Part 5 - Classes

Tyler Swann

Agenda

- Classes
- Templates
- Generics
- Concepts
- Discussion
- Presentation of Assignment 1

Classes

Classes

- Classes are a way for users to create custom types for our programs.
- Classes allow us to customize the behaviour of type from its construction and destruction, to the available operations that can be performed on a class.
- Classes and structures are identical in C++ except the members of a class are private by default (inaccessible).

```cpp
#include <iostream>

class Point
{
    int x;
    int y;
};

auto main() -> int
{
    auto p = Point{ 2, 5 };
    /// Fails as `x` and `y` are private
    std::cout << "( " << p.x << ", " << p.y << " )" << std::endl;

    return 0;
}
```

Access Specifiers

Accessor Category	Meaning
`private`	Member can only be used by member functions of the same class or friends (functions or classes).
`protected`	Member can only be used by member functions of the same class, friends (functions or classes) or derived classes.
`public`	Can be accessed by anyone.

Access Specifiers Example

```cpp
#include <iostream>

class Point
{
public:                 ///< Declare members `x` and `y` as public
    int x;
    int y;
};

auto main() -> int
{
    Point p{ 2, 5 };  ///< Now succeeds

    std::cout << "( " << p.x << ", " << p.y << " )" << std::endl;  ///< Now succeeds

    return 0;
}
```

Derived Access to Members

Base Classes Access Policy	*`private`*	*`protected`*	*`public`*
	Always inaccessible with any derivation access	private in derived class if you use private derivation	private in derived class if you use private derivation
		protected in derived class if you use protected derivation	protected in derived class if you use protected derivation
		protected in derived class if you use public derivation	public in derived class if you use public derivation

Derived Access Specifiers Example

```cpp
#include <iostream>

class Point
{
public:
    int x;
    int y;
};

class Point3D
    : protected Point       ///< Points members are `protected`
{
public:
    int z;
};

auto main() -> int
{
    Point p{ 2, 5 };
    Point3D p3d{};

    std::cout << "( " << p.x << ", " << p.y << " )" << std::endl;
    std::cout << "p3d.z = " << p3d.z << std::endl;
    std::cout << "( " << p3d.x << ", " << p3d.y << ", " << p3d.z << " )" << std::endl;  ///< Fails, `x` and `y` are inaccessible

    return 0;
}
```

Constructors and Destructors

- Constructors give us a way to define how we want our types to be created from resource acquisition to member initialisation.
- Constructors can be of many forms from explicit ones that directly initialise members to ones that define the copying and ownership transfer functionality.
- Default constructors take no arguments and are used as the base initialise form of a class
- Constructors allow us to ensure type safety through RAII (resource acquisition is initialisation), ensuring that that a type only gets initialised if it can successfully obtain all its resources at the point of construction.
- Types with open() / close(), lock() / unlock(), or init() / copyFrom() / destroy() methods are typically non-RAII and are not fully type safe.
- Destructors allow us to customize the behaviour of how our types release resources and will never throw exceptions ensuring any acquired resources are returned even in the event of failed construction.
- Constructors and destructors can be implemented by the compiler using the `= default` pattern instead of a function body and constructors can be deleted using the `= delete` pattern (again, instead of a function body).

```
class A
{
    /// Default Constructor
    A() {};

    /// Copy Constructor
    A(const A& a) {};

    /// Move Constructor
    A(A&& a) {};

    /// Explicit Constructor
    A(...) {};

    /// Destructor
    ~A() {};
};
```

Point Example – Constructors and Destructors

```
class Point
{
public:

    /// Default Constructor
    constexpr Point() noexcept = default;

    /// Explicit Constructor for initialising `x` and `y`
    explicit constexpr Point(int x, int y) noexcept
        : x{ x }, y{ y } { }

    /// Copy Constructor
    constexpr Point(const Point& p) noexcept
        : x{ p.x }, y{ p.y } { }

    /// Move Constructor
    constexpr Point(Point&& p) noexcept
        : x{ std::move(p.x) }, y{ std::move(p.y) }
    {
        p.x = int();
        p.y = int();
    }

    /// ...
```

```
/// ...

constexpr auto operator= (const Point& p) noexcept -> Point&
{
    if (p != *this)
    {
        x = p.x;
        y = p.y;
    }
    return *this;
}

constexpr auto operator= (Point&& p) noexcept -> Point&
{
    if (p != *this)
    {
        x = std::move(p.x);
        y = std::move(p.y);
     }
    return *this;
}

constexpr ~Point() noexcept = default;

/// ... Implementation details

};
```

