

C++ ONLINE

ABEL SEN

THE EVOLUTION OF
FUNCTIONAL PROGRAMMING
IN C++

2024

About Me

Functional and systems programmer focused on Haskell,
C and C++

Socials

 abel-sen

 neuroevolutus

 neuroevolutus

 neuroevolutus

 neuroevolutus

Agenda

- C++ Origins and Ethos
- The evolution of FP in C++ from C++98 through C++23
 - Understanding the benefits of each FP feature
 - Comparisons with Haskell and its functional features
- FP in C++26 and beyond

What is C++?

- Inspired by the powerful type-checking and abstractions provided by Simula
 - Dr. Bjarne Stroustrup used Simula for his Ph.D. thesis to implement a simulator for distributed systems
- Influenced by the difficulty of rewriting the simulator in BCPL

- Dr. Stroustrup went on to work at Bell Labs
- Found himself writing a distributed system again
 - Could no longer repeat the Herculean effort of writing a complex system in BCPL
- Early C++, i.e. C with Classes, was born

So what can we say about C++ as a language?

- A systems language built to solve *real-world* programming problems
- Offers lightweight to zero-cost abstractions for effectively modelling *concepts* from any problem domain
- Can *scale* to handle the complexity of the problems you face

C++98

Core Functional Features

- Functions as first-class citizens
- Generic programming

Core Functional Features

- Function pointers
- Function references
- Pointers to members
- Function objects
- Templates

Core Functional Features

- Function pointers

Functions as first-class citizens

- Pass functions to functions
- Return functions from functions
- Bind functions to names

Function pointers

- A feature shared with C
- Make functions an (arguably) first-class feature

Key benefit: refactoring high-level algorithms out of concrete implementations

A real-world motivating example

So you need to sort a list...

- Need to do it two ways

- Need to do it two ways
- Prefer for it to be maintainable

- Need to do it two ways
- Prefer for it to be maintainable
- What do you do? 🤔

Our initial attempt


```
69     v1.push_back(8);
70     v1.push_back(9);
71     v1.push_back(7);
72     v1.push_back(5);
73     v1.push_back(6);
74     v1.push_back(4);
75     std::vector<int> v2(v1);
76     lomuto_quicksort(v1);
77     hoare_quicksort(v2);
78     for (std::vector<int>::size_type i = 0; i < v1.size(); ++i) {
79         std::cout << v1[i] << ' ';
80     }
81     std::cout << '\n';
82     for (std::vector<int>::size_type i = 0; i < v2.size(); ++i) {
83         std::cout << v2[i] << ' ';
84     }
```

<https://godbolt.org/z/Wr5KnGzeY>

```
--      last_element_index
51      );
52  );
53  hoare_quicksort_impl(v, 0, pivot_index - 1);
54  hoare_quicksort_impl(v, pivot_index + 1, last_element_index);
55 }
56
57 void hoare_quicksort(std::vector<int>& v) {
58     hoare_quicksort_impl(v, 0, static_cast<int>(v.size()) - 1);
59 }
60
61 int main()
62 {
63     std::vector<int> v1;
64     v1.reserve(10);
65     v1.push_back(3);
--
```

<https://godbolt.org/z/Wr5KnGzeY>

```
41 void hoare_quicksort_impl(
42     std::vector<int>& v,
43     int const first_element_index,
44     int const last_element_index
45 ) {
46     if (first_element_index ≥ last_element_index
47         || first_element_index < 0) return;
48     int const pivot_index = hoare_partition(
49         v,
50         first_element_index,
51         last_element_index
52     );
53     hoare_quicksort_impl(v, 0, pivot_index - 1);
54     hoare_quicksort_impl(v, pivot_index + 1, last_element_index);
55 }
```

<https://godbolt.org/z/Wr5KnGzeY>

```
30 void lomuto_quicksort(std::vector<int>& v)
31 {
32     lomuto_quicksort_impl(v, 0, static_cast<int>(v.size()) - 1);
33 }
34
35 int hoare_partition(
36     std::vector<int>& v,
37     int const first_element_index,
38     int const last_element_index
39 ); // Definition elided
40
41 void hoare_quicksort_impl(
42     std::vector<int>& v,
43     int const first_element_index,
44     int const last_element_index
45 )
```

<https://godbolt.org/z/Wr5KnGzeY>

```
24     last_element_index
25 );
26 lomuto_quicksort_impl(v, first_element_index, pivot_index - 1);
27 lomuto_quicksort_impl(v, pivot_index + 1, last_element_index);
28 }
29
30 void lomuto_quicksort(std::vector<int>& v)
31 {
32     lomuto_quicksort_impl(v, 0, static_cast<int>(v.size()) - 1);
33 }
34
35 int hoare_partition(
36     std::vector<int>& v,
37     int const first_element_index,
38     int const last_element_index
39 ): // Definition elided
```

<https://godbolt.org/z/Wr5KnGzeY>

```
--  
14 void lomuto_quicksort_impl(  
15     std::vector<int>& v,  
16     int const first_element_index,  
17     int const last_element_index  
18 ) {  
19     if (first_element_index ≥ last_element_index  
20         || first_element_index < 0) return;  
21     int const pivot_index = lomuto_partition(  
22         v,  
23         first_element_index,  
24         last_element_index  
25     );  
26     lomuto_quicksort_impl(v, first_element_index, pivot_index - 1);  
27     lomuto_quicksort_impl(v, pivot_index + 1, last_element_index);  
28 }
```

<https://godbolt.org/z/Wr5KnGzeY>

```
-----  
3 #include <iostream>  
4 #include <vector>  
5  
6 // Referenced from: https://en.wikipedia.org/wiki/Quicksort  
7  
8 int lomuto_partition(  
9     std::vector<int>& v,  
10    int const first_element_index,  
11    int const last_element_index  
12 ); // Definition elided  
13  
14 void lomuto_quicksort_impl(  
15     std::vector<int>& v,  
16     int const first_element_index,  
17     int const last_element_index  
18 ) {
```

<https://godbolt.org/z/Wr5KnGzeY>

Same driver skeleton

```
void variant_quicksort(std::vector<int>& v) {  
    variant_quicksort_impl(v, 0, static_cast<int>(v.size()) - 1);  
}
```

Same algorithm skeleton

```
1 void variant_quicksort_impl(
2     std::vector<int>& v,
3     int const first_element_index,
4     int const last_element_index
5 ) {
6     if (first_element_index ≥ last_element_index
7         || first_element_index < 0) return;
8     int const pivot_index = variant_partition(
9         v,
10        first_element_index,
11        last_element_index
12    );
13    variant_quicksort_impl(v, first_element_index, pivot_index - 1);
14    variant_quicksort_impl(v, pivot_index + 1, last_element_index);
15 }
```

Same algorithm skeleton

```
1 void variant_quicksort_impl(
2     std::vector<int>& v,
3     int const first_element_index,
4     int const last_element_index
5 ) {
6     if (first_element_index ≥ last_element_index
7         || first_element_index < 0) return;
8     int const pivot_index = variant_partition(
9         v,
10        first_element_index,
11        last_element_index
12    );
13    variant_quicksort_impl(v, first_element_index, pivot_index - 1);
14    variant_quicksort_impl(v, pivot_index + 1, last_element_index);
15 }
```

Same algorithm skeleton

```
1 void variant_quicksort_impl(
2     std::vector<int>& v,
3     int const first_element_index,
4     int const last_element_index
5 ) {
6     if (first_element_index ≥ last_element_index
7         || first_element_index < 0) return;
8     int const pivot_index = variant_partition(
9         v,
10        first_element_index,
11        last_element_index
12    );
13    variant_quicksort_impl(v, first_element_index, pivot_index - 1);
14    variant_quicksort_impl(v, pivot_index + 1, last_element_index);
15 }
```

Only differing partition strategies!

 Extract out the function as a parameter


```
34 }
35
36 void quicksort(
37     std::vector<int>& v,
38     int (*partition)(std::vector<int>&, int, int)
39 ) {
40     quicksort_impl(
41         v,
42         0,
43         static_cast<int>(v.size()) - 1,
44         partition
45     );
46 }
47
48 int main()
```

<https://godbolt.org/z/dv1n3q79T>

```
20
21 void quicksort_impl(
22     std::vector<int>& v,
23     int const first_element_index,
24     int const last_element_index,
25     int (*partition)(std::vector<int>&, int, int)
26 ) {
27     if (first_element_index ≥ last_element_index
28         || first_element_index < 0) return;
29     int const pivot_index = partition(
30         v, first_element_index, last_element_index
31     );
32     quicksort_impl(v, 0, pivot_index - 1, partition);
33     quicksort_impl(v, pivot_index + 1, last_element_index, partition);
34 }
35
```

<https://godbolt.org/z/dv1n3q79T>

```
6 // Referenced from: https://en.wikipedia.org/wiki/Quicksort
7
8 int lomuto_partition(
9     std::vector<int>& v,
10    int const first_element_index,
11    int const last_element_index
12 ); // Definition elided
13
14
15 int hoare_partition(
16     std::vector<int>& v,
17     int const first_element_index,
18     int const last_element_index
19 ); // Definition elided
20
21 void quicksort impl(
```

<https://godbolt.org/z/dv1n3q79T>

Benefits

- Sorting algorithm separated from concrete partition strategy
- Code can be refactored or extended with new strategies without unnecessary code bloat

Cons

- Parameter drilling of function pointer
- Runtime penalty of function pointer dereference

Core Functional Features

- Function pointers
- Function references
- Pointers to members
- Function objects
- Templates

Function references

- Similar to function pointers
- But cannot be reseated
- And can never be null

Let's revisit how we sort


```
30 );
31 quicksort_impl(v, 0, pivot_index - 1, partition);
32 quicksort_impl(v, pivot_index + 1, last_element_index, partition);
33 }
34
35 void quicksort(
36     std::vector<int>& v,
37     int (&partition)(std::vector<int>&, int, int)
38 ) {
39     quicksort_impl(
40         v,
41         0,
42         static_cast<int>(v.size()) - 1,
43         partition
44     );

```

<https://godbolt.org/z/53853noP6>

<https://godbolt.org/z/53853noP6>

Core Functional Features

- Function pointers
- Function references
- Pointers to members
- Function objects
- Templates

The <functional> header

The <functional> header

- Defines a set of function objects that compose well with algorithm templates
 - A function object is any object for which operator() is defined
- Introduces *closures* and *partial application* to the world of C++

Let's first do another refactor

Key principle: Allow user to pick the type of customization
similar to before


```
1 #ifndef QUICKSORT_HPP_INCLUDED
2 #define QUICKSORT_HPP_INCLUDED
3
4 #include <vector>
5
6 // Referenced from: https://en.wikipedia.org/wiki/Quicksort
7
8 template <typename PartitionStrategy>
9 void quicksort_impl(
10     std::vector<int>& v,
11     int const first_element_index,
12     int const last_element_index,
13     PartitionStrategy partition
14 {
15     if (first_element_index ≥ last_element_index)
16         // First element index is greater than or equal to last element index
```

