2023

Boost.MP11

A Christmas Story

Jody Hagins

C++ now

Why This Talk?

C++ Now 2023

Boost.Mp11: A Christmas Story

Learning Enough Functional Metaprogramming to Save Christmas

Jody Hagins
jhagins@maystreet.com
coachhagins@gmail.com

Why Boost.Mp11?

Why Boost.Mp11?

Why Boost.Mp11?

boost C++ LIBRARIES

"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

Boost.Mp11: A C++11 metaprogramming library

Peter Dimov

Overview

Mp11 is a C++11 metaprogramming library for compile-time manipulation of data structures that contain types. It's based on template aliases and variadic templates and implements the approach outlined in the article "Simple C++ metaprogramming" and its sequel. Reading these articles before proceeding with this documentation is *highly* recommended.

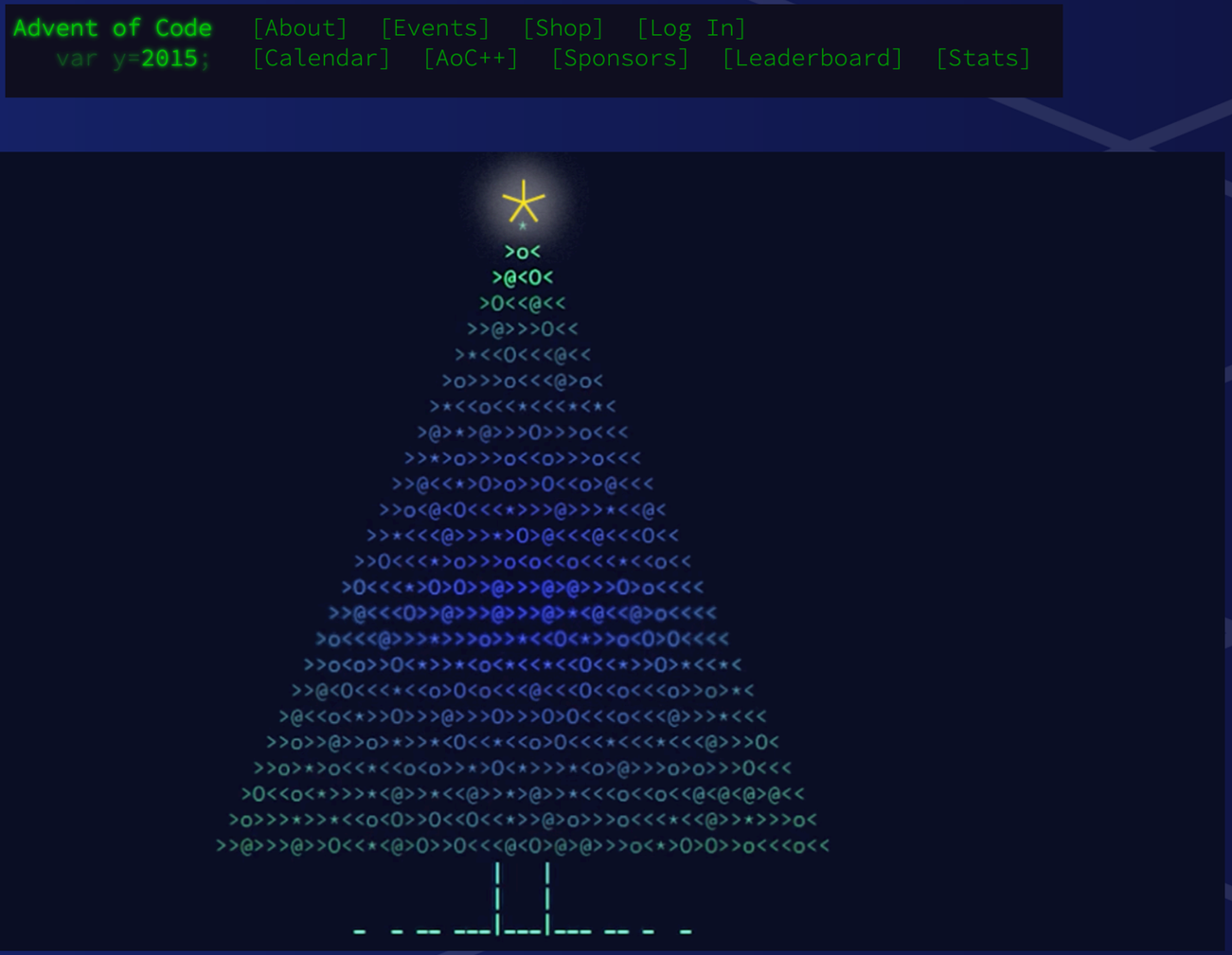
The general principles upon which Mp11 is built are that algorithms and metafunctions are template aliases of the form `F<T…>` and data structures are lists of the form `L<T…>`, with the library placing no requirements on `L`. `mp_list<T…>` is the built-in list type, but `std::tuple<T…>`, `std::pair<T1, T2>` and `std::variant<T…>` are also perfectly legitimate list types, although of course `std::pair<T1, T2>`, due to having exactly two elements, is not resizeable and will consequently not work with algorithms that need to add or remove elements.

Why A Christmas Story?

Why A Christmas Story?

You'll Shoot Your Eye Out!

Why A Christmas Story?



Why This Conference For This Talk?

Why This Conference For This Talk?

Why This Conference For This Talk?

Ground Rules

- Tutorial
  - Solve each problem at compile time
  - Not going for the best or simplest solution
  - Explore the Boost.Mp11 API by example
  - Learn Functional Metaprogramming
  - Save Christmas!
  - Have Fun

Boost.Mp11

https://www.boost.org/doc/libs/1_82_0/libs/mp11/doc/html/mp11.html

Boost.Mp11 : Definitions

- Metafunction
- Type

Boost.Mp11 : Definitions

- Traditional Metafunction

```
using SomePointerType = typename std::add_pointer<SomeType>::type;
```

Boost.Mp11 : Definitions

- Traditional Metafunction

```
using SomePointerType = typename std::add_pointer<SomeType>::type;
```

- MP11 Metafunction

```
using SomePointerType = std::add_pointer_t<SomeType>;
```

Boost.Mp11 : Definitions

- Metafunction
- List
- Quoted Metafunction
- Integral Constant Type
- Set
- Map

Boost.Mp11 : Definitions

- Metafunction
- List
- Quoted Metafunction
- Integral Constant Type
- Set
- Map

Boost.Mp11 : Definitions

- Metafunction
- List
- Quoted Metafunction
- Integral Constant Type
- Set
- Map

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename... T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- Metafunction

```
A metafunction is a class template or alias template whose
parameters are all types, for example

std::add_pointer_t   std::is_const   std::tuple   std::pair
std::shared_ptr      mp_push_front   mp_second    mp_list

template <typename...> using F1 = void;

template <typename T> using F2 = T *;

template <typename...> T>
using F3 = std::integral_constant<std::size_t, sizeof...(T)>;
```

Boost.Mp11 : Definitions

- List

```
A list is a – usually but not necessarily a variadic – class
template whose parameters are all types
```

Boost.Mp11 : Definitions

- List

```
A list is a – usually but not necessarily a variadic – class
template whose parameters are all types, for example

    mp_list<char[], void>
    mp_list<>
    std::tuple<int, float, char>
    std::pair<int, float>
    std::shared_ptr<X>
```

Boost.Mp11 : Definitions

- List

```
A list is a – usually but not necessarily a variadic – class
template whose parameters are all types, for example

    mp_list<char[], void>
    mp_list<>
    std::tuple<int, float, char>
    std::pair<int, float>
    std::shared_ptr<X>
```

Boost.Mp11 : Definitions

- List

```
A list is a – usually but not necessarily a variadic – class
template whose parameters are all types, for example

    mp_list<char[], void>
    mp_list<>
    std::tuple<int, float, char>
    std::pair<int, float>
    std::shared_ptr<X>
```

Boost.Mp11 : Definitions

- Quoted Metafunction

```
A quoted metafunction is a class with a public metafunction
member called fn
```

Boost.Mp11 : Definitions

- Quoted Metafunction

```
A quoted metafunction is a class with a public metafunction
member called fn, for example

struct Q1 { template<typename...> using fn = void; };

struct Q2 { template<typename T> using fn = T*; };

struct Q3 {
    template<typename... T>
    using fn = std::integral_constant<std::size_t, sizeof...(T)>;
};
```

Boost.Mp11 : Definitions

- Quoted Metafunction

```
A quoted metafunction is a class with a public metafunction
member called fn, for example




struct Q4 {
    template <typename... Ts>
    struct fn {
        // ...
    };
};
```

Boost.Mp11 : Definitions

- Integral Constant Type

```
An integral constant type is a class with a public member value
that is an integral constant in the C++ sense.
```

Boost.Mp11 : Definitions

- Integral Constant Type

```
An integral constant type is a class with a public member value
that is an integral constant in the C++ sense. For example

std::integral_constant<int, 7>

struct N { static int constexpr value = 2; };

struct E { enum { value = 42 }; };
```

Boost.Mp11 : Definitions

- Integral Constant Type

```
An integral constant type is a class with a public member value
that is an integral constant in the C++ sense. For example

std::integral_constant<int, 7>

struct N { static int constexpr value = 2; };

struct E { enum { value = 42 }; };
```

Boost.Mp11 : Definitions

- Integral Constant Type

```
An integral constant type is a class with a public member value
that is an integral constant in the C++ sense. For example

std::integral_constant<int, 7>

struct N { static int constexpr value = 2; };

struct E { enum { value = 42 }; };
```

Boost.Mp11 : Definitions

- Set

```
A set is a list whose elements are unique.
```

Boost.Mp11 : Definitions

- Set

```
A set is a list whose elements are unique.

std::tuple<int, char, float, double>
mp_list<char, void, std::tuple<int, float>>
std::string
```

Boost.Mp11 : Definitions

- Set

```
A set is a list whose elements are unique.

std::tuple<int, char, float, double>
mp_list<char, void, std::tuple<int, float>>
std::string
```

Boost.Mp11 : Definitions

- Set

```
A set is a list whose elements are unique.

std::tuple<int, char, float, double>
mp_list<char, void, std::tuple<int, float>>
std::string
```

Boost.Mp11 : Definitions

- Set

```
A set is a list whose elements are unique.

std::tuple<int, char, float, double>
mp_list<char, void, std::tuple<int, float>>
std::basic_string<char>
```

Boost.Mp11 : Definitions

- Set

```
A set is a list whose elements are unique.

std::tuple<int, char, float, double>
mp_list<char, void, std::tuple<int, float>>
std::basic_string<char, std::char_traits<char>, std::allocator<char>>
```

Boost.Mp11 : Definitions

- Map

```
A map is a list of lists, the inner lists having at least one
element (the key). The keys of the map must be unique.
```

Boost.Mp11 : Definitions

- Map

```
A map is a list of lists, the inner lists having at least one
element (the key.) The keys of the map must be unique. For
example

using M1 = std::tuple<
    std::pair<int, int*>,
    std::pair<float, float*>,
    std::pair<void, void*>>;

using M2 = mp_list<
    mp_list<int, int*>,
    mp_list<float>,
    mp_list<char, char[1], char[2]>>;
```

Boost.Mp11 : Definitions

- Map

```
A map is a list of lists, the inner lists having at least one
element (the key.) The keys of the map must be unique. For
example

using M1 = std::tuple<
    std::pair<int, int*>,
    std::pair<float, float*>,
    std::pair<void, void*>>;

using M2 = mp_list<
    mp_list<int, int*>,
    mp_list<float>,
    mp_list<char, char[1], char[2]>>;
```

Boost.Mp11 : Definitions

- Map

```
A map is a list of lists, the inner lists having at least one
element (the key.) The keys of the map must be unique. For
example

using M1 = std::tuple<
    std::pair<int, int*>,
    std::pair<float, float*>,
    std::pair<void, void*>>;

using M2 = mp_list<
    mp_list<int, int*>,
    mp_list<float>,
    mp_list<char, char[1], char[2]>>;
```

Boost.Mp11 : Definitions

- Map

```
A map is a list of lists, the inner lists having at least one
element (the key.) The keys of the map must be unique. For
example

using M1 = std::tuple<
    std::pair<int, int*>,
    std::pair<float, float*>,
    std::pair<void, void*>>;

using M2 = mp_list<
    mp_list<int, int*>,
    mp_list<float>,
    mp_list<char, char[1], char[2]>>;
```

Boost.Mp11 : Definitions

- Map

```
A map is a list of lists, the inner lists having at least one
element (the key.) The keys of the map must be unique. For
example

using M1 = std::tuple<
    std::pair<int, int*>,
    std::pair<float, float*>,
    std::pair<void, void*>>;

using M2 = mp_list<
    mp_list<int, int*>,
    mp_list<float>,
    mp_list<char, char[1], char[2]>>;
```

Boost.Mp11 : Definitions

- Map

```
A map is a list of lists, the inner lists having at least one
element (the key.) The keys of the map must be unique. For
example

using M1 = std::tuple<
    std::pair<int, int*>,
    std::pair<float, float*>,
    std::pair<void, void*>>;

using M2 = mp_list<
    std::pair<int, int*>,
    mp_list<float>,
    mp_list<char, char[1], char[2]>>;
```

Boost.Mp11 : Conventions

Boost.Mp11 : Conventions

boost C++ LIBRARIES

"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

Boost.Mp11: A C++11 metaprogramming library

Peter Dimov

Overview

Boost.Mp11 : Conventions

boost C++ LIBRARIES

"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

Boost.Mp11: A C++11 metaprogramming library

Peter Dimov

Overview

Boost.Mp11 : Conventions

mp_list<T...>

```
template<class... T> struct mp_list {};
```

`mp_list` is the standard list type of Mp11, although the library is not restricted to it and can operate on arbitrary class templates such as `std::tuple` or `std::variant`. Even `std::pair` can be used if the transformation does not alter the number of the elements in the list.

Boost.Mp11 : Conventions

mp_to_bool<T>

```
template<class T> using mp_to_bool = mp_bool<static_cast<bool>(T::value)>;
```

Boost.Mp11 : Conventions

mp_to_bool<T>

```
template<class T> using mp_to_bool = mp_bool<static_cast<bool>(T::value)>;
```

Boost.Mp11 : Conventions

mp_size<L>

```
template<class L> using mp_size = /*...*/;
```

`mp_size<L>` returns the number of elements in the list `L`, as a `mp_size_t`. In other words, `mp_size<L<T…>>` is an alias for `mp_size_t<sizeof…(T)>`.

Boost.Mp11 : Conventions - Pop Quiz

mp_size<L>

```
template<class L> using mp_size = /*...*/;
```

`mp_size<L>` returns the number of elements in the list `L`, as a `mp_size_t`. In other words, `mp_size<L<T…>>` is an alias for `mp_size_t<sizeof…(T)>`.

```
mp_size<mp_list<int, char>>
```

Boost.Mp11 : Conventions - Pop Quiz

mp_size<L>

```
template<class L> using mp_size = /*...*/;
```

`mp_size<L>` returns the number of elements in the list `L`, as a `mp_size_t`. In other words, `mp_size<L<T…>>` is an alias for `mp_size_t<sizeof…(T)>`.

```
mp_size<mp_list<int, char>>                                    mp_size_t<2>
```

Boost.Mp11 : Conventions - Pop Quiz

mp_size<L>

```
template<class L> using mp_size = /*...*/;
```

`mp_size<L>` returns the number of elements in the list `L`, as a `mp_size_t`. In other words, `mp_size<L<T…>>` is an alias for `mp_size_t<sizeof…(T)>`.

```
mp_size<std::string>
```

Boost.Mp11 : Conventions - Pop Quiz

mp_size<L>

```
template<class L> using mp_size = /*...*/;
```

`mp_size<L>` returns the number of elements in the list `L`, as a `mp_size_t`. In other words, `mp_size<L<T…>>` is an alias for `mp_size_t<sizeof…(T)>`.

```
mp_size<std::string>                                    mp_size_t<3>
```

Boost.Mp11 : Conventions - Pop Quiz

mp_size<L>

```
template<class L> using mp_size = /*...*/;
```

`mp_size<L>` returns the number of elements in the list `L`, as a `mp_size_t`. In other words, `mp_size<L<T…>>` is an alias for `mp_size_t<sizeof…(T)>`.

```
mp_size<std::unique_ptr<Foo>>
```

Boost.Mp11 : Conventions - Pop Quiz

mp_size<L>

```
template<class L> using mp_size = /*...*/;
```

`mp_size<L>` returns the number of elements in the list `L`, as a `mp_size_t`. In other words, `mp_size<L<T…>>` is an alias for `mp_size_t<sizeof…(T)>`.

```
mp_size<std::unique_ptr<Foo>>                    mp_size_t<2>
```

Boost.Mp11 : Conventions

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Boost.Mp11 : Conventions

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Boost.Mp11 : Conventions

mp_transform_q<Q, L...>

```
template<class Q, class... L> using mp_transform_q =
    mp_transform<Q::template fn, L...>;
```

As `mp_transform`, but takes a quoted metafunction.

