

# Concepts

Rainer Grimm

Training, Coaching, and  
Technology Consulting

[www.ModernesCpp.de](http://www.ModernesCpp.de)

# Italian C++ 2019

## HOST



## PATRON



Community Crumbs

## SPONSORS

The KDAB logo, featuring a stylized 'K' icon followed by the text "KDAB" in white on a blue speech bubble background.

CONAN  
C/C++ package manager

The develer logo, featuring an orange arc above the text "develer" in blue.

ACCADEMIA  
ITALIANA  
VIDEOGIOCHI

The sigeu logo, featuring a stylized 's' icon with the text "sigeu" and "Servizi Informativi Geografici" below it.

HEXAGON

The Leica Geosystems logo, featuring the text "Leica Geosystems" in red.

# Concepts

A first Overview

The long, long History

Functions and Classes

Placeholder Syntax

Syntactic Sugar

Define your Concepts



πάντα ρεῖ



# Concepts

## A first Overview

The long, long History

Functions and Classes

Placeholder Syntax

Syntactic Sugar

Define your Concepts

# Two Extremes



## Too Specific

- Concrete functions

➡ Type conversions

- Narrowing conversion
- Integral promotion

## Too Generic

- Generic functions

➡ Ugly compile time errors

# Two Extremes

## Too Specific

```
#include <iostream>

void needInt(int i){
    std::cout << i << std::endl;
}

int main(){

    double d{1.234};
    needInt(d);

}
```

## Too Generic

```
#include <iostream>

template<typename T>
T gcd(T a, T b){
    if( b == 0 ){ return a; }
    else{
        return gcd(b, a % b);
    }
}

int main(){

    std::cout << gcd(100, 10) << std::endl;
    std::cout << gcd(3.5, 4.0) << std::endl;

}
```



# Concepts to the Rescue

## Advantages

- Express the template parameter requirements as part of the interface
- Support the overloading of functions and the specialisation of class templates
- Produce drastically improved error messages by comparing the requirements of the template parameter with the template arguments
- Use them as placeholders for generic programming
- Empower you to define your concepts
- Can be used for all kinds of templates



# Concepts

A first Overview

The long, long History

Functions and Classes

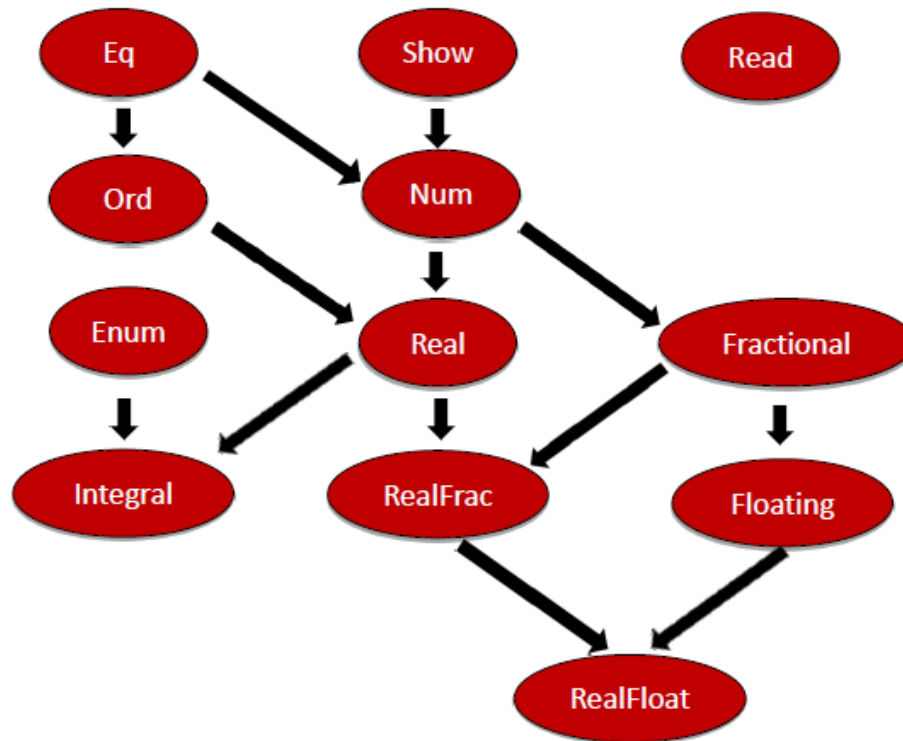
Placeholder Syntax

Syntactic Sugar

Define your Concepts

# The Inspiration

- Concepts are inspired by Haskell's typeclasses.
- Typeclasses are interfaces for similar types.



# The long Way

- 2009: removed from the C++11 standard  
"The C++0x concept design evolved into a monster of complexity."  
(Bjarne Stroustrup)
- 2017: "Concept Lite" removed from the C++17 standard
- 2020: part of the C++20 standard

# Concepts

A first Overview

The long, long History

Functions and Classes

Placeholder Syntax

Syntactic Sugar

Define your Concepts

# Functions

Using of the concept `Sortable`.

## implicit

```
template<Sortable Cont>
void sort(Cont& container){
    ...
}
```

## explicite

```
template<typename Cont>
    requires Sortable<Cont>()
void sort(Cont& container){
    ...
}
```

### ■ Usage:

```
std::list<int> lst = {1998, 2014, 2003, 2011};
sort(lst); // ERROR: lst is no random-access container with <
```

### ■ Sortable

- has to be a constant expression and predicate



# Classes

```
template<Object T>  
class MyVector{};
```

```
MyVector<int> v1; // OK
```

```
MyVector<int&> v2 // ERROR: int& does not satisfy the  
constraint Object
```

➡ A reference is not an object.

