C++ Sydney Meetup
Using Concepts through the Ranges TS

Author

Christopher Di Bella

March 29, 2017

Who am I?

- Software developer in finance
- Tutor for UNSW
- C++ enthusiast

Thanks

- Casey Carter, Eric Niebler, Andrew Sutton
- Sergey Zubkov
- Daryl D'Souza
- Oswyn Brent, Manuel Chakravarty

Aims

- Start using Concepts
- Start using Ranges
- Do it now!

Concepts TS

- Predicates over templates
  - Impose requirements on type deduction
  - Check *syntax* requirements
  - Similar to a bouncer at a club
  - Better diagnostics
- Defined in the Concepts Technical Specification

Ranges TS

- Specifies standard concepts
  - Standard concepts *designed* to enforce *semantic* requirements
- Specifies STL algorithm replacements
  - New algorithms formally check concepts
  - Also specify range-based algorithms
- TS should be ratified in July

But Concepts and Ranges aren't Standard!

- It's true, they didn't make it into C++17 ☹
- But they *are* Standard C++
- A Technical Specification is like a beta branch for C++

Warm-up problem

- UNSW COMP6771 students had to solve this on their own

What's the issue?

- Templates don't provide a clear solution
- Unclear diagnostic
- Solutions involve too much:
  - Developer work + maintenance
  - Clever, but unclear solutions (e.g. `iterator_traits`/`enable_if`, etc.)
  - Ewww....

Ranges TS give a clear, easy solution

1. Include appropriate header
2. Rename namespace to something humanly typeable
3. Add requires clause (maybe?)
4. That's it!

What is this `Sentinel`?

- Denotes the end of a range
- Endcodes the end of a range into the type system
- Offers more generality
- Eliminates dummy end iterators
- Must be `EqualityComparable` with the `Iterator` parameter

Constrasting diagnostics

- Remove `make_vector(size_t, double)`
- Recompile with Concepts fix
- Diagnostics are *explicitly* informative

Compiling with Concepts

- Need GCC 6.1 or later
  - GCC 6.2+ *strongly* preferred due to a bug in 6.1
- Need to compile with `-fconcepts`
- Not in Visual C++ or Clang (yet)

Compiling with Ranges

- Need to compile with `-std=c++1z`
- Need to download the prototype implementation
- Isn't compiling with experimental stuff risky?

Compiling with Ranges

- Need to compile with `-std=c++1z`
- Need to download the prototype implementation
- Isn't compiling with experimental stuff risky?
- No!

Restricting automatic type deduction

- You've surely been burned by this...

What’s the problem?

- You’ve surely been burned by this...
- What’s the problem?
  - Going off the diagnostics, I have no clue...
  - At least not for a while
  - Let’s see how Ranges fares...

Argument dependent lookup

- Cool feature that lets you skip the namespace qualification on a function
- Notice that we haven't used `std::` on any function
- We *have* qualified everything else
- Can't do this with Ranges
  - Algorithms in `std` match better with ADL ☹
  - *Always* qualify algorithms from Ranges

Problem 3

Really `requires ranges::Regular`

- Strive to make your types `Regular` or `Semiregular`
- No `typename`?
- `ranges::Regular` replaces it!
- Why?

Terse concept syntax

- Convergence of compile-time polymorphism and run-time polymorphism
- Doesn't matter if it's a template or a different type

```
// why do this
template <typename T>
requires Regular<T>
T foo(T, T);

// when you can do this?
Regular foo(Regular, Regular);
```

This code works...

- ...but it falls flat on its face (twice!)

What if I can't use Concepts or Ranges in my project

- Business is conservative!
- Several solutions (ordered best to worst):
  1. Range-v3
     - C++11 support
     - Testing ground for Ranges TS before concepts
     - Eric Niebler's CppCon talk
  2. Boost.Range
  3. Another well-supported library
  4. Roll your own with `enable_if` (last resort)

References

- *N3351 A Concept Design for the STL* (Stepanov, A et al. 2012.)
- *N4128 Ranges for the Standard Library Part 1* (Niebler, E et al. 2014.)
- *N4651 Working Draft – C++ Extensions for Ranges* (Niebler E, Carter C. 2017.)
- *CppCon 2016: Casey Carter "Iterator Haiku"* (Carter, C. 2016.)