<https://godbolt.org/z/fMoe3r1Yv>

<https://godbolt.org/z/fMoe3r1Yv>

```
--  -----
12  int const last_element_index,
13  PartitionStrategy partition
14 {
15  if (first_element_index >= last_element_index
16      || first_element_index < 0) return;
17  int const pivot_index = partition(
18      v, first_element_index, last_element_index
19 );
20  quicksort_impl(v, 0, pivot_index - 1, partition);
21  quicksort_impl(v, pivot_index + 1, last_element_index, partition);
22 }
23
24 template <typename PartitionStrategy>
25 void quicksort(
26     std::vector<int>& v,
27     ...
```

<https://godbolt.org/z/fMoe3r1Yv>

...and the client code can stay the same, too

Now, we can pass function objects to `quicksort` to specify the partition strategy.

```
1 #include "quicksort.hpp"
2
3 #include <algorithm>
4 #include <cstdlib>
5 #include <iostream>
6 #include <vector>
7
8 // Referenced from: https://en.wikipedia.org/wiki/Quicksort
9
10 struct LomutoPartition {
11     int operator()(
12         std::vector<int>& v,
13         int const first_element_index,
14         int const last_element_index
15     ) const; // Definition elided
16 }
```

<https://godbolt.org/z/PP4szz8sn>

```
--      -----
14     int const last_element_index
15   ) const; // Definition elided
16 };
17
18 struct HoarePartition{
19   int operator()(
20     std::vector<int>& v,
21     int const first_element_index,
22     int const last_element_index
23   ) const; // Definition elided
24 };
25
26 int main()
27 {
28   std::vector<int> v1;
29 }
```

<https://godbolt.org/z/PP4szz8sn>

```
54     v1.push_back(8);
55     v1.push_back(9);
56     v1.push_back(7);
57     v1.push_back(5);
58     v1.push_back(6);
59     v1.push_back(4);
60     std::vector<int> v2(v1);
61     quicksort(v1, LomutoPartition());
62     quicksort(v2, HoarePartition());
63     for (std::vector<int>::size_type i = 0; i < v1.size(); ++i) {
64         std::cout << v1[i] << ' ';
65     }
66     std::cout << '\n';
67     for (std::vector<int>::size_type i = 0; i < v2.size(); ++i) {
68         std::cout << v2[i] << ' ';
69     }
```

<https://godbolt.org/z/PP4szz8sn>

- Benefit: potentially better inlining due to using templates and stateless objects
- Con: potentially more binary bloat because of template instantiation

Function objects allow us to emulate closures by capturing values or variables upon construction.

```
1 #include <algorithm>
2 #include <cstdlib>
3 #include <iostream>
4 #include <vector>
5
6 class AddNumber {
7     int number;
8     public:
9     AddNumber(int number) {
10         this->number = number;
11     }
12     int operator()(int other_number) const {
13         return number + other_number;
14     }
15 };
//
```

<https://godbolt.org/z/5h95sTv8x>

```
17 int main() {
18     std::vector<int> v1, v2;
19     v1.push_back(1);
20     v1.push_back(2);
21     v1.push_back(3);
22     // Pretend x is a user-supplied int
23     int const x = 5;
24     std::transform(
25         v1.begin(), v1.end(), std::back_inserter(v2), AddNumber(x)
26     );
27     for (std::vector<int>::size_type i = 0; i < v2.size(); ++i) {
28         std::cout << v2[i] << ' ';
29     }
30     std::cout << '\n';
31     return EXIT_SUCCESS;
32 }
```

<https://godbolt.org/z/5h95sTv8x>

A slight problem

A slight problem

- Already have a function object that expresses our desired computation

A slight problem

- Already have a function object that expresses our desired computation
- But its signature is a bit too general

A slight problem

- Already have a function object that expresses our desired computation
- But its signature is a bit too general
- Can we avoid reinventing the wheel?

```
1 #include <algorithm>
2 #include <cstdlib>
3 #include <iostream>
4 #include <numeric>
5 #include <vector>
6
7 int main() {
8     std::vector<int> v1;
9     v1.push_back(1);
10    v1.push_back(2);
11    v1.push_back(3);
12    // Printing out the sum
13    std::cout << std::accumulate(
14        v1.begin(), v1.end(), 0, std::plus<int>()
15    ) << '\n';
16 }
```

<https://godbolt.org/z/Ts4b84Ezf>

```
- -----
3 #include <iostream>
4 #include <numeric>
5 #include <vector>
6
7 int main() {
8     std::vector<int> v1;
9     v1.push_back(1);
10    v1.push_back(2);
11    v1.push_back(3);
12    // Printing out the sum
13    std::cout << std::accumulate(
14        v1.begin(), v1.end(), 0, std::plus<int>()
15    ) << '\n';
16    return EXIT_SUCCESS;
17 }
```

<https://godbolt.org/z/Ts4b84Ezf>

Solution: specialising `std::plus` with the parameter we already have on hand

```
1 #include <algorithm>
2 #include <cstdlib>
3 #include <functional>
4 #include <iostream>
5 #include <vector>
6
7 int main()
8 {
9     std::vector<int> v1, v2;
10    v1.push_back(1);
11    v1.push_back(2);
12    v1.push_back(3);
13    int const x = 5;
14    std::transform(
15        v1.begin(),
16        v1.end(),
17        v2.begin(),
18        std::bind2nd(std::multiplies<int>(), x));
```

<https://godbolt.org/z/vba5dsGM5>

```
10 // --- P R O G R A M ---\n11 v1.push_back(2);\n12 v1.push_back(3);\n13 int const x = 5;\n14 std::transform(\n15     v1.begin(),\n16     v1.end(),\n17     std::back_inserter(v2),\n18     std::bind1st(std::plus<int>(), x)\n19 );\n20 for (std::vector<int>::size_type i = 0; i < v2.size(); ++i) {\n21     std::cout << v2[i] << ' ';\n22 }\n23 std::cout << '\n';\n24 return EXIT_SUCCESS;\n25 }
```

<https://godbolt.org/z/vba5dsGM5>

- Similar to `bind1st`, `bind2nd` specialises a function object by binding the *second* argument.
- Many other function objects in `<functional>` like `minus<T>`, `equal_to<T>`, etc.

Core Functional Features

- Function pointers
- Function references
- Pointers to members
- Function objects
- Templates

Pointers to members

- Can be turned into function objects that take an object and return a member
- <functional> also provides function objects and function templates like `mem_fun_t` and `mem_fn` for this purpose

Core Functional Features

- Function pointers
- Function references
- Pointers to members
- Function objects
- Templates

Generic programming with templates

- Templates allow you to specify generic *recipes* for baking concrete classes or functions.
- Enable one to *specialise* the recipe for specific classes

A (somewhat) realistic example

```
1 #ifndef STACK_HPP_INCLUDED
2 #define STACK_HPP_INCLUDED
3
4 #include <cstdlib>
5 #include <stdexcept>
6
7 template <typename T>
8 struct StackNode {
9     T value;
10    StackNode* rest;
11    StackNode* previous;
12    StackNode(T const value) {
13         this->value = value;
14     }
15    StackNode() {
16        // ...
17    }
18 }
```

<https://godbolt.org/z/6nszxW8az>

```
15     StackNode() {
16         rest = previous = NULL;
17     }
18 };
19
20 template <typename T>
21 class Stack {
22     StackNode<T>* head;
23
24     public:
25     Stack() {
26         head = NULL;
27     }
28
29     Stack(T const value) {
30         head = new StackNode<T>(value);
31         previous = head;
32         rest = NULL;
33     }
34 }
```

<https://godbolt.org/z/6nszxW8az>

```
18 };
19
20 template <typename T>
21 class Stack {
22     StackNode<T>* head;
23
24     public:
25     Stack() {
26         head = NULL;
27     }
28
29     Stack(T const value) {
30         push(value);
31     }
32
33 // Definitions elided
```

<https://godbolt.org/z/6nszxW8az>

```
--  
23  
24     public:  
25     Stack() {  
26         head = NULL;  
27     }  
28  
29     Stack(T const value) {  
30         push(value);  
31     }  
32  
33     // Definitions elided  
34     void push(T const value);  
35     T front() const;  
36     T pop();  
37     ~Stack();  
--
```

<https://godbolt.org/z/6nszxW8az>

```
--  
26     head = NULL;  
27 }  
28  
29 Stack(T const value) {  
30     push(value);  
31 }  
32  
33 // Definitions elided  
34 void push(T const value);  
35 T front() const;  
36 T pop();  
37 ~Stack();  
38 };  
39  
40 #endif
```

<https://godbolt.org/z/6nszxW8az>

```
1 #include "Stack.hpp"
2
3 #include <cstdlib>
4 #include <iostream>
5
6 int main() {
7     Stack<int> stack;
8     for (std::size_t i = 0; i < 3; ++i) {
9         stack.push(i);
10    }
11    for (std::size_t i = 0; i < 3; ++i) {
12        std::cout << stack.pop() << '\n';
13    }
14    return EXIT_SUCCESS;
15 }
```

<https://godbolt.org/z/6nszxW8az>

An example of specialisation

```
1 #ifndef STACK_HPP_INCLUDED
2 #define STACK_HPP_INCLUDED
3
4 #include <cstdlib>
5 #include <stdexcept>
6
7 template <typename T>
8 struct StackNode {
9     T value;
10    StackNode* rest;
11    StackNode* previous;
12    StackNode(T const value) {
13         this->value = value;
14     }
15    StackNode() {
16        // ...
17    }
18 }
```

<https://godbolt.org/z/f46q345P6>

```
44     std::size_t capacity;
45
46     void resize()
47     {
48         unsigned char* old_bitset = bitset;
49         std::size_t const num_chars_used
50             = capacity / sizeof(unsigned char) / 8;
51         bitset = new unsigned char[num_chars_used * 2];
52         for (std::size_t i = 0; i < num_chars_used; ++i) {
53             bitset[i] = old_bitset[i];
54         }
55         capacity *= 2;
56     }
57
58     static std::size_t char_index(std::size_t i)
59     {
60         if (i > capacity)
61             throw std::out_of_range("Index out of range");
62         return i / 8;
63     }
64 }
```

<https://godbolt.org/z/f46q345P6>

```
55     capacity *= 2;
56 }
57
58 static std::size_t char_index(std::size_t i)
59 {
60     return i / (sizeof(unsigned char) * 8);
61 }
62
63 static std::size_t bit_index(std::size_t i)
64 {
65     return i % (sizeof(unsigned char) * 8);
66 }
67
68 public:
69 Stack()
70 {
```

<https://godbolt.org/z/f46q345P6>

```
64    {
65        return i % (sizeof(unsigned char) * 8);
66    }
67
68    public:
69    Stack()
70    {
71        bitset = NULL;
72        num_elements = 0;
73        capacity = sizeof(unsigned char) * 8;
74    }
75
76    Stack(bool b)
77    {
78        push(b);
79    }
```

<https://godbolt.org/z/f46q345P6>


```
73     } // Stack, stack-allocated objects, etc,
74 }
75
76 Stack(bool b)
77 {
78     push(b);
79 }
80
81 // Definitions elided
82 void push(bool b);
83 bool front() const;
84 bool pop();
85 ~Stack();
86 };
87
88 #endif
```