# Methods

```
template<Object T>
class MyVector{
    ...
    requires Copyable<T>()
    void push_back(const T& e);
    ...
};
```

- The type parameter  $T$  must be copyable.
- The concepts has to be placed before the method declaration.

# More Requirements

```
template <SequenceContainer S,  
          EqualityComparable<value_type<S>> T>  
Iterator_type<S> find(S&& seq, const T& val){  
    ...  
}
```

- `find` requires that the elements of the container must
  - build a sequence
  - be equality comparable


# Overloading

```
template<InputIterator I>  
void advance(I& iter, int n){...}
```

```
template<BidirectionalIterator I>  
void advance(I& iter, int n){...}
```

```
template<RandomAccessIterator I>  
void advance(I& iter, int n){...}
```

- `std::advance` puts its iterator `n` positions further
- depending on the iterator, another function template is used

```
std::list<int> lst{1,2,3,4,5,6,7,8,9};  
std::list<int>::iterator i = lst.begin();  
 std::advance(i, 2);    // BidirectionalIterator
```

# Specialisation

```
template<typename T>  
class MyVector{};
```

```
template<Object T>  
class MyVector{};
```

```
➡ MyVector<int> v1; // Object T  
   MyVector<int&> v2 // typename T
```

`MyVector<int&>` goes to the unconstrained template parameter.

`MyVector<int>` goes to the constrained template parameter.



# Concepts

A first Overview

The long, long History

Functions and Classes

Placeholder Syntax

Syntactic Sugar

Define your Concepts

# auto

## Detour: Asymmetry in C++14

```
auto genLambdaFunction = [](auto a, auto b) {  
    return a < b;  
};
```

```
template <typename T, typename T2>  
auto genFunction(T a, T2 b) {  
    return a < b;  
}
```

➡ Generic lambdas introduced a new way to define templates.

# auto

C++20 unifies this asymmetry.

- `auto`: Unconstrained placeholder
- `Concept`: Constrained placeholder

➡ Usage of a placeholder generates a function template.

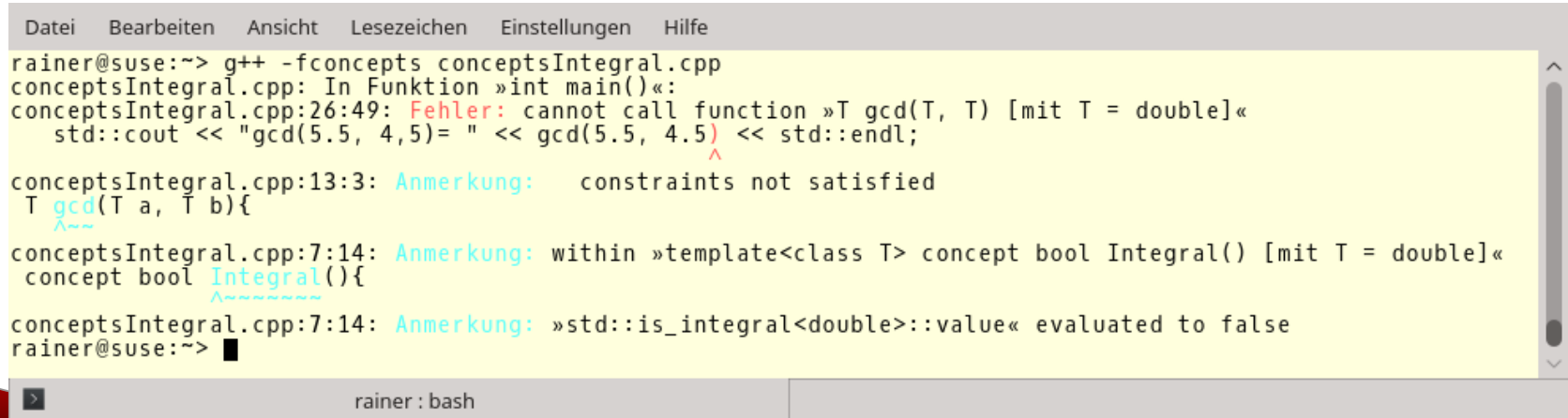
# The Concept Integral

```
#include <type_traits>
#include <iostream>

template<typename T>
concept bool Integral() {
    return std::is_integral<T>::value;
}

template<typename T>
requires Integral<T>()
T gcd(T a, T b) {
    if( b == 0 ) { return a; }
    else return gcd(b, a % b;
}

int main() {
    std::cout << "gcd(5.5, 4.5)= "
               << gcd(5.5, 4.5) << std::endl;
}
```



The screenshot shows a terminal window with the following content:

```
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
rainer@suse:~> g++ -fconcepts conceptsIntegral.cpp
conceptsIntegral.cpp: In Funktion »int main()«:
conceptsIntegral.cpp:26:49: Fehler: cannot call function »T gcd(T, T) [mit T = double]«
    std::cout << "gcd(5.5, 4.5)= " << gcd(5.5, 4.5) << std::endl;
                                   ^
conceptsIntegral.cpp:13:3: Anmerkung: constraints not satisfied
  T gcd(T a, T b){
  ^~~
conceptsIntegral.cpp:7:14: Anmerkung: within »template<class T> concept bool Integral() [mit T = double]«
  concept bool Integral(){
               ^~~~~~
conceptsIntegral.cpp:7:14: Anmerkung: »std::is_integral<double>::value« evaluated to false
rainer@suse:~> █
```

The terminal window has a title bar with buttons: Datei, Bearbeiten, Ansicht, Lesezeichen, Einstellungen, Hilfe. The status bar at the bottom shows "rainer : bash".