Copies and Moves

- Certain constructors create different semantics based on their arguments. These are called copy and move semantics.
- A class imposes copy semantics when a constant reference can be taken from it allowing another instance of a class to access the underlying data without modifying the existing one.
- A class imposes move semantics when a rvalue reference can be taken from it ie. it is a temporary value and ownership of the data can be taken.
- Moves don't actually move any data, instead the term 'move' refers to moving or transfer of ownership. After a move, that data will remain in the same memory space but now has a new owner.
- The old owner can still be used to copy of take ownership of new data
- Moves occur when a value is newly constructed and assigned to a variable or when using `std::move()` to manufacture a rvalue reference.
- Copy and move assignments are achieved using the same semantics (parameter types) on an overloaded `=` operator.

```cpp
auto main() -> int
{
    /// Default Constructor
    A a1();

    /// Copy constructor, copies `a1` data to `a2`
    A a2(a1);

    /// Move constructor, transfers ownership of `a1`
resources to `a3`
    A a3(std::move(a1));

    /// Move assignment, transfers temp. A() to `a1`
    auto a4 = A();

    /// Copy assignment, copies `a4` data to `a5`
    auto a5 = a4;

    /// Move, transfers `a4` data to `a6`
    auto a6 = std::move(a4);

    return 0;
}
```

Members and Methods

- You can define variables and functions with classes just like normal scopes. These members will be bound to an instance of the class and a classes methods (functions) are able to access all parts of the class even if it is `private` or `protected`. This allows for us to make stateful changes to a class without manually interacting with the underlying data.
- The normal rules of functions such as overloading and `noexcept` optimization apply for methods. We can even overload operators for classes to extend their functionality.
- Methods can be postfixed with a `const` , `&` or `&&` specifiers indicating that they can only be used if the class object is constant, a lvalue or a rvalue.
- All methods are implicitly given a pointer called `this` which is the pointer to the current instance of a class in memory. This can be accessed as if it was a regular pointer to a class.
- Methods and other types can be declared as a `friend`, allowing access to even `private` or `protected` members of the class without needed to expose the classes internals. Friendship is not transitive nor inherited. A friend of a friend is not your friend and your friends' children are not your friends.

Point Example – Members and Methods

```cpp
class Point
{
    /// ... Constructor & Destructor Implementations

    constexpr auto operator+ (const Point& p) noexcept -> Point
    { return Point{ x + p.x, y + p.y }; }

    constexpr auto operator- (const Point& p) noexcept -> Point
    { return Point{ x - p.x, y - p.y }; }

    constexpr auto operator== (const Point& p) noexcept -> bool
    { return (x == p.x) && (y == p.y); }

    constexpr auto operator!= (const Point& p) noexcept -> bool
    { return !(*this == p); }

    friend auto operator<< (std::ostream& os, const Point& p) noexcept -> std::ostream&
    {
        os << "( " << p.x << ", " << p.y << " )";
        return os;
    }

private:
    int x;
    int y;

};
```

Dynamic Inheritance and OOP

- C++ has a large language level support for Object Oriented Programming using classes.
- Classes can be inherited from allowing for the creation of hierarchical relationships between types.
- Virtual functions can be used in order to create overridable function implementations using inheritance between base and derived classes. Abstract classes can be created using pure virtual functions (methods with no implementation) that derived classes must override. C++ also has full support for runtime polymorphism and dynamic dispatch through vtables.
- Access to a base class's members can be controlled using access specifiers at the point of inheritance.
- Base classes can be used at function parameters allow for classes derived from the base class to be operated on from those functions.
- OOP is not the focus of the C++ series for a couple reasons; (i) OOP is a complex enough concept that is best taught independently, (ii) C++ supports OOP but does not rely on it except in some of the oldest parts of the standard library and this rarely impacts the writing of any C++ code unless you seek it out, the C++ standard since day one (1998) has move more and more away from using it as better features achieve the same result in 95% of use cases and finally I don’t like OOP as it is currently used and taught for my own reasons ☺. OOP is still important to know about for its most basic use cases.

OOP Example

```cpp
#include <iostream>

struct A
{
    int n;
};

struct B : public A
{
    float f;
};

auto main() -> int
{
    A a();
    std::cout << a.n << std::endl;  ///< 0
    a.n = 7;
    std::cout << a.n << std::endl;  ///< 7

    B b();
    std::cout << b.n << std::endl;  ///< 0
    std::cout << b.f << std::endl;  ///< 0.0
    b.n = 4;
    std::cout << b.n << std::endl;  ///< 4
    std::cout << a.n << std::endl;  ///< 7

    b.f = 8.53464f;
    std::cout << b.f << std::endl;  ///< 8.53464

    return 0;
}
```

Templates

Templates

Templates allow for creating blueprints for a function, class or variable that is instantiated with parameters at compile time.

Template parameters can be either type parameters using the `typename` keyword as a type introducer or value parameter that has a fixed type or a template type.

Templates are far more flexible and customizable compared to object-oriented type polymorphism as templates are only instantiated when the template parameters are satisfied.