Boost.Mp11 : Conventions

mp_transform_q<Q, L...>

```
template<class Q, class... L> using mp_transform_q =
    mp_transform<Q::template fn, L...>;
```

As `mp_transform`, but takes a quoted metafunction.

Boost.Mp11 : Conventions

mp_transform_q<Q, L...>

```
template<class Q, class... L> using mp_transform_q =
    mp_transform<Q::template fn, L...>;
```

As `mp_transform`, but takes a quoted metafunction.

Boost.Mp11 : Conventions

mp_int<I>

```
template<int I> using mp_int = std::integral_constant<int, I>;
```

Boost.Mp11 : Conventions

mp_at_c<L, I>

```
template<class L, std::size_t I> using mp_at_c = /*...*/;
```

`mp_at_c<L, I>` returns the `I`-th element of `L`, zero-based.

Boost.Mp11 : Conventions

mp_at<L, I>

```
template<class L, class I> using mp_at = /*...*/;
```

Same as `mp_at_c`, but with a type argument `I`. `I::value` must be a nonnegative number.

Boost.Mp11 : Conventions

mp_at<L, I>

```
template<class L, class I> using mp_at = /*...*/;
```

Same as `mp_at_c`, but with a type argument `I`. `I::value` must be a nonnegative number.

Boost.Mp11 : Conventions

mp_iota_c<N>

```
template<std::size_t N> using mp_iota_c = /*...*/;
```

`mp_iota_c<N>` is an alias for `mp_list<mp_size_t<0>, mp_size_t<1>, …, mp_size_t<N-1>>`.

mp_iota<N>

```
template<class N> using mp_iota = /*...*/;
```

Boost.Mp11 : Summary

- Metafunction - class/alias template with all type template parameters - F
- List - instantiated metafunction - L
- Quoted Metafunction - class with nested metafunction fn - Q
- Integral Constant Type - class with nested constant value - I/N
- Set - List with all types unique - S
- Map - List of lists, first type in each list is the key - M

Advent of Code

Santa's reindeer typically eat regular reindeer food, but they need a lot of magical energy to deliver presents on Christmas. For that, their favorite snack is a special type of star fruit that only grows deep in the jungle. The Elves have brought you on their annual expedition to the grove where the fruit grows.

To supply enough magical energy, the expedition needs to retrieve a minimum of fifty stars by December 25th. Although the Elves assure you that the grove has plenty of fruit, you decide to grab any fruit you see along the way, just in case.

Collect stars by solving puzzles. Two puzzles will be made available on each day in the Advent calendar; the second puzzle is unlocked when you complete the first. Each puzzle grants one star. Good luck!

Day 1

The jungle must be too overgrown and difficult to navigate in vehicles or access from the air; the Elves' expedition traditionally goes on foot. As your boats approach land, the Elves begin taking inventory of their supplies. One important consideration is food - in particular, the number of **Calories** each Elf is carrying (your puzzle input).

The Elves take turns writing down the number of Calories contained by the various meals, snacks, rations, etc. that they've brought with them, one item per line. Each Elf separates their own inventory from the previous Elf's inventory (if any) by a blank line.

Day 1

For example, suppose the Elves finish writing their items' Calories and end up with the following list:

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

Day 1

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

This list represents the Calories of the food carried by five Elves:

- The first Elf is carrying food with `1000`, `2000`, and `3000` Calories, a total of `6000` Calories.
- The second Elf is carrying one food item with `4000` Calories.
- The third Elf is carrying food with `5000` and `6000` Calories, a total of `11000` Calories.
- The fourth Elf is carrying food with `7000`, `8000`, and `9000` Calories, a total of `24000` Calories.
- The fifth Elf is carrying one food item with `10000` Calories.

In case the Elves get hungry and need extra snacks, they need to know which Elf to ask: they'd like to know how many Calories are being carried by the Elf carrying the **most** Calories. In the example above, this is `24000` (carried by the fourth Elf).

Find the Elf carrying the most Calories. **How many total Calories is that Elf carrying?**

Day 1

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

This list represents the Calories of the food carried by five Elves:

- The first Elf is carrying food with `1000`, `2000`, and `3000` Calories, a total of `6000` Calories.
- The second Elf is carrying one food item with `4000` Calories.
- The third Elf is carrying food with `5000` and `6000` Calories, a total of `11000` Calories.
- The fourth Elf is carrying food with `7000`, `8000`, and `9000` Calories, a total of `24000` Calories.
- The fifth Elf is carrying one food item with `10000` Calories.

In case the Elves get hungry and need extra snacks, they need to know which Elf to ask: they'd like to know how many Calories are being carried by the Elf carrying the **most** Calories. In the example above, this is `24000` (carried by the fourth Elf).

Find the Elf carrying the most Calories. How many total Calories is that Elf carrying?

Day 1 - Actual Input

```
2228 6014
2229
2230 3531
2231 4133
2232 1549
2233 9146
2234 8227
2235 5186
2236 5159
2237 1952
input.txt
```

Day 1 - "Traditional" Programming

Day 1 - "Traditional" Programming

```
int
day01_part01(std::vector<std::vector<int>> const & input)
{
    int max_total = 0;
    for (auto const & elf : input) {
        int total = 0;
        for (auto item : elf) {
            total += item;
        }
        if (total > max_total) {
            max_total = total;
        }
    }
    return max_total;
}
```

Day 1 - "Traditional" Programming

```
int
day01_part01(std::vector<std::vector<int>> const & input)
{
    int max_total = 0;
    for (auto const & elf : input) {
        int total = 0;
        for (auto item : elf) {
            total += item;
        }
        if (total > max_total) {
            max_total = total;
        }
    }
    return max_total;
}
```

Day 1 - "Traditional" Programming

```
int
day01_part01(std::vector<std::vector<int>> const & input)
{
    int max_total = 0;
    for (auto const & elf : input) {
        int total = 0;
        for (auto item : elf) {
            total += item;
        }
        if (total > max_total) {
            max_total = total;
        }
    }
    return max_total;
}
```

Day 1 - "Traditional" Programming

```
int
day01_part01(std::vector<std::vector<int>> const & input)
{
    int max_total = 0;
    for (auto const & elf : input) {
        int total = 0;
        for (auto item : elf) {
            total += item;
        }
        if (total > max_total) {
            max_total = total;
        }
    }
    return max_total;
}
```

Day 1 - "Traditional" Programming

```
int
day01_part01(std::vector<std::vector<int>> const & input)
{
    int max_total = 0;
    for (auto const & elf : input) {
        int total = 0;
        for (auto item : elf) {
            total += item;
        }
        if (total > max_total) {
            max_total = total;
        }
    }
    return max_total;
}
```

Day 1 - "Traditional" Programming

```
int
day01_part01(std::vector<std::vector<int>> const & input)
{
    int max_total = 0;
    for (auto const & elf : input) {
        int total = 0;
        for (auto item : elf) {
            total += item;
        }
        if (total > max_total) {
            max_total = total;
        }
    }
    return max_total;
}
```

Day 1 - "Traditional" Programming

For example, suppose the Elves finish writing their items' Calories and end up with the following list:

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - Meta-Programming

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 - "Traditional" Metaprogramming

Day 1 - "Traditional" Metaprogramming

```
template <std::size_t V>
using Size = std::integral_constant<std::size_t, V>;

static_assert(std::is_same_v<
    std::integral_constant<std::size_t, 42>,
    Size<42>>);
```

Day 1 - "Traditional" Metaprogramming

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

For example, suppose the Elves finish writing their items' Calories and end up with the following list:

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

6000

4000

11000

24000

10000

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - "Traditional" Metaprogramming

```
1000
2000
3000

4000

5000
6000

7000
8000
9000

10000
```

```
class Day01Part01 {
    template <typename T> struct Exec : Size<0> { };

    template <typename... Ts, typename... Us>
    struct Exec<TypeList<TypeList<Ts...>, Us...>>
    : Size<std::max(Exec<TypeList<Ts...>>::value,
                    Exec<TypeList<Us...>>::value)>
    { };

    template <std::size_t V, typename... Ts>
    struct Exec<TypeList<Size<V>, Ts...>>
    : Size<V + Exec<TypeList<Ts...>>::value>
    { };
public:
    template <typename InputT>
    using fn = typename Exec<InputT>::type;
};
```

Day 1 - Functional Programming

C++ Now 2023

Boost.MP11: A Christmas Story

Enough Functional Metaprogramming to Save Christmas

Jody Hagins
jhagins@maystreet.com
coachhagins@gmail.com

Day 1 - Functional Programming (Soln 1)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    std::vector<int> totals;
    std::transform(
        input.begin(),
        input.end(),
        std::back_inserter(totals),
        [](std::vector<int> const & v) {
            return std::accumulate(v.begin(), v.end(), 0);
        });
    return *std::max_element(totals.begin(), totals.end());
}
```

Day 1 - Functional Programming (Soln 1)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    std::vector<int> totals;
    std::transform(
        input.begin(),
        input.end(),
        std::back_inserter(totals),
        [](std::vector<int> const & v) {
            return std::accumulate(v.begin(), v.end(), 0);
        });
    return *std::max_element(totals.begin(), totals.end());
}
```

Day 1 - Functional Programming (Soln 1)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    std::vector<int> totals;
    std::transform(
        input.begin(),
        input.end(),
        std::back_inserter(totals),
        [](std::vector<int> const & v) {
            return std::accumulate(v.begin(), v.end(), 0);
        });
    return *std::max_element(totals.begin(), totals.end());
}
```

Day 1 - Functional Programming (Soln 1)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    std::vector<int> totals;
    std::transform(
        input.begin(),
        input.end(),
        std::back_inserter(totals),
        [](std::vector<int> const & v) {
            return std::accumulate(v.begin(), v.end(), 0);
        });
    return *std::max_element(totals.begin(), totals.end());
}
```

Day 1 - Functional Programming (Soln 1)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    std::vector<int> totals;
    std::transform(
        input.begin(),
        input.end(),
        std::back_inserter(totals),
        [](std::vector<int> const & v) {
            return std::accumulate(v.begin(), v.end(), 0);
        });
    return *std::max_element(totals.begin(), totals.end());
}
```

Day 1 - Functional Programming (accumulate)

```
inline constexpr auto accumulate = [](
    auto const & collection,
    typename std::decay_t<decltype(collection)>::value_type
        initial = {})
{
    return std::accumulate(
        collection.begin(),
        collection.end(),
        initial);
};
```

Day 1 - Functional Programming (accumulate)

```
inline constexpr auto accumulate = [](
    auto const & collection,
    typename std::decay_t<decltype(collection)>::value_type
        initial = {})
{
    return std::accumulate(
        collection.begin(),
        collection.end(),
        initial);
};
```

Day 1 - Functional Programming (accumulate)

```
inline constexpr auto accumulate = [](
    auto const & collection,
    typename std::decay_t<decltype(collection)>::value_type
        initial = {})
{
    return std::accumulate(
        collection.begin(),
        collection.end(),
        initial);
};
```

Day 1 - Functional Programming (accumulate)

```
inline constexpr auto accumulate = [](
    auto const & collection,
    typename std::decay_t<decltype(collection)>::value_type
        initial = {})
{
    return std::accumulate(
        collection.begin(),
        collection.end(),
        initial);
};
```

Day 1 - Functional Programming (transform)

```
inline constexpr auto transform = [](auto const & collection, auto && f)
{
    using Collection = std::decay_t<decltype(collection)>;
    using F = decltype(f);
    using T = std::invoke_result_t<F, typename Collection::value_type>;
    auto result = mp_assign<Collection, mp_list<T>>{};
    std::transform(
        collection.begin(),
        collection.end(),
        std::back_inserter(result),
        std::forward<F>(f));
    return result;
};
```

Day 1 - Functional Programming (transform)

```
inline constexpr auto transform = [](auto const & collection, auto && f)
{
    using Collection = std::decay_t<decltype(collection)>;
    using F = decltype(f);
    using T = std::invoke_result_t<F, typename Collection::value_type>;
    auto result = mp_assign<Collection, mp_list<T>>{};
    std::transform(
        collection.begin(),
        collection.end(),
        std::back_inserter(result),
        std::forward<F>(f));
    return result;
};
```

Day 1 - Functional Programming (transform)

```
inline constexpr auto transform = [](auto const & collection, auto && f)
{
    using Collection = std::decay_t<decltype(collection)>;
    using F = decltype(f);
    using T = std::invoke_result_t<F, typename Collection::value_type>;
    auto result = mp_assign<Collection, mp_list<T>>{};
    std::transform(
        collection.begin(),
        collection.end(),
        std::back_inserter(result),
        std::forward<F>(f));
    return result;
};
```

Day 1 - Functional Programming (transform)

```
inline constexpr auto transform = [](auto const & collection, auto && f)
{
    using Collection = std::decay_t<decltype(collection)>;
    using F = decltype(f);
    using T = std::invoke_result_t<F, typename Collection::value_type>;
    auto result = mp_assign<Collection, mp_list<T>>{};
    std::transform(
        collection.begin(),
        collection.end(),
        std::back_inserter(result),
        std::forward<F>(f));
    return result;
};
```

Day 1 - Functional Programming (transform)

```
inline constexpr auto transform = [](auto const & collection, auto && f)
{
    using Collection = std::decay_t<decltype(collection)>;
    using F = decltype(f);
    using T = std::invoke_result_t<F, typename Collection::value_type>;
    auto result = mp_assign<Collection, mp_list<T>>{};
    std::transform(
        collection.begin(),
        collection.end(),
        std::back_inserter(result),
        std::forward<F>(f));
    return result;
};
```

Day 1 - Functional Programming (transform)

```
inline constexpr auto transform = [](auto const & collection, auto && f)
{
    using Collection = std::decay_t<decltype(collection)>;
    using F = decltype(f);
    using T = std::invoke_result_t<F, typename Collection::value_type>;
    auto result = mp_assign<Collection, mp_list<T>>{};
    std::transform(
        collection.begin(),
        collection.end(),
        std::back_inserter(result),
        std::forward<F>(f));
    return result;
};
```

Day 1 - Functional Programming (max_element)

```
inline constexpr auto max_element = [](auto const & collection)
{
    assert(collection.begin() != collection.end());
    return *std::max_element(
        collection.begin(),
        collection.end());
};
```

Day 1 - Functional Programming (max_element)

```
inline constexpr auto max_element = [](auto const & collection)
{
    assert(collection.begin() != collection.end());
    return *std::max_element(
        collection.begin(),
        collection.end());
};
```

Day 1 - Functional Programming (max_element)