# Constrained and Unconstrained

Constrained concepts can be used where `auto` is usable.

```
int main(){

#include <iostream>
#include <type_traits>
#include <vector>

template<typename T>
concept bool Integral(){
    return std::is_integral<T>::value;
}

Integral auto getIntegral(int val){
    return val;
}

    std::vector<int> vec{1, 2, 3, 4, 5};
    for (Integral auto i: vec)
        std::cout << i << " ";

    Integral auto b = true;
    std::cout << b << std::endl;

    Integral auto integ = getIntegral(10);
    std::cout << integ << std::endl;

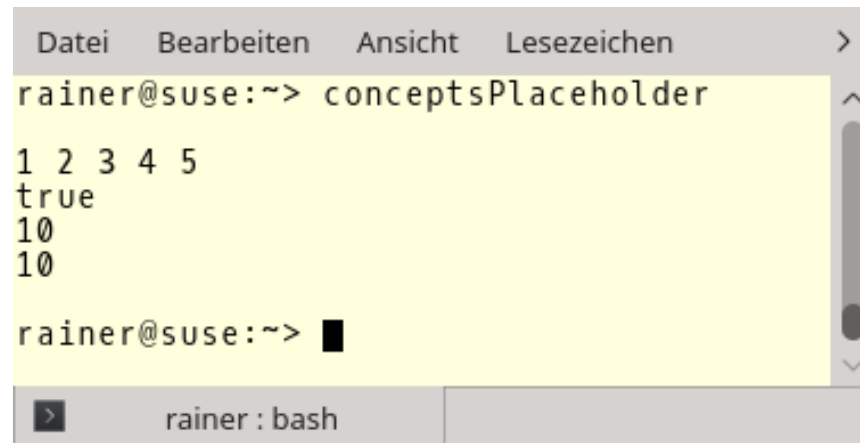
    auto integ1 = getIntegral(10);
    std::cout << integ1 << std::endl;

}
```



# Constrained and Unconstrained

Constraint and unconstrained placeholder behave as expected.



```

Datei  Bearbeiten  Ansicht  Lesezeichen  >
rainer@suse:~> conceptsPlaceholder
1 2 3 4 5
true
10
10
rainer@suse:~> █
rainer : bash
```

The image shows a terminal window with a menu bar at the top containing 'Datei', 'Bearbeiten', 'Ansicht', 'Lesezeichen', and a right arrow. The main area has a yellow background and displays the output of the 'conceptsPlaceholder' command: '1 2 3 4 5', 'true', '10', and '10'. Below this, the prompt 'rainer@suse:~>' is followed by a black cursor. At the bottom, a status bar shows a left arrow icon and the text 'rainer : bash'. A vertical scrollbar is on the right side of the terminal area.

# Concepts

A first Overview

The long, long History

Functions and Classes

Placeholder Syntax

**Syntactic Sugar**

Define your Concepts

# Syntactic Variations

## Classical

```
template<typename T>
requires Integral<T>()
T gcd(T a, T b){
    if( b == 0 ) return a;
    else return gcd(b, a % b); }
}
```

```
template<Integral T>
T gcd1(T a, T b){
    if( b == 0 ) return a;
    else return gcd(b, a % b); }
}
```

## Abbreviated Function Templates

```
Integral auto gcd2(Integral auto a,
                    Integral auto b){
    if( b == 0 ) return a;
    else return gcd(b, a % b); }
```

```
auto gcd3(auto a, auto b){
    if( b == 0 ) return a;
    else return gcd(b, a % b); }
```

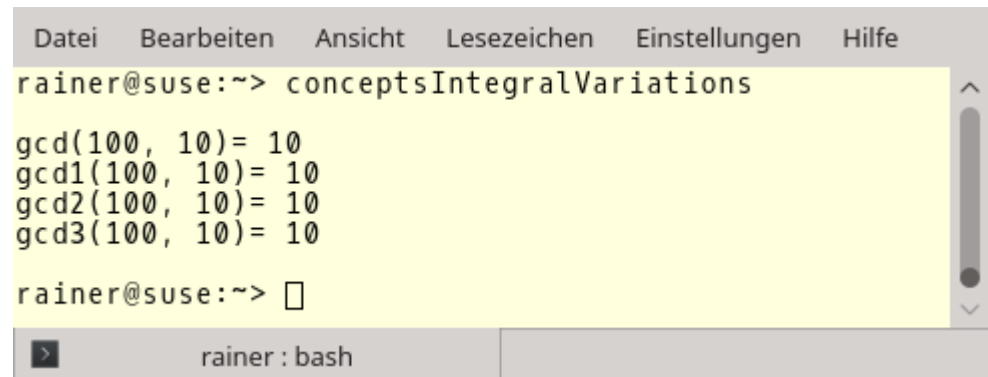
# Syntactic Variations

```
int main(){

    std::cout << std::endl;
    std::cout << "gcd(100, 10)= " << gcd(100, 10) << std::endl;
    std::cout << "gcd1(100, 10)= " << gcd1(100, 10) << std::endl;
    std::cout << "gcd2(100, 10)= " << gcd2(100, 10) << std::endl;
    std::cout << "gcd3(100, 10)= " << gcd3(100, 10) << std::endl;
    std::cout << std::endl;

}
```