<https://godbolt.org/z/f46q345P6>

```
1 #include "Stack.hpp"
2
3 #include <cstdlib>
4 #include <iostream>
5
6 int main()
7 {
8     Stack<bool> stack;
9     for (std::size_t i = 0; i < 16; ++i) {
10         stack.push(bool(i & 1));
11     }
12     std::cout << std::boolalpha;
13     for (std::size_t i = 0; i < 16; ++i) {
14         std::cout << stack.pop() << '\n';
15     }
16 }
```

<https://godbolt.org/z/f46q345P6>

Comparing with Haskell

- First-class functions = 🍞🧈
- No OOP-style classes or objects 😱
- Currency of programs are sum and product types + ×

Let's see it in code.

```
1 module Main where
2
3 import Data.Foldable (foldl')
4
5 add :: Float → Float → Float
6 add a b = a + b
7
8 mult :: Float → Float → Float
9 mult a b = a * b
10
11 listSum :: [Float] → Float
12 listSum xs = foldl' add 0 xs
13 -- alternatively use foldl' (+) 0 xs
14
15 listProd :: [Float] → Float
16 listProd xs = foldl' mult 1 xs
```

<https://godbolt.org/z/hYv5Ebo5z>

```
7
8 mult :: Float → Float → Float
9 mult a b = a * b
10
11 listSum :: [Float] → Float
12 listSum xs = foldl' add 0 xs
13 -- alternatively use foldl' (+) 0 xs
14
15 listProd :: [Float] → Float
16 listProd xs = foldl' mult 1 xs
17 -- alternatively use foldl' (*) 1 xs
18
19 useListSum :: Bool → [Float] → Float
20 useListSum (True) = listSum
21 useListSum (False) = listProd
```

<https://godbolt.org/z/hYv5Ebo5z>

```
14
15 listProd :: [Float] → Float
16 listProd xs = foldl' mult 1 xs
17 -- alternatively use foldl' (*) 1 xs
18
19 useListSum :: Bool → [Float] → Float
20 useListSum (True) = listSum
21 useListSum (False) = listProd
22 -- alternatively use an if expression
23
24 main :: IO ()
25 main = do
26   let list = [1.0, 2.0, 3.0]
27   putStrLn (show (useListSum True list))
28   putStrLn (show (useListSum False list))
```

<https://godbolt.org/z/hYv5Ebo5z>

```
14
15 listProd :: [Float] → Float
16 listProd xs = foldl' mult 1 xs
17 -- alternatively use foldl' (*) 1 xs
18
19 useListSum :: Bool → [Float] → Float
20 useListSum (True) = listSum
21 useListSum (False) = listProd
22 -- alternatively use an if expression
23
24 main :: IO ()
25 main = do
26   let list = [1.0, 2.0, 3.0]
27   putStrLn (show (useListSum True list))
28   putStrLn (show (useListSum False list))
```

<https://godbolt.org/z/hYv5Ebo5z>

Haskell lambdas and closures

```
1 {-# LANGUAGE BlockArguments #-}
2
3 module Main where
4
5 import Control.Monad (forM_)
6
7 main :: IO ()
8 main = do
9   let x = 3
10  let l = map (+ x) [1..5]
11  forM_ l \x → do
12    putStrLn $ show x
```

<https://godbolt.org/z/KnT1Tb35P>

How user-defined data types work in Haskell


```
15 -- but we use this custom type for the sake of
14 -- example
15
16 -- A data "template"
17 -- i.e. a polymorphic data type
18 data Optional a = None | Some a
19
20 -- A simple stack
21 data Stack a = Nil | Cons a (Stack a)
22   deriving (Show, Foldable)
23
24 empty :: Stack a
25 empty = Nil
26
27 push :: a → Stack a → Stack a
28 push = Cons
```

<https://godbolt.org/z/a3MWf94n5>

```
--  
21 data Stack a = Nil | Cons a (Stack a)  
22   deriving (Show, Foldable)  
23  
24 empty :: Stack a  
25 empty = Nil  
26  
27 push :: a → Stack a → Stack a  
28 push = Cons  
29  
30 pop :: Stack a → (Optional a, Stack a)  
31 pop Nil = (None, Nil)  
32 pop (Cons a s) = (Some a, s)  
33  
34 main :: IO ()  
35 main = do  
-- - . . ~ - - - - - - -
```

<https://godbolt.org/z/a3MWf94n5>

```
-- moon.vcs
29
30 pop :: Stack a → (Optional a, Stack a)
31 pop Nil = (None, Nil)
32 pop (Cons a s) = (Some a, s)
33
34 main :: IO ()
35 main = do
36   let x = Some 3 :: Optional Int
37   let y = None :: Optional Int
38   let z = None :: Optional String
39   let w = Some "Hello, world!" :: Optional String
40
41 let stack = foldl' (flip push) Nil [1..3]
42 forM_ stack \x → do
43   putStrLn (show x)
```

<https://godbolt.org/z/a3MWf94n5>

What about specialisation in Haskell?

- Emulating the specialisability of C++ templates is more difficult
 - Requires leveraging type and data families
- Example at: <https://godbolt.org/z/G3G53YPG3>

C++ facilitates creating ergonomic and *transparently* performant data structures.

Now, what about C++98 template meta-programming?

Probably best left for another talk 😊

C++98 Recap

C++98 Recap

- Functions as first-class citizens

C++98 Recap

- Functions as first-class citizens
- Generic programming with templates

C++98 Recap

- Functions as first-class citizens
- Generic programming with templates
- λ more succinct

C++98 Recap

- Functions as first-class citizens
- Generic programming with templates
- λ more succinct
 - But with more painful specialisation

C++11

Core Functional Features

- Lambdas
- `std::function`
- `std::bind` and partial application
- `auto` type deduction
- More compile-time meta-programming

Core Functional Features

- Lambdas

Lambdas

- Can be easily passed around and bound to names
- Can capture variables from surrounding scope to form true lexical closures

The anatomy of a lambda

```
[captures](parameters){ body }
```

A string/file processing example

```
1 #include <algorithm>
2 #include <cstdlib>
3 #include <iostream>
4 #include <iterator>
5 #include <string>
6 #include <vector>
7
8 int main() {
9     std::vector<std::string> const filenames{
10         "logs.txt", "app.hs", "mock.cpp"
11     };
12     std::vector<std::string> programs;
13     std::copy_if(
14         filenames.cbegin(),
15         filenames.cend(),
16         std::back_inserter(programs))
```

<https://godbolt.org/z/PWz633ezv>

```
9 std::vector<std::string> const filenames{  
10    "logs.txt", "app.hs", "mock.cpp"  
11};  
12 std::vector<std::string> programs;  
13 std::copy_if(  
14    filenames.cbegin(),  
15    filenames.cend(),  
16    std::back_inserter(programs),  
17    [](std::string filename){  
18        return filename.rfind(".txt") == std::string::npos;  
19    }  
20);  
21 for (auto const& program: programs) {  
22     std::cout << program << '\n';  
23 }  
24 return EXIT_SUCCESS;
```

<https://godbolt.org/z/PWz633ezv>

```
10     };
11 
12     std::vector<std::string> programs;
13 
14     std::copy_if(
15         filenames.cbegin(),
16         filenames.cend(),
17         std::back_inserter(programs),
18         [] (std::string filename) {
19             return filename.rfind(".txt") == std::string::npos;
20         }
21     );
22     for (auto const& program: programs) {
23         std::cout << program << '\n';
24     }
25     return EXIT_SUCCESS;
26 }
```

<https://godbolt.org/z/PWz633ezv>

Capturing variables with closures

```
1 #ifndef LIB_HPP_INCLUDED
2 #define LIB_HPP_INCLUDED
3
4 #include <algorithm>
5 #include <iostream>
6 #include <string>
7 #include <vector>
8
9 std::vector<std::string> filter_files(
10     std::vector<std::string> const& filenames
11 ) {
12     std::vector<std::string> desired_files;
13     std::string ignored_file_extension;
14     std::cout << "What file extension would you like to ignore? ";
15     std::cin >> ignored_file_extension;
16 }
```

<https://godbolt.org/z/vK4eGe87Y>

```
6 #include <string>
7 #include <vector>
8
9 std::vector<std::string> filter_files(
10    std::vector<std::string> const& filenames
11 ) {
12    std::vector<std::string> desired_files;
13    std::string ignored_file_extension;
14    std::cout << "What file extension would you like to ignore? ";
15    std::cin >> ignored_file_extension;
16    std::copy_if(
17        filenames.cbegin(),
18        filenames.cend(),
19        std::back_inserter(desired_files),
20        [&ignored_file_extension](std::string const& filename) {
21            return filename.rfind(ignored_file_extension)
```

<https://godbolt.org/z/vK4eGe87Y>

```
--  
13     std::string ignored_file_extension;  
14     std::cout << "What file extension would you like to ignore? ";  
15     std::cin >> ignored_file_extension;  
16     std::copy_if(  
17         filenames.cbegin(),  
18         filenames.cend(),  
19         std::back_inserter(desired_files),  
20         [&ignored_file_extension](std::string const& filename) {  
21             return filename.rfind(ignored_file_extension)  
22                 = std::string::npos;  
23         }  
24     );  
25     return desired_files;  
26 }  
27  
-- ... ...
```

<https://godbolt.org/z/vK4eGe87Y>

```
13     std::string ignored_file_extension;
14     std::cout << "What file extension would you like to ignore? ";
15     std::cin >> ignored_file_extension;
16     std::copy_if(
17         filenames.cbegin(),
18         filenames.cend(),
19         std::back_inserter(desired_files),
20         [&ignored_file_extension](std::string const& filename) {
21             return filename.rfind(ignored_file_extension)
22                 = std::string::npos;
23         }
24     );
25     return desired_files;
26 }
27
28 ...
```

<https://godbolt.org/z/vK4eGe87Y>

```
13 // Asking for file extension...
14 std::cout << "What file extension would you like to ignore? ";
15 std::cin >> ignored_file_extension;
16 std::copy_if(
17     filenames.cbegin(),
18     filenames.cend(),
19     std::back_inserter(desired_files),
20     [&ignored_file_extension](std::string const& filename) {
21         return filename.rfind(ignored_file_extension)
22             == std::string::npos;
23     }
24 );
25 return desired_files;
26 }
27
28 #endif
```

<https://godbolt.org/z/vK4eGe87Y>

```
1 #include "lib.hpp"
2
3 #include <cstdlib>
4 #include <iostream>
5 #include <string>
6 #include <vector>
7
8 int main()
9 {
10     std::vector<std::string> const filenames {
11         "logs.txt", "app.hs", "profiler.cpp"
12     };
13     std::vector<std::string> const programs { filter_files(filenames) };
14     for (auto const& program : programs) {
15         std::cout << program << '\n';
16     }
}
```

<https://godbolt.org/z/vK4eGe87Y>

Core Functional Features

- Lambdas
- `std::function`
- `std::bind` and partial application
- `auto` type deduction
- More compile-time meta-programming

`std::function`

- Lambdas → unique types
- `std::function`: erases concrete type to provide general function signature
- Useful for when a desired function can only be known at *runtime*

```
1 #include <algorithm>
2 #include <cstdlib>
3 #include <functional>
4 #include <iostream>
5 #include <vector>
6
7 enum class Plan {
8     standard,
9     plus,
10    ultra
11 };
12
13 struct Subscriber {
14     Plan plan;
15     bool active_last_month;
16     .
17 }
```

