

C++ Club Meeting Notes

Gleb Dolgich

2018-10-04

C++ developers for hire

- ▶ Malar Linkeshwaran, London
- ▶ Jeremy Levine, New York

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (1/13)

- ▶ Video
 - ▶ Reddit
 - ▶ Arthur O'Dwyer: Concepts as door-opening robots

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (2/13)



cppcon | 2018
THE C++ CONFERENCE • BELLEVUE, WASHINGTON

BJARNE STROUSTRUP

Concepts: The Future of Generic Programming (the future is here)

CppCon.org

Morgan Stanley

Concepts support status

- Concepts TS approved 2016
 - Available in GCC since GCC 6 (soon: Clang)
- Concepts in WP for C++20
 - Explicit **requires** clauses

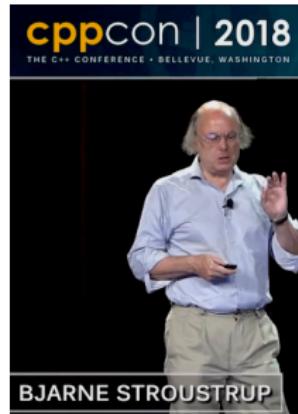
```
template<typename Iter> requires RandomAccessIterator<Iter> void sort(Iter,Iter);
```
 - Shorthand notation

```
template<RandomAccessIterator Iter> void sort(Iter,Iter);
```
 - **requires** expressions
 - Basic concepts for Ranges in standard library
- Not quite yet
 - Ranges (defined using concepts) in standard library (soon)
 - A more function declaration like syntax (compromise being worked out)
 - Concept type-name introducers (won't make C++20)

Stroustrup - Good Concepts - CppCon'18

7

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (3/13)



The image shows Bjarne Stroustrup, a man with white hair and glasses, wearing a light blue shirt and khaki pants, standing on a stage and gesturing with his hands while speaking. The background is dark.

BJARNE STROUSTRUP

Concepts: The Future of Generic Programming (the future is here)

CppCon.org

Morgan Stanley

Types and concepts

- A type
 - Specifies the set of operations that can be applied to an object
 - Implicitly and explicitly
 - Relies on function declarations and language rules
 - Specifies how an object is laid out in memory
- A concept
 - Specifies the set of operations that can be applied to an object
 - Implicitly and explicitly
 - Relies on use patterns
 - reflecting function declarations and language rules
 - Says nothing about the layout of the object

My ideal: to be able use concepts wherever we use a type, in the same way,
Except for defining layout

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (4/13)



The image shows Bjarne Stroustrup, a man with white hair and glasses, wearing a light blue shirt and khaki pants, standing on a stage and gesturing with his hands while speaking. He is positioned next to a large screen displaying the title of his talk.

cppcon | 2018
THE C++ CONFERENCE • BELLEVUE, WASHINGTON

BJARNE STROUSTRUP

**Concepts: The Future of Generic Programming
(the future is here)**

Technical issue

Morgan Stanley

- Immovable opposition to the natural/conventional syntax in WG21
 - `void sort(Sortable&);` *// deemed confusing and error prone*
 - `void sort(Sortable auto&);` *// deemed much better by some*
- I don't see it
 - I have used and taught concepts for years
 - `void sort(Sortable&&);` *// Key "anti" example: rvalue or forward reference?*
- I guess I can live with
 - `void sort(Sortable&&);` *// error*
 - `void sort(Sortable auto&&);` *// forward reference*
 - But it breaks the equivalence between types and concepts

Stroustrup - Good Concepts - CppCon'18

19

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (5/13)



The image shows Bjarne Stroustrup, a man with white hair and glasses, wearing a light blue shirt and khaki pants, standing on a stage and speaking into a microphone. He is gesturing with his hands. The background is dark.

BJARNE STROUSTRUP

Concepts: The Future of
Generic Programming
(the future is here)

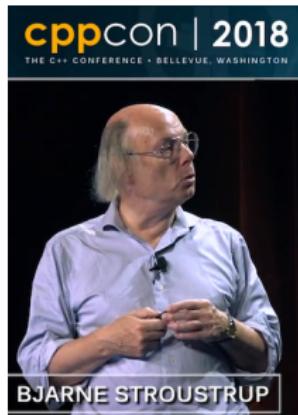
CppCon.org

Morgan Stanley

Readability

- Don't expect optimal readability from
 - Older libraries converted to use concepts
 - They often need to be “bug compatible”
 - “Advanced foundation libraries”
 - They often have to offer extreme flexibility
- Design new libraries with readability in mind

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (6/13)