Compiled with GCC 6.3 and the  
Flag `-fconcepts`

A terminal window with a menu bar (Datei, Bearbeiten, Ansicht, Lesezeichen, Einstellungen, Hilfe) and a title bar (rainer@suse:~). The terminal shows the command 'conceptsIntegralVariations' being executed, which outputs 'gcd(100, 10)= 10', 'gcd1(100, 10)= 10', 'gcd2(100, 10)= 10', and 'gcd3(100, 10)= 10'. The prompt 'rainer@suse:~>' is shown at the bottom.

```
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
rainer@suse:~> conceptsIntegralVariations
gcd(100, 10)= 10
gcd1(100, 10)= 10
gcd2(100, 10)= 10
gcd3(100, 10)= 10
rainer@suse:~> 
```

# Overloading

```
void overload(auto t){
    std::cout << "auto : " << t << std::endl;
}

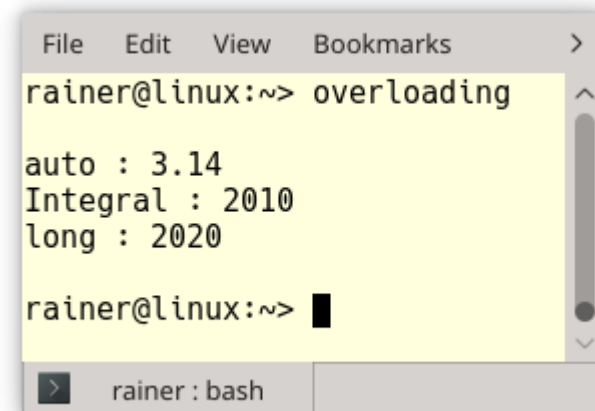
void overload(Integral auto t){
    std::cout << "Integral : " << t << std::endl;
}

void overload(long t){
    std::cout << "long : " << t << std::endl;
}
```

```
int main(){

    overload(3.14);
    overload(2010);
    overload(20201);

}
```





```
File Edit View Bookmarks >
rainer@linux:~> overloading
auto : 3.14
Integral : 2010
long : 2020
rainer@linux:~> █
> rainer : bash
```



# Template Introduction

Template introduction is a simplified syntax for declaring templates

- `template <Integral T>`  `Integral{T}`
- Syntax is only available for constrained placeholders (concepts) but not for unconstrained placeholders (`auto`)
  -  Create a constrained placeholder which always evaluates to `true`

# Template Introduction

## Constrained Placeholder

```
Integral{T}
Integral gcd(T a, T b){
    if( b == 0 )return a;
    else return gcd(b, a % b);
}
```

```
Integral{T}
class ConstrainedClass{};
```

## Unconstrained Placeholder

```
auto{T}
T gcd(T a, T b){
    if( b == 0 )return a;
    else return gcd(b, a % b);
}
```

```
auto{T}
class ConstrainedClass{};
```

# Template Introduction

```
template<typename T>
concept bool Generic(){
    return true;
}
```

```
Generic{T}
Generic gcd(T a, T b){
    if( b == 0 ) return a;
    else return gcd(b, a % b);
}
```

```
Generic{T}
class ConstrainedClass{
public:
    ConstrainedClass(){
        std::cout << typeid(decltype(std::declval<T>())) .name() << std::endl;
    }
};
```

# Template Introduction

```
int main() {  
  
    std::cout << "gcd(100, 10): " << gcd(100, 10) << std::endl;  
  
    std::cout << std::endl;  
  
    ConstrainedClass<int> genericClassInt;  
    ConstrainedClass<std::string> genericClassString;  
    ConstrainedClass<double> genericClassDouble;  
  
}
```



```
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe  
rainer@suse:~> templateIntroductionGeneric  
gcd(100, 10): 10  
  
i  
NSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEE  
d  
  
rainer@suse:~> █  
rainer : bash
```

# Concepts

A first Overview

The long, long History

Functions and Classes

Placeholder Syntax

Syntactic Sugar

Define your Concepts

# Predefined Concepts

- Core language concepts
  - Same
  - DerivedFrom
  - ConvertibleTo
  - Common
  - Integral
  - Signed Integral
  - Unsigned Integral
  - Assignable
  - Swappable
- Comparison concepts
  - Boolean
  - EqualityComparable
  - StrictTotallyOrdered
- Object concepts
  - Destructible
  - Constructible
  - DefaultConstructible
  - MoveConstructible
  - CopyConstructible
  - Movable
  - Copyable
  - Semiregular
  - Regular
- Callable concepts
  - Callable
  - RegularCallable
  - Predicate
  - Relation
  - StrictWeakOrder

# Variable Concept

```
template<typename T>
concept bool Integral =
    std::is_integral<T>::value;
}
```

- T fulfils the variable concept if  
`std::integral<T>::value` evaluates to true

# Function Concept

## Concepts TS

```
template<typename T>
concept bool Equal() {
    return requires(T a, T b) {
        { a == b } -> bool;
        { a != b } -> bool;
    };
}
```

## Draft C++20 standard

```
template<typename T>
concept Equal =
    requires(T a, T b) {
        { a == b } -> bool;
        { a != b } -> bool;
    };
};
```

- T fulfils the function concept if == and != are overloaded and return a boolean