<https://godbolt.org/z/js64rTcdj>

```
18 std::ostream& operator<<(std::ostream& out, Plan plan);
19
20 std::ostream& operator<<(
21     std::ostream& out,
22     Subscriber const& subscriber
23 );
24
25 std::function<bool(Subscriber)> get_qos_filter() {
26     int choice;
27     std::cout << "Do you wish to differentiate by "
28         "ultra membership (1) or by activity (2)?"
29         "\nPlease enter a digit for your choice: ";
30     std::cin >> choice;
31     if (choice == 1) {
32         return [] (Subscriber s) { return s.plan == Plan::ultra; };
33     }
34 }
```

<https://godbolt.org/z/js64rTcdj>

```
21     std::ostream& out,
22     Subscriber const& subscriber
23 );
24
25 std::function<bool(Subscriber)> get_qos_filter() {
26     int choice;
27     std::cout << "Do you wish to differentiate by "
28         "ultra membership (1) or by activity (2)?"
29         "\nPlease enter a digit for your choice: ";
30     std::cin >> choice;
31     if (choice == 1) {
32         return [] (Subscriber s) { return s.plan == Plan::ultra; };
33     } else return &Subscriber::active_last_month;
34 }
35
36 . . . . .
```

<https://godbolt.org/z/js64rTcdj>

```
22 //  
23  
24 std::function<bool(Subscriber)> get_qos_filter() {  
25     int choice;  
26     std::cout << "Do you wish to differentiate by "  
27         "ultra membership (1) or by activity (2)?"  
28         "\nPlease enter a digit for your choice: ";  
29     std::cin >> choice;  
30     if (choice == 1) {  
31         return [](Subscriber s) { return s.plan == Plan::ultra; };  
32     } else return &Subscriber::active_last_month;  
33 }  
34  
35  
36 int main()  
37 {  
38     std::vector<Subscriber> const subscribers{  
39         { "John", "john@example.com", "ultra", "2023-01-01" },  
40         { "Jane", "jane@example.com", "standard", "2023-02-01" },  
41         { "Mike", "mike@example.com", "ultra", "2023-03-01" },  
42         { "Sarah", "sarah@example.com", "standard", "2023-04-01" },  
43         { "David", "david@example.com", "ultra", "2023-05-01" },  
44         { "Emily", "emily@example.com", "standard", "2023-06-01" },  
45         { "Olivia", "olivia@example.com", "ultra", "2023-07-01" },  
46         { "Noah", "noah@example.com", "standard", "2023-08-01" },  
47         { "Ava", "ava@example.com", "ultra", "2023-09-01" },  
48         { "Liam", "liam@example.com", "standard", "2023-10-01" },  
49         { "Mia", "mia@example.com", "ultra", "2023-11-01" },  
50         { "Lucas", "lucas@example.com", "standard", "2023-12-01" }  
51     };  
52     for (const auto& subscriber : subscribers) {  
53         if (get_qos_filter()(subscriber)) {  
54             std::cout << "Subscription accepted for " << subscriber.name << std::endl;  
55         } else {  
56             std::cout << "Subscription rejected for " << subscriber.name << std::endl;  
57         }  
58     }  
59 }
```

<https://godbolt.org/z/js64rTcdj>

```
25 std::function<bool(Subscriber)> get_qos_filter() {
26     int choice;
27     std::cout << "Do you wish to differentiate by "
28         "ultra membership (1) or by activity (2)?"
29         "\nPlease enter a digit for your choice: ";
30     std::cin >> choice;
31     if (choice == 1) {
32         return [] (Subscriber s) { return s.plan == Plan::ultra; };
33     } else return &Subscriber::active_last_month;
34 }
35
36 int main()
37 {
38     std::vector<Subscriber> const subscribers{
39         { Plan::standard, false },
40         { Plan::ultra, true }
```

<https://godbolt.org/z/js64rTcdj>

```
26     int choice;
27     std::cout << "Do you wish to differentiate by "
28         "ultra membership (1) or by activity (2)?"
29         "\nPlease enter a digit for your choice: ";
30     std::cin >> choice;
31     if (choice == 1) {
32         return [] (Subscriber s) { return s.plan == Plan::ultra; };
33     } else return &Subscriber::active_last_month;
34 }
35
36 int main()
37 {
38     std::vector<Subscriber> const subscribers{
39         { Plan::standard, false },
40         { Plan::ultra, true },
41         { Plan::gold, true },
42         { Plan::platinum, true }
```

<https://godbolt.org/z/js64rTcdj>

```
34 }
35
36 int main()
37 {
38     std::vector<Subscriber> const subscribers{
39         { Plan::standard, false },
40         { Plan::ultra, true },
41         { Plan::ultra, false },
42         { Plan::plus, true },
43         { Plan::standard, true }
44     };
45     std::vector<Subscriber> elite_subscribers;
46     std::copy_if(
47         subscribers.cbegin(),
48         subscribers.cend(),
49         elite_subscribers.begin(),
50         &Subscriber::is_elite);
51 }
```

<https://godbolt.org/z/js64rTcdj>

```
38     std::vector<Subscriber> const subscribers{
39         { Plan::standard, false },
40         { Plan::ultra, true },
41         { Plan::ultra, false },
42         { Plan::plus, true },
43         { Plan::standard, true }
44     };
45     std::vector<Subscriber> elite_subscribers;
46     std::copy_if(
47         subscribers.cbegin(),
48         subscribers.cend(),
49         std::back_inserter(elite_subscribers),
50         get_qos_filter()
51     );
52     for (auto subscriber: elite_subscribers) {
53         if (subscriber.get_qos() >= qos)
54             subscribers.push_back(subscriber);
55     }
56 }
```

<https://godbolt.org/z/js64rTcdj>

```
41     { Plan::ultra, false },
42     { Plan::plus, true },
43     { Plan::standard, true }
44 };
45 std::vector<Subscriber> elite_subscribers;
46 std::copy_if(
47     subscribers.cbegin(),
48     subscribers.cend(),
49     std::back_inserter(elite_subscribers),
50     get_qos_filter()
51 );
52 for (auto subscriber: elite_subscribers) {
53     std::cout << subscriber << '\n';
54 }
55 return EXIT_SUCCESS;
56 }
```

<https://godbolt.org/z/js64rTcdj>

```
15     { Plan::elite, true },
16     { Plan::plus, true },
17     { Plan::standard, true }
18 };
19 std::vector<Subscriber> elite_subscribers;
20 std::copy_if(
21     subscribers.cbegin(),
22     subscribers.cend(),
23     std::back_inserter(elite_subscribers),
24     get_qos_filter()
25 );
26 for (auto subscriber: elite_subscribers) {
27     std::cout << subscriber << '\n';
28 }
29 return EXIT_SUCCESS;
30 }
```

<https://godbolt.org/z/js64rTcdj>

Core Functional Features

- Lambdas
- `std::function`
- `std::bind` and partial application
- `auto` type deduction
- More compile-time meta-programming

`std::bind` and partial application

- `std::bind` allows you to create ad-hoc specialised functions from more general ones
- Great for when you know some arguments ahead of time
 - e.g. the `this` argument of a member function
- Can do this so-called *partial application* for as many arguments as needed more generally

An example with machine learning

- Already have a trained model object with a predict method
- Receive a set of new data
- Wish to transform the data to get a set of predictions

```
1 #ifndef PERCEPTRON_HPP_INCLUDED
2 #define PERCEPTRON_HPP_INCLUDED
3
4 #include "Coordinate.hpp"
5
6 #include <algorithm>
7 #include <cstddef>
8 #include <limits>
9 #include <numeric>
10 #include <random>
11
12 class Perceptron {
13     public:
14     Perceptron();
15     void train(
16         std::vector<Coordinate> const& data,
```

<https://godbolt.org/z/644xjh3xq>

<https://godbolt.org/z/644xjh3xq>

```
15 };
16 std::vector<double> const training_labels{
17     1.0, 1.0, -1.0, -1.0
18 };
19 Perceptron model{};
20 model.train(training_data, training_labels);
21 std::vector<Coordinate> const test_data{
22     { 1.8, 3.8 }, { -100.1, -1.0 }, { 20.0, 5.0 }, { -3.0, -15.0 }
23 };
24 std::vector<double> predictions(test_data.size());
25 std::transform(
26     test_data.cbegin(),
27     test_data.cend(),
28     predictions.begin(),
29     std::bind(&Perceptron::predict, &model, _1)
30 );
```

<https://godbolt.org/z/644xjh3xq>

```
20 model.train(training_data, training_labels);
21 std::vector<Coordinate> const test_data{
22     { 1.8, 3.8 }, { -100.1, -1.0 }, { 20.0, 5.0}, { -3.0, -15.0 }
23 };
24 std::vector<double> predictions(test_data.size());
25 std::transform(
26     test_data.cbegin(),
27     test_data.cend(),
28     predictions.begin(),
29     std::bind(&Perceptron::predict, &model, _1)
30 );
31 for (auto prediction: predictions) {
32     std::cout << prediction << '\n';
33 }
34 return EXIT_SUCCESS;
35 }
```

<https://godbolt.org/z/644xjh3xq>

```
--> #include <iostream>
--> #include <vector>
--> #include <array>
--> #include <functional>
--> #include "perceptron.h"
-->
--> int main() {
21     std::vector<Coordinate> const test_data{
22         { 1.8, 3.8 }, { -100.1, -1.0 }, { 20.0, 5.0 }, { -3.0, -15.0 }
23     };
24     std::vector<double> predictions(test_data.size());
25     std::transform(
26         test_data.cbegin(),
27         test_data.cend(),
28         predictions.begin(),
29         std::bind(&Perceptron::predict, &model, _1)
30     );
31     for (auto prediction: predictions) {
32         std::cout << prediction << '\n';
33     }
34     return EXIT_SUCCESS;
35 }
```

<https://godbolt.org/z/644xjh3xq>

Core Functional Features

- Lambdas
- `std::function`
- `std::bind` and partial application
- `auto` type deduction
- More compile-time meta-programming

auto type deduction

- Allow you to bind values to names without having to specify types
 - Can make code more readable and maintainable
- A hallmark of advanced functional languages

```
1 #include <cstdlib>
2
3 int main() {
4     double x = 2.0;
5     auto const x_squared{ x * x };
6     return EXIT_SUCCESS;
7 }
```

Type-inference in Haskell

Core Functional Features

- Lambdas
- `std::function`
- `std::bind` and partial application
- `auto` type deduction
- More compile-time meta-programming

C++11 Template Meta-programming

- Provided better support for primitive "concepts"
 - Supported by C++11 variadic generics, alias templates and the `<type_traits>` header
 - Builds on C++98 SFINAE

Some fun with monoids

What is a monoid?