Much of the C++ standard library uses templates.

```
template<typename T, std::size_t N>
/// ... template entity details
```

Templates Functions

- Template functions are identical to regular functions except they feature a template declaration before the function signature.
- Template type parameters can be used as any objects type throughout the scope of the function including in the function's parameters.
- The templated function is called with two sets of parenthesis, the first is the template arguments which are passed in angle brackets (<>) and the second are the regular function parameters passed with parenthesis (( )).
- Since C++17, if the template parameters are directly related to the type of value of the parameters of the function, the template parameters can be deduced by the compiler using argument deduction.

```cpp
#include <array>
#include <iostream>
#include <string>

template<typename T, std::size_t N>
auto print(const std::array<T, N>& arr) -> void
{
    std::cout << "[ ";

    for (auto n { N }; const auto& e : arr)
        if (--n > 0)
            std::cout << e << ", ";
        else
            std::cout << e;

    std::cout << " ]" << std::endl;
}

auto main() -> int
{
    auto a1 = std::to_array<int>({ 1, 2, 3, 4, 5 });

    auto a2 = std::to_array<double>({ 1.576, 0.0002, 3756348.34646, 5e-14,
465.7657, 358.0, 237437.456756 });

    auto a3 = std::to_array<std::string>({ "John", "Anna", "Grace", "Bob" });

    print<int, 5>(a1);  ///< Explicit template argument instantiation
    print(a2);          ///< template argument deduction (same below)
    print(a3);

    return 0;
}
```

Templates Classes

- Classes can also be templated to create generic types.
- This allows for classes to be instantiated with a particular types or values at compile time by the end user.
- Most of the standard library is made of templated classes.
- Every instantiation of a template class is fundamentally a different class type.

```cpp
#include <iostream>
#include <utility>

template<typename T>
class Point
{
public:

    explicit constexpr Point(T x, T y) noexcept
        : x{ x }, y{ y } { }

/// ... Implementation details

private:
    T x;
    T y;
};

auto main() -> int
{
    auto p1 = Point<int>{ 2, 5 };
    auto p2 = Point{ 6, 7 };
    auto p3 = p1 + p2;
    auto p4 = Point<double>{ 5.6 , -0.007 };
    auto p5 = Point{ 4.576 , 24.012 };
    auto p6 = p4 - p5;

    std::cout << p1 << std::endl;
    std::cout << p2 << std::endl;
    std::cout << p3 << std::endl;
    std::cout << p4 << std::endl;
    std::cout << p5 << std::endl;
    std::cout << p6 << std::endl;

    return 0;
}
```

Templated Methods

- Classes; regular or template, can have templated methods.
- Template members are declared like templated functions, and the same rules of methods apply such as the implicit *this* pointer.
- Template methods also allow for template classes of different instantiations to interact with each other allowing for type conversion constructors and mixed type operations. This does require the other instantiations of the template class to be *friend*.

```cpp
template<typename T>
class Point
{
public:
/// ... Point details

    template<typename U>
    constexpr auto operator+ (const Point<U>& p)
        noexcept -> Point<typename std::common_type<T, U>::type>
    { return Point<typename std:: common_type<T, U>::type>{ x + p.x, y + p.y }; }

    template<typename U>
    constexpr auto operator- (const Point<U>& p)
        noexcept -> Point<typename std:: common_type<T, U>::type>
    { return Point<typename std:: common_type<T, U>::type>{ x - p.x, y - p.y }; }

    template<typename U>
    constexpr auto operator== (const Point<U>& p) noexcept -> bool
    { return (x == p.x) && (y == p.y); }

    template<typename U>
    constexpr auto operator!= (const Point<U>& p) noexcept -> bool
    { return !(*this == p); }

/// ... Point details

private:

    T x;
    T y;

    template<typename U>
    friend class Point;
};
```

Templated Variables and Template Metaprogramming

- We can also create compile time templated variables using *constexpr*.
- These are useful for creating numeric values of constants that could use a variety of underlying base types based on the need.
- Metaprogramming is the process of using compile time programming to generate code.
- Template metaprogramming allows for type introspection and specialization that can be used to generate different code for different types.

```cpp
#include <iomanip>
#include <iostream>
#include <numeric>

template<typename T>
constexpr T e = T(2.7182818284590452353602874713527);

auto main() -> int
{
    std::cout << std::setprecision(std::numeric_limits<long double>::max_digits10);
    std::cout << "True e            = 2.7182818284590452353602874713527" << std::endl;
    std::cout << "e<long double>    = " << e<long double> << std::endl;
    std::cout << "e<double>         = " << e<double> << std::endl;
    std::cout << "e<float>          = " << e<float> << std::endl;
    std::cout << "e<int>            = " << e<int> << std::endl;
    std::cout << "e<char>           = " << e<char> << std::endl;

    return 0;
}
```

```
True e            = 2.7182818284590452353602874713527
e<long double>    = 2.71828182845904508
e<double>         = 2.71828182845904508
e<float>          = 2.71828174591064453125
e<int>            = 2
e<char>           = 
```

Generics

Generic Programming and Parameter Packs

- Generic programming is a style of programming that allows for the definition of types that are to be specified later. In C++, templates are used for generic programming.
- Parameter packs are used to hold an arbitrary number of either type or value template parameters.
- Type parameter packs can have different types.
- To get the size of a parameter pack we use the `sizeof...` operator.
- Parameter packs can allow for compile time computation using fold expressions which apply functions and operators to element in a parameter pack.

```cpp
#include <iostream>

auto tprintf(const char* format) -> void  ///< Base function
{ std::cout << format; }

template<typename T, typename... Args>
auto tprintf(const char* format, T value, Args&&... args) -> void  ///< recursive variadic
function
{
    for (; *format != '\0'; format++)
    {
        if (*format == '%')
        {
            std::cout << value;
            tprintf(format + 1, std::forward<Args>(args)...);  ///< recursive call,
`std::forward` is called on the expanded pack
            return;
        }
        std::cout << *format;
    }
}

auto main() -> int
{
    tprintf("% world% % - %\n", "Hello", '!', 123, -0.3575);

    return 0;
}
```

