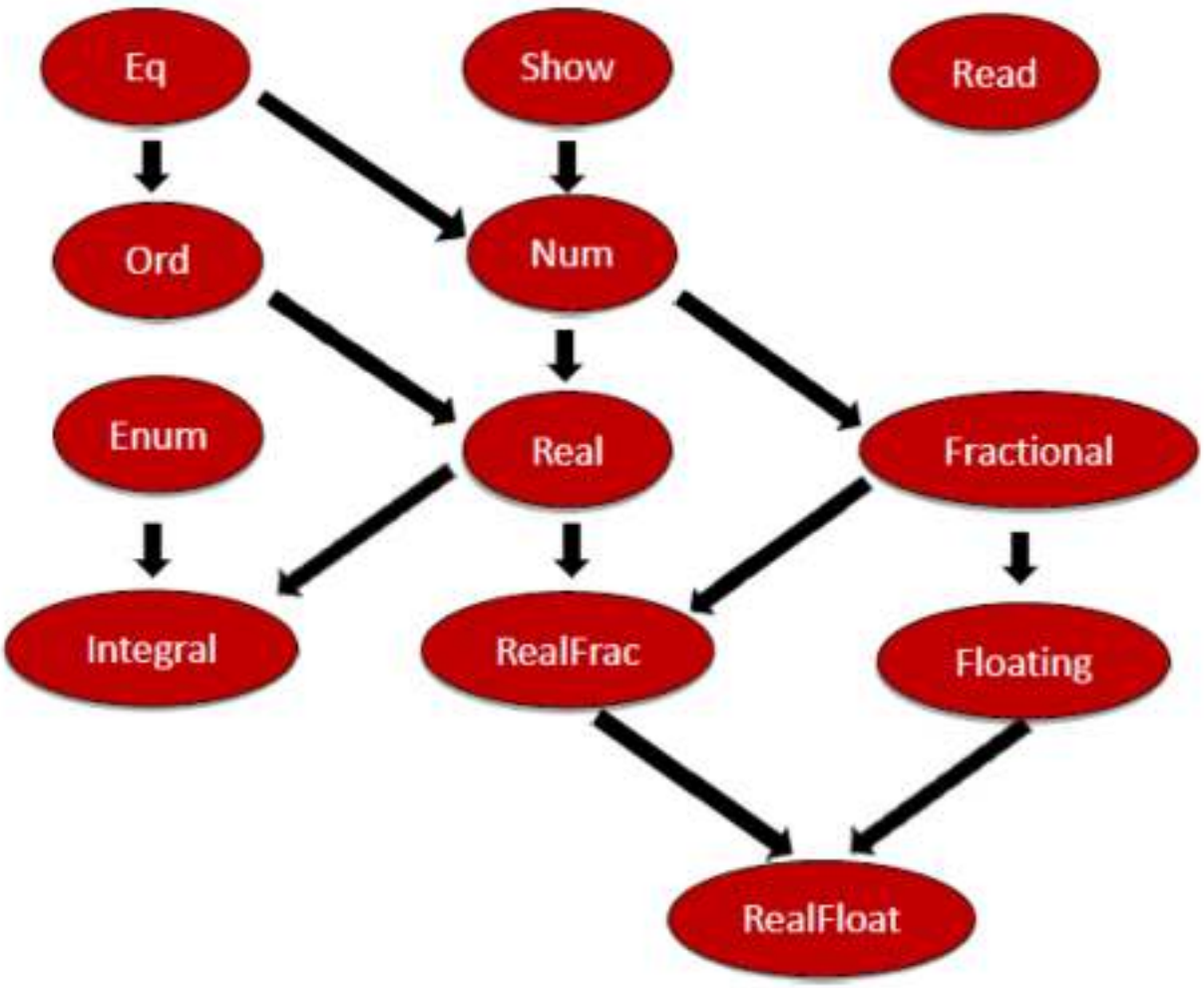
History

In this lesson, we'll learn about the history of C++ and talk about future concepts.

WE'LL COVER THE FOLLOWING

The Inspiration from Haskell

Type classes are interfaces for similar types. If a type is a member of a type class, it has to have specific properties. Type classes play a similar role in generic programming as interfaces play in object-oriented programming. Here you can see a part of Haskell's type classes hierarchy.



Haskell Type Class Hierarchy

Haskell's type classes build a hierarchy. The type class `Ord` is a subclass of the type class `Eq`. Therefore, instances of the type class `Ord` have to be members of the type class `Eq` and in addition support the comparison operators.

Classical Concepts

The key idea of generic programming with templates is, to define functions and classes that can be used with different types. But it will often happen that you instantiate a template with the wrong type. The result may be a cryptic error message that is many pages long. It is sad but true: templates in C++ are known for this. Therefore, classical concepts were planned as one of the great features of C++11. They should allow you to specify constraints for templates that can be verified by the compiler. Due to their complexity, they were removed in July 2009 from the standard: "*The C++0x concept design evolved into a monster of complexity.*" (**Bjarne Stroustrup**)

Concepts

With C++20, we will get concepts. Although concepts are in the first implementations simplified classical concepts, they have a lot to offer.

The Advantages of Concepts

Concepts

- Empower the programmer to express their requirements as part of the interface directly.
- Support the overloading of functions and the specialization of class templates based on the requirements of the template parameters.
- Produce drastically improved error messages by comparing the requirements of the template parameter with the applied template arguments.
- Can be used as placeholders for generic programming.
- Empower you to define your own concepts.

We will get the benefits without additional compile-time or runtime time costs. Concepts are similar to Haskell's type classes. Concepts describe semantic categories and not syntactic restrictions. For types of the standard library, we get library concepts such as `DefaultConstructible`, `MoveConstructible`, `CopyConstructible`, `MoveAssignable`, `CopyAssignable`, or `Destructible`. For the containers, we get concepts such as `ReversibleContainer`, `AllocatorAwareContainer`, `SequenceContainer`, `ContiguousContainer`, `AssociativeContainer`, or `UnorderedAssociativeContainer`. You can read more about concepts and their constraints here: cppreference.com.

In the next lesson, we'll learn about the details of functions and classes with reference to concepts.