```
inline constexpr auto max_element = [](auto const & collection)
{
    assert(collection.begin() != collection.end());
    return *std::max_element(
        collection.begin(),
        collection.end());
};
```

Day 1 - Functional Programming (Soln 2)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

Day 1 - Functional Programming (Soln 2)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

Day 1 - Functional Programming (Soln 2)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

Day 1 - Functional Programming (Soln 2)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

Day 1 - Functional Programming (Soln 2)

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

Day 1 - Functional Metaprogramming

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

Day 1 - Functional Metaprogramming

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>>;
};
```

Day 1 - Functional Metaprogramming

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>>;
};
```

Day 1 - Functional Metaprogramming

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>>;
};
```

Day 1 - Functional Metaprogramming

```
int
day01_part01_fn(std::vector<std::vector<int>> const & input)
{
    return max_element(transform(input, accumulate));
}
```

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>, mp_less>;
};
```

Day 1 / Part 1 Breakdown

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>, mp_less>;
};
```

Day 1 / Part 1 Breakdown : mp_max_element

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>, mp_less>;
};
```

Day 1 / Part 1 Breakdown : mp_max_element

mp_max_element<L, P>

```
template<class L, template<class...> class P> using mp_max_element = /*...*/;
```

`mp_max_element<L, P>` returns the maximal element of the list `L` according to the ordering `mp_to_bool<P<T, U>>`.

It's equivalent to `mp_fold<mp_rest<L>, mp_first<L>, F>`, where `F<T, U>` returns `mp_if<P<U, T>, T, U>`.

Day 1 / Part 1 Breakdown : mp_max_element

mp_max_element<L, P>

```
template<class L, template<class...> class P> using mp_max_element = /*...*/;
```

`mp_max_element<L, P>` returns the maximal element of the list `L` according to the ordering `mp_to_bool<P<T, U>>`.

It's equivalent to `mp_fold<mp_rest<L>, mp_first<L>, F>`, where `F<T, U>` returns `mp_if<P<U, T>, T, U>`.

Day 1 / Part 1 Breakdown : mp_max_element

mp_max_element<L, P>

```
template<class L, template<class...> class P> using mp_max_element = /*...*/;
```

`mp_max_element<L, P>` returns the maximal element of the list `L` according to the ordering `mp_to_bool<P<T, U>>`.

It's equivalent to `mp_fold<mp_rest<L>, mp_first<L>, F>`, where `F<T, U>` returns `mp_if<P<U, T>, T, U>`.

Day 1 / Part 1 Breakdown : mp_max_element

mp_max_element<L, P>

```
template<class L, template<class...> class P> using mp_max_element = /*...*/;
```

`mp_max_element<L, P>` returns the maximal element of the list `L` according to the ordering `mp_to_bool<P<T, U>>`.

It's equivalent to `mp_fold<mp_rest<L>, mp_first<L>, F>`, where `F<T, U>` returns `mp_if<P<U, T>, T, U>`.

Day 1 / Part 1 Breakdown : mp_to_bool

mp_to_bool<T>

```
template<class T> using mp_to_bool = mp_bool<static_cast<bool>(T::value)>;
```

Day 1 / Part 1 Breakdown : mp_to_bool

mp_to_bool<T>

```
template<class T> using mp_to_bool = mp_bool<static_cast<bool>(T::value)>;
```

Day 1 / Part 1 Breakdown : mp_bool

mp_bool<B>

```
template<bool B> using mp_bool = std::integral_constant<bool, B>;
```

Day 1 / Part 1 Breakdown : mp_max_element

mp_max_element<L, P>

```
template<class L, template<class...> class P> using mp_max_element = /*...*/;
```

`mp_max_element<L, P>` returns the maximal element of the list `L` according to the ordering `mp_to_bool<P<T, U>>`.

It's equivalent to `mp_fold<mp_rest<L>, mp_first<L>, F>`, where `F<T, U>` returns `mp_if<P<U, T>, T, U>`.

Day 1 / Part 1 Breakdown : mp_less

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>, mp_less>;
};
```

Day 1 / Part 1 Breakdown : mp_less

mp_less<T1, T2>

```
template<class T1, class T2> using mp_less = /*...*/;
```

`mp_less<T1, T2>` is `mp_true` when the numeric value of `T1::value` is less than the numeric value of `T2::value`, `mp_false` otherwise.

(Note that this is not necessarily the same as `T1::value < T2::value` when comparing between signed and unsigned types; `-1 < 1u` is `false`, but `mp_less<mp_int<-1>, mp_size_t<1>>` is `mp_true`.)

Day 1 / Part 1 Breakdown : mp_less

mp_less<T1, T2>

```
template<class T1, class T2> using mp_less = /*...*/;
```

`mp_less<T1, T2>` is `mp_true` when the numeric value of `T1::value` is less than the numeric value of `T2::value`, `mp_false` otherwise.

(Note that this is not necessarily the same as `T1::value < T2::value` when comparing between signed and unsigned types; `-1 < 1u` is `false`, but `mp_less<mp_int<-1>, mp_size_t<1>>` is `mp_true`.)

Day 1 / Part 1 Breakdown : mp_less

mp_less<T1, T2>

```
template<class T1, class T2> using mp_less = /*...*/;
```

`mp_less<T1, T2>` is `mp_true` when the numeric value of `T1::value` is less than the numeric value of `T2::value`, `mp_false` otherwise.

(Note that this is not necessarily the same as `T1::value < T2::value` when comparing between signed and unsigned types; `-1 < 1u` is `false`, but `mp_less<mp_int<-1>, mp_size_t<1>>` is `mp_true`.)

Day 1 / Part 1 Breakdown : mp_less

mp_less<T1, T2>

```
template<class T1, class T2>
using mp_less = mp_bool<(T1::value < 0 && T2::value >= 0) ||
                        ((T1::value < T2::value) &&
                            !(T1::value >= 0 && T2::value < 0))>;
```

(Note: signed and unsigned types; `-1 < 1u` is `false`, but `mp_less<mp_int<-1>, mp_size_t<1>>` is `mp_true`.)

Day 1 / Part 1 Breakdown : mp_transform

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>, mp_less>;
};
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L = std::tuple<void, int, float>;
using R = mp_transform<std::add_pointer_t, L>;
static_assert(std::is_same_v<std::tuple<void*, int*, float*>, R>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L = std::tuple<void, int, float>;
using R = mp_transform<std::add_pointer_t, L>;
static_assert(std::is_same_v<std::tuple<void*, int*, float*>, R>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L = std::tuple<void, int, float>;
using R = mp_transform<std::add_pointer_t, L>;
static_assert(std::is_same_v<std::tuple<void*, int*, float*>, R>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L1 = std::tuple<void, int, float>;
using L2 = mp_list<void, int, double>;
using R = mp_transform<std::is_same, L1, L2>;
static_assert(std::is_same_v<R,
    std::tuple<std::is_same<void, void>,
               std::is_same<int, int>,
               std::is_same<float, double>>>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L1 = std::tuple<void, int, float>;
using L2 = mp_list<void, int, double>;
using R = mp_transform<std::is_same, L1, L2>;
static_assert(std::is_same_v<R,
    std::tuple<std::is_same<void, void>,
               std::is_same<int, int>,
               std::is_same<float, double>>>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L1 = std::tuple<void, int, float>;
using L2 = mp_list<void, int, double>;
using R = mp_transform<std::is_same, L1, L2>;
static_assert(std::is_same_v<R,
    std::tuple<std::is_same<void, void>,
               std::is_same<int, int>,
               std::is_same<float, double>>>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L1 = std::tuple<void, int, float>;
using L2 = mp_list<void, int, double>;
using R = mp_transform<std::is_same, L1, L2>;
static_assert(std::is_same_v<R,
    std::tuple<std::is_same<void, void>,
               std::is_same<int, int>,
               std::is_same<float, double>>>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L1 = std::tuple<void, int, float>;
using L2 = mp_list<void, int, double>;
using R = mp_transform<std::is_same, L1, L2>;
static_assert(std::is_same_v<R,
    std::tuple<std::is_same<void, void>,
               std::is_same<int, int>,
               std::is_same<float, double>>>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L1 = std::tuple<void, int, float>;
using L2 = mp_list<void, int, double>;
using R = mp_transform<std::is_same, L1, L2>;
static_assert(std::is_same_v<R,
    std::tuple<std::is_same<void, void>,
               std::is_same<int, int>,
               std::is_same<float, double>>>);
```

Day 1 / Part 1 Breakdown : mp_transform

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
using L1 = std::tuple<void, int, float>;
using L2 = mp_list<void, int, double>;
using R = mp_transform<std::is_same, L1, L2>;
static_assert(std::is_same_v<R,
    std::tuple<std::is_same<void, void>,
               std::is_same<int, int>,
               std::is_same<float, double>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>, mp_less>;
};
```

Day 1 / Part 1 Breakdown : mp_sum

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>;
```

Day 1 / Part 1 Breakdown : mp_sum

```
template <typename...> struct TypeList { };
using Input = mp_transform<mp_sum, TypeList<
    TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>,
    TypeList<
        std::integral_constant<std::size_t, 4000>>,
    TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>,
    TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>,
    TypeList<
        std::integral_constant<std::size_t, 10000>>>>;
```

Day 1 / Part 1 Breakdown : mp_sum

```
template <typename...> struct TypeList { };
using Input = TypeList<
    mp_sum<TypeList<
        std::integral_constant<std::size_t, 1000>,
        std::integral_constant<std::size_t, 2000>,
        std::integral_constant<std::size_t, 3000>>>,
    mp_sum<TypeList<
        std::integral_constant<std::size_t, 4000>>>,
    mp_sum<TypeList<
        std::integral_constant<std::size_t, 5000>,
        std::integral_constant<std::size_t, 6000>>>,
    mp_sum<TypeList<
        std::integral_constant<std::size_t, 7000>,
        std::integral_constant<std::size_t, 8000>,
        std::integral_constant<std::size_t, 9000>>>,
    mp_sum<TypeList<
        std::integral_constant<std::size_t, 10000>>>>;
```

Day 1 / Part 1 Breakdown : mp_sum

mp_plus<T...>

```
template<class... T> using mp_plus = /*...*/;
```

`mp_plus<T…>` is an integral constant type with a value that is the sum of `U::value` for all types `U` in `T…`. `mp_plus<>` is `mp_int<0>`.

Day 1 / Part 1 Breakdown : mp_sum

mp_plus<T...>

```
template<class... T> using mp_plus = /*...*/;
```

`mp_plus<T…>` is an integral constant type with a value that is the sum of `U::value` for all types `U` in `T…`. `mp_plus<>` is `mp_int<0>`.

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(3 ==
    mp_apply<mp_plus, mp_list<mp_int<1>, mp_int<2>>>::value);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(3 ==
    mp_apply<mp_plus, mp_list<mp_int<1>, mp_int<2>>>::value);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(3 ==
    mp_apply<mp_plus, mp_list<mp_int<1>, mp_int<2>>>::value);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(3 ==
    mp_apply<mp_plus, mp_list<mp_int<1>, mp_int<2>>>::value);

static_assert(3 ==
    mp_plus<mp_int<1>, mp_int<2>>::value);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_apply<mp_plus, mp_list<mp_int<1>, mp_int<2>>>,
    mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_apply<mp_plus, mp_list<mp_int<1>, mp_int<2>>>,
    mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
    mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

mp_apply<F, L>

```
template<template<class...> class F, class L> using mp_apply = mp_rename<L, F>;
```

`mp_apply<F, L>` applies the metafunction `F` to the contents of the list `L`, that is, `mp_apply<F, L<T…>>` is an alias for `F<T…>`. (`mp_apply` is the same as `mp_rename` with the arguments reversed.)

```
static_assert(std::is_same_v<
    mp_rename<mp_list<mp_int<1>, mp_int<2>>, mp_plus>,
              mp_plus<mp_int<1>, mp_int<2>>>);
```

Day 1 / Part 1 Breakdown : mp_sum

```
template <typename L>
using mp_sum = mp_apply<mp_plus, L>;
```

Day 1 / Part 1 Solution

```
struct Day01Part01
{
    template <typename T>
    using fn = mp_max_element<mp_transform<mp_sum, T>, mp_less>;
};
```

Day 1 - Part 2

--- Part Two ---

By the time you calculate the answer to the Elves' question, they've already realized that the Elf carrying the most Calories of food might eventually run out of snacks.

To avoid this unacceptable situation, the Elves would instead like to know the total Calories carried by the top three Elves carrying the most Calories. That way, even if one of those Elves runs out of snacks, they still have two backups.

In the example above, the top three Elves are the fourth Elf (with `24000` Calories), then the third Elf (with `11000` Calories), then the fifth Elf (with `10000` Calories). The sum of the Calories carried by these three elves is `45000`.

Find the top three Elves carrying the most Calories. How many Calories are those Elves carrying in total?

Day 1 - Part 2

--- Part Two ---

By the time you calculate the answer to the Elves' question, they've already realized that the Elf carrying the most Calories of food might eventually run out of snacks.

To avoid this unacceptable situation, the Elves would instead like to know the total Calories carried by the top three Elves carrying the most Calories. That way, even if one of those Elves runs out of snacks, they still have two backups.

In the example above, the top three Elves are the fourth Elf (with `24000` Calories), then the third Elf (with `11000` Calories), then the fifth Elf (with `10000` Calories). The sum of the Calories carried by these three elves is `45000`.

Find the top three Elves carrying the most Calories. How many Calories are those Elves carrying in total?

Day 1 - Functional Programming (sort)

```
inline constexpr auto sort = [](auto collection, auto pred)
{
    std::sort(collection.begin(), collection.end(), pred);
    return collection;
};
```

Day 1 - Functional Programming (sort)

```
inline constexpr auto sort = [](auto collection, auto pred)
{
    std::sort(collection.begin(), collection.end(), pred);
    return collection;
};
```

Day 1 - Functional Programming (sort)

```
inline constexpr auto sort = [](auto collection, auto pred)
{
    std::sort(collection.begin(), collection.end(), pred);
    return collection;
};
```

Day 1 - Functional Programming (sort)

```
inline constexpr auto sort = [](auto collection, auto pred)
{
    std::sort(collection.begin(), collection.end(), pred);
    return collection;
};
```

Day 1 - Functional Programming (take)

```
inline constexpr auto take = [](auto const & collection, std::size_t n)
{
    auto result = std::decay_t<decltype(collection)>{};
    std::copy(
        collection.begin(),
        std::next(collection.begin(), n),
        std::back_inserter(result));
    return result;
};
```

Day 1 - Functional Programming (take)

```
inline constexpr auto take = [](auto const & collection, std::size_t n)
{
    auto result = std::decay_t<decltype(collection)>{};
    std::copy(
        collection.begin(),
        std::next(collection.begin(), n),
        std::back_inserter(result));
    return result;
};
```

Day 1 - Functional Programming (take)

```
inline constexpr auto take = [](auto const & collection, std::size_t n)
{
    auto result = std::decay_t<decltype(collection)>{};
    std::copy(
        collection.begin(),
        std::next(collection.begin(), n),
        std::back_inserter(result));
    return result;
};
```

Day 1 - Functional Programming (take)

```
inline constexpr auto take = [](auto const & collection, std::size_t n)
{
    auto result = std::decay_t<decltype(collection)>{};
    std::copy(
        collection.begin(),
        std::next(collection.begin(), n),
        std::back_inserter(result));
    return result;
};
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                          3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                          3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                          3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 1 Breakdown : mp_max_element

mp_max_element<L, P>

```
template<class L, template<class...> class P> using mp_max_element = /*...*/;
```

`mp_max_element<L, P>` returns the maximal element of the list `L` according to the ordering `mp_to_bool<P<T, U>>`.

It's equivalent to `mp_fold<mp_rest<L>, mp_first<L>, F>`, where `F<T, U>` returns `mp_if<P<U, T>, T, U>`.

Day 1 / Part 1 Breakdown : mp_less

mp_less<T1, T2>

```
template<class T1, class T2> using mp_less = /*...*/;
```

`mp_less<T1, T2>` is `mp_true` when the numeric value of `T1::value` is less than the numeric value of `T2::value`, `mp_false` otherwise.