Fold Expressions Example

```cpp
#include <iostream>

template<typename... Args>
auto sum(Args&&... args) -> std::common_type<Args...>::type
{ return (... + args); }

template<typename T, typename... Args>
auto product(T init, Args&&... args) -> std::common_type<T, Args...>::type
{ return (init * ... * args); }

auto main() -> int
{
    std::cout << sum(2, 0.6, 0.5313f) << std::endl;
    std::cout << product(0, 6, 1.356, 90.5313f) << std::endl;
    std::cout << product(6, 1.356, 90.5313f) << std::endl;

    return 0;
}
```

Tuples and Pairs

TUPLE

- Parameter packs are used to build heterogeneous container types such as `std::tuple`.
- To obtain the element of a tuple you can index by type of by number using `std::get<>()`.
- Tuples allow for efficient packing of different typed data into a single structure with a common interface.
- `std::tuple` come with a host of helper functions for introspecting the types and size of a tuple.

PAIR

- `std::pair` is a specialized case of std::tuple for two heterogenous types.
- Its member variables can be accessed directly and are named first and second respectively.

Tuple Example

```cpp
#include <iostream>
#include <tuple>
#include <utility>

template<typename Tup, std::size_t... I>
auto print(const Tup& t, std::index_sequence<I...>) -> void
{
    std::cout << "( ";

    (..., (std::cout << (I == 0 ? "" : ", ") << std::get<I>(t)) );

    std::cout << " )" << std::endl;
}


template<typename... Ts, std::size_t... I>
auto print(const std::tuple<Ts...>& t) -> void
{ print(t, std::make_index_sequence<sizeof...(Ts)>()); }


auto main() -> int
{
    auto t1 = std::tuple<int, double>{ 2, 0.6 };        ///< Types explicitly declared
    auto t2 = std::tuple{ 15.2f, "Hello" };             ///< With type deduction
    auto t3 = std::make_tuple("Bye", 15, 78, 343.546);  ///< With type deducing maker

    print(t1);
    print(t2);
    print(t3);

    return 0;
}
```

Pair Example

```cpp
#include <iostream>
#include <utility>

template<typename F, typename S>
auto print(const std::pair<F, S>& p) -> void
{
    std::cout << "( " << p.first << ", " << p.second << " )" << std::endl;
}

auto main() -> int
{
    auto p1 = std::pair<int, double>{ 2, 0.6 };     ///< Types explicitly declared
    auto p2 = std::pair{ 15.2f, "Hello" };          ///< With type deduction
    auto p3 = std::make_pair(78, 343.546);          ///< With type deducing maker

    print(p1);
    print(p2);
    print(p3);

    return 0;
}
```

Structured Bindings

- Tuples and *Tuple-Like* structures can be de-structured using structured bindings.
- This is a process of efficiently binding variables in an outer scope to the member variables of *Tuple-Like* objects.
- It involves declaring a list of variable names with automatic storage duration where each variable is bound to a member of the class in the declaration order they appear in the class.
- Structured bindings can be reference and constant qualified.

```cpp
#include <iostream>
#include <tuple>

auto main() -> int
{

    auto t = std::make_tuple(78, 343.546, "Hello");
    auto& [i1, f1, s1] = t;

    std::cout << "( " << i1 << ", " << f1 << ", " << s1 << " )" << std::endl;

    i1 = 576876;

    const auto& [i2, f2, s2] = t;
    std::cout << "( " << i2 << ", " << f2 << ", " << s2 << " )" << std::endl;

    return 0;
}
```

Concepts

Concepts

- Templates allow for powerful generic programming, compile time type instantiation and metaprogramming.
- However, templates are open ended, there are no restrictions on what you can instantiate a template with which can lead to failures later in the program when invalid operations are called on the instantiated type.
- To fix this C++ introduced *concepts*, a way to impose compile time conditions on a template type parameter.
- Concepts are paired with constraints to create custom 'recipes' for restricting types and ensuring that only valid types are used for template arguments.

```cpp
#include <concepts>
#include <iomanip>
#include <iostream>
#include <string>
#include <utility>

/// Concept defining a type that can be hashed using `std::hash`
template<typename H>
concept Hashable = requires (H a)
{
    { std::hash<H>{}(a) } -> std::convertible_to<std::size_t>;
};

struct NotHashable {};

using namespace std::literals;

auto main() -> int
{
   std::cout << std::boolalpha;
   std::cout << "Is Hashable<int>:           " << Hashable<int> << "   :
std::hash<int>{}(69)               = " << std::hash<int>{}(69) << std::endl;
   std::cout << "Is Hashable<float>:         " << Hashable<float> << "   :
std::hash<float>{}(4.5756f)        = " << std::hash<float>{}(4.5756f) << std::endl;
   std::cout << "Is Hashable<double>:        " << Hashable<double> << "   :
std::hash<double>{}(-0.0036565764) = " << std::hash<double>{}(-0.0036565764) <<
std::endl;
   std::cout << "Is Hashable<std::string>:   " << Hashable<std::string> << "   :
std::hash<std::string>{}()         = " << std::hash<std::string>{}(""s) << std::endl;
   std::cout << "Is Hashable<NotHashable>:   " << Hashable<NotHashable> << std::endl;

    return 0;
}
```