- A set of elements S equipped with an operation \circ
- $a \circ (b \circ c) = (a \circ b) \circ c$
- An identity element e

Ad-hoc polymorphism in Haskell with typeclasses

```
1 module Main where
2
3 import Data.Foldable (foldl')
4 import Prelude hiding (Monoid, mappend, mconcat, mempty)
5
6 class Monoid a where
7   mempty :: a
8   mappend :: a → a → a
9   mconcat :: [a] → a
10
11   mempty = mconcat []
12   mappend a b = mconcat [a, b]
13   mconcat = foldl' mappend mempty
14
15 {-# MINIMAL (mempty), (mappend) | (mconcat) #-}
16
```

<https://godbolt.org/z/7f3Y89fEn>

```
1 module Main where
2
3 import Data.Foldable (foldl')
4 import Prelude hiding (Monoid, mappend, mconcat, mempty)
5
6 class Monoid a where
7   mempty :: a
8   mappend :: a → a → a
9   mconcat :: [a] → a
10
11   mempty = mconcat []
12   mappend a b = mconcat [a, b]
13   mconcat = foldl' mappend mempty
14
15 {-# MINIMAL (mempty), (mappend) | (mconcat) #-}
16
```

<https://godbolt.org/z/7f3Y89fEn>

```
5
6 class Monoid a where
7   mempty :: a
8   mappend :: a → a → a
9   mconcat :: [a] → a
10
11 mempty = mconcat []
12 mappend a b = mconcat [a, b]
13 mconcat = foldl' mappend mempty
14
15 {-# MINIMAL (mempty), (mappend) | (mconcat) #-}
16
17 data Product a = Product { getProduct :: a }
18 data Sum a = Sum { getSum :: a }
19
20 . . .
```

<https://godbolt.org/z/7f3Y89fEn>


```
10
11 mempty = mconcat []
12 mappend a b = mconcat [a, b]
13 mconcat = foldl' mappend mempty
14
15 {-# MINIMAL (mempty), (mappend) | (mconcat) #-}
16
17 data Product a = Product { getProduct :: a }
18 data Sum a = Sum { getSum :: a }
19
20 instance Num a => Monoid (Product a) where
21     mempty = Product 1
22     mappend (Product a) (Product b)
23         = Product (a * b)
24
25 instance Num a => Monoid (Sum a) where
```

<https://godbolt.org/z/7f3Y89fEn>

```
16
17 data Product a = Product { getProduct :: a }
18 data Sum a = Sum { getSum :: a }
19
20 instance Num a => Monoid (Product a) where
21     mempty = Product 1
22     mappend (Product a) (Product b)
23         = Product (a * b)
24
25 instance Num a => Monoid (Sum a) where
26     mconcat
27         = foldl' (\(Sum a) (Sum b) → Sum (a + b)) (Sum 0)
28
29 main :: IO ()
30 main = do
31     let nums = [1..4]
```

<https://godbolt.org/z/7f3Y89fEn>

```
18 {-# LANGUAGE DataKinds #-} -- DataKinds is required for Product and Sum
19 instance Num a => Monoid (Product a) where
20     mempty = Product 1
21     mappend (Product a) (Product b)
22         = Product (a * b)
23
24 instance Num a => Monoid (Sum a) where
25     mconcat
26         = foldl' (\(Sum a) (Sum b) → Sum (a + b)) (Sum 0)
27
28 main :: IO ()
29 main = do
30     let nums = [1..4]
31     print $ getProduct . mconcat . map Product $ nums -- 24
32     print $ getSum . mconcat . map Sum $ nums -- 10
```

<https://godbolt.org/z/7f3Y89fEn>


```
30 template <typename T> struct Product<
31   T,
32   my_enable_if_t<std::is_arithmetic<my_remove_cvref_t<T>>::value>
33 > {
34   T value;
35   constexpr Product(T value): value { value } { }
36 };
37
38 template <typename T> struct Sum<
39   T,
40   my_enable_if_t<std::is_arithmetic<my_remove_cvref_t<T>>::value>
41 > {
42   T value;
43   constexpr Sum(T value): value { value } { }
44 };

```

<https://godbolt.org/z/E5vsxr9o8>

```
87     >::value
88   >
89 >::mempty { "" };
90
91 template <typename T> struct Monoid<Product<T>> {
92   static Product<T> const mempty;
93   static constexpr Product<T> mappend(
94     Product<T> a,
95     Product<T> b
96   ) { return { a.value * b.value }; }
97 };
98
99 template <typename T> struct Monoid<Sum<T>> {
100   template <
101     typename Collection,
```

<https://godbolt.org/z/E5vsxr9o8>

```
85     my_remove_cvref_t<T>,
86     std::string
87     >::value
88   >
89 >::mempty { "" };
90
91 template <typename T> struct Monoid<Product<T>> {
92   static Product<T> const mempty;
93   static constexpr Product<T> mappend(
94     Product<T> a,
95     Product<T> b
96   ) { return { a.value * b.value }; }
97 };
98
99 template <typename T> struct Monoid<Sum<T>> {
100   static Sum<T> const mempty;
101   static constexpr Sum<T> mappend(
102     Sum<T> a,
103     Sum<T> b
104   ) { return { a.value + b.value }; }
```

<https://godbolt.org/z/E5vsxr9o8>

```
87     >::value
88   >
89 >::mempty { "" };
90
91 template <typename T> struct Monoid<Product<T>> {
92   static Product<T> const mempty;
93   static constexpr Product<T> mappend(
94     Product<T> a,
95     Product<T> b
96   ) { return { a.value * b.value }; }
97 };
98
99 template <typename T> struct Monoid<Sum<T>> {
100   template <
101     typename Collection,
102     typename U = typename Collection::value_type
```

<https://godbolt.org/z/E5vsxr9o8>

```
98  // ...
99  template <typename T> struct Monoid<Sum<T>> {
100    template <
101        typename Collection,
102        typename U = typename Collection::value_type
103    > static constexpr Sum<T> mconcat(Collection&& c) {
104      return std::accumulate(
105          c.cbegin(), c.cend(), Sum<T>{0},
106          [](Sum<T> acc, Sum<T> e) {
107            return Sum<T>(acc.value + e.value);
108          }
109      );
110    }
111  };
112
```

<https://godbolt.org/z/E5vsxr9o8>

```
96     ) { return { a.value * b.value }; }
97 };
98
99 template <typename T> struct Monoid<Sum<T>> {
100     template <
101         typename Collection,
102         typename U = typename Collection::value_type
103     > static constexpr Sum<T> mconcat(Collection&& c) {
104         return std::accumulate(
105             c.cbegin(), c.cend(), Sum<T>{0},
106             [] (Sum<T> acc, Sum<T> e) {
107                 return Sum<T>(acc.value + e.value);
108             }
109         );
110     }
111 }
```

<https://godbolt.org/z/E5vsxr9o8>

```
110    }
111 };
112
113 template <typename T>
114 Product<T> const Monoid<Product<T>>::mempty { 1 };
115
116 template <typename T>
117 constexpr decltype(Monoid<T>::mempty)
118 mempty() { return Monoid<T>::mempty; }
119
120 template <typename T>
121 constexpr decltype(
122     Monoid<T>::template mconcat<std::vector<T>>(std::vector<T>{}))
123 ) mempty() {
124     return Monoid<T>::template mconcat<std::vector<T>>({});
```

<https://godbolt.org/z/E5vsxr9o8>

```
115
116 template <typename T>
117 constexpr decltype(Monoid<T>::mempty)
118 mempty() { return Monoid<T>::mempty; }
119
120 template <typename T>
121 constexpr decltype(
122     Monoid<T>::template mconcat<std::vector<T>>(std::vector<T>{})
123 ) mempty() {
124     return Monoid<T>::template mconcat<std::vector<T>>({});
125 }
126
127 template <typename T, typename U>
128 constexpr auto mappend(T&& a, U&& b) →
129     my_enable_if_t<
130         std::is_same<
```

<https://godbolt.org/z/E5vsxr9o8>

```
180 template <
181     typename Monoid, typename Collection, typename F,
182     typename Element
183         = typename my_remove_cvref_t<Collection>::value_type
184 > constexpr my_enable_if_t<
185     std::is_convertible<
186         F, std::function<Monoid(my_remove_cvref_t<Element>)>
187     >::value, Monoid
188 > foldMap(Collection&& c, F&& f)
189 {
190     return std::accumulate(c.cbegin(), c.cend(), mempty<Monoid>(),
191     [&](Monoid acc, Element e) {
192         return mappend(acc, std::forward<F>(f)(e)); }
193     );
194 }
```

<https://godbolt.org/z/E5vsxr9o8>

```
-->    typename Element
182    = typename my_remove_cvref_t<Collection>::value_type
183 > constexpr my_enable_if_t<
184     std::is_convertible<
185         F, std::function<Monoid(my_remove_cvref_t<Element>)>
186     >::value, Monoid
187 > foldMap(Collection&& c, F&& f)
188 {
189     return std::accumulate(c.cbegin(), c.cend(), mempty<Monoid>(),
190     [&](Monoid acc, Element e) {
191         return mappend(acc, std::forward<F>(f)(e)); }
192     );
193 }
194 }
195
196 #endif
```

<https://godbolt.org/z/E5vsxr9o8>

C++11 Recap

C++11 Recap

- Lambdas

C++11 Recap

- Lambdas
- `std::function`

C++11 Recap

- Lambdas
- std::function
- std::bind and partial application

C++11 Recap

- Lambdas
- `std::function`
- `std::bind` and partial application
- `auto` type deduction

C++11 Recap

- Lambdas
- `std::function`
- `std::bind` and partial application
- `auto` type deduction
- Ad-hoc polymorphism with templates

C++14

Core Functional Features

- auto return type deduction
- auto lambda parameters, i.e. generic lambdas


```
100 template <
101     typename Collection,
102     typename U = typename Collection::value_type
103 > static constexpr Sum<T> mconcat(Collection&& c) {
104     return std::accumulate(
105         c.cbegin(), c.cend(), Sum<T>{0},
106         [] (Sum<T> acc, Sum<T> e) {
107             return Sum<T>(acc.value + e.value);
108         }
109     );
110 }
111 };
112
113 template <typename T>
114 Product<T> const Monoid<Product<T>>::mempty { 1 };
115
```

<https://godbolt.org/z/voo9Wo9Eb>

```
100 template <
101     typename Collection,
102     typename U = typename Collection::value_type
103 > static constexpr Sum<T> mconcat(Collection&& c) {
104     return std::accumulate(
105         c.cbegin(), c.cend(), Sum<T>{0},
106         [] (Sum<T> acc, Sum<T> e) {
107             return Sum<T>(acc.value + e.value);
108         }
109     );
110 }
111 };
112
113 template <typename T>
114 Product<T> const Monoid<Product<T>>::mempty { 1 };
115
```

<https://godbolt.org/z/voo9Wo9Eb>

```
99 template <typename T> struct Monoid<Sum<T>> {
100     template <
101         typename Collection,
102         typename U = typename Collection::value_type
103     > static constexpr Sum<T> mconcat(Collection&& c) {
104         return std::accumulate(
105             c.cbegin(), c.cend(), Sum<T>{0},
106             [](Sum<T> acc, Sum<T> e) {
107                 return Sum<T>(acc.value + e.value);
108             }
109         );
110     }
111 };
112
113 template <typename T>
```

<https://godbolt.org/z/voo9Wo9Eb>


```
158     typename Collection,
159     typename Element =
160         typename my_remove_cvref_t<Collection>::value_type
161 > constexpr decltype(Monoid<Element>::mempty)
162 mconcat(Collection&& c)
163 {
164     return std::accumulate(
165         c.cbegin(), c.cend(), mempty<Element>(),
166         [] (Element acc, Element e) { return mappend(acc, e); }
167     );
168 }
169
170 template <
171     typename Collection,
172     typename Element
173     = typename my_remove_cvref_t<Collection>::value_type
```