# The Concept Equal

```
bool areEqual(Equal auto a, Equal auto b) return a == b;

/*
struct WithoutEqual{
    bool operator == (const WithoutEqual& other) = delete;
};
struct WithoutUnequal{
    bool operator != (const WithoutUnequal& other) = delete;
};
*/
. . .
std::cout << "areEqual(1, 5): " << areEqual(1, 5) << std::endl;

/*
bool res = areEqual(WithoutEqual(), WithoutEqual());
bool res2 = areEqual(WithoutUnequal(), WithoutUnequal());
*/
```

# The Concept Equal

```
File Edit View Bookmarks Settings Help
rainer@suse:~> conceptsDefinitionEqual
areEqual(1, 5): false
rainer@suse:~> █
rainer : bash
```

```
File Edit View Bookmarks Settings Help
rainer@suse:~> g++ -fconcepts conceptsDefinitionEqual.cpp -o conceptsDefinitionEqual
conceptsDefinitionEqual.cpp: In function 'int main()':
conceptsDefinitionEqual.cpp:37:54: error: cannot call function 'bool areEqual(auto:1, auto:1) [with auto:1 = WithoutEqual]'
    bool res = areEqual(WithoutEqual(), WithoutEqual());
                                   ^
conceptsDefinitionEqual.cpp:13:6: note: constraints not satisfied
    bool areEqual(Equal a, Equal b){
    ^~~~~~
conceptsDefinitionEqual.cpp:6:14: note: within 'template<class T> concept bool Equal() [with T = WithoutEqual]'
    concept bool Equal(){
    ^~~~~~
conceptsDefinitionEqual.cpp:6:14: note: with 'WithoutEqual a'
conceptsDefinitionEqual.cpp:6:14: note: with 'WithoutEqual b'
conceptsDefinitionEqual.cpp:6:14: note: the required expression '(a == b)' would be ill-formed
conceptsDefinitionEqual.cpp:6:14: note: 'b->a.WithoutEqual::operator==()' is not implicitly convertible to 'bool'
conceptsDefinitionEqual.cpp:6:14: note: the required expression '(a != b)' would be ill-formed
conceptsDefinitionEqual.cpp:39:59: error: cannot call function 'bool areEqual(auto:1, auto:1) [with auto:1 = WithoutUnequal]'
    bool res2 = areEqual(WithoutUnequal(), WithoutUnequal());
                                   ^
conceptsDefinitionEqual.cpp:13:6: note: constraints not satisfied
    bool areEqual(Equal a, Equal b){
    ^~~~~~
conceptsDefinitionEqual.cpp:6:14: note: within 'template<class T> concept bool Equal() [with T = WithoutUnequal]'
    concept bool Equal(){
    ^~~~~~
conceptsDefinitionEqual.cpp:6:14: note: with 'WithoutUnequal a'
conceptsDefinitionEqual.cpp:6:14: note: with 'WithoutUnequal b'
conceptsDefinitionEqual.cpp:6:14: note: the required expression '(a == b)' would be ill-formed
conceptsDefinitionEqual.cpp:6:14: note: the required expression '(a != b)' would be ill-formed
conceptsDefinitionEqual.cpp:6:14: note: 'b->a.WithoutUnequal::operator!=()' is not implicitly convertible to 'bool'
rainer@suse:~> █
rainer : bash
```

# Eq versus Equal

## The Typeclass Eq

```
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
```

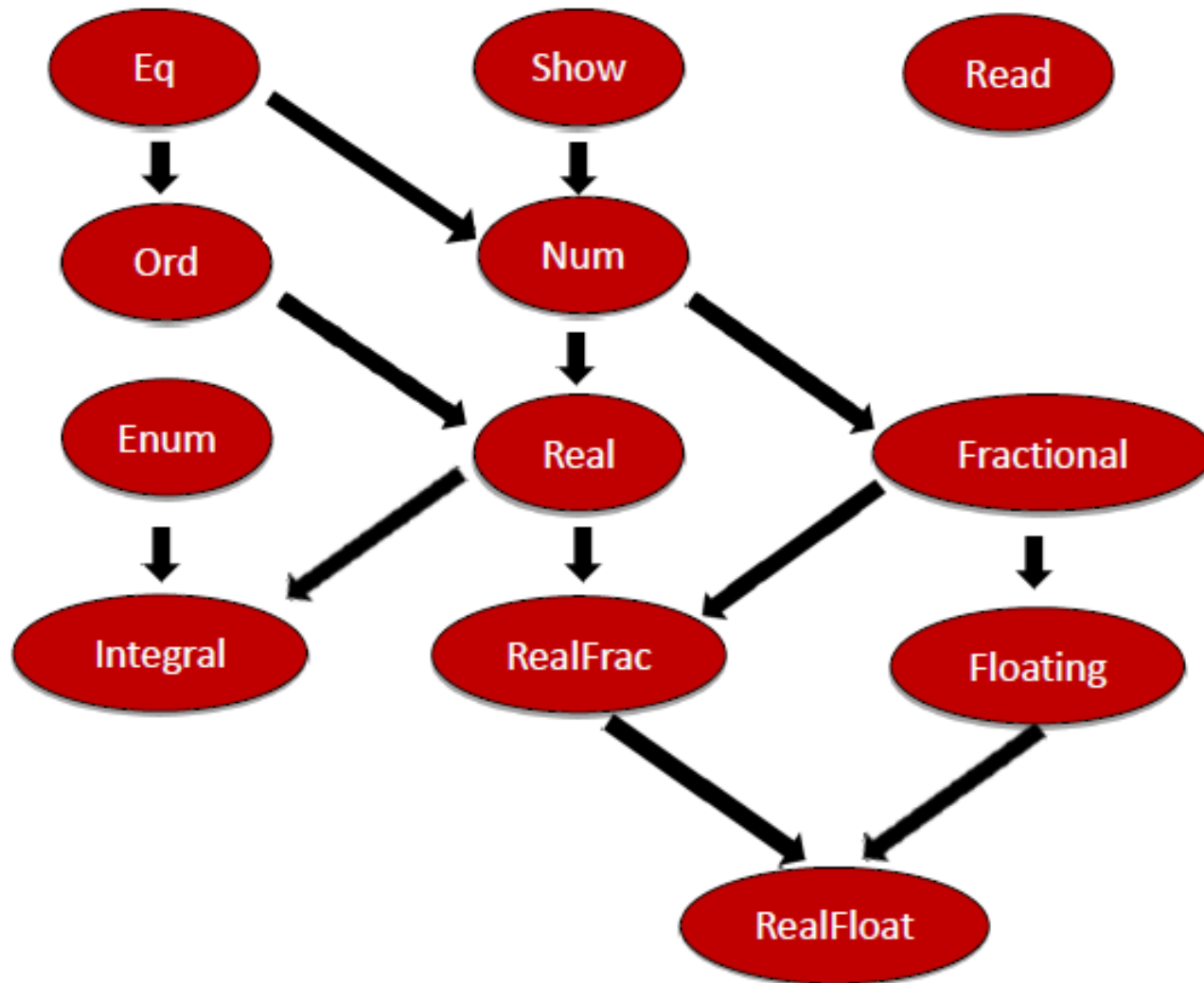
## The Concept Equal

```
template <typename T>
concept bool Equal() {
    return requires(T a, T b) {
        { a == b } -> bool;
        { a != b } -> bool;
    };
}
```