(Note that this is not necessarily the same as `T1::value < T2::value` when comparing between signed and unsigned types; `-1 < 1u` is `false`, but `mp_less<mp_int<-1>, mp_size_t<1>>` is `mp_true`.)

Day 1 / Part 1 Breakdown : mp_less

mp_less<T1, T2>

```
template<class T1, class T2> using mp_less = /*...*/;
```

`mp_less<T1, T2>` is `mp_true` when the numeric value of `T1::value` is less than the numeric value of `T2::value`, `mp_false` otherwise.

(Note that this is not necessarily the same as `T1::value < T2::value` when comparing between signed and unsigned types; `-1 < 1u` is `false`, but `mp_less<mp_int<-1>, mp_size_t<1>>` is `mp_true`.)

Day 1 / Part 2 - Functional Metaprogramming

mp_sort<L, P>

```
template<class L, template<class...> class P> using mp_sort = /*...*/;
```

`mp_sort<L, P>` sorts the list `L` according to the strict weak ordering `mp_to_bool<P<T, U>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_sort<L, P>

```
template<class L, template<class...> class P> using mp_sort = /*...*/;
```

`mp_sort<L, P>` sorts the list `L` according to the strict weak ordering `mp_to_bool<P<T, U>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_sort<L, P>

```
template<class L, template<class...> class P> using mp_sort = /*...*/;
```

`mp_sort<L, P>` sorts the list `L` according to the strict weak ordering `mp_to_bool<P<T, U>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_sort<L, P>

```
template<class L, template<class...> class P> using mp_sort = /*...*/;
```

`mp_sort<L, P>` sorts the list `L` according to the strict weak ordering `mp_to_bool<P<T, U>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_sort<L, P>

```
template<class L, template<class...> class P> using mp_sort = /*...*/;
```

`mp_sort<L, P>` sorts the list `L` according to the strict weak ordering `mp_to_bool<P<T, U>>`.

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename T1, typename T2>
using mp_greater = mp_less<T2, T1>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

```
struct Day01Part02
{
   template <typename T>
    using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                        mp_greater>,
                                3>>;
};
```

Day 1 / Part 2 - Functional Metaprogramming

mp_take_c<L, N>

```
template<class L, std::size_t N> using mp_take_c = /*...*/;
```

`mp_take_c<L, N>` returns a list of the same form as `L` containing the first `N` elements of `L`.

Day 1 / Part 2 - Functional Metaprogramming

mp_take_c<L, N>

```
template<class L, std::size_t N> using mp_take_c = /*...*/;
```

`mp_take_c<L, N>` returns a list of the same form as `L` containing the first `N` elements of `L`.

Day 1 / Part 2 - Functional Metaprogramming

mp_take_c<L, N>

```
template<class L, std::size_t N> using mp_take_c = /*...*/;
```

`mp_take_c<L, N>` returns a list of the same form as `L` containing the first `N` elements of `L`.

Day 1 / Part 2 - Functional Metaprogramming

mp_take_c<L, N>

```
template<class L, std::size_t N> using mp_take_c = /*...*/;
```

`mp_take_c<L, N>` returns a list of the same form as `L` containing the first `N` elements of `L`.

Day 1 / Part 2 - Functional Metaprogramming

mp_take_c<L, N>

```
template<class L, std::size_t N> using mp_take_c = /*...*/;
```

`mp_take_c<L, N>` returns a list of the same form as `L` containing the first `N` elements of `L`.

Day 1 / Part 2 - Functional Metaprogramming

mp_take_c<L, N>

```
template<class L, std::size_t N> using mp_take_c = /*...*/;
```

`mp_take_c<L, N>` returns a list of the same form as `L` containing the first `N` elements of `L`.

Day 1 / Part 2 - Functional Metaprogramming

mp_take_c<L, N>

```
template<class L, std::size_t N> using mp_take_c = /*...*/;
```

`mp_take_c<L, N>` returns a list of the same form as `L` containing the first `N` elements of `L`.

Day 1 / Part 2 - Functional Metaprogramming

mp_take<L, N>

```
template<class L, class N> using mp_take = /*...*/;
```

Same as `mp_take_c`, but with a type argument `N`. `N::value` must be a nonnegative number.

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(sort(transform(input, accumulate),
                                std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(partial_sort(transform(input, accumulate),
                                        2,
                                        std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

```
int day01_part02_fn(std::vector<std::vector<int>> const & input)
{
    return accumulate(take(nth_element(transform(input, accumulate),
                                       2,
                                       std::greater<>{}),
                           3));
}
```

Day 1 / Part 2 - Functional Metaprogramming

mp_nth_element_c<L, I, P>

```
template<class L, std::size_t I, template<class...> class P> using mp_nth_element_c =
    /*...*/;
```

Returns the element at position `I` in `mp_sort<L, P>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_nth_element_c<L, I, P>

```
template<class L, std::size_t I, template<class...> class P> using mp_nth_element_c =
    /*...*/;
```

Returns the element at position `I` in `mp_sort<L, P>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_nth_element_c<L, I, P>

```
template<class L, std::size_t I, template<class...> class P> using mp_nth_element_c =
    /*...*/;
```

Returns the element at position `I` in `mp_sort<L, P>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_nth_element_c<L, I, P>

```
template<class L, std::size_t I, template<class...> class P> using mp_nth_element_c =
    /*...*/;
```

Returns the element at position `I` in `mp_sort<L, P>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_nth_element_c<L, I, P>

```
template<class L, std::size_t I, template<class...> class P> using mp_nth_element_c =
    /*...*/;
```

Returns the element at position `I` in `mp_sort<L, P>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_nth_element_c<L, I, P>

```
template<class L, std::size_t I, template<class...> class P> using mp_nth_element_c =
    /*...*/;
```

Returns the element at position `I` in `mp_sort<L, P>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_nth_element_c<L, I, P>

```
template<class L, std::size_t I, template<class...> class P> using mp_nth_element_c =
    /*...*/;
```

Returns the element at position `I` in `mp_sort<L, P>`.

Day 1 / Part 2 - Functional Metaprogramming

cppreference.com

Page Discussion

C++ Algorithm library

std::nth_element

`nth_element` is a partial sorting algorithm that rearranges elements in [`first` , `last`) such that:

- The element pointed at by `nth` is changed to whatever element would occur in that position if [`first` , `last`) were sorted.
- All of the elements before this new `nth` element are less than or equal to the elements after the new `nth` element.

Day 1 / Part 2 - Functional Metaprogramming

cppreference.com

Page Discussion

C++ Algorithm library

std::nth_element

`nth_element` is a partial sorting algorithm that rearranges elements in [`first` , `last`) such that:

- The element pointed at by `nth` is changed to whatever element would occur in that position if [`first` , `last`) were sorted.
- All of the elements before this new `nth` element are less than or equal to the elements after the new `nth` element.

Day 1 / Part 2 - Functional Metaprogramming

cppreference.com

Page Discussion

C++ Algorithm library

std::nth_element

`nth_element` is a partial sorting algorithm that rearranges elements in [`first` , `last`) such that:

- The element pointed at by `nth` is changed to whatever element would occur in that position if [`first` , `last`) were sorted.
- All of the elements before this new `nth` element are less than or equal to the elements after the new `nth` element.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_partition<L, P>

```
template<class L, template<class...> class P> using mp_partition = /*...*/;
```

`mp_partition<L<T…>, P>` partitions `L` into two lists `L<U1…>` and `L<U2…>` such that `mp_to_bool<P<T>>` is `mp_true` for the elements of `L<U1…>` and `mp_false` for the elements of `L<U2…>`. Returns `L<L<U1…>, L<U2…>>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind;
```

`mp_bind<F, T…>` is a quoted metafunction that implements the type-based equivalent of `boost::bind`. Its nested template `fn<U…>` returns `F<V…>`, where `V…` is `T…` with the placeholders replaced by the corresponding element of `U…` and the `mp_bind`, `mp_bind_front`, and `mp_bind_back` expressions replaced with their corresponding evaluations against `U…`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind;
```

`mp_bind<F, T…>` is a quoted metafunction that implements the type-based equivalent of `boost::bind`. Its nested template `fn<U…>` returns `F<V…>`, where `V…` is `T…` with the placeholders replaced by the corresponding element of `U…` and the `mp_bind`, `mp_bind_front`, and `mp_bind_back` expressions replaced with their corresponding evaluations against `U…`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind_front<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind_front;
```

`mp_bind_front<F, T…>` binds the leftmost arguments of `F` to `T…`. Its nested template `fn<U…>` returns `F<T…, U…>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind_front<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind_front;
```

`mp_bind_front<F, T…>` binds the leftmost arguments of `F` to `T…`. Its nested template `fn<U…>` returns `F<T…, U…>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind_front<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind_front;
```

`mp_bind_front<F, T…>` binds the leftmost arguments of `F` to `T…`. Its nested template `fn<U…>` returns `F<T…, U…>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind_front<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind_front;
```

`mp_bind_front<F, T…>` binds the leftmost arguments of `F` to `T…`. Its nested template `fn<U…>` returns `F<T…, U…>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind_back<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind_back;
```

`mp_bind_back<F, T…>` binds the rightmost arguments of `F` to `T…`. Its nested template `fn<U…>` returns `F<U…, T…>`.

Day 1 / Part 2 - Functional Metaprogramming

mp_bind_back<F, T...>

```
template<template<class...> class F, class... T> struct mp_bind_back;
```

`mp_bind_back<F, T…>` binds the rightmost arguments of `F` to `T…`. Its nested template `fn<U…>` returns `F<U…, T…>`.

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT>
using fn = mp_sum<
    mp_first<
        mp_partition_q<
            mp_transform<mp_sum, InputT>,
            mp_bind_back<
                mp_greater,
                mp_nth_element_c<
                    mp_transform<mp_sum, InputT>,
                    3,
                    mp_greater>>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT>
using fn = mp_sum<
    mp_first<
        mp_partition_q<
            mp_transform<mp_sum, InputT>,
            mp_bind_back<
                mp_greater,
                mp_nth_element_c<
                    mp_transform<mp_sum, InputT>,
                    3,
                    mp_greater>>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT>
using fn = mp_sum<
    mp_first<
        mp_partition_q<
            mp_transform<mp_sum, InputT>,
            mp_bind_back<
                mp_greater,
                mp_nth_element_c<
                    mp_transform<mp_sum, InputT>,
                    3,
                    mp_greater>>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT>
using fn = mp_sum<
    mp_first<
        mp_partition_q<
            mp_transform<mp_sum, InputT>,
            mp_bind_back<
                mp_greater,
                mp_nth_element_c<
                    mp_transform<mp_sum, InputT>,
                    3,
                    mp_greater>>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT>
using fn = mp_sum<
    mp_first<
        mp_partition_q<
            mp_transform<mp_sum, InputT>,
            mp_bind_back<
                mp_greater,
                mp_nth_element_c<
                    mp_transform<mp_sum, InputT>,
                    3,
                    mp_greater>>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT>
using fn = mp_sum<
    mp_first<
        mp_partition_q<
            mp_transform<mp_sum, InputT>,
            mp_bind_back<
                mp_greater,
                mp_nth_element_c<
                    mp_transform<mp_sum, InputT>,
                    3,
                    mp_greater>>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT>
using fn = mp_sum<
    mp_first<
        mp_partition_q<
            mp_transform<mp_sum, InputT>,
            mp_bind_back<
                mp_greater,
                mp_nth_element_c<
                    mp_transform<mp_sum, InputT>,
                    3,
                    mp_greater>>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

Day 1 / Part 2 - Functional Metaprogramming

```
template <typename InputT,
          typename SumsT = mp_transform<mp_sum, InputT>,
          typename NthT = mp_nth_element_c<SumsT, 3, mp_greater>>
using fn = mp_sum<
    mp_first<mp_partition_q<SumsT, mp_bind_back<mp_greater, NthT>>>>;
```

```
template <typename T>
using fn = mp_sum<mp_take_c<mp_sort<mp_transform<mp_sum, T>,
                                    mp_greater>,
                            3>>;
```

Advent of Code : Day 2

Advent of Code : Day 2

--- Day 2: Rock Paper Scissors ---

The Elves begin to set up camp on the beach. To decide whose tent gets to be closest to the snack storage, a giant Rock Paper Scissors tournament is already in progress.

Rock Paper Scissors is a game between two players. Each game contains many rounds; in each round, the players each simultaneously choose one of Rock, Paper, or Scissors using a hand shape. Then, a winner for that round is selected: Rock defeats Scissors, Scissors defeats Paper, and Paper defeats Rock. If both players choose the same shape, the round instead ends in a draw.

Appreciative of your help yesterday, one Elf gives you an **encrypted strategy guide** (your puzzle input) that they say will be sure to help you win. "The first column is what your opponent is going to play: `A` for Rock, `B` for Paper, and `C` for Scissors. The second column--" Suddenly, the Elf is called away to help with someone's tent.

Advent of Code : Day 2

--- Day 2: Rock Paper Scissors ---

The Elves begin to set up camp on the beach. To decide whose tent gets to be closest to the snack storage, a giant Rock Paper Scissors tournament is already in progress.

Rock Paper Scissors is a game between two players. Each game contains many rounds; in each round, the players each simultaneously choose one of Rock, Paper, or Scissors using a hand shape. Then, a winner for that round is selected: Rock defeats Scissors, Scissors defeats Paper, and Paper defeats Rock. If both players choose the same shape, the round instead ends in a draw.

Appreciative of your help yesterday, one Elf gives you an encrypted strategy guide (your puzzle input) that they say will be sure to help you win. "The first column is what your opponent is going to play: `A` for Rock, `B` for Paper, and `C` for Scissors. The second column--" Suddenly, the Elf is called away to help with someone's tent.

Advent of Code : Day 2

The second column, you reason, must be what you should play in response: `X` for Rock, `Y` for Paper, and `Z` for Scissors. Winning every time would be suspicious, so the responses must have been carefully chosen.

The winner of the whole tournament is the player with the highest score. Your **total score** is the sum of your scores for each round. The score for a single round is the score for the **shape you selected** (1 for Rock, 2 for Paper, and 3 for Scissors) plus the score for the **outcome of the round** (0 if you lost, 3 if the round was a draw, and 6 if you won).

Since you can't be sure if the Elf is trying to help you or trick you, you should calculate the score you would get if you were to follow the strategy guide.

For example, suppose you were given the following strategy guide:

```
A Y
B X
C Z
```

Advent of Code : Day 2

The second column, you reason, must be what you should play in response: X for Rock, Y for Paper, and Z for Scissors. Winning every time would be suspicious, so the responses must have been carefully chosen.

The winner of the whole tournament is the player with the highest score. Your total score is the sum of your scores for each round. The score for a single round is the score for the shape you selected (1 for Rock, 2 for Paper, and 3 for Scissors) plus the score for the outcome of the round (0 if you lost, 3 if the round was a draw, and 6 if you won).

Since you can't be sure if the Elf is trying to help you or trick you, you should calculate the score you would get if you were to follow the strategy guide.

For example, suppose you were given the following strategy guide:

```
A Y
B X
C Z
```

Advent of Code : Day 2

The second column, you reason, must be what you should play in response: `X` for Rock, `Y` for Paper, and `Z` for Scissors. Winning every time would be suspicious, so the responses must have been carefully chosen.

The winner of the whole tournament is the player with the highest score. Your **total score** is the sum of your scores for each round. The score for a single round is the score for the **shape you selected** (1 for Rock, 2 for Paper, and 3 for Scissors) plus the score for the **outcome of the round** (0 if you lost, 3 if the round was a draw, and 6 if you won).

Since you can't be sure if the Elf is trying to help you or trick you, you should calculate the score you would get if you were to follow the strategy guide.

For example, suppose you were given the following strategy guide:

```
A Y
B X
C Z
```

Advent of Code : Day 2

```
A Y
B X
C Z

This strategy guide predicts and recommends the following:

  - In the first round, your opponent will choose Rock (A), and you
    should choose Paper (Y). This ends in a win for you with a score of 8
    (2 because you chose Paper + 6 because you won).
  - In the second round, your opponent will choose Paper (B), and you
    should choose Rock (X). This ends in a loss for you with a score of 1
    (1 + 0).
  - The third round is a draw with both players choosing Scissors, giving
    you a score of 3 + 3 = 6.