<https://godbolt.org/z/voo9Wo9Eb>

```
159     typename Element =  
160         typename my_remove_cvref_t<Collection>::value_type  
161 > constexpr decltype(Monoid<Element>::mempty)  
162 mconcat(Collection&& c)  
163 {  
164     return std::accumulate(  
165         c.cbegin(), c.cend(), mempty<Element>(),  
166         [](Element acc, Element e) { return mappend(acc, e); }  
167     );  
168 }  
169  
170 template <  
171     typename Collection,  
172     typename Element  
173     = typename my_remove_cvref_t<Collection>::value_type  
174     > constexpr decltype(Monoid<Element>::mempty)  
175 mconcat(Collection c)
```

<https://godbolt.org/z/voo9Wo9Eb>


```
160     typename my_remove_cvref_t<Collection>::value_type
161 > constexpr decltype(Monoid<Element>::mempty)
162 mconcat(Collection&& c)
163 {
164     return std::accumulate(
165         c.cbegin(), c.cend(), mempty<Element>(),
166         [](auto&& acc, auto&& e) {
167             return mappend(std::forward<decltype(acc)>(acc),
168                         std::forward<decltype(e)>(e));
169         });
170 }
171
172 template <
173     typename Collection,
174     typename Element
```

<https://godbolt.org/z/68janhx4x>

```
161 > constexpr decltype(Monoid<Element>::mempty)
162 mconcat(Collection&& c)
163 {
164     return std::accumulate(
165         c.cbegin(), c.cend(), mempty<Element>(),
166         [](auto&& acc, auto&& e) {
167             return mappend(std::forward<decltype(acc)>(acc),
168                            std::forward<decltype(e)>(e)); }
169     );
170 }
171
172 template <
173     typename Collection,
174     typename Element
175     = typename my_remove_cvref_t<Collection>::value_type
176 > decltype(mconcat(Collection, Element)) mconcat(Collection, Element)
```

<https://godbolt.org/z/68janhx4x>

C++14 Recap

C++14 Recap

- Basic type inference

C++14 Recap

- Basic type inference
- Polymorphism based on value category

C++14 Recap

- Basic type inference
- Polymorphism based on value category
 - Possible with λ , but requires working with linear types

C++17

Core Functional Features

- Basic destructuring
- Algebraic data types

Core Functional Features

- Structured bindings
- `std::optional`
- `std::variant` and `std::visit`

Core Functional Features

- Structured bindings

```
1 #include <cstdlib>
2 #include <iostream>
3 #include <string>
4 #include <vector>
5
6 struct Subscriber {
7     std::size_t id;
8     std::string name;
9     std::size_t years_as_member;
10 };
11
12 // An imaginary database of subscribers
13 static const std::vector<Subscriber> subscribers{
14     { 0, "Abel", 2 },
15     { 1, "John", 5 }
16 };
```

<https://godbolt.org/z/Y8MjY59Y1>

```
7     std::size_t id;
8     std::string name;
9     std::size_t years_as_member;
10 };
11
12 // An imaginary database of subscribers
13 static const std::vector<Subscriber> subscribers{
14     { 0, "Abel", 2 },
15     { 1, "John", 5 }
16 };
17
18 int main() {
19     for (auto const [<-, name, years_as_member]: subscribers) {
20         std::cout << name << " has been a member for " <<
21         years_as_member << " years.\n";
22 }
```

<https://godbolt.org/z/Y8MjY59Y1>

```
    years_as_member,
10 };
11
12 // An imaginary database of subscribers
13 static const std::vector<Subscriber> subscribers{
14     { 0, "Abel", 2 },
15     { 1, "John", 5 }
16 };
17
18 int main() {
19     for (auto const [_, name, years_as_member]: subscribers) {
20         std::cout << name << " has been a member for " <<
21         years_as_member << " years.\n";
22     }
23     return EXIT_SUCCESS;
24 }
```

<https://godbolt.org/z/Y8MjY59Y1>

```
    years_as_member,
10 };
11
12 // An imaginary database of subscribers
13 static const std::vector<Subscriber> subscribers{
14     { 0, "Abel", 2 },
15     { 1, "John", 5 }
16 };
17
18 int main() {
19     for (auto const [_, name, years_as_member]: subscribers) {
20         std::cout << name << " has been a member for " <<
21         years_as_member << " years.\n";
22     }
23     return EXIT_SUCCESS;
24 }
```

<https://godbolt.org/z/Y8MjY59Y1>

```
    years_as_member,
10 };
11
12 // An imaginary database of subscribers
13 static const std::vector<Subscriber> subscribers{
14     { 0, "Abel", 2 },
15     { 1, "John", 5 }
16 };
17
18 int main() {
19     for (auto const [_, name, years_as_member]: subscribers) {
20         std::cout << name << " has been a member for " <<
21         years_as_member << " years.\n";
22     }
23     return EXIT_SUCCESS;
24 }
```

<https://godbolt.org/z/Y8MjY59Y1>

Core Functional Features

- Structured bindings
- `std::optional`
- `std::variant` and `std::visit`

std::optional

- Gives us the ability to meaningfully compose computations that have *at most* one meaningful result¹
 - Essentially provides us with the Maybe type constructor from Haskell
 - Provides a *value-semantic* alternative to misappropriating pointers, heap allocation and exceptions for working with nullable values
1. At least starting with C++23

```
1 #include <cstdlib>
2 #include <optional>
3 #include <string>
4 #include <string_view>
5
6 struct PhoneNumber {};
7
8 class User {
9     std::size_t age;
10    std::string first_name;
11    std::string last_name;
12    std::string email;
13    // Don't use a pointer and heap
14    // allocation for a nullable member
15    std::optional<PhoneNumber> phone;
16}
```

<https://godbolt.org/z/8KnjszeTK>

```
23     std::optional<PhoneNumber> phone
24     ): age{ age }, first_name{ first_name },
25     last_name{ last_name }, email{ email },
26     phone{ phone } {}
27 };
28
29 // Do not throw if we cannot validate,
30 // just have a failed parse
31 std::optional<PhoneNumber>
32 parse_phone_number(std::string_view phone);
33
34 bool is_valid(std::string_view email);
35
36 std::optional<User> make_user(
37     std::size_t age,
38     std::string_view first_name,
```

<https://godbolt.org/z/8KnjszeTK>

```
--  
29 // Do not throw if we cannot validate,  
30 // just have a failed parse  
31 std::optional<PhoneNumber>  
32 parse_phone_number(std::string_view phone);  
33  
34 bool is_valid(std::string_view email);  
35  
36 std::optional<User> make_user(  
37     std::size_t age,  
38     std::string_view first_name,  
39     std::string_view last_name,  
40     std::string_view email,  
41     std::string_view phone  
42 ) {  
43     if (auto p{ parse_phone_number(phone) };  
..
```

<https://godbolt.org/z/8KnjszeTK>

```
56 std::optional<User> make_user(
57     std::size_t age,
58     std::string_view first_name,
59     std::string_view last_name,
60     std::string_view email,
61     std::string_view phone
62 ) {
63     if (auto p{ parse_phone_number(phone) });
64         age > 18 && is_valid(email) && p)
65     return User(
66         age,
67         first_name,
68         last_name,
69         email,
70         p
71     );

```

<https://godbolt.org/z/8KnjszeTK>

```
--> std::string_view first_name,
39   std::string_view last_name,
40   std::string_view email,
41   std::string_view phone
42 ) {
43   if (auto p{ parse_phone_number(phone) };
44       age > 18 && is_valid(email) && p)
45     return User(
46       age,
47       first_name,
48       last_name,
49       email,
50       p
51     );
52   else return {};
53 }
```

<https://godbolt.org/z/8KnjszeTK>

C++17 Core Functional Features

- Structured bindings
- `std::optional`
- `std::variant` and `std::visit`

- A type-safe generalization of enum / union
- A tool for defining *sum types* from FP languages
- Offers an alternative to rigid inheritance hierarchies

```
1 #include <cstdlib>
2 #include <functional>
3 #include <optional>
4 #include <string>
5 #include <variant>
6
7 class Slime;
8 class Goblin;
9 // Avoid creating an inheritance hierarchy
10 // you will regret later!
11 using Enemy = std::variant<Slime, Goblin>;
12
13 class Slime {
14     int x;
15     int y;
16     ... and ...
17 }
```

<https://godbolt.org/z/ovqxxjY9K>

```
20     Slime(
21         int x, int y, int speed_x, int speed_y,
22         std::size_t hunger
23     ): x{ x }, y{ y }, speed_x{ speed_x },
24       speed_y{ speed_y }, hunger{ hunger } {}
25
26     public:
27     static Enemy make_slime(std::size_t hunger);
28     static Enemy feed(Slime s, std::size_t food);
29     static std::size_t get_hunger(Slime s);
30 };
31
32 class Goblin {
33     int x;
34     int y;
35     . .
36 }
```

<https://godbolt.org/z/ovqxxjY9K>

```
43     std::size_t hunger, bool aggravated,
44     std::size_t level
45 ): x{ x }, y{ y }, speed_x{ speed_x },
46     speed_y{ speed_y }, hunger{ hunger },
47     aggravated{ aggravated }, level{ level } {}
48
49 public:
50     static std::optional<Enemy> make_goblin(std::size_t hunger);
51     static Enemy feed(Goblin g, std::size_t food);
52     static std::size_t get_hunger(Goblin g);
53     static std::size_t get_level(Goblin g);
54 };
55
56 std::string show(Enemy e);
57 Enemy feed(Enemy e, std::size_t food);
```

<https://godbolt.org/z/ovqxxjY9K>


```
--  
66 std::string show(Enemy e) { return std::visit(  
67     overloaded {  
68         [=](Slime s) {  
69             std::ostringstream o;  
70             o << "Slime with hunger " << Slime::get_hunger(s);  
71             return o.str();  
72         },  
73         [=](Goblin g) {  
74             std::ostringstream o;  
75             o << "Goblin at level " << Goblin::get_level(g)  
76                 << " with hunger " << Goblin::get_hunger(g);  
77             return o.str();  
78         }  
79     }, e);  
80 }
```

<https://godbolt.org/z/ovqxxjY9K>

```
--  
66 std::string show(Enemy e) { return std::visit(  
67     overloaded {  
68         [=](Slime s) {  
69             std::ostringstream o;  
70             o << "Slime with hunger " << Slime::get_hunger(s);  
71             return o.str();  
72         },  
73         [=](Goblin g) {  
74             std::ostringstream o;  
75             o << "Goblin at level " << Goblin::get_level(g)  
76                 << " with hunger " << Goblin::get_hunger(g);  
77             return o.str();  
78         }  
79     }, e);  
80 }
```

<https://godbolt.org/z/ovqxxjY9K>

```
--  
63 template <typename... Ts>  
64 overloaded(Ts...) → overloaded<Ts...>;  
65  
66 std::string show(Enemy e) { return std::visit(  
67     overloaded {  
68         [=](Slime s) {  
69             std::ostringstream o;  
70             o << "Slime with hunger " << Slime::get_hunger(s);  
71             return o.str();  
72         },  
73         [=](Goblin g) {  
74             std::ostringstream o;  
75             o << "Goblin at level " << Goblin::get_level(g)  
76             << " with hunger " << Goblin::get_hunger(g);  
77             return o.str();  
78     }  
    ,
```