THE C++ CONFERENCE • BELLEVUE, WASHINGTON

BJARNE STROUSTRUP

Concepts: The Future of Generic Programming (the future is here)

CppCon.org

Morgan Stanley

Typed vs. untyped styles

- **auto** is the weakest concept
 - An unconstrained type
- In theory we could do without **auto** as a language construct
 - `template<typename T> concept Auto = true; // Auto means auto`
- Ideal:
 - Accept a concept wherever an **auto** is
 - Actually, that's backwards: Accept an **auto** wherever a concept is
 - More historically accurate
 - The committee accepted **auto** before concepts
- Historical factoid
 - I proposed **auto f(auto);** in 2003

Stroustrup - Good Concepts - CppCon'18

31

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (7/13)



The image shows Bjarne Stroustrup, a man with white hair and glasses, wearing a light blue shirt, standing on a stage and gesturing with his hands while speaking. The background is dark.

BJARNE STROUSTRUP

Concepts: The Future of Generic Programming (the future is here)

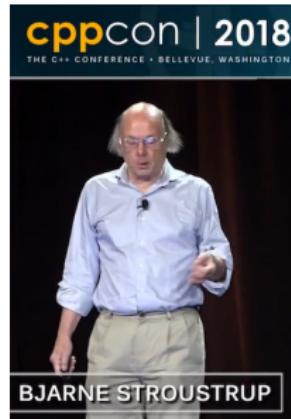
CppCon.org

Morgan Stanley

Concepts weren't born yesterday

- 1981: Alex Stepanov: “Algebraic structures”
- 1988: My attempts to find a way of constraining template arguments (failed)
- 1994: STL was specified in terms of concepts
- 2003: “Texas” design: use patterns and many alternatives
- 2003: “Indiana” design: functions signatures and initial implementation
- 2006: “Texas” + “Indiana” merger (leading to C++0x concepts and failure)
- 2011: Palo Alto meeting (leading to the Concepts TS and GCC implementation)
- 2012: “Concepts are predicates” design and implementation
- 2017: Concepts TS + GCC implementation + Ranges TS
- C++20: (most in current WP)

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (8/13)



Concepts: The Future of
Generic Programming
(the future is here)

CppCon.org

Morgan Stanley

What is a concept?

- Concepts are compile-time predicates
 - `ForwardIterator<T>`: Is `T` a forward iterator?
 - `EqualityComparable<T,U>`: Can a `T` be compared to a `U` using `==` or `!=`?
- Concepts are fundamental
 - They represent fundamental concepts of an application area
 - Concepts are come in “clusters” describing an application area
 - Monoid, group, field, and ring
 - Input, forward, bidirectional, and random access operators

Stroustrup - Good Concepts - CppCon'18



36

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (9/13)



BJARNE STROUSTRUP

Concepts: The Future of
Generic Programming
(the future is here)

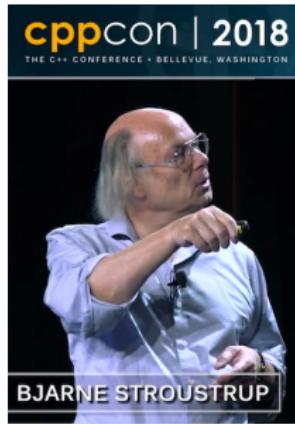
CppCon.org

Morgan Stanley

Operations come in “clusters”

- Useful concepts describe “clusters” of operations
 - E.g. algebra: The mathematicians used centuries to work out the few meaningful concepts in this area
 - Monoid, group, ring, field, vector space, ...
- An operation typically cannot be defined in isolation
 - Numbers: +, -, *, /, +=, -=, ++, --, ...
 - Iterators: ++, --, *, []
 - Stacks: `push()`, `B`
- Only rarely does a concept characterize a single operation
 - “HasPlus” is very suspect
 - is `std::string` a HasPlus?
 - Do we need a separate “HasMinus”? (no way!)
- Just like for types

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (10/13)



BJARNE STROUSTRUP

Concepts: The Future of
Generic Programming
(the future is here)

CppCon.org

Morgan Stanley

Defining concepts

- Avoid ad-hoc constraints

```
template<typename T> T sum(T& a, T& b)  
    requires requires(T a, T b) { {a+b} -> T; }; // misuse
```

```
template<typename T> concept Addable = requires {  
    { a+b } -> T;  
    a+=b; // can increment  
    a=b; // can copy  
    T{0}; // can construct from zero  
};
```