The typeclass `Eq` (Haskell) and the concept `Equal` (C++) require for the concrete types

- they have to support equal and the unequal operations
- the operations have to return a boolean
- both types have to be the same

# Haskells Typeclasses



# Haskells Typeclass Ord

```
class Eq a => Ord a where
  compare :: a -> a -> Ordering
  (<)    :: a -> a -> Bool
  (<=)   :: a -> a -> Bool
  (>)    :: a -> a -> Bool
  (>=)   :: a -> a -> Bool
  max    :: a -> a -> a
```

➡ Each type supporting `Ord` must support `Eq`.

# The Concept Ord

## The concept Equal

```
template<typename T>
concept bool Equal() {
    return requires(T a, T b) {
        { a == b } -> bool;
        { a != b } -> bool;
    };
}
```

## The concept Ord

```
template <typename T>
concept bool Ord() {
    return requires(T a, T b) {
        requires Equal<T>();
        { a <= b } -> bool;
        { a < b } -> bool;
        { a > b } -> bool;
        { a >= b } -> bool;
    };
}
```

# The Concept Ord

```
bool areEqual(Equal auto a,
              Equal auto b){
    return a == b;
}

Ord getSmaller(Ord auto a,
              Ord auto b){
    return (a < b) ? a : b;
}

int main(){

    std::cout << areEqual(1, 5);

    std::cout << getSmaller(1, 5);

    std::unordered_set<int> firSet{1, 2, 3, 4, 5};
    std::unordered_set<int> secSet{5, 4, 3, 2, 1};

    std::cout << areEqual(firSet, secSet);

    // auto smallerSet = getSmaller(firSet, secSet);
}
```

# The Concept Ord

```
File Edit View Bookmarks Settings Help
rainer@suse:~> conceptsDefinitionOrd

areEqual(1, 5): false
getSmaller(1, 5): 1
areEqual(firSet, secSet): true

rainer@suse:~> █
```

rainer: bash

```
File Edit View Bookmarks Settings Help
rainer@suse:~> g++ -fconcepts conceptsDefinitionOrd.cpp -o conceptsDefinitionOrd
conceptsDefinitionOrd.cpp: In function 'int main()':
conceptsDefinitionOrd.cpp:44:45: error: cannot call function 'auto getSmaller(auto:2, auto:2)
[with auto:2 = std::unordered_set<int>]'
    auto smallerSet= getSmaller(firSet, secSet);
                           ^
conceptsDefinitionOrd.cpp:27:5: note: constraints not satisfied
    Ord getSmaller(Ord a, Ord b){
    ^~~~~~
conceptsDefinitionOrd.cpp:13:14: note: within 'template<class T> concept bool Ord() [with T =
std::unordered_set<int>]'
    concept bool Ord(){
    ^~~
conceptsDefinitionOrd.cpp:13:14: note: with 'std::unordered_set<int> a'
conceptsDefinitionOrd.cpp:13:14: note: with 'std::unordered_set<int> b'
conceptsDefinitionOrd.cpp:13:14: note: the required expression '(a <= b)' would be ill-formed
conceptsDefinitionOrd.cpp:13:14: note: the required expression '(a < b)' would be ill-formed
conceptsDefinitionOrd.cpp:13:14: note: the required expression '(a > b)' would be ill-formed
conceptsDefinitionOrd.cpp:13:14: note: the required expression '(a >= b)' would be ill-formed
rainer@suse:~> █
```