<https://godbolt.org/z/ovqxxjY9K>

```
68     [=](Slime s) {
69         std::ostringstream o;
70         o << "Slime with hunger " << Slime::get_hunger(s);
71         return o.str();
72     },
73     [=](Goblin g) {
74         std::ostringstream o;
75         o << "Goblin at level " << Goblin::get_level(g)
76             << " with hunger " << Goblin::get_hunger(g);
77         return o.str();
78     }
79 }, e);
80 }
81
82 Enemy feed(Enemy e, std::size_t food)
83 {
```

<https://godbolt.org/z/ovqxxjY9K>


```
1 #include "lib.hpp"
2
3 #include <cstdlib>
4 #include <iostream>
5 #include <optional>
6
7 int main() {
8     Enemy const s{ Slime::make_slime(5) };
9     // Skip optional handling for the sake
10    // of the example
11    Enemy const g{ *Goblin::make_goblin(10) };
12    std::vector<Enemy> const enemies{
13        s, g
14    };
15    for (auto const e: enemies) {
16        std::cout << e << std::endl;
17    }
18}
```

<https://godbolt.org/z/ovqxxjY9K>

```
8  Enemy const s{ Slime::make_slime(5) };
9  // Skip optional handling for the sake
10 // of the example
11 Enemy const g{ *Goblin::make_goblin(10) };
12 std::vector<Enemy> const enemies{
13     s, g
14 };
15 for (auto const e: enemies) {
16     std::cout << show(e) << '\n';
17 }
18 auto const new_enemies{
19     feed_all(enemies, 3)
20 };
21 for (auto const e: new_enemies) {
22     std::cout << show(e) << '\n';
`--
```

<https://godbolt.org/z/ovqxxjY9K>

- Benefits
 - Can make new supertypes by adding types to other `std::variant` aliases
 - Can add new subtypes in similar fashion
 - Refactoring can be guided by resulting compiler errors

C++17 Recap

C++17 Recap

- Basic destructuring

C++17 Recap

- Basic destructuring
- Algebraic data types

C++20

Core Functional Features

- Coroutines
- Concepts
- Ranges

Core Functional Features

- Coroutines
- Concepts
- Ranges

Coroutines

- Facilitate lazy computations
- Similar to one-shot continuations
- See Dr. Ivan Čukić's Prog C++ talk for more!

Core Functional Features

- Coroutines
- Concepts
- Ranges

Concepts

- A powerful mechanism for expressing your *intent* as a programmer
- Think `consteval` type predicate template
- Similar to Haskell's typeclasses for expressing ad-hoc polymorphism

```
1 #ifndef MONOID_HPP_INCLUDED
2 #define MONOID_HPP_INCLUDED
3
4 #include <functional>
5 #include <numeric>
6 #include <sstream>
7 #include <string>
8 #include <type_traits>
9 #include <utility>
10 #include <vector>
11
12 template <typename T>
13 struct Monoid;
14
15 template <typename T>
16 struct Product;
```

<https://godbolt.org/z/oxcav3Mdq>

```
-- -----
19 struct Sum;
20
21 template <typename T>
22 concept arithmetic = requires(T&& a, T&& b) {
23     { std::forward<T>(a) + std::forward<T>(b) }
24         → std::common_reference_with<T>;
25     { std::forward<T>(a) - std::forward<T>(b) }
26         → std::common_reference_with<T>;
27     { std::forward<T>(a) * std::forward<T>(b) }
28         → std::common_reference_with<T>;
29     { std::forward<T>(a) / std::forward<T>(b) }
30         → std::common_reference_with<T>;
31 };
32
33 template <typename T>
-- -----
```

<https://godbolt.org/z/oxcav3Mdq>

```
58 concept product_on = arithmetic<U> &&
59   std::is_same_v<Product<U>, std::remove_cvref_t<T>>;
60
61 template <typename T, typename U>
62 concept sum_on = arithmetic<U>
63   && std::is_same_v<Sum<U>, std::remove_cvref_t<T>>;
64
65 template <arithmetic T>
66 struct Product<T> {
67     T value;
68     template <typename U>
69         requires arithmetic<U> && std::common_reference_with<T, U>
70     constexpr Product(U&& value): value{ std::forward<U>(value) } {}
71     constexpr Product(Product const& other): value{ other.value } {}
72     constexpr Product(Product& other): value{ other.value } {}
73     constexpr Product(Product&& other):
```

<https://godbolt.org/z/oxcav3Mdq>

```
205 {
206     return Monoid<Element>::template mconcat<Collection>(c);
207 }
208
209 template <typename T>
210 concept monoid = requires(T&& a, T&& b) {
211     { mconcat(
212         std::vector<T>{ std::forward<T>(a), std::forward<T>(b) }
213     ) }
214     } → std::common_reference_with<T>;
215 };
216
217 template <typename T>
218 concept monoidal = monoid<std::remove_cvref_t<T>>;
219
220 }
```

<https://godbolt.org/z/oxcav3Mdq>

```
220 template <
221     monoidal Monoid, typename Collection, typename Element
222 = typename std::remove_cvref_t<Collection>::value_type,
223     std::convertible_to<
224         std::function<Monoid(std::remove_cvref_t<Element>)>
225     > F
226 > constexpr Monoid foldMap(Collection&& c, F&& f)
227 {
228     return std::accumulate(c.cbegin(), c.cend(), mempty<Monoid>(),
229     [&]<typename T, typename U>(T&& acc, U&& e) {
230         return mappend(std::forward<T>(acc),
231             std::forward<F>(f)(std::forward<U>(e))
232         );
233     });
234 }
```

<https://godbolt.org/z/oxcav3Mdq>

```
214     } → std::common_reference_with<T>;
215 };
216
217 template <typename T>
218 concept monoidal = monoid<std::remove_cvref_t<T>>;
219
220 template <
221     monoidal Monoid, typename Collection, typename Element
222     = typename std::remove_cvref_t<Collection>::value_type,
223     std::convertible_to<
224         std::function<Monoid(std::remove_cvref_t<Element>)>
225     > F
226 > constexpr Monoid foldMap(Collection&& c, F&& f)
227 {
228     return std::accumulate(c.cbegin(), c.cend(), mempty<Monoid>(),
229     [f](Monoid t, Element e) { return f(t, e); });
230 }
```

<https://godbolt.org/z/oxcav3Mdq>

Core Functional Features

- Coroutines
- Concepts
- Ranges

Ranges

- Unify a iterator-sentinel pair
- Views on them enable *lazy* traversals of data structures
- Can be easily composed in a fluent, functional style

Querying some subscribers

```
1 #include <cstdlib>
2 #include <functional>
3 #include <iostream>
4 #include <iterator>
5 #include <ranges>
6 #include <vector>
7
8 enum class Plan {
9     standard,
10    plus,
11    ultra
12 };
13
14 struct Subscriber {
15     int age;
16     // additional fields omitted
```

<https://godbolt.org/z/vb8MK8x9n>

```
-- 19
20 };
21
22 int main()
23 {
24     std::vector<Subscriber> subscribers {
25         { 25, "See", "Plus", { "romance", "comedy" }, Plan::ultra },
26         { 35, "Hask", "Ell", { "docs", "comedy" }, Plan::plus },
27         { 30, "Cloh", "Jeur", { "docs", "thriller", "feature" },
28             Plan::standard },
29         { 60, "Ski", "Murr", { "romance", "drama" }, Plan::plus }
30     };
31     auto target_subscribers {
32         subscribers
33         | std::views::filter([](auto const& s) { return s.age < 36; })
34         | std::views::filter([](auto const& s) { return s.age > 24; })
-- }
```

<https://godbolt.org/z/vb8MK8x9n>

```
--> 28     Plan::standard },
29     { 60, "Ski", "Murr", { "romance", "drama" }, Plan::plus }
30 };
31 auto target_subscribers {
32     subscribers
33     | std::views::filter([](auto const& s) { return s.age < 36; })
34     | std::views::filter([](auto const& s) { return s.age > 24; })
35     | std::views::filter(
36         [](auto const& s) { return s.plan != Plan::standard; }
37     )
38     | std::views::transform(&Subscriber::first_name)
39 };
40 for (auto const& subscriber: target_subscribers) {
41     std::cout << subscriber << '\n';
42 }
```

--> . FVTT SUCCESS

<https://godbolt.org/z/vb8MK8x9n>

```
    };
}
auto target_subscribers {
    subscribers
    | std::views::filter([](auto const& s) { return s.age < 36; })
    | std::views::filter([](auto const& s) { return s.age > 24; })
    | std::views::filter(
        [](auto const& s) { return s.plan != Plan::standard; })
    )
    | std::views::transform(&Subscriber::first_name)
};
for (auto const& subscriber: target_subscribers) {
    std::cout << subscriber << '\n';
}
return EXIT_SUCCESS;
}
```

<https://godbolt.org/z/vb8MK8x9n>

Is it easier in Haskell?

Yes, because of ordinary functions and function composition!

C++20 Recap

C++20 Recap

- Coroutines

C++20 Recap

- Coroutines
- Concepts

C++20 Recap

- Coroutines
- Concepts
- Ranges

C++23

Core Functional Concepts

- std::expected
- std::optional monadic operations
- More ranges library support
 - std::views::join, std::views::enumerate etc.
 - std::generator

Core Functional Concepts

- std::expected

`std::expected`

- Similar to `std::optional` but with an error tag
- Another great tool for clarifying your intent as a programmer


```
25     = std::variant<division_by_zero<double>, overflow<double>, NaN>;
26
27 constexpr auto divide(double a, double b) →
28     std::expected<double, arithmetic_error>
29 {
30     if (std::isinf(a)) return std::unexpected(overflow{ a });
31     if (std::isinf(b)) return std::unexpected(overflow{ b });
32     if (std::isnan(a) || std::isnan(b)) return std::unexpected(NaN{});
33     if (std::fpclassify(b) == FP_ZERO) return std::unexpected(
34         division_by_zero{ a }
35     );
36     return a / b;
37 }
38
39 constexpr auto square(double a) → double {
40     . . .
41 }
```

<https://godbolt.org/z/Es3sbovYK>

```
25     = std::variant<division_by_zero<double>, overflow<double>, NaN>;
26
27 constexpr auto divide(double a, double b) →
28     std::expected<double, arithmetic_error>
29 {
30     if (std::isinf(a)) return std::unexpected(overflow{ a });
31     if (std::isinf(b)) return std::unexpected(overflow{ b });
32     if (std::isnan(a) || std::isnan(b)) return std::unexpected(NaN{});
33     if (std::fpclassify(b) == FP_ZERO) return std::unexpected(
34         division_by_zero{ a }
35     );
36     return a / b;
37 }
38
39 constexpr auto square(double a) → double {
40     return a * a;
```

<https://godbolt.org/z/Es3sbovYK>


```
-- -----
33     if (std::fpclassify(b) == FP_ZERO) return std::unexpected(
34         division_by_zero{ a }
35     );
36     return a / b;
37 }
38
39 constexpr auto square(double a) → double {
40     return a * a;
41 }
42
43 int main() {
44     if (auto result{ divide(3.0, 0).transform(square) }; result) {
45         std::print("The value is {}.\n", *result);
46     } else {
47         std::print("The result is invalid.");
48     }
}
```