In this example, if you were to follow the strategy guide, you would get
a total score of 15 (8 + 1 + 6).

What would your total score be if everything goes exactly according to
your strategy guide?
```

Advent of Code : Day 2

```
A Y
B X
C Z
```

This strategy guide predicts and recommends the following:

- In the first round, your opponent will choose Rock (A), and you should choose Paper (Y). This ends in a win for you with a score of 8 (2 because you chose Paper + 6 because you won).
- In the second round, your opponent will choose Paper (B), and you should choose Rock (X). This ends in a loss for you with a score of 1 (1 + 0).
- The third round is a draw with both players choosing Scissors, giving you a score of 3 + 3 = 6.

In this example, if you were to follow the strategy guide, you would get a total score of 15 (8 + 1 + 6).

What would your total score be if everything goes exactly according to your strategy guide?

Advent of Code : Day 2

```
A Y
B X
C Z
```

This strategy guide predicts and recommends the following:

- In the first round, your opponent will choose Rock (A), and you should choose Paper (Y). This ends in a win for you with a score of 8 (2 because you chose Paper + 6 because you won).
- In the second round, your opponent will choose Paper (B), and you should choose Rock (X). This ends in a loss for you with a score of 1 (1 + 0).
- The third round is a draw with both players choosing Scissors, giving you a score of 3 + 3 = 6.

In this example, if you were to follow the strategy guide, you would get a total score of 15 (8 + 1 + 6).

What would your total score be if everything goes exactly according to your strategy guide?

Advent of Code : Day 2

```
A Y
B X
C Z
```

This strategy guide predicts and recommends the following:

- In the first round, your opponent will choose Rock (A), and you should choose Paper (Y). This ends in a win for you with a score of 8 (2 because you chose Paper + 6 because you won).
- In the second round, your opponent will choose Paper (B), and you should choose Rock (X). This ends in a loss for you with a score of 1 (1 + 0).
- The third round is a draw with both players choosing Scissors, giving you a score of 3 + 3 = 6.

In this example, if you were to follow the strategy guide, you would get a total score of 15 (8 + 1 + 6).

What would your total score be if everything goes exactly according to your strategy guide?

Advent of Code : Day 2

```
A Y
B X
C Z
```

This strategy guide predicts and recommends the following:

- In the first round, your opponent will choose Rock (A), and you should choose Paper (Y). This ends in a win for you with a score of 8 (2 because you chose Paper + 6 because you won).
- In the second round, your opponent will choose Paper (B), and you should choose Rock (X). This ends in a loss for you with a score of 1 (1 + 0).
- The third round is a draw with both players choosing Scissors, giving you a score of 3 + 3 = 6.

In this example, if you were to follow the strategy guide, you would get a total score of 15 (8 + 1 + 6).

What would your total score be if everything goes exactly according to your strategy guide?

Advent of Code : Day 2 : Test Input

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<std::integral_constant<char, 'A'>,  // Their ROCK
             std::integral_constant<char, 'Y'>>, // My PAPER
    TypeList<std::integral_constant<char, 'B'>,  // Their PAPER
             std::integral_constant<char, 'X'>>, // MY ROCK
    TypeList<std::integral_constant<char, 'C'>,  // Their SCISSORS
             std::integral_constant<char, 'Z'>>  // My SCISSORS
    >;
```

Advent of Code : Day 2 : Test Input

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<std::integral_constant<char, 'A'>,  // Their ROCK
             std::integral_constant<char, 'Y'>>, // My PAPER
    TypeList<std::integral_constant<char, 'B'>,  // Their PAPER
             std::integral_constant<char, 'X'>>, // MY ROCK
    TypeList<std::integral_constant<char, 'C'>,  // Their SCISSORS
             std::integral_constant<char, 'Z'>>  // My SCISSORS
    >;
```

Advent of Code : Day 2 : Test Input

```
template <typename...> struct TypeList { };
using Input = TypeList<
    TypeList<std::integral_constant<char, 'A'>,  // Their ROCK
             std::integral_constant<char, 'Y'>>, // My PAPER
    TypeList<std::integral_constant<char, 'B'>,  // Their PAPER
             std::integral_constant<char, 'X'>>, // MY ROCK
    TypeList<std::integral_constant<char, 'C'>,  // Their SCISSORS
             std::integral_constant<char, 'Z'>>  // My SCISSORS
    >;
```

Advent of Code : Day 2

mp_int<I>

```
template<int I> using mp_int = std::integral_constant<int, I>;
```

mp_size_t<N>

```
template<std::size_t N> using mp_size_t = std::integral_constant<std::size_t, N>;
```

Advent of Code : Day 2

mp_int<I>

```
template<int I> using mp_int = std::integral_constant<int, I>;
```

mp_size_t<N>

```
template<std::size_t N> using mp_size_t = std::integral_constant<std::size_t, N>;
```

Advent of Code : Day 2

```
class Day02
{
protected:
    template <typename MineT, typename I>
    using choice_score = mp_plus<mp_int<MineT::value - 'X' + 1>, I>;
    template <typename MineT>
    using lose_score = choice_score<MineT, mp_int<0>>;
    template <typename MineT>
    using draw_score = choice_score<MineT, mp_int<3>>;
    template <typename MineT>
    using win_score = choice_score<MineT, mp_int<6>>;
```

Advent of Code : Day 2

```
class Day02
{
protected:
    template <typename MineT, typename I>
    using choice_score = mp_plus<mp_int<MineT::value - 'X' + 1>, I>;
    template <typename MineT>
    using lose_score = choice_score<MineT, mp_int<0>>;
    template <typename MineT>
    using draw_score = choice_score<MineT, mp_int<3>>;
    template <typename MineT>
    using win_score = choice_score<MineT, mp_int<6>>;
```

Advent of Code : Day 2

```
class Day02
{
protected:
    template <typename MineT, typename I>
    using choice_score = mp_plus<mp_int<MineT::value - 'X' + 1>, I>;
    template <typename MineT>
    using lose_score = choice_score<MineT, mp_int<0>>;
    template <typename MineT>
    using draw_score = choice_score<MineT, mp_int<3>>;
    template <typename MineT>
    using win_score = choice_score<MineT, mp_int<6>>;
```

Advent of Code : Day 2

The second column, you reason, must be what you should play in response: `X` for Rock, `Y` for Paper, and `Z` for Scissors. Winning every time would be suspicious, so the responses must have been carefully chosen.

The winner of the whole tournament is the player with the highest score. Your **total score** is the sum of your scores for each round. The score for a single round is the score for the **shape you selected** (1 for Rock, 2 for Paper, and 3 for Scissors) plus the score for the **outcome of the round** (0 if you lost, 3 if the round was a draw, and 6 if you won).

Since you can't be sure if the Elf is trying to help you or trick you, you should calculate the score you would get if you were to follow the strategy guide.

For example, suppose you were given the following strategy guide:

```
A Y
B X
C Z
```

Advent of Code : Day 2

The second column, you reason, must be what you should play in response: X for Rock, Y for Paper, and Z for Scissors. Winning every time would be suspicious, so the responses must have been carefully chosen.

The winner of the whole tournament is the player with the highest score. Your total score is the sum of your scores for each round. The score for a single round is the score for the shape you selected (1 for Rock, 2 for Paper, and 3 for Scissors) plus the score for the outcome of the round (0 if you lost, 3 if the round was a draw, and 6 if you won).

Since you can't be sure if the Elf is trying to help you or trick you, you should calculate the score you would get if you were to follow the strategy guide.

For example, suppose you were given the following strategy guide:

```
A Y
B X
C Z
```

Advent of Code : Day 2

The second column, you reason, must be what you should play in response: X for Rock, Y for Paper, and Z for Scissors. Winning every time would be suspicious, so the responses must have been carefully chosen.

The winner of the whole tournament is the player with the highest score. Your total score is the sum of your scores for each round. The score for a single round is the score for the shape you selected (1 for Rock, 2 for Paper, and 3 for Scissors) plus the score for the outcome of the round (0 if you lost, 3 if the round was a draw, and 6 if you won).

Since you can't be sure if the Elf is trying to help you or trick you, you should calculate the score you would get if you were to follow the strategy guide.

For example, suppose you were given the following strategy guide:

```
A Y
B X
C Z
```

Advent of Code : Day 2

```
class Day02
{
protected:
    template <typename MineT, typename I>
    using choice_score = mp_plus<mp_int<MineT::value - 'X' + 1>, I>;
    template <typename MineT>
    using lose_score = choice_score<MineT, mp_int<0>>;
    template <typename MineT>
    using draw_score = choice_score<MineT, mp_int<3>>;
    template <typename MineT>
    using win_score = choice_score<MineT, mp_int<6>>;
```

Advent of Code : Day 2

```
class Day02
{
protected:
    template <typename MineT, typename I>
    using choice_score = mp_plus<mp_int<MineT::value - 'X' + 1>, I>;
    template <typename MineT>
    using lose_score = choice_score<MineT, mp_int<0>>;
    template <typename MineT>
    using draw_score = choice_score<MineT, mp_int<3>>;
    template <typename MineT>
    using win_score = choice_score<MineT, mp_int<6>>;
```

Advent of Code : Day 2

```
class Day02
{
protected:
    template <typename MineT, typename I>
    using choice_score = mp_plus<mp_int<MineT::value - 'X' + 1>, I>;
    template <typename MineT>
    using lose_score = choice_score<MineT, mp_int<0>>;
    template <typename MineT>
    using draw_score = choice_score<MineT, mp_int<3>>;
    template <typename MineT>
    using win_score = choice_score<MineT, mp_int<6>>;
```

Advent of Code : Day 2

```
class Day02
{
protected:
    template <typename MineT, typename I>
    using choice_score = mp_plus<mp_int<MineT::value - 'X' + 1>, I>;
    template <typename MineT>
    using lose_score = choice_score<MineT, mp_int<0>>;
    template <typename MineT>
    using draw_score = choice_score<MineT, mp_int<3>>;
    template <typename MineT>
    using win_score = choice_score<MineT, mp_int<6>>;
```

Advent of Code : Day 2

```
using TheirRock = mp_constant<'A'>;
using TheirPaper = mp_constant<'B'>;
using TheirScissors = mp_constant<'C'>;
using MyRock = mp_constant<'X'>;
using MyPaper = mp_constant<'Y'>;
using MyScissors = mp_constant<'Z'>;
```

Advent of Code : Day 2

```
using TheirRock = mp_constant<'A'>;
using TheirPaper = mp_constant<'B'>;
using TheirScissors = mp_constant<'C'>;
using MyRock = mp_constant<'X'>;
using MyPaper = mp_constant<'Y'>;
using MyScissors = mp_constant<'Z'>;
```

Advent of Code : Day 2

```
    using TheirRock = mp_constant<'A'>;
    using TheirPaper = mp_constant<'B'>;
    using TheirScissors = mp_constant<'C'>;
    using MyRock = mp_constant<'X'>;
    using MyPaper = mp_constant<'Y'>;
    using MyScissors = mp_constant<'Z'>;
```

```
template <auto I>
using mp_constant = std::integral_constant<decltype(I), I>;
```

Advent of Code : Day 2

```
using TheirRock = mp_constant<'A'>;
using TheirPaper = mp_constant<'B'>;
using TheirScissors = mp_constant<'C'>;
using MyRock = mp_constant<'X'>;
using MyPaper = mp_constant<'Y'>;
using MyScissors = mp_constant<'Z'>;
```

```
template <auto I>
using mp_constant = std::integral_constant<decltype(I), I>;
```

Advent of Code : Day 2

```
    using TheirRock = mp_constant<'A'>;
    using TheirPaper = mp_constant<'B'>;
    using TheirScissors = mp_constant<'C'>;
    using MyRock = mp_constant<'X'>;
    using MyPaper = mp_constant<'Y'>;
    using MyScissors = mp_constant<'Z'>;
```

```
template <auto I>
using mp_constant = std::integral_constant<decltype(I), I>;
```

Advent of Code : Day 2

```
    using TheirRock = mp_constant<'A'>;
    using TheirPaper = mp_constant<'B'>;
    using TheirScissors = mp_constant<'C'>;
    using MyRock = mp_constant<'X'>;
    using MyPaper = mp_constant<'Y'>;
    using MyScissors = mp_constant<'Z'>;
```

```
template <auto I>
using mp_constant = std::integral_constant<decltype(I), I>;
```

Advent of Code : Day 2

```
template <typename TheirsT, typename MineT>
struct Score : draw_score<MineT> { };
template <> struct Score<TheirRock, MyPaper>
: win_score<MyPaper> { };
template <> struct Score<TheirRock, MyScissors>
: lose_score<MyScissors> { };
template <> struct Score<TheirPaper, MyRock>
: lose_score<MyRock> { };
template <> struct Score<TheirPaper, MyScissors>
: win_score<MyScissors> { };
template <> struct Score<TheirScissors, MyRock>
: win_score<MyRock> { };
template <> struct Score<TheirScissors, MyPaper>
: lose_score<MyPaper> { };
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

mp_cond<C, T, R...>

```
template<class C, class T, class... R> using mp_cond = /*...*/;
```

`mp_cond<C, T, R…>` is an alias for `T` when `static_cast<bool>(C::value)` is `true`. When `static_cast<bool>(C::value)` is `false`, it's an alias for `mp_cond<R…>`.

(If `static_cast<bool>(C::value)` is a substitution failure, the result is too a substitution failure.)

```
template<int N> using unsigned_ = mp_cond<
    mp_bool<N ==  8>, uint8_t,
    mp_bool<N == 16>, uint16_t,
    mp_bool<N == 32>, uint32_t,
    mp_bool<N == 64>, uint64_t,
    mp_true, unsigned // default case
>;
```

Advent of Code : Day 2

```
template <typename ThemT, typename MeT>
using score = mp_cond<
  mp_all<mp_same<ThemT, TheirRock>,  mp_same<MeT, MyPaper>>, win<MeT>,
  mp_all<mp_same<ThemT, TheirRock>,  mp_same<MeT, MyScsrs>>, lose<MeT>,
  mp_all<mp_same<ThemT, TheirPaper>, mp_same<MeT, MyRock>>,  lose<MeT>,
  mp_all<mp_same<ThemT, TheirPaper>, mp_same<MeT, MyScsrs>>, win<MeT>,
  mp_all<mp_same<ThemT, TheirScsrs>, mp_same<MeT, MyRock>>,  win<MeT>,
  mp_all<mp_same<ThemT, TheirScsrs>, mp_same<MeT, MyPaper>>, lose<MeT>,
  mp_true, draw<MeT>>;
```

Advent of Code : Day 2

```
using outcomes = mp_list<
    mp_list<mp_list<TheirRock, MyPaper>,      win_score<MyPaper>>,
    mp_list<mp_list<TheirRock, MyScissors>,   lose_score<MyScissors>>,
    mp_list<mp_list<TheirPaper, MyRock>,      lose_score<MyRock>>,
    mp_list<mp_list<TheirPaper, MyScissors>,  win_score<MyScissors>>,
    mp_list<mp_list<TheirScissors, MyRock>,   win_score<MyRock>>,
    mp_list<mp_list<TheirScissors, MyPaper>,  lose_score<MyPaper>>>;
```

Advent of Code : Day 2

```
using outcomes = mp_list<
    mp_list<mp_list<TheirRock, MyPaper>,       win_score<MyPaper>>,
    mp_list<mp_list<TheirRock, MyScissors>,    lose_score<MyScissors>>,
    mp_list<mp_list<TheirPaper, MyRock>,       lose_score<MyRock>>,
    mp_list<mp_list<TheirPaper, MyScissors>,   win_score<MyScissors>>,
    mp_list<mp_list<TheirScissors, MyRock>,    win_score<MyRock>>,
    mp_list<mp_list<TheirScissors, MyPaper>,   lose_score<MyPaper>>>;
```

Advent of Code : Day 2

```
using outcomes = mp_list<
    mp_list<mp_list<TheirRock, MyPaper>,        win_score<MyPaper>>,
    mp_list<mp_list<TheirRock, MyScissors>,     lose_score<MyScissors>>,
    mp_list<mp_list<TheirPaper, MyRock>,        lose_score<MyRock>>,
    mp_list<mp_list<TheirPaper, MyScissors>,    win_score<MyScissors>>,
    mp_list<mp_list<TheirScissors, MyRock>,     win_score<MyRock>>,
    mp_list<mp_list<TheirScissors, MyPaper>,    lose_score<MyPaper>>>;
```

Advent of Code : Day 2

```
using outcomes = mp_list<
    mp_list<mp_list<TheirRock, MyPaper>,        win_score<MyPaper>>,
    mp_list<mp_list<TheirRock, MyScissors>,     lose_score<MyScissors>>,
    mp_list<mp_list<TheirPaper, MyRock>,        lose_score<MyRock>>,
    mp_list<mp_list<TheirPaper, MyScissors>,    win_score<MyScissors>>,
    mp_list<mp_list<TheirScissors, MyRock>,     win_score<MyRock>>,
    mp_list<mp_list<TheirScissors, MyPaper>,    lose_score<MyPaper>>>;
```

Advent of Code : Day 2

Map Operations, <boost/mp11/map.hpp>

A map is a list of lists, the inner lists having at least one element (the key.) The keys of the map must be unique.