rainer: bash



# Regular and SemiRegular

## Regular

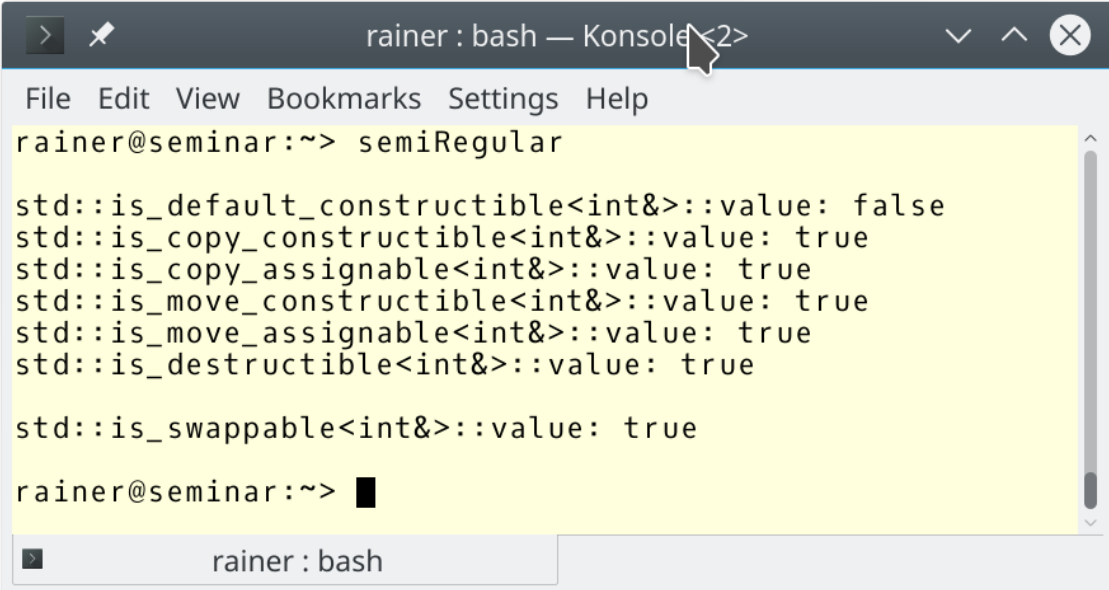
- `DefaultConstructible`
- `CopyConstructible`,  
`CopyAssignable`
- `MoveConstructible`,  
`MoveAssignable`
- `Destructible`
- `Swappable`
- `EqualityComparable`

## SemiRegular

- **Regular** –  
`EqualityComparable`

# Regular **and** SemiRegular

```
std::cout << std::is_default_constructible<int&>::value;  
std::cout << std::is_copy_constructible<int&>::value;  
std::cout << std::is_copy_assignable<int&>::value;  
std::cout << std::is_move_constructible<int&>::value;  
std::cout << std::is_move_assignable<int&>::value;  
std::cout << std::is_destructible<int&>::value;  
std::cout << std::is_swappable<int&>::value;
```



A screenshot of a terminal window titled "rainer : bash — Konsole". The window shows the execution of a program named "semiRegular". The output of the program is displayed on a yellow background, showing the results of the C++ type traits for `int&`. The results are: `std::is_default_constructible<int&>::value: false`, `std::is_copy_constructible<int&>::value: true`, `std::is_copy_assignable<int&>::value: true`, `std::is_move_constructible<int&>::value: true`, `std::is_move_assignable<int&>::value: true`, `std::is_destructible<int&>::value: true`, and `std::is_swappable<int&>::value: true`. The prompt "rainer@seminar:~>" is visible at the bottom of the terminal.

```
rainer : bash — Konsole  
File Edit View Bookmarks Settings Help  
rainer@seminar:~> semiRegular  
  
std::is_default_constructible<int&>::value: false  
std::is_copy_constructible<int&>::value: true  
std::is_copy_assignable<int&>::value: true  
std::is_move_constructible<int&>::value: true  
std::is_move_assignable<int&>::value: true  
std::is_destructible<int&>::value: true  
  
std::is_swappable<int&>::value: true  
  
rainer@seminar:~> █  
rainer : bash
```

# Regular **and** SemiRegular

The type-trait `isEqualityComparable`:

```
template<typename T>
using equal_comparable_t = decltype(std::declval<T&>() ==
                                   std::declval<T&>()));
```

```
template<typename T>
struct isEqualityComparable:
    std::experimental::is_detected<equal_comparable_t, T> {};
```

# Regular **and** SemiRegular

## The type-traits Regular **and** SemiRegular

```
template<typename T>
struct isSemiRegular: std::integral_constant<bool,
    std::is_default_constructible<T>::value &&
    std::is_copy_constructible<T>::value &&
    std::is_copy_assignable<T>::value &&
    std::is_move_constructible<T>::value &&
    std::is_move_assignable<T>::value &&
    std::is_destructible<T>::value &&
    std::is_swappable<T>::value >{};
```

```
template<typename T>
struct isRegular: std::integral_constant<bool,
    isSemiRegular<T>::value &&
    isEqualityComparable<T>::value >{};
```

