



Using Concepts through the Ranges TS

Author

Christopher Di Bella

2017-03-29

Who am I?

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



cjdb.ns@gmail.com



<https://github.com/cjdb>



Applied Modern C++



@cjdb_ns

Quora [Christopher Di Bella](#)

Thanks

- Casey Carter¹, Eric Niebler¹ (editors for Ranges TS)
- Andrew Sutton¹ (editor for Concepts TS)
- Sergey Zubkov
- Daryl D'Souza
- Oswyn Brent, Manuel Chakravarty

¹These guys have been instrumental in providing support for Concepts and Ranges, and without their efforts, neither TS would likely exist.

Aims

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

Concepts TS

- Predicates over templates
 - Impose requirements on type deduction
 - Check *syntax* requirements
- Defined in the Concepts Technical Specification

Ranges TS

- Specifies 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 own²

²`motivation1a.cpp` **and** `motivation1b.cpp`

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.)³
 - Ewww....

³`motivation1c.cpp` **and** `motivation1d.cpp`

Ranges TS give a clear, easy solution⁴

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

⁴solution1a.cpp

What is this Sentinel⁵?

- Denotes the end of a range
- Encodes the end of a range into the type system
- Offers more generality

⁵`solution1b.cpp`

Constrasting compiler 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...⁶

⁶`motivation2.cpp`

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...⁷

⁷ `solution2.cpp`

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 3⁸

⁸`motivation3.cpp`

Really `requires` `ranges::Regular`⁹

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

⁹`solution3.cpp`

Terse concept syntax

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

```
1 // why do this
2 template <typename T>
3 requires Regular<T>
4 T foo(T, T);
5
6 // when you can do this?
7 Regular foo(Regular, Regular);
```

This code works...¹⁰

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

¹⁰`motivation4.cpp` **and** `solution4.cpp`

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

- Stepanov, A et al. *N3351 A Concept Design for the STL* 2012.
- Niebler, E et al. *N4128 Ranges for the Standard Library Part 1* 2014.
- Niebler, E and Carter, C. *N4651 Working Draft – C++ Extensions for Ranges* 2017.
- Carter, C. *CppCon 2016: Casey Carter “Iterator Haiku”* 2016.
- Stroustrup, B. *P0557R0 Concepts: The Future of Generic Programming or How to design good concepts and use them well*¹¹. 2017.
- Di Bella, C J. *Sydney C++ Meetup notes* 2017.

¹¹Someone asked about the benefits of concepts over runtime polymorphism at the meetup. This document should articulate answers that might be more convincing than mine.