Advent of Code : Day 2

Map Operations, <boost/mp11/map.hpp>

A map is a list of lists, the inner lists having at least one element (the key.) The keys of the map must be unique.

```
using outcomes = mp_list<
    mp_list<mp_list<TheirRock, MyPaper>,      win_score<MyPaper>>,
    mp_list<mp_list<TheirRock, MyScissors>,   lose_score<MyScissors>>,
    mp_list<mp_list<TheirPaper, MyRock>,      lose_score<MyRock>>,
    mp_list<mp_list<TheirPaper, MyScissors>,  win_score<MyScissors>>,
    mp_list<mp_list<TheirScissors, MyRock>,   win_score<MyRock>>,
    mp_list<mp_list<TheirScissors, MyPaper>,  lose_score<MyPaper>>>;
```

Advent of Code : Day 2

Map Operations, <boost/mp11/map.hpp>

A map is a list of lists, the inner lists having at least one element (the key.) The keys of the map must be unique.

```
using outcomes = mp_list<
    mp_list<mp_list<TheirRock, MyPaper>,      win_score<MyPaper>>,
    mp_list<mp_list<TheirRock, MyScissors>,   lose_score<MyScissors>>,
    mp_list<mp_list<TheirPaper, MyRock>,      lose_score<MyRock>>,
    mp_list<mp_list<TheirPaper, MyScissors>,  win_score<MyScissors>>,
    mp_list<mp_list<TheirScissors, MyRock>,   win_score<MyRock>>,
    mp_list<mp_list<TheirScissors, MyPaper>,  lose_score<MyPaper>>>;
```

Advent of Code : Day 2

mp_map_find<M, K>

```
template<class M, class K> using mp_map_find = /*...*/;
```

`mp_map_find<M, K>` is an alias for the element of the map `M` with a key `K`, or for `void`, if there is no such element.

Advent of Code : Day 2

mp_map_find<M, K>

```
template<class M, class K> using mp_map_find = /*...*/;
```

`mp_map_find<M, K>` is an alias for the element of the map `M` with a key `K`, or for `void`, if there is no such element.

Advent of Code : Day 2

mp_map_find<M, K>

```
template<class M, class K> using mp_map_find = /*...*/;
```

`mp_map_find<M, K>` is an alias for the element of the map `M` with a key `K`, or for `void`, if there is no such element.

Advent of Code : Day 2

mp_map_find<M, K>

```
template<class M, class K> using mp_map_find = /*...*/;
```

`mp_map_find<M, K>` is an alias for the element of the map `M` with a key `K`, or for `void`, if there is no such element.

Advent of Code : Day 2

mp_map_find<M, K>

```
template<class M, class K> using mp_map_find = /*...*/;
```

`mp_map_find<M, K>` is an alias for the element of the map `M` with a key `K`, or for `void`, if there is no such element.

Advent of Code : Day 2

```
template <typename TheirsT,
          typename MineT,
          template <typename...> class F>
using outcome = mp_list<mp_list<TheirsT, MineT>, F<MineT>>;
```

Advent of Code : Day 2

```
template <typename TheirsT,
          typename MineT,
          template <typename...> class F>
using outcome = mp_list<mp_list<TheirsT, MineT>, F<MineT>>;
```

```
using outcomes = mp_list<
    outcome<TheirRock, MyPaper, win_score>,
    outcome<TheirRock, MyScissors, lose_score>,
    outcome<TheirPaper, MyRock, lose_score>,
    outcome<TheirPaper, MyScissors, win_score>,
    outcome<TheirScissors, MyRock, win_score>,
    outcome<TheirScissors, MyPaper, lose_score>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

mp_eval_or<T, F, U...>

```
template<class T, template<class...> class F, class... U> using mp_eval_or =
    mp_eval_if_not<mp_valid<F, U...>, T, F, U...>;
```

`mp_eval_or<T, F, U…>` is an alias for `F<U…>` when this expression is valid, for `T` otherwise.

Advent of Code : Day 2

mp_eval_or<T, F, U...>

```
template<class T, template<class...> class F, class... U> using mp_eval_or =
    mp_eval_if_not<mp_valid<F, U...>, T, F, U...>;
```

`mp_eval_or<T, F, U…>` is an alias for `F<U…>` when this expression is valid, for `T` otherwise.

Advent of Code : Day 2

mp_eval_or<T, F, U...>

```
template<class T, template<class...> class F, class... U> using mp_eval_or =
    mp_eval_if_not<mp_valid<F, U...>, T, F, U...>;
```

`mp_eval_or<T, F, U…>` is an alias for `F<U…>` when this expression is valid, for `T` otherwise.

Advent of Code : Day 2

mp_eval_or<T, F, U...>

```
template<class T, template<class...> class F, class... U> using mp_eval_or =
    mp_eval_if_not<mp_valid<F, U...>, T, F, U...>;
```

`mp_eval_or<T, F, U…>` is an alias for `F<U…>` when this expression is valid, for `T` otherwise.

Advent of Code : Day 2

mp_eval_or<T, F, U...>

```
template<class T, template<class...> class F, class... U> using mp_eval_or =
    mp_eval_if_not<mp_valid<F, U...>, T, F, U...>;
```

`mp_eval_or<T, F, U…>` is an alias for `F<U…>` when this expression is valid, for `T` otherwise.

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2

```
template <typename GameT>
using score = mp_eval_or<
    draw_score<mp_second<L>>,
    mp_second,
    mp_map_find<outcomes, mp_rename<GameT, mp_list>>>;
```

```
template <typename L>
using fn = mp_sum<mp_transform<score, L>>;
```

Advent of Code : Day 2 / Part 2

--- Part Two ---

The Elf finishes helping with the tent and sneaks back over to you. "Anyway, the second column says how the round needs to end: X means you need to lose, Y means you need to end the round in a draw, and Z means you need to win. Good luck!"

The total score is still calculated in the same way, but now you need to figure out what shape to choose so the round ends as indicated. The example above now goes like this:

- In the first round, your opponent will choose Rock (A), and you need the round to end in a draw (Y), so you also choose Rock. This gives you a score of 1 + 3 = 4.
- In the second round, your opponent will choose Paper (B), and you choose Rock so you lose (X) with a score of 1 + 0 = 1.
- In the third round, you will defeat your opponent's Scissors with Rock for a score of 1 + 6 = 7.

Advent of Code : Day 2 / Part 2

--- Part Two ---

The Elf finishes helping with the tent and sneaks back over to you. "Anyway, the second column says how the round needs to end: X means you need to lose, Y means you need to end the round in a draw, and Z means you need to win. Good luck!"

The total score is still calculated in the same way, but now you need to figure out what shape to choose so the round ends as indicated. The example above now goes like this:

- In the first round, your opponent will choose Rock (A), and you need the round to end in a draw (Y), so you also choose Rock. This gives you a score of 1 + 3 = 4.
- In the second round, your opponent will choose Paper (B), and you choose Rock so you lose (X) with a score of 1 + 0 = 1.
- In the third round, you will defeat your opponent's Scissors with Rock for a score of 1 + 6 = 7.

Advent of Code : Day 2 / Part 2

```
--- Part Two ---

The Elf finishes helping with the tent and sneaks back over to you.
"Anyway, the second column says how the round needs to end: X means you
need to lose, Y means you need to end the round in a draw, and Z means
you need to win. Good luck!"

The total score is still calculated in the same way, but now you need to
figure out what shape to choose so the round ends as indicated. The
example above now goes like this:

  - In the first round, your opponent will choose Rock (A), and you need
    the round to end in a draw (Y), so you also choose Rock. This gives
    you a score of 1 + 3 = 4.
  - In the second round, your opponent will choose Paper (B), and you
    choose Rock so you lose (X) with a score of 1 + 0 = 1.
  - In the third round, you will defeat your opponent's Scissors with
    Rock for a score of 1 + 6 = 7.
```

Advent of Code : Day 2 / Part 2

Now that you're correctly decrypting the ultra top secret strategy guide, you would get a total score of `12`.

Following the Elf's instructions for the second column, what would your total score be if everything goes exactly according to your strategy guide?

Advent of Code : Day 2 / Part 2

```
struct Day02Part02
: Day02
{
    using NeedToLose = mp_constant<'X'>;
    using NeedToDraw = mp_constant<'Y'>;
    using NeedToWin = mp_constant<'Z'>;
```

Advent of Code : Day 2 / Part 2

```
template <typename TheirsT, typename NeedT, typename MineT>
using outcome = mp_list<
    mp_list<TheirsT, NeedT>,
    Day02Part01::score<mp_list<TheirsT, MineT>>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename TheirsT, typename NeedT, typename MineT>
using outcome = mp_list<
    mp_list<TheirsT, NeedT>,
    Day02Part01::score<mp_list<TheirsT, MineT>>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename TheirsT, typename NeedT, typename MineT>
using outcome = mp_list<
    mp_list<TheirsT, NeedT>,
    Day02Part01::score<mp_list<TheirsT, MineT>>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename TheirsT, typename NeedT, typename MineT>
using outcome = mp_list<
    mp_list<TheirsT, NeedT>,
    Day02Part01::score<mp_list<TheirsT, MineT>>>;
```

```
using outcomes = mp_list<
    outcome<TheirRock,     NeedToLose,  MyScissors>,
    outcome<TheirRock,     NeedToDraw,  MyRock>,
    outcome<TheirRock,     NeedToWin,   MyPaper>,
    outcome<TheirPaper,    NeedToLose,  MyRock>,
    outcome<TheirPaper,    NeedToDraw,  MyPaper>,
    outcome<TheirPaper,    NeedToWin,   MyScissors>,
    outcome<TheirScissors, NeedToLose,  MyPaper>,
    outcome<TheirScissors, NeedToDraw,  MyScissors>,
    outcome<TheirScissors, NeedToWin,   MyRock>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename OutcomesT, typename T>
using score_impl = mp_eval_or<
    draw_score<mp_second<T>>,
    mp_second,
    mp_map_find<OutcomesT, mp_rename<T, mp_list>>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename OutcomesT, typename T>
using score_impl = mp_eval_or<
    draw_score<mp_second<T>>,
    mp_second,
    mp_map_find<OutcomesT, mp_rename<T, mp_list>>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename OutcomesT, typename T>
using score_impl = mp_eval_or<
    draw_score<mp_second<T>>,
    mp_second,
    mp_map_find<OutcomesT, mp_rename<T, mp_list>>>;
```

```
template <typename T>
using score = score_impl<outcomes, T>;
```

Advent of Code : Day 2 / Part 2

```
template <typename OutcomesT, typename T>
using score_impl = mp_eval_or<
    draw_score<mp_second<T>>,
    mp_second,
    mp_map_find<OutcomesT, mp_rename<T, mp_list>>>;

template <typename T>
using score = score_impl<outcomes, T>;
```

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_bind_front<Day02Part01::score_impl, outcomes>,
    T>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename OutcomesT, typename T>
using score_impl = mp_eval_or<
    draw_score<mp_second<T>>,
    mp_second,
    mp_map_find<OutcomesT, mp_rename<T, mp_list>>>;

template <typename T>
using score = score_impl<outcomes, T>;
```

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_bind_front<Day02Part01::score_impl, outcomes>,
    T>>;
```

Advent of Code : Day 2 / Part 2

```
template <typename OutcomesT, typename T>
using score_impl = mp_eval_or<
    draw_score<mp_second<T>>,
    mp_second,
    mp_map_find<OutcomesT, mp_rename<T, mp_list>>>;

template <typename T>
using score = score_impl<outcomes, T>;
```

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_bind_front<Day02Part01::score_impl, outcomes>,
    T>>;
```

Advent of Code : Day 3

Advent of Code : Day 3

--- Day 3: Rucksack Reorganization ---

One Elf has the important job of loading all of the rucksacks with supplies for the jungle journey. Unfortunately, that Elf didn't quite follow the packing instructions, and so a few items now need to be rearranged.

Each rucksack has two large compartments. All items of a given type are meant to go into exactly one of the two compartments. The Elf that did the packing failed to follow this rule for exactly one item type per rucksack.

The Elves have made a list of all of the items currently in each rucksack (your puzzle input), but they need your help finding the errors. Every item type is identified by a single lowercase or uppercase letter (that is, `a` and `A` refer to different types of items).

Advent of Code : Day 3

--- Day 3: Rucksack Reorganization ---

One Elf has the important job of loading all of the rucksacks with supplies for the jungle journey. Unfortunately, that Elf didn't quite follow the packing instructions, and so a few items now need to be rearranged.

Each rucksack has two large compartments. All items of a given type are meant to go into exactly one of the two compartments. The Elf that did the packing failed to follow this rule for exactly one item type per rucksack.

The Elves have made a list of all of the items currently in each rucksack (your puzzle input), but they need your help finding the errors. Every item type is identified by a single lowercase or uppercase letter (that is, `a` and `A` refer to different types of items).

Advent of Code : Day 3

--- Day 3: Rucksack Reorganization ---

One Elf has the important job of loading all of the rucksacks with supplies for the jungle journey. Unfortunately, that Elf didn't quite follow the packing instructions, and so a few items now need to be rearranged.

Each rucksack has two large compartments. All items of a given type are meant to go into exactly one of the two compartments. The Elf that did the packing failed to follow this rule for exactly one item type per rucksack.

The Elves have made a list of all of the items currently in each rucksack (your puzzle input), but they need your help finding the errors. Every item type is identified by a single lowercase or uppercase letter (that is, `a` and `A` refer to different types of items).

Advent of Code : Day 3

The list of items for each rucksack is given as characters all on a single line. A given rucksack always has the same number of items in each of its two compartments, so the first half of the characters represent items in the first compartment, while the second half of the characters represent items in the second compartment.