<https://godbolt.org/z/Es3sbovYK>

```
--  ''
36     return a / b;
37 }
38
39 constexpr auto square(double a) → double {
40     return a * a;
41 }
42
43 int main() {
44     if (auto result{ divide(3.0, 0).transform(square) }; result) {
45         std::print("The value is {}.\n", *result);
46     } else {
47         std::print("The result is invalid.");
48     }
49     return EXIT_SUCCESS;
50 }
```

<https://godbolt.org/z/Es3sbovYK>

```
--  ''
36     return a / b;
37 }
38
39 constexpr auto square(double a) → double {
40     return a * a;
41 }
42
43 int main() {
44     if (auto result{ divide(3.0, 0).transform(square) }; result) {
45         std::print("The value is {}.\n", *result);
46     } else {
47         std::print("The result is invalid.");
48     }
49     return EXIT_SUCCESS;
50 }
```

<https://godbolt.org/z/Es3sbovYK>

```
-- 36     return a / b;
37 }
38
39 constexpr auto square(double a) → double {
40     return a * a;
41 }
42
43 int main() {
44     if (auto result{ divide(3.0, 0).transform(square) }; result) {
45         std::print("The value is {}.\n", *result);
46     } else {
47         std::print("The result is invalid.");
48     }
49     return EXIT_SUCCESS;
50 }
```

<https://godbolt.org/z/Es3sbovYK>

```
--  ''
36     return a / b;
37 }
38
39 constexpr auto square(double a) → double {
40     return a * a;
41 }
42
43 int main() {
44     if (auto result{ divide(3.0, 0).transform(square) }; result) {
45         std::print("The value is {}.\n", *result);
46     } else {
47         std::print("The result is invalid.");
48     }
49     return EXIT_SUCCESS;
50 }
```

<https://godbolt.org/z/Es3sbovYK>

Core Functional Concepts

- `std::expected`
- `std::optional` monadic operations
- More ranges library support
 - `std::views::join`, `std::views::enumerate` etc.
 - `std::generator`

`std::optional` monadic operations

- `transform` and `and_then` methods similar to that of `std::expected`

Core Functional Concepts

- std::expected
- std::optional monadic operations
- More ranges library support
 - std::views::join, std::views::enumerate etc.
 - std::generator

```
1 #include "graph.hpp"
2
3 #include <compare>
4 #include <cstdlib>
5 #include <print>
6 #include <ranges>
7 #include <string>
8
9 struct User {
10     std::string first_name;
11     std::string last_name;
12     friend auto operator==(User const&, User const&)
13         → std::strong_ordering = default;
14 };
15
16 // main.cpp
```

<https://godbolt.org/z/bv71onv4z>

```
15
16 int main()
17 {
18     Graph<User> network {};
19     User const u1 { "Ada", "Lovelace" };
20     User const u2 { "Abraham", "Lincoln" };
21     User const u3 { "Nikola", "Tesla" };
22     User const u4 { "Frederick", "Douglass" };
23     User const u5 { "Henry", "Thoreau" };
24     User const u6 { "Srinivasa", "Ramanujan" };
25     User const u7 { "Yogen", "Dalal" };
26     User const u8 { "Bertrand", "Russell" };
27     /*
28      Ada's connected to Abraham and Nikola.
29      Abraham's connected to Frederick and Henry.
30      ...
```

<https://godbolt.org/z/bv71onv4z>

```
30     Nikola's connected to Srinivasa, Bertrand and Yogen.
31     Bertrand follows Ada.
32 */
33 network
34     .add_node(u1)
35     .add_node(u2)
36     .add_node(u3)
37     .add_node(u4)
38     .add_node(u5)
39     .add_node(u6)
40     .add_node(u7)
41     .add_node(u8)
42     .add_edge(u1, u2)
43     .add_edge(u2, u1)
44     .add_edge(u1, u3)
45     .add_edge(u3, u1)
```

<https://godbolt.org/z/bv71onv4z>

```
--  
42     .add_edge(u1, u2)  
43     .add_edge(u2, u1)  
44     .add_edge(u1, u3)  
45     .add_edge(u3, u1)  
46     .add_edge(u2, u4)  
47     .add_edge(u4, u2)  
48     .add_edge(u2, u5)  
49     .add_edge(u5, u2)  
50     .add_edge(u3, u6)  
51     .add_edge(u6, u3)  
52     .add_edge(u3, u7)  
53     .add_edge(u7, u3)  
54     .add_edge(u3, u8)  
55     .add_edge(u8, u3)  
56     .add_edge(u8, u1);  
--
```

<https://godbolt.org/z/bv71onv4z>

```
55     .add_edge(u8, u5)
56     .add_edge(u8, u1);
57 auto not_ada {
58     [&u1](auto node) { return **node != u1; }
59 };
60 // Get Ada's first-degree connections
61 auto firsts {
62     (*network.find_node(u1))→get_neighbours()
63     | std::views::filter(not_ada)
64 };
65 // Ignore duplicates among second-degree
66 // connections for simplicity
67 auto seconds {
68     firsts | std::views::transform([](auto node) {
69         return node→get_neighbours();
70     })
```

<https://godbolt.org/z/bv71onv4z>

```
62     (*network.find_node(u1))→get_neighbours()
63     | std::views::filter(not_ada)
64 };
65 // Ignore duplicates among second-degree
66 // connections for simplicity
67 auto seconds {
68     firsts | std::views::transform([](auto node) {
69         return node→get_neighbours();
70     })
71     | std::views::join | std::views::filter(not_ada)
72 };
73 for (auto user : seconds) {
74     std::print("{}\n", (*user)→first_name);
75 }
76 return EXIT_SUCCESS;
77 }
```

<https://godbolt.org/z/bv71onv4z>

```
62     (*network.find_node(u1))→get_neighbours()
63     | std::views::filter(not_ada)
64 };
65 // Ignore duplicates among second-degree
66 // connections for simplicity
67 auto seconds {
68     firsts | std::views::transform([](auto node) {
69         return node→get_neighbours();
70     })
71     | std::views::join | std::views::filter(not_ada)
72 };
73 for (auto user : seconds) {
74     std::print("{}\n", (*user)→first_name);
75 }
76 return EXIT_SUCCESS;

```

<https://godbolt.org/z/bv71onv4z>

```
52
53     | std::views::filter(not_ada)
54 };
55 // Ignore duplicates among second-degree
56 // connections for simplicity
57 auto seconds {
58     firsts | std::views::transform([](auto node) {
59         return node->get_neighbours();
60     })
61     | std::views::join | std::views::filter(not_ada)
62 };
63 for (auto user : seconds) {
64     std::print("{}\n", (*user)->first_name);
65 }
66 return EXIT_SUCCESS;
67 }
```

<https://godbolt.org/z/bv71onv4z>

```
52
53     | std::views::filter(not_ada)
54 };
55 // Ignore duplicates among second-degree
56 // connections for simplicity
57 auto seconds {
58     firsts | std::views::transform([](auto node) {
59         return node->get_neighbours();
60     })
61     | std::views::join | std::views::filter(not_ada)
62 };
63 for (auto user : seconds) {
64     std::print("{}\n", (*user)->first_name);
65 }
66 return EXIT_SUCCESS;
67 }
```

<https://godbolt.org/z/bv71onv4z>

Refactoring with std::generator


```
20 {  
21     auto not_user {  
22         [&u](auto node) { return **node != u; }  
23     };  
24     for (auto first : (*network.find_node(u))→get_neighbours()  
25             | std::views::filter(not_user)) {  
26         for (auto second : first→get_neighbours()  
27              | std::views::filter(not_user)) {  
28             co_yield *second;  
29         }  
30     }  
31 }  
32  
33 int main()  
34 {  
    ...  
}
```

<https://godbolt.org/z/qEGf9o7o7>

```
17 std::generator<User> second_degree_connections(
18     Graph<User> const& network,
19     User const& u)
20 {
21     auto not_user {
22         [&u](auto node) { return **node != u; }
23     };
24     for (auto first : (*network.find_node(u))→get_neighbours()
25             | std::views::filter(not_user)) {
26         for (auto second : first→get_neighbours()
27               | std::views::filter(not_user)) {
28             co_yield *second;
29         }
30     }
31 }
32 }
```

<https://godbolt.org/z/qEGf9o7o7>

```
19     User const& u)
20 {
21     auto not_user {
22         [&u](auto node) { return **node != u; }
23     };
24     for (auto first : (*network.find_node(u))→get_neighbours()
25           | std::views::filter(not_user)) {
26         for (auto second : first→get_neighbours()
27               | std::views::filter(not_user)) {
28             co_yield *second;
29         }
30     }
31 }
32
33 int main()
34 {
```

<https://godbolt.org/z/qEGf9o7o7>

```
21     auto not_user {
22         [&u](auto node) { return **node != u; }
23     };
24     for (auto first : (*network.find_node(u))→get_neighbours()
25             | std::views::filter(not_user)) {
26         for (auto second : first→get_neighbours()
27             | std::views::filter(not_user)) {
28             co_yield *second;
29         }
30     }
31 }
32
33 int main()
34 {
35     Graph<User> network {};
36     ...
37 }
```

<https://godbolt.org/z/qEGf9o7o7>

```
--  
67     .add_edge(u3, u6)  
68     .add_edge(u6, u3)  
69     .add_edge(u3, u7)  
70     .add_edge(u7, u3)  
71     .add_edge(u3, u8)  
72     .add_edge(u8, u3)  
73     .add_edge(u8, u1);  
74     for (auto const& user : second_degree_connections(  
75         network,  
76         u1  
77     )) {  
78         std::print("{}\n", user.first_name);  
79     }  
80     return EXIT_SUCCESS;  
81 }
```

<https://godbolt.org/z/qEGf9o7o7>

C++23 Recap

C++23 Recap

- std::expected

C++23 Recap

- `std::expected`
- `std::optional` monadic operations

C++23 Recap

- `std::expected`
- `std::optional` monadic operations
- Ranges updates

C++26 and beyond

- More pattern matching / destructuring
- do expressions
- More ranges improvements

Pattern matching / destructuring

- Discussed recently in P2392, P2688, P2940, P2941 and P2761
- Would allow programs to work on the *shape* of data and avail of ADTs' full potential
- Potentially allow fine-grained control of referencing and ownership transfer

do expressions

- Described by P2806
- Eliminates the need for immediately invoked lambda expressions
- Makes C++ into a more *expression-oriented* language

More ranges improvements

- Papers include P2760, P3060 and P1255
- Better encoding the idea of *nullable types as foldable objects*
- Other quality-of-life improvements

A bright future for FP in C++!

I am looking for a job!

Thanks for coming!

Bibliography

1. p0963
2. p2714
3. p2841
4. p2986
5. p2761
6. p2996
7. p2806
8. p1255
9. p2900
10. Passing Overload Sets
11. Functional Data Structures in C++
12. The Design of C++
13. C++ Weekly - Episode 332
14. A History of C++: 1979 – 1991
15. *The Design and Evolution of C++*
16. *Modern C++ Design*
17. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*
18. *Functional Programming in C++*