Template Failure Example

- Here is an example of code that will fail to compile; which is better than crashing but it isn't obvious why it fails.
- Can you find the error?

```cpp
/// ... Point implementation

using namespace std::literals;

auto main() -> int
{
   auto p1 = Point{ "Hello"s, "Hi"s };
   auto p2 = Point{ "Goobye"s, "Bye"s };

   auto p3 = p1 - p2;

   std::cout << p3 << std::endl;

    return 0;
}
```

Template Failure Error

```
<source>: In instantiation of 'constexpr Point<typename std::common_type<T, U>::type> Point<T>::operator-(const Point<U>&) [with U = std::__cxx11::basic_string<char>; T = std::__cxx11::basic_string<char>; typename std::common_type<T, U>::type = std::__cxx11::basic_string<char>]':
<source>:96:19:   required from here
<source>:61:62: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                           ~~^~~~~
In file included from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/string:47,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/locale_classes.h:40,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/ios_base.h:41,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ios:42,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ostream:38,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/iostream:39,
                 from <source>:1:
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                           ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                           ~~^~~~~
<source>:61:71: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                    ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                    ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                    ~~^~~~~
<source>:61:77: error: no matching function for call to 'Point<std::__cxx11::basic_string<char> >::Point(<brace-enclosed initializer list>)'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                            ^
<source>:20:15: note: candidate: 'constexpr Point<T>::Point(Point<T>&&) [with T = std::__cxx11::basic_string<char>]'
   20 |     constexpr Point(Point&& p) noexcept
      |               ^~~~~
<source>:20:15: note:   candidate expects 1 argument, 2 provided
<source>:16:15: note: candidate: 'constexpr Point<T>::Point(const Point<T>&) [with T = std::__cxx11::basic_string<char>]'
   16 |     constexpr Point(const Point& p) noexcept
      |               ^~~~~
<source>:16:15: note:   candidate expects 1 argument, 2 provided
<source>:12:5: note: candidate: 'constexpr Point<T>::Point(T, T) [with T = std::__cxx11::basic_string<char>]'
   12 |     Point(T x, T y) noexcept
      |     ^~~~~
<source>:12:5: note:   conversion of argument 1 would be ill-formed:
<source>:9:15: note: candidate: 'constexpr Point<T>::Point() [with T = std::__cxx11::basic_string<char>]'
    9 |     constexpr Point() = default;
      |               ^~~~~
<source>:9:15: note:   candidate expects 0 arguments, 2 provided
ASM generation compiler returned: 1
<source>: In instantiation of 'constexpr Point<typename std::common_type<T, U>::type> Point<T>::operator-(const Point<U>&) [with U = std::__cxx11::basic_string<char>; T = std::__cxx11::basic_string<char>; typename std::common_type<T, U>::type = std::__cxx11::basic_string<char>]':
<source>:96:19:   required from here
<source>:61:62: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                           ~~^~~~~
In file included from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/string:47,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/locale_classes.h:40,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/ios_base.h:41,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ios:42,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ostream:38,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/iostream:39,
                 from <source>:1:
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                           ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                           ~~^~~~~
<source>:61:71: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                    ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                    ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype ((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                    ~~^~~~~
<source>:61:77: error: no matching function for call to 'Point<std::__cxx11::basic_string<char> >::Point(<brace-enclosed initializer list>)'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                            ^
<source>:20:15: note: candidate: 'constexpr Point<T>::Point(Point<T>&&) [with T = std::__cxx11::basic_string<char>]'
   20 |     constexpr Point(Point&& p) noexcept
      |               ^~~~~
<source>:20:15: note:   candidate expects 1 argument, 2 provided
<source>:16:15: note: candidate: 'constexpr Point<T>::Point(const Point<T>&) [with T = std::__cxx11::basic_string<char>]'
   16 |     constexpr Point(const Point& p) noexcept
      |               ^~~~~
<source>:16:15: note:   candidate expects 1 argument, 2 provided
<source>:12:5: note: candidate: 'constexpr Point<T>::Point(T, T) [with T = std::__cxx11::basic_string<char>]'
   12 |     Point(T x, T y) noexcept
      |     ^~~~~
<source>:12:5: note:   conversion of argument 1 would be ill-formed:
<source>:9:15: note: candidate: 'constexpr Point<T>::Point() [with T = std::__cxx11::basic_string<char>]'
    9 |     constexpr Point() = default;
      |               ^~~~~
<source>:9:15: note:   candidate expects 0 arguments, 2 provided
Execution build compiler returned: 1
```