For example, suppose you have the following list of contents from six rucksacks:

```
vJrwpWtwJgWrhcsFMMfFFhFp
jqHRNqRjqzjGDLGLrsFMfFZSrLrFZsSL
PmmdzqPrVvPwwTWBwg
wMqvLMZHhHMvwLHjbvcjnnSBnvTQFn
ttgJtRGJQctTZtZT
CrZsJsPPZsGzwwsLwLmpwMDw
```

Advent of Code : Day 3

The list of items for each rucksack is given as characters all on a single line. A given rucksack always has the same number of items in each of its two compartments, so the first half of the characters represent items in the first compartment, while the second half of the characters represent items in the second compartment.

For example, suppose you have the following list of contents from six rucksacks:

```
vJrwpWtwJgWrhcsFMMfFFhFp
jqHRNqRjqzjGDLGLrsFMfFZSrLrFZsSL
PmmdzqPrVvPwwTWBwg
wMqvLMZHhHMvwLHjbvcjnnSBnvTQFn
ttgJtRGJQctTZtZT
CrZsJsPPZsGzwwsLwLmpwMDw
```

Advent of Code : Day 3

- The first rucksack contains the items `vJrwpWtwJgWrhcsFMMfFFhFp`, which means its first compartment contains the items `vJrwpWtwJgWr`, while the second compartment contains the items `hcsFMMfFFhFp`. The only item type that appears in both compartments is lowercase `p`.
- The second rucksack's compartments contain `jqHRNqRjqzjGDLGL` and `rsFMfFZSrLrFZsSL`. The only item type that appears in both compartments is uppercase `L`.
- The third rucksack's compartments contain `PmmdzqPrV` and `vPwwTWBwg`; the only common item type is uppercase `P`.
- The fourth rucksack's compartments only share item type `v`.
- The fifth rucksack's compartments only share item type `t`.
- The sixth rucksack's compartments only share item type `s`.

Advent of Code : Day 3

To help prioritize item rearrangement, every item type can be converted to a priority:

- Lowercase item types a through z have priorities 1 through 26.
- Uppercase item types A through Z have priorities 27 through 52.

In the above example, the priority of the item type that appears in both compartments of each rucksack is 16 (p), 38 (L), 42 (P), 22 (v), 20 (t), and 19 (s); the sum of these is 157.

Find the item type that appears in both compartments of each rucksack. What is the sum of the priorities of those item types?

Advent of Code : Day 3

To help prioritize item rearrangement, every item type can be converted to a priority:

- Lowercase item types a through z have priorities 1 through 26.
- Uppercase item types A through Z have priorities 27 through 52.

In the above example, the priority of the item type that appears in both compartments of each rucksack is 16 (p), 38 (L), 42 (P), 22 (v), 20 (t), and 19 (s); the sum of these is 157.

Find the item type that appears in both compartments of each rucksack. What is the sum of the priorities of those item types?

Advent of Code : Day 3

```
vJrwpWtwJgWrhcsFMMfFFhFp
jqHRNqRjqzjGDLGLrsFMfFZSrLrFZsSL
PmmdzqPrVvPwwTWBwg
wMqvLMZHhHMvwLHjbvcjnnSBnvTQFn
ttgJtRGJQctTZtZT
CrZsJsPPZsGzwwsLwLmpwMDw
```

Advent of Code : Day 3

```
template <char... cs>
using Chars = std::integer_sequence<char, cs...>;

using Input = TypeList<
Chars<'v','J','r','w','p','W','t','w','J','g','W','r','h','c','s','F',
      'M','M','f','F','F','h','F','p'>,
Chars<'j','q','H','R','N','q','R','j','q','z','j','G','D','L','G','L',
      'r','s','F','M','f','F','Z','S','r','L','r','F','Z','s','S','L'>,
Chars<'P','m','m','d','z','q','P','r','V','v','P','w','w','T','W','B',
      'w','g'>,
Chars<'w','M','q','v','L','M','Z','H','h','H','M','v','w','L','H','j',
      'b','v','c','j','n','n','S','B','n','v','T','Q','F','n'>,
Chars<'t','t','g','J','t','R','G','J','Q','c','t','T','Z','t','Z','T'>,
Chars<'C','r','Z','s','J','s','P','P','Z','s','G','z','w','w','s','L',
      'w','L','m','p','w','M','D','w'>>;
```

Advent of Code : Day 3

```
template <char... cs>
using Chars = std::integer_sequence<char, cs...>;

using Input = TypeList<
Chars<'v','J','r','w','p','W','t','w','J','g','W','r','h','c','s','F',
      'M','M','f','F','F','h','F','p'>,
Chars<'j','q','H','R','N','q','R','j','q','z','j','G','D','L','G','L',
      'r','s','F','M','f','F','Z','S','r','L','r','F','Z','s','S','L'>,
Chars<'P','m','m','d','z','q','P','r','V','v','P','w','w','T','W','B',
      'w','g'>,
Chars<'w','M','q','v','L','M','Z','H','h','H','M','v','w','L','H','j',
      'b','v','c','j','n','n','S','B','n','v','T','Q','F','n'>,
Chars<'t','t','g','J','t','R','G','J','Q','c','t','T','Z','t','Z','T'>,
Chars<'C','r','Z','s','J','s','P','P','Z','s','G','z','w','w','s','L',
      'w','L','m','p','w','M','D','w'>>;
```

Advent of Code : Day 3

```
template <char... cs>
using Chars = std::integer_sequence<char, cs...>;

using Input = TypeList<
Chars<'v','J','r','w','p','W','t','w','J','g','W','r','h','c','s','F',
      'M','M','f','F','F','h','F','p'>,
Chars<'j','q','H','R','N','q','R','j','q','z','j','G','D','L','G','L',
      'r','s','F','M','f','F','Z','S','r','L','r','F','Z','s','S','L'>,
Chars<'P','m','m','d','z','q','P','r','V','v','P','w','w','T','W','B',
      'w','g'>,
Chars<'w','M','q','v','L','M','Z','H','h','H','M','v','w','L','H','j',
      'b','v','c','j','n','n','S','B','n','v','T','Q','F','n'>,
Chars<'t','t','g','J','t','R','G','J','Q','c','t','T','Z','t','Z','T'>,
Chars<'C','r','Z','s','J','s','P','P','Z','s','G','z','w','w','s','L',
      'w','L','m','p','w','M','D','w'>>;
```

Advent of Code : Day 3

```
template <char... cs>
using Chars = std::integer_sequence<char, cs...>;

using Input = TypeList<
Chars<'v','J','r','w','p','W','t','w','J','g','W','r','h','c','s','F',
      'M','M','f','F','F','h','F','p'>,
Chars<'j','q','H','R','N','q','R','j','q','z','j','G','D','L','G','L',
      'r','s','F','M','f','F','Z','S','r','L','r','F','Z','s','S','L'>,
Chars<'P','m','m','d','z','q','P','r','V','v','P','w','w','T','W','B',
      'w','g'>,
Chars<'w','M','q','v','L','M','Z','H','h','H','M','v','w','L','H','j',
      'b','v','c','j','n','n','S','B','n','v','T','Q','F','n'>,
Chars<'t','t','g','J','t','R','G','J','Q','c','t','T','Z','t','Z','T'>,
Chars<'C','r','Z','s','J','s','P','P','Z','s','G','z','w','w','s','L',
      'w','L','m','p','w','M','D','w'>>;
```

Advent of Code : Day 3

- Convert integer sequence into list of types
- Split the input in two parts
- Find the one item in both parts
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3

mp_from_sequence<S>

```
template<class S> using mp_from_sequence = /*...*/
```

`mp_from_sequence` transforms an integer sequence produced by `make_integer_sequence` into an `mp_list` of the corresponding `std::integral_constant` types. Given

```
template<class T, T... I> struct S;
```

`mp_from_sequence<S<T, I…>>` is an alias for `mp_list<std::integral_constant<T, I>…>`.

Advent of Code : Day 3

mp_from_sequence<S>

```
template<class S> using mp_from_sequence = /*...*/
```

`mp_from_sequence` transforms an integer sequence produced by `make_integer_sequence` into an `mp_list` of the corresponding `std::integral_constant` types. Given

```
template<class T, T... I> struct S;
```

`mp_from_sequence<S<T, I…>>` is an alias for `mp_list<std::integral_constant<T, I>…>`.

Advent of Code : Day 3

mp_from_sequence<S>

```
template<class S> using mp_from_sequence = /*...*/
```

`mp_from_sequence` transforms an integer sequence produced by `make_integer_sequence` into an `mp_list` of the corresponding `std::integral_constant` types. Given

```
template<class T, T... I> struct S;
```

`mp_from_sequence<S<T, I…>>` is an alias for `mp_list<std::integral_constant<T, I>…>`.

Advent of Code : Day 3

- Convert integer sequence into list of types
- Split the input in two parts
- Find the one item in both parts
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3

mp_split<L, S>

```
template<class L, class S> using mp_split = /*...*/;
```

Splits the list `L` into segments at each separator `S` and returns a list of the segments.

`mp_split<L<>, S>` is `L<L<>>`. `mp_split<L<T…>, S>`, where `S` does not occur in `T…`, is `L<L<T…>>`.
`mp_split<L<T1…, S, T2…, S, T3…>, S>` is `L<L<T1…>, L<T2…>, L<T3…>>`.

The segments may be empty; `mp_split<L<S, X, Y, S, S>, S>` is `L<L<>, L<X, Y>, L<>, L<>>`.

Advent of Code : Day 3

mp_split<L, S>

```
template<class L, class S> using mp_split = /*...*/;
```

Splits the list `L` into segments at each separator `S` and returns a list of the segments.

`mp_split<L<>, S>` is `L<L<>>`. `mp_split<L<T…>, S>`, where `S` does not occur in `T…`, is `L<L<T…>>`.
`mp_split<L<T1…, S, T2…, S, T3…>, S>` is `L<L<T1…>, L<T2…>, L<T3…>>`.

The segments may be empty; `mp_split<L<S, X, Y, S, S>, S>` is `L<L<>, L<X, Y>, L<>, L<>>`.

Advent of Code : Day 3

mp_split<L, S>

```
template<class L, class S> using mp_split = /*...*/;
```

Splits the list `L` into segments at each separator `S` and returns a list of the segments.

`mp_split<L<>, S>` is `L<L<>>`. `mp_split<L<T…>, S>`, where `S` does not occur in `T…`, is `L<L<T…>>`.
`mp_split<L<T1…, S, T2…, S, T3…>, S>` is `L<L<T1…>, L<T2…>, L<T3…>>`.

The segments may be empty; `mp_split<L<S, X, Y, S, S>, S>` is `L<L<>, L<X, Y>, L<>, L<>>`.

Advent of Code : Day 3

mp_split<L, S>

```
template<class L, class S> using mp_split = /*...*/;
```

Splits the list `L` into segments at each separator `S` and returns a list of the segments.

`mp_split<L<>, S>` is `L<L<>>`. `mp_split<L<T…>, S>`, where `S` does not occur in `T…`, is `L<L<T…>>`.
`mp_split<L<T1…, S, T2…, S, T3…>, S>` is `L<L<T1…>, L<T2…>, L<T3…>>`.

The segments may be empty; `mp_split<L<S, X, Y, S, S>, S>` is `L<L<>, L<X, Y>, L<>, L<>>`.

Advent of Code : Day 3

mp_drop_c<L, N>

```
template<class L, std::size_t N> using mp_drop_c = /*...*/;
```

`mp_drop_c<L, N>` removes the first `N` elements of `L` and returns the result.

Advent of Code : Day 3

mp_drop_c<L, N>

```
template<class L, std::size_t N> using mp_drop_c = /*...*/;
```

`mp_drop_c<L, N>` removes the first `N` elements of `L` and returns the result.

Advent of Code : Day 3

```
template <typename L>
using split = mp_list<
    mp_take<L, mp_constant<mp_size<T>::value/2>>,
    mp_drop<L, mp_constant<mp_size<T>::value/2>>>;
```

Advent of Code : Day 3

```
template <typename L>
using split = mp_list<
    mp_take<L, mp_constant<mp_size<T>::value/2>>,
    mp_drop<L, mp_constant<mp_size<T>::value/2>>>;
```

Advent of Code : Day 3

```
template <typename L>
using split = mp_list<
    mp_take<L, mp_constant<mp_size<T>::value/2>>,
    mp_drop<L, mp_constant<mp_size<T>::value/2>>>;
```

Advent of Code : Day 3

```
template <typename L>
using split = mp_list<
    mp_take<L, mp_constant<mp_size<T>::value/2>>,
    mp_drop<L, mp_constant<mp_size<T>::value/2>>>;
```

Advent of Code : Day 3

```
template <typename L>
using split = mp_list<
    mp_take<L, mp_constant<mp_size<T>::value/2>>,
    mp_drop<L, mp_constant<mp_size<T>::value/2>>>;
```

Advent of Code : Day 3

- Convert integer sequence into list of types
- Split the input in two parts
- Find the one item in both parts
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3

Set Operations, <boost/mp11/set.hpp>

A set is a list whose elements are unique.

Advent of Code : Day 3

Set Operations, <boost/mp11/set.hpp>

A set is a list whose elements are unique.

mp_unique<L>

```
template<class L> using mp_unique = /*...*/;
```

`mp_unique<L>` returns a list of the same form as `L` with the duplicate elements removed.

Advent of Code : Day 3

Set Operations, <boost/mp11/set.hpp>

A set is a list whose elements are unique.

mp_unique<L>

```
template<class L> using mp_unique = /*...*/;
```

`mp_unique<L>` returns a list of the same form as `L` with the duplicate elements removed.

Advent of Code : Day 3

Set Operations, <boost/mp11/set.hpp>

A set is a list whose elements are unique.

mp_unique<L>

```
template<class L> using mp_unique = /*...*/;
```

`mp_unique<L>` returns a list of the same form as `L` with the duplicate elements removed.

mp_set_intersection<S...>

```
template<class... S> using mp_set_intersection = /*...*/;
```

`mp_set_intersection<S…>` returns a set that contains the elements that occur in all of the sets `S…` .
`mp_set_intersection<>` is `mp_list<>` .

Advent of Code : Day 3

Set Operations, <boost/mp11/set.hpp>

A set is a list whose elements are unique.

mp_unique<L>

```
template<class L> using mp_unique = /*...*/;
```

`mp_unique<L>` returns a list of the same form as `L` with the duplicate elements removed.

mp_set_intersection<S...>

```
template<class... S> using mp_set_intersection = /*...*/;
```

`mp_set_intersection<S…>` returns a set that contains the elements that occur in all of the sets `S…`.
`mp_set_intersection<>` is `mp_list<>`.