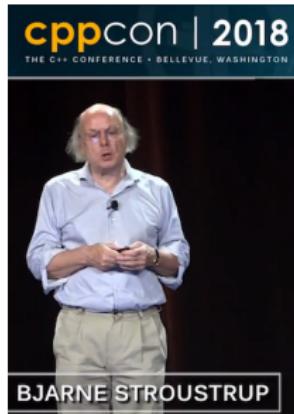
```
template<Addable T> T sum(T& a, T& b); // better
```

- “**requires requires**” is usually a design error
 - Leads to incomplete constraints

Stroustrup - Good Concepts - CppCon'18

47

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (11/13)



The image shows Bjarne Stroustrup, a man with white hair, wearing a light blue shirt and khaki pants, standing on a stage and speaking. He is gesturing with his hands. The background is dark.

BJARNE STROUSTRUP

Concepts: The Future of Generic Programming
(the future is here)

CppCon.org

Morgan Stanley

Definition checking: Why not?

- 90% of benefits come from point of use checking
 - And design improvements
- We know how to do definition checking (Gabriel Dos Reis experiments)
- Any significant change to a definition requires interface changes
 - What about debugging aids?
 - What about telemetry/logging?
- How to manage transition?
 - Can an unconstrained template call a constrained one?
 - Must: implies late checking
 - Can a constrained template call an unconstrained one?
 - Must: implies instantiation for checking
- Maybe never
 - Requires serious experimentation

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (12/13)



The C++ CONFERENCE • BELLEVUE, WASHINGTON

BJARNE STROUSTRUP

Concepts: The Future of Generic Programming (the future is here)

CppCon.org

Morgan Stanley

Use higher-level concepts

- Now we just need to define **Mergeable**:

```
template<typename T1, typename T2, typename T3>
concept Mergeable =
    ForwardIterator<For>
    && ForwardIterator<For2>
    && OutputIterator<Out>
    && Assignable<Value_type<For>, Value_type<Out>>
    && Assignable<Value_type<For2>, Value_type<Out>>
    && Comparable<Value_type<For>, Value_type<For2>>;
```

Stroustrup - Good Concepts - CppCon'18

56

CppCon 2018: Bjarne Stroustrup “Concepts: The Future of Generic Programming (the future is here)” (13/13)



The image shows Bjarne Stroustrup, a man with white hair and glasses, wearing a light blue polo shirt, speaking on stage at CppCon 2018. He is gesturing with his right hand. The background is dark.

BJARNE STROUSTRUP

Concepts: The Future of Generic Programming (the future is here)

CppCon.org

Morgan Stanley

Concrete suggestions

- User-level/application-level concepts should have semantics
- Incomplete concepts are better than no concepts
- Use named concepts
 - Never **requires requires**
- Use **static_asserts** to
 - Check (sets of) types against concepts
 - Check algorithms against (sets of) types
- Define algorithms using general types
 - For plug-and-play
 - Not absolute minimal requirements for an implementation
- Constrain variables with concepts to improve readability
 - Tighten type checking compared to auto
 - Relax type checking **compared to specific types**

Stroustrup - Good Concepts - CppCon'18

58

C++ Antipathy (1/3)

- ▶ Video: CppCon 2018: Simplicity: Not Just For Beginners, by Kate Gregory
- ▶ Hacker News
- ▶ Reddit

I am genuinely curious about the passion of anti C++ commenters - could you comment on your journey towards your current antipathy towards C++? Did you use it and get burnt? What did you use it for? What were the short comings? How long did you use it for? When did you use it (ie which C++ standard version)?

C++ Antipathy (2/3)

- ▶ “Everything is so irredeemably hard”
- ▶ “Headers and package management is a total mess”
- ▶ “Rust will take over”
- ▶ “D avoids most C++ mistakes”
- ▶ “Backward compatibility is preventing progress”
- ▶ “Too many changes are being pushed by evangelists”
- ▶ “I still don’t understand why people bother with template metaprogramming”
- ▶ “With every new version of the standard it gets more unwieldy, more difficult to use correctly”
- ▶ “I was falling back in love with C++ and then I read Scott Meyers’ book”

C++ Antipathy (3/3)

There has been something of a push towards a simpler style of C++, and the speaker (Kate Gregory) is one of the best at articulating how and why it can work.

- ▶ “Honestly, C++ isn’t *that* bad”
- ▶ “Concepts should hopefully end (or reduce) the 30 year streak of insane error messages on template substitution failures.”

What would you strip out of C++ or change if backwards compatibility wasn't a concern?

Reddit

Quote

David Wheeler:

All problems in computer science can be solved by another level of indirection, except for the problem of too many layers of indirection.