Template Failure Error

```
<source>: In instantiation of 'constexpr Point<typename std::common_type<T, U>::type> Point<T>::operator-(const Point<U>&) [with U =
std::__cxx11::basic_string<char>; T = std::__cxx11::basic_string<char>; typename std::common_type<T, U>::type = std::__cxx11::basic_string<char>]':
<source>:96:19:   required from here
<source>:61:62: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                            ~~^~~~~
In file included from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/string:47,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/locale_classes.h:40,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/ios_base.h:41,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ios:42,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ostream:38,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/iostream:39,
                 from <source>:1:
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class _IteratorR>
constexpr decltype ((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                            ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class _IteratorR>
constexpr decltype ((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                            ~~^~~~~
<source>:61:71: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                     ~~^~~~~
```

```
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class
_IteratorR> constexpr decltype ((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const
reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution
failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class
_IteratorR> constexpr decltype ((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const
move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution
failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                ~~^~~~~
<source>:61:77: error: no matching function for call to 'Point<std::__cxx11::basic_string<char> >::Point(<brace-enclosed initializer
list>)'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                            ^
<source>:20:15: note: candidate: 'constexpr Point<T>::Point(Point<T>&&) [with T = std::__cxx11::basic_string<char>]'
   20 |     constexpr Point(Point&& p) noexcept
      |               ^~~~~
<source>:20:15: note:   candidate expects 1 argument, 2 provided
<source>:16:15: note: candidate: 'constexpr Point<T>::Point(const Point<T>&) [with T = std::__cxx11::basic_string<char>]'
   16 |     constexpr Point(const Point& p) noexcept
      |               ^~~~~
<source>:16:15: note:   candidate expects 1 argument, 2 provided
<source>:12:5: note: candidate: 'constexpr Point<T>::Point(T, T) [with T = std::__cxx11::basic_string<char>]'
   12 |     Point(T x, T y) noexcept
      |     ^~~~~
```

```
<source>:12:5: note:   conversion of argument 1 would be ill-formed:
<source>:9:15: note: candidate: 'constexpr Point<T>::Point() [with T = std::__cxx11::basic_string<char>]'
    9 |     constexpr Point() = default;
      |               ^~~~~
<source>:9:15: note:   candidate expects 0 arguments, 2 provided
ASM generation compiler returned: 1
<source>: In instantiation of 'constexpr Point<typename std::common_type<T, U>::type> Point<T>::operator-(const Point<U>&) [with U =
std::__cxx11::basic_string<char>; T = std::__cxx11::basic_string<char>; typename std::common_type<T, U>::type = std::__cxx11::basic_string<char>]':
<source>:96:19:   required from here
<source>:61:62: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                            ~~^~~~~
In file included from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/string:47,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/locale_classes.h:40,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/ios_base.h:41,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ios:42,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/ostream:38,
                 from /opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/iostream:39,
                 from <source>:1:
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype
((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                            ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype
((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution failed:
<source>:61:62: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                            ~~^~~~~
```

```
<source>:61:71: error: no match for 'operator-' (operand types are 'std::__cxx11::basic_string<char>' and 'const std::__cxx11::basic_string<char>')
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype
((__y.base() - __x.base())) std::operator-(const reverse_iterator<_IteratorL>&, const reverse_iterator<_IteratorR>&)'
  621 |     operator-(const reverse_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:621:5: note:   template argument deduction/substitution failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                ~~^~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note: candidate: 'template<class _IteratorL, class _IteratorR> constexpr decltype
((__x.base() - __y.base())) std::operator-(const move_iterator<_IteratorL>&, const move_iterator<_IteratorR>&)'
 1778 |     operator-(const move_iterator<_IteratorL>& __x,
      |     ^~~~~~~~
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/bits/stl_iterator.h:1778:5: note:   template argument deduction/substitution failed:
<source>:61:71: note:   'std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_IteratorL>'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                ~~^~~~~
<source>:61:77: error: no matching function for call to 'Point<std::__cxx11::basic_string<char> >::Point(<brace-enclosed initializer list>)'
   61 |     { return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }
      |                                                                        ^
<source>:20:15: note: candidate: 'constexpr Point<T>::Point(Point<T>&&) [with T = std::__cxx11::basic_string<char>]'
   20 |     constexpr Point(Point&& p) noexcept
      |               ^~~~~
<source>:20:15: note:   candidate expects 1 argument, 2 provided
<source>:16:15: note: candidate: 'constexpr Point<T>::Point(const Point<T>&) [with T = std::__cxx11::basic_string<char>]'
   16 |     constexpr Point(const Point& p) noexcept
      |               ^~~~~
<source>:16:15: note:   candidate expects 1 argument, 2 provided
<source>:12:5: note: candidate: 'constexpr Point<T>::Point(T, T) [with T = std::__cxx11::basic_string<char>]'
   12 |     Point(T x, T y) noexcept
      |     ^~~~~
<source>:12:5: note:   conversion of argument 1 would be ill-formed:
<source>:9:15: note: candidate: 'constexpr Point<T>::Point() [with T = std::__cxx11::basic_string<char>]'
    9 |     constexpr Point() = default;
      |               ^~~~~
<source>:9:15: note:   candidate expects 0 arguments, 2 provided
Execution build compiler returned: 1
```