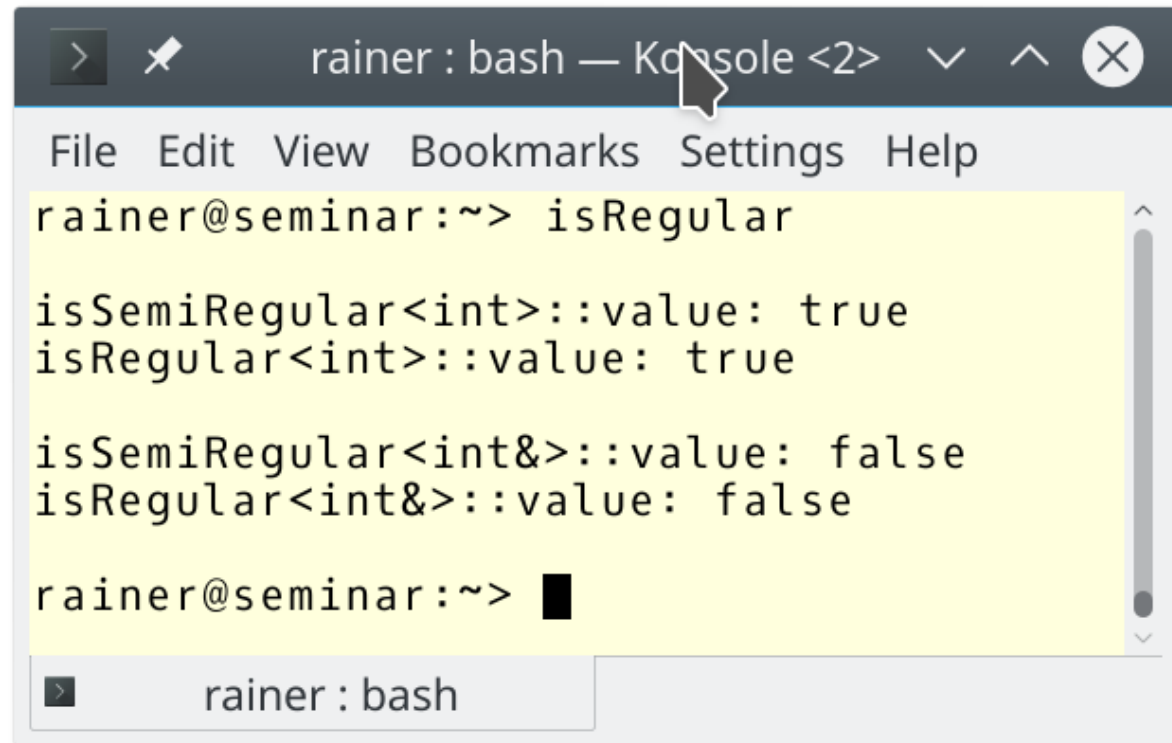
# Regular **and** SemiRegular

```
std::cout << isSemiRegular<int>::value;
```

```
std::cout << isRegular<int>::value;
```

```
std::cout << isSemiRegular<int&>::value;
```

```
std::cout << isRegular<int&>::value;
```



The screenshot shows a terminal window titled "rainer : bash — Konsole <2>". The window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal content is as follows:

```
rainer@seminar:~> isRegular  
  
isSemiRegular<int>::value: true  
isRegular<int>::value: true  
  
isSemiRegular<int&>::value: false  
isRegular<int&>::value: false  
  
rainer@seminar:~> █
```

The terminal window has a tab at the bottom labeled "rainer : bash".

# Regular **and** SemiRegular

```
template<typename T>
concept bool Regular =
    std::isRegular<T>::value;
}
```

```
template<typename T>
concept bool SemiRegular =
    std::isSemiRegular<T>::value;
}
```

# Concepts

A first Overview

The long, long History

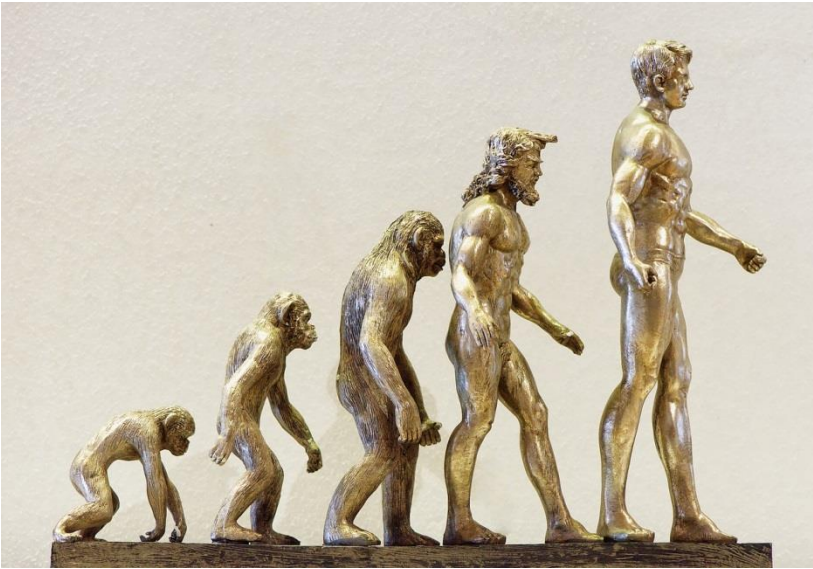
Functions and Classes

Placeholder Syntax

Syntactic Sugar

Define your Concepts

# Evolution or Revolution in C++?





# Evolution or Revolution in C++

## Evolution

- `auto` as unconstrained placeholders
- Generic lambdas as new way to define templates

```
auto add = [](auto a, auto b) {  
    return a + b;  
}
```

## Revolution

- Template requirements are verified by the compiler
- Declaration and definition of templates radically improved
- **Concepts define semantic categories and not syntactic requirements**

# Blogs

[www.grimm-jaud.de](http://www.grimm-jaud.de) [De]

[www.ModernesCpp.com](http://www.ModernesCpp.com) [En]

Rainer Grimm

Training, Coaching, and  
Technology Consulting

[www.ModernesCpp.de](http://www.ModernesCpp.de)