- My solutions: https://github.com/tcbrindle/advent_of_code_2018

- Oli's solutions: https://github.com/Olipro/AdventOfCode-2018

Last week

- End-of-module quiz! (again)

- Quiz solutions

- Introduction to the STL

This week

- Further STL intro

- Iterators

Last week's homework

- https://github.com/CPPLondonUni/stl_week1_class_exercise

  - Exercise 1: given a vector of words, can you print them in alphabetical order?

  - Exercise 2: given a list of random numbers, what is the minimum and maximum? What is the median?

Solution

- https://github.com/CPPLondonUni/stl_week1_class_exercise/tree/solution

Any questions before we move on?

The standard template library

- The **standard template library** (STL) was invented by Alexander Stepanov in the early 1990s

- It provided a set of **container classes** and fundamental **algorithms**

- The STL pioneered the concept of *generic programming*, revolutionising the way C++ was written and used

- Stepanov's STL formed the basis for much of the C++ standard library that we use today

The standard template library

- It is common (although technically incorrect) to refer to the containers and algorithms part of the standard library as “the STL”

- Today’s standard library provides around a dozen containers and more than 90 algorithms which we can (and should) use in our programs

- Get to know the standard library algorithms! Use them whenever possible.

Introducing Iterators

- *Iterators* are the “glue” that binds together containers and algorithms

- Containers *provide* iterators, and algorithms *use* them

- For example:

```
std::vector<int> vec{5, 4, 3, 2, 1};
auto first = std::begin(vec); // iterator to start of container
auto last = std::end(vec); // iterator to end of container
std::sort(first, last); // call algorithm on iterator pair
```

- There is no single iterator *class* — rather, an iterator is a generic *concept* (or family of concepts) which classes can *model*

What is an iterator?

- The simple answer: an iterator is just an *index* into some collection

- (Trivia: Stepanov originally called them *linear coordinates*)

- The complicated answer: iterators are a *generalisation of pointers*

- Just as a raw pointer can point to an element of a C array, so an iterator points to an element of a more complex container

Iterators and ranges

- Iterators are generally used in *pairs* — almost all standard algorithms operate on a pair of iterators

- A pair of iterators denotes a *range*

- The first iterator in the pair points to the *start of the range*

- The second iterator in the pair points to one place *past the end of the range*

- **It is an error to dereference a past-the-end iterator!**

Iterators and ranges

- We can obtain an iterator to the start of a container by calling `container.begin()`
- We can obtain an iterator to (one past) the end of a container by calling `container.end()`
- The standard library has free functions `std::begin(c)` and `std::end(c)` which do the same thing

Iterator fundamentals

- An iterator is an object which represents a *position* in a collection

- If an element exists at that position, we say the iterator is *valid*; otherwise we say it is *invalid*

- Iterators are *value types*: that is, copy and assignment have their usual meanings

- We can also compare iterators for equality: two iterators are equal if they point to the *same position in the same collection*

Iterator fundamentals


```cpp
std::vector<int> vec{1, 2, 3};

auto it1 = vec.begin();
// We can copy iterators
auto it2 = it1;
// it2 denotes the same position in the same collection
assert(it1 == it2);
// We can assign to iterators
it2 = vec.end();
// The iterators are no longer equal
assert(it1 != it2);
```

Iterator fundamentals

- For a valid iterator, we can obtain a reference to the element of the collection that it points to

- This is called *dereferencing* the iterator

- We write this as `*iter`

- If the returned reference is *read-only*, we call the iterator a *const iterator*.

- **It is an error to dereference an invalid iterator!**

Iterator fundamentals


```cpp
std::vector<int> vec{1, 2, 3};

auto it1 = vec.begin();
std::cout << *it1 << '\n';
// prints 1
*it1 = 42;
std::cout << *it1 << '\n';
```

```cpp
const std::vector<int> cvec{3, 2, 1};
auto it2 = cvec.begin();
std::cout << *it2 << '\n';
// prints 3
*it2 = 42;
// Compile error -- it2 is a const iterator

auto it3 = vec.end();
std::cout << *it3 << '\n';
// Undefined behaviour -- it3 is not a valid iterator
// (May print junk, or just crash)
```

Iterator fundamentals

- We can *increment* a valid iterator so that it points to the next position in the collection

- We write this as `++iter` (as with `int`s)

- As with `int`s, we can also write `iter++`, which increments the iterator but returns the previous position

- The standard library function `std::next(iter)` returns a new iterator which points to one place after `iter`.

Iterator fundamentals


```cpp
std::array<float, 12> arr{0.0f, };

auto it1 = arr.begin();
// it1 points to the element at position zero
++it1;
// it1 points to the element at position one
auto it2 = std::next(it1);
// it2 points to the element at position two
// it1 still points to the element at position one

// What does this do?
for (auto it = arr.begin(); it != arr.end(); ++it) {
    *it = 99;
}
```

Online resources

- https://isocpp.org/get-started
- cppreference.com — The bible, but aimed at experts
- cplusplus.com — Another reference site, also has a tutorial section
- learncpp.com — Free online tutorial, very up-to-date
- https://www.pluralsight.com/authors/kate-gregory - Comprehensive set of courses from an experienced C++ trainer (free trial)
- reddit.com/r/cpp_questions
- Cpplang Slack channel — https://cpplang.now.sh/ for an “invite”
- StackOverflow (but…)