Constrained Template Parameters

- We can constrain a template type parameter by a single concept by replacing `typename` with the concept we want to impose on the type parameter.
- This automatically passes the type parameter to the concepts template type parameter.
- On the left we can see our `Point` example being constrained to a standard type concept `std::integral` constraining `T` to types that are integrals (`int`, `char`, `long` etc.).
- What do you think the compiler will do?
    - Compile because we don't us the `-` on any `Point`.
    - Fail with an error.
    - Blow up.

```cpp
template<std::integral T>
class Point
{
    /// ... implementation
};

using namespace std::literals;

auto main() -> int
{
   auto p1 = Point{ "Hello"s, "Hi"s };

    return 0;
}
```

Concept Failure Error

```
<source>: In function 'int main()':
<source>:94:37: error: class template argument deduction failed:
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:94:37: error: no matching function for call to 'Point(std::__cxx11::basic_string<char>, std::__cxx11::basic_string<char>)'
<source>:21:15: note: candidate: 'template<class T> Point(Point<T>&&)-> Point<T>'
   21 |     constexpr Point(Point&& p) noexcept
      |               ^~~~~
<source>:21:15: note:   template argument deduction/substitution failed:
<source>:94:37: note:   'std::__cxx11::basic_string<char>' is not derived from 'Point<T>'
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:17:15: note: candidate: 'template<class T> Point(const Point<T>&)-> Point<T>'
   17 |     constexpr Point(const Point& p) noexcept
      |               ^~~~~
<source>:17:15: note:   template argument deduction/substitution failed:
<source>:94:37: note:   'std::__cxx11::basic_string<char>' is not derived from 'const Point<T>'
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:13:5: note: candidate: 'template<class T> Point(T, T)-> Point<T>'
   13 |     Point(T x, T y) noexcept
      |     ^~~~~
<source>:13:5: note:   template argument deduction/substitution failed:
<source>: In substitution of 'template<class T> Point(T, T)-> Point<T> [with T = std::__cxx11::basic_string<char>]':
<source>:94:37:   required from here
<source>:13:5: error: template constraint failure for 'template<class T>  requires  integral<T> class Point'
<source>:13:5: note: constraints not satisfied
In file included from <source>:1:
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts: In substitution of 'template<class T>  requires  integral<T> class Point [with T = std::__cxx11::basic_string<char>]':
<source>:13:5:   required by substitution of 'template<class T> Point(T, T)-> Point<T> [with T = std::__cxx11::basic_string<char>]'
<source>:94:37:   required from here
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts:100:13:   required for the satisfaction of 'integral<T>' [with T = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts:100:24: note: the expression 'is_integral_v<_Tp> [with _Tp = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]' evaluated to 'false'
  100 |     concept integral = is_integral_v<_Tp>;
      |                        ^~~~~~~~~~~~~~~~~~
<source>:10:15: note: candidate: 'template<class T> Point()-> Point<T>'
   10 |     constexpr Point() = default;
      |               ^~~~~
<source>:10:15: note:   template argument deduction/substitution failed:
<source>:94:37: note:   candidate expects 0 arguments, 2 provided
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:6:7: note: candidate: 'template<class T> Point(Point<T>)-> Point<T>'
    6 | class Point
      |       ^~~~~
<source>:6:7: note:   template argument deduction/substitution failed:
<source>:94:37: note:   'std::__cxx11::basic_string<char>' is not derived from 'Point<T>'
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
ASM generation compiler returned: 1
<source>: In function 'int main()':
<source>:94:37: error: class template argument deduction failed:
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:94:37: error: no matching function for call to 'Point(std::__cxx11::basic_string<char>, std::__cxx11::basic_string<char>)'
<source>:21:15: note: candidate: 'template<class T> Point(Point<T>&&)-> Point<T>'
   21 |     constexpr Point(Point&& p) noexcept
      |               ^~~~~
<source>:21:15: note:   template argument deduction/substitution failed:
<source>:94:37: note:   'std::__cxx11::basic_string<char>' is not derived from 'Point<T>'
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:17:15: note: candidate: 'template<class T> Point(const Point<T>&)-> Point<T>'
   17 |     constexpr Point(const Point& p) noexcept
      |               ^~~~~
<source>:17:15: note:   template argument deduction/substitution failed:
<source>:94:37: note:   'std::__cxx11::basic_string<char>' is not derived from 'const Point<T>'
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:13:5: note: candidate: 'template<class T> Point(T, T)-> Point<T>'
   13 |     Point(T x, T y) noexcept
      |     ^~~~~
<source>:13:5: note:   template argument deduction/substitution failed:
<source>: In substitution of 'template<class T> Point(T, T)-> Point<T> [with T = std::__cxx11::basic_string<char>]':
<source>:94:37:   required from here
<source>:13:5: error: template constraint failure for 'template<class T>  requires  integral<T> class Point'
<source>:13:5: note: constraints not satisfied
In file included from <source>:1:
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts: In substitution of 'template<class T>  requires  integral<T> class Point [with T = std::__cxx11::basic_string<char>]':
<source>:13:5:   required by substitution of 'template<class T> Point(T, T)-> Point<T> [with T = std::__cxx11::basic_string<char>]'
<source>:94:37:   required from here
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts:100:13:   required for the satisfaction of 'integral<T>' [with T = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts:100:24: note: the expression 'is_integral_v<_Tp> [with _Tp = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]' evaluated to 'false'
  100 |     concept integral = is_integral_v<_Tp>;
      |                        ^~~~~~~~~~~~~~~~~~
<source>:10:15: note: candidate: 'template<class T> Point()-> Point<T>'
   10 |     constexpr Point() = default;
      |               ^~~~~
<source>:10:15: note:   template argument deduction/substitution failed:
<source>:94:37: note:   candidate expects 0 arguments, 2 provided
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
<source>:6:7: note: candidate: 'template<class T> Point(Point<T>)-> Point<T>'
    6 | class Point
      |       ^~~~~
<source>:6:7: note:   template argument deduction/substitution failed:
<source>:94:37: note:   'std::__cxx11::basic_string<char>' is not derived from 'Point<T>'
   94 |    auto p1 = Point{ "Hello"s, "Hi"s };
      |                                     ^
Execution build compiler returned: 1
```

Why does it fail?

Concept Failure Error – A Closer Look

```
<source>: In substitution of 'template<class T> Point(T, T)-> Point<T> [with T = std::__cxx11::basic_string<char>]':
<source>:94:37:   required from here
<source>:13:5: error: template constraint failure for 'template<class T>  requires  integral<T> class Point'
<source>:13:5: note: constraints not satisfied
In file included from <source>:1:
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts: In substitution of 'template<class T>  requires  integral<T> class
Point [with T = std::__cxx11::basic_string<char>]':
<source>:13:5:   required by substitution of 'template<class T> Point(T, T)-> Point<T> [with T =
std::__cxx11::basic_string<char>]'
<source>:94:37:   required from here
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts:100:13:   required for the satisfaction of 'integral<T>' [with T =
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]
/opt/compiler-explorer/gcc-12.2.0/include/c++/12.2.0/concepts:100:24: note: the expression 'is_integral_v<_Tp> [with _Tp =
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]' evaluated to 'false'
  100 |     concept integral = is_integral_v<_Tp>;
```

This is a snippet from the error message. Can you see what caused the error?

Requirement Expressions

- We can further customize the requirements of template type parameters using `requires` expressions. This use conjunctions (`&&`) and disjunctions (`||`) of concepts to build a specific constraint.
- When using a `requires` expression, concepts must be explicitly instantiated with their template type parameters.
- For `Point` we have imposed that `T` must be a type that satisfies either `std::integral` or `std::floating_point`.

```cpp
template<typename T>
    requires std::integral<T> || std::floating_point<T>
class Point
{
    /// ... implementation
};

auto main() -> int
{
   auto p1 = Point{ 0.567, 45.657 };
   auto p2 = Point{ 1, 9 };

   std::cout << p1 + p2 << std::endl;
   std::cout << p1 - p2 << std::endl;

    return 0;
}
```

Requirement Clauses

- Sometimes we need to impose requirements based on runtime properties of a type or depends on the result of two objects or types. This is where `requires` clauses can be used to create an instantiation of the type and validate these requirements.
- `requires` clauses are introduced with an additional `requires` keyword followed a *function-like* signature specifying mock runtime parameter.
- With these parameters you can specifiy expressions that must be valid
- For `Point` we have imposed that `+` and `-` must be valid between the types `T` and `U`.

```
/// ... Point details

template<typename U>
    requires requires (T a, U b)
    {
        a + b;
    }
constexpr auto
operator+ (const Point<U>& p)
    noexcept -> Point<typename std::common_type<T, U>::type>
{ return Point<typename std::common_type<T, U>::type>{ x + p.x, y + p.y }; }

template<typename U>
    requires requires (T a, U b)
    {
        a - b;
    }
constexpr auto
operator- (const Point<U>& p)
    noexcept -> Point<typename std::common_type<T, U>::type>
{ return Point<typename std::common_type<T, U>::type>{ x - p.x, y - p.y }; }

/// ... Point details
```

Constrained Variables

- We can also constrain variables and parameters with automatic storage duration.
- This allows us to ensure that variables only accept values whose type satisfy the concept.
- Constant and reference qualifiers can still be specified

```cpp
#include <concepts>
#include <iostream>

auto println(const std::integral auto& v) -> void
{ std::cout << v << std::endl; }

auto main() -> int
{
    println(9);
    // println(46.567);  ///< fails

    return 0;
}
```

Discussion

- Any questions?
- Need help?
- Open discussion.
- Concerns?
- Assignment 2

Next Week

- Iterators
- Data Structures
- Algorithms
- Ranges
- Views

Thank You

Tyler Swann

https://github.com/MonashDeepNeuron/HPP