Advent of Code : Day 3

```
template <typename L>
using find_unique = mp_set_intersection<
    mp_unique<mp_first<L>>,
    mp_unique<mp_second<L>>>;
```

Advent of Code : Day 3

```
template <typename L>
using find_unique = mp_set_intersection<
    mp_unique<mp_first<L>>,
    mp_unique<mp_second<L>>>;
```

Advent of Code : Day 3

```
template <typename L>
using find_unique = mp_set_intersection<
    mp_unique<mp_first<L>>,
    mp_unique<mp_second<L>>>;
```

Advent of Code : Day 3

```
template <typename L>
using find_unique = mp_set_intersection<
    mp_unique<mp_first<L>>,
    mp_unique<mp_second<L>>>;
```

Advent of Code : Day 3

```
template <typename L>
using find_unique = mp_set_intersection<
    mp_unique<mp_first<L>>,
    mp_unique<mp_second<L>>>;
```

Advent of Code : Day 3

```
template <typename L>
using find_unique = mp_set_intersection<
    mp_unique<mp_first<L>>,
    mp_unique<mp_second<L>>>;
```

Advent of Code : Day 3

```
template <typename L>
using find_unique = mp_first<
    mp_set_intersection<
        mp_unique<mp_first<L>>,
        mp_unique<mp_second<L>>>>;
```

Advent of Code : Day 3

- Convert integer sequence into list of types
- Split the input in two parts
- Find the one item in both parts
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3

- Convert integer sequence into list of types
- Split the input in two parts
- Find the one item in both parts
- Get the priority for that item

```
To help prioritize item rearrangement, every item type can be converted
to a priority:

  - Lowercase item types a through z have priorities 1 through 26.
  - Uppercase item types A through Z have priorities 27 through 52.
```

Advent of Code : Day 3

```
template <typename T>
using priority = mp_constant<
    (T::value <= 'z' && T::value >= 'a')
        ? T::value - 'a' + 1
        : T::value - 'A' + 27>;
```

Advent of Code : Day 3

```
template <typename T>
using priority = mp_constant<
    (T::value <= 'z' && T::value >= 'a')
        ? T::value - 'a' + 1
        : T::value - 'A' + 27>;
```

Advent of Code : Day 3

```
template <typename T>
using priority = mp_constant<
    (T::value <= 'z' && T::value >= 'a')
        ? T::value - 'a' + 1
        : T::value - 'A' + 27>;
```

Advent of Code : Day 3

```
template <typename T>
using priority = mp_constant<
    (T::value <= 'z' && T::value >= 'a')
        ? T::value - 'a' + 1
        : T::value - 'A' + 27>;
```

Advent of Code : Day 3

- Convert integer sequence into list of types
- Split the input in two parts
- Find the one item in both parts
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,
            split,
            find_unique,
            priority>,
        T>>;
```

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,          mp_from_sequence
            split,                   | split
            find_unique,             | find_unique
            priority>,               | priority
        T>>;
```

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,          mp_from_sequence
            split,                   | split
            find_unique,             | find_unique
            priority>,               | priority
        T>>;
```

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,           mp_from_sequence
            split,                    | split
            find_unique,              | find_unique
            priority>,                | priority
        T>>;
```

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,           mp_from_sequence
            split,                    | split
            find_unique,              | find_unique
            priority>,                | priority
        T>>;
```

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,           mp_from_sequence
            split,                    | split
            find_unique,              | find_unique
            priority>,                | priority
        T>>;
```

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,           mp_from_sequence
            split,                    | split
            find_unique,              | find_unique
            priority>,                | priority
        T>>;
```

Advent of Code : Day 3

```
template <typename T>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            mp_from_sequence,          mp_from_sequence
            split,                   | split
            find_unique,             | find_unique
            priority>,               | priority
        T>>;
```

Advent of Code : Day 3

mp_compose<F...>

```
template<template<class...> class... F> struct mp_compose;
```

`mp_compose<F1, F2, …, Fn>` is a quoted metafunction that applies `F1`, `F2`, ..., `Fn` to its argument, in sequence. That is, `mp_compose<F1, F2, …, Fn>::fn<T…>` is `Fn<…F2<F1<T…>>…>`.

Advent of Code : Day 3

mp_compose<F...>

```
template<template<class...> class... F> struct mp_compose;
```

`mp_compose<F1, F2, …, Fn>` is a quoted metafunction that applies `F1`, `F2`, …, `Fn` to its argument, in sequence. That is, `mp_compose<F1, F2, …, Fn>::fn<T…>` is `Fn<…F2<F1<T…>>…>`.

Advent of Code : Day 3

mp_compose<F...>

```
template<template<class...> class... F> struct mp_compose;
```

`mp_compose<F1, F2, …, Fn>` is a quoted metafunction that applies `F1`, `F2`, …, `Fn` to its argument, in sequence. That is, `mp_compose<F1, F2, …, Fn>::fn<T…>` is `Fn<…F2<F1<T…>>…>`.

Advent of Code : Day 3 / Part 2

Advent of Code : Day 3 / Part 2

```
--- Part Two ---
```

As you finish identifying the misplaced items, the Elves come to you with another issue.

For safety, the Elves are divided into groups of three. Every Elf carries a badge that identifies their group. For efficiency, within each group of three Elves, the badge is the **only item type carried by all three Elves**. That is, if a group's badge is item type `B`, then all three Elves will have item type `B` somewhere in their rucksack, and at most two of the Elves will be carrying any other item type.

The problem is that someone forgot to put this year's updated authenticity sticker on the badges. All of the badges need to be pulled out of the rucksacks so the new authenticity stickers can be attached.

Additionally, nobody wrote down which item type corresponds to each group's badges. The only way to tell which item type is the right one is by finding the one item type that is **common between all three Elves** in each group.

Advent of Code : Day 3 / Part 2

```
--- Part Two ---

As you finish identifying the misplaced items, the Elves come to you with
another issue.

For safety, the Elves are divided into groups of three. Every Elf carries
a badge that identifies their group. For efficiency, within each group of
three Elves, the badge is the only item type carried by all three Elves.
That is, if a group's badge is item type B, then all three Elves will
have item type B somewhere in their rucksack, and at most two of the
Elves will be carrying any other item type.
```

```
The problem is that someone forgot to put this year's updated
authenticity sticker on the badges. All of the badges need to be pulled
out of the rucksacks so the new authenticity stickers can be attached.

Additionally, nobody wrote down which item type corresponds to each
group's badges. The only way to tell which item type is the right one is
by finding the one item type that is common between all three Elves in
each group.
```

Advent of Code : Day 3 / Part 2

Every set of three lines in your list corresponds to a single group, but each group can have a different badge item type. So, in the above example, the first group's rucksacks are the first three lines:

```
vJrwpWtwJgWrhcsFMMfFFhFp
jqHRNqRjqzjGDLGLrsFMfFZSrLrFZsSL
PmmdzqPrVvPwwTWBwg
```

And the second group's rucksacks are the next three lines:

```
wMqvLMZHhHMvwLHjbvcjnnSBnvTQFn
ttgJtRGJQctTZtZT
CrZsJsPPZsGzwwsLwLmpwMDw
```

In the first group, the only item type that appears in all three rucksacks is lowercase `r`; this must be their badges. In the second group, their badge item type must be `Z`.

Advent of Code : Day 3 / Part 2

Every set of three lines in your list corresponds to a single group, but each group can have a different badge item type. So, in the above example, the first group's rucksacks are the first three lines:

```
vJrwpWtwJgWrhcsFMMfFFhFp
jqHRNqRjqzjGDLGLrsFMfFZSrLrFZsSL
PmmdzqPrVvPwwTWBwg
```

And the second group's rucksacks are the next three lines:

```
wMqvLMZHhHMvwLHjbvcjnnSBnvTQFn
ttgJtRGJQctTZtZT
CrZsJsPPZsGzwwsLwLmpwMDw
```

In the first group, the only item type that appears in all three rucksacks is lowercase `r`; this must be their badges. In the second group, their badge item type must be `Z`.

Advent of Code : Day 3 / Part 2

Every set of three lines in your list corresponds to a single group, but each group can have a different badge item type. So, in the above example, the first group's rucksacks are the first three lines:

```
vJrwpWtwJgWrhcsFMMfFFhFp
jqHRNqRjqzjGDLGLrsFMfFZSrLrFZsSL
PmmdzqPrVvPwwTWBwg
```

And the second group's rucksacks are the next three lines:

```
wMqvLMZHhHMvwLHjbvcjnnSBnvTQFn
ttgJtRGJQctTZtZT
CrZsJsPPZsGzwwsLwLmpwMDw
```

In the first group, the only item type that appears in all three rucksacks is lowercase `r`; this must be their badges. In the second group, their badge item type must be `Z`.

Advent of Code : Day 3 / Part 2

In the first group, the only item type that appears in all three rucksacks is lowercase `r`; this must be their badges. In the second group, their badge item type must be `Z`.

Priorities for these items must still be found to organize the sticker attachment efforts: here, they are 18 (`r`) for the first group and 52 (`Z`) for the second group. The sum of these is `70`.

Find the item type that corresponds to the badges of each three-Elf group. What is the sum of the priorities of those item types?

Advent of Code : Day 3 / Part 2

In the first group, the only item type that appears in all three rucksacks is lowercase `r`; this must be their badges. In the second group, their badge item type must be `Z`.

Priorities for these items must still be found to organize the sticker attachment efforts: here, they are 18 (`r`) for the first group and 52 (`Z`) for the second group. The sum of these is `70`.

Find the item type that corresponds to the badges of each three-Elf group. **What is the sum of the priorities of those item types?**

Advent of Code : Day 3 / Part 2

- Convert integer sequence into list of types
- Create groupings of every 3 consecutive lists
- Find the one item in all three lists
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3 / Part 2

- Convert integer sequence into list of types
- Create groupings of every 3 consecutive lists
- Find the one item in all three lists
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, …>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, …>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, ...>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, ...>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, …>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>`...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, …>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, ...>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, ...>`.

Advent of Code : Day 3 / Part 2

mp_iterate<V, F, R>

```
template<class V, template<class...> class F, template<class...> class R>
    using mp_iterate = /*...*/;
```

`mp_iterate<V, F, R>` applies `R` to `V` successively until that's no longer possible, yielding the sequence `V`, `R<V>`, `R<R<V>>`, `R<R<R<V>>>` ...

It then returns an `mp_list` whose elements are formed by applying `F` to the above sequence of values. That is, it returns `mp_list<F<V>, F<R<V>>, F<R<R<V>>>, …>`.

Advent of Code : Day 3 / Part 2

```
template <typename L>
using grouped_by_3 = mp_iterate_q<
    L,
    mp_bind_back<mp_take, mp_int<3>>,
    mp_bind_back<mp_drop, mp_int<3>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using grouped_by_3 = mp_iterate_q<
    L,
    mp_bind_back<mp_take, mp_int<3>>,
    mp_bind_back<mp_drop, mp_int<3>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using grouped_by_3 = mp_iterate_q<
    L,
    mp_bind_back<mp_take, mp_int<3>>,
    mp_bind_back<mp_drop, mp_int<3>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using grouped_by_3 = mp_iterate_q<
    L,
    mp_bind_back<mp_take, mp_int<3>>,
    mp_bind_back<mp_drop, mp_int<3>>>;
```

Advent of Code : Day 3 / Part 2

- Convert integer sequence into list of types
- Create groupings of every 3 consecutive lists
- Find the one item in all three lists
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3 / Part 2

```
template <typename L>
using find_unique = mp_first<
    mp_set_intersection<
        mp_unique<mp_first<L>>,
        mp_unique<mp_second<L>>>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using find_unique = mp_first<
    mp_set_intersection<
        mp_unique<mp_first<L>>,
        mp_unique<mp_second<L>>>>;
```

```
template <typename L>
using find_unique = mp_first<
    mp_apply<
        mp_set_intersection,
        mp_transform<mp_unique, L>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using find_unique = mp_first<
    mp_set_intersection<
        mp_unique<mp_first<L>>,
        mp_unique<mp_second<L>>>>;
```

```
template <typename L>
using find_unique = mp_first<
    mp_apply<
        mp_set_intersection,
        mp_transform<mp_unique, L>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using find_unique = mp_first<
    mp_set_intersection<
        mp_unique<mp_first<L>>,
        mp_unique<mp_second<L>>>>;
```

```
template <typename L>
using find_unique = mp_first<
    mp_apply<
        mp_set_intersection,
        mp_transform<mp_unique, L>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using find_unique = mp_first<
    mp_set_intersection<
        mp_unique<mp_first<L>>,
        mp_unique<mp_second<L>>>>;
```

```
template <typename L>
using find_unique = mp_first<
    mp_apply<
        mp_set_intersection,
        mp_transform<mp_unique, L>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using find_unique = mp_first<
    mp_set_intersection<
        mp_unique<mp_first<L>>,
        mp_unique<mp_second<L>>>>;
```

```
template <typename L>
using find_unique = mp_first<
    mp_apply<
        mp_set_intersection,
        mp_transform<mp_unique, L>>>;
```

Advent of Code : Day 3 / Part 2

- Convert integer sequence into list of types
- Create groupings of every 3 consecutive lists
- Find the one item in all three lists
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3 / Part 2

- Convert integer sequence into list of types
- Create groupings of every 3 consecutive lists
- Find the one item in all three lists
- Get the priority for that item
- Sum the priorities

Advent of Code : Day 3 / Part 2

```
template <typename L>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            find_unique,
            priority>,
        grouped<mp_transform<mp_from_sequence, L>>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            find_unique,
            priority>,
        grouped<mp_transform<mp_from_sequence, L>>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            find_unique,
            priority>,
        grouped<mp_transform<mp_from_sequence, L>>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            find_unique,
            priority>,
        grouped<mp_transform<mp_from_sequence, L>>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            find_unique,
            priority>,
        grouped<mp_transform<mp_from_sequence, L>>>>;
```

Advent of Code : Day 3 / Part 2

```
template <typename L>
using fn = mp_sum<
    mp_transform_q<
        mp_compose<
            find_unique,
            priority>,
        grouped<mp_transform<mp_from_sequence, L>>>>;
```

Advent of Code : Day 4

Advent of Code : Day 4

--- Day 4: Camp Cleanup ---

Space needs to be cleared before the last supplies can be unloaded from the ships, and so several Elves have been assigned the job of cleaning up sections of the camp. Every section has a unique ID number, and each Elf is assigned a range of section IDs.

However, as some of the Elves compare their section assignments with each other, they've noticed that many of the assignments overlap. To try to quickly find overlaps and reduce duplicated effort, the Elves pair up and make a big list of the section assignments for each pair (your puzzle input).

Advent of Code : Day 4

```
--- Day 4: Camp Cleanup ---

Space needs to be cleared before the last supplies can be unloaded from
the ships, and so several Elves have been assigned the job of cleaning up
sections of the camp. Every section has a unique ID number, and each Elf
is assigned a range of section IDs.

However, as some of the Elves compare their section assignments with each
other, they've noticed that many of the assignments overlap. To try to
quickly find overlaps and reduce duplicated effort, the Elves pair up and
make a big list of the section assignments for each pair (your puzzle
input).
```

Advent of Code : Day 4

For example, consider the following list of section assignment pairs:

```
2-4,6-8
2-3,4-5
5-7,7-9
2-8,3-7
6-6,4-6
2-6,4-8
```

For the first few pairs, this list means:

- Within the first pair of Elves, the first Elf was assigned sections 2-4 (sections 2, 3, and 4), while the second Elf was assigned sections 6-8 (sections 6, 7, 8).
- The Elves in the second pair were each assigned two sections.
- The Elves in the third pair were each assigned three sections: one got sections 5, 6, and 7, while the other also got 7, plus 8 and 9.

Advent of Code : Day 4

For example, consider the following list of section assignment pairs:

```
2-4,6-8
2-3,4-5
5-7,7-9
2-8,3-7
6-6,4-6
2-6,4-8
```

For the first few pairs, this list means:

- Within the first pair of Elves, the first Elf was assigned sections 2-4 (sections 2, 3, and 4), while the second Elf was assigned sections 6-8 (sections 6, 7, 8).
- The Elves in the second pair were each assigned two sections.
- The Elves in the third pair were each assigned three sections: one got sections 5, 6, and 7, while the other also got 7, plus 8 and 9.

Advent of Code : Day 4

```
.234.....  2-4
.....678.  6-8

.23......  2-3
...45....  4-5

....567..  5-7
......789  7-9

.2345678.  2-8
..34567..  3-7

.....6...  6-6
...456...  4-6

.23456...  2-6
...45678.  4-8
```

This example list uses single-digit section IDs to make it easier to draw; your actual list might contain larger numbers. Visually, these pairs of section assignments look like this:

Some of the pairs have noticed that one of their assignments **fully contains** the other. For example, `2-8` fully contains `3-7`, and `6-6` is fully contained by `4-6`. In pairs where one assignment fully contains the other, one Elf in the pair would be exclusively cleaning sections their partner will already be cleaning, so these seem like the most in need of reconsideration. In this example, there are `2` such pairs.

In how many assignment pairs does one range fully contain the other?

Advent of Code : Day 4

```
.234.....  2-4
.....678.  6-8

.23......  2-3
...45....  4-5

....567..  5-7
......789  7-9

.2345678.  2-8
..34567..  3-7

.....6...  6-6
...456...  4-6

.23456...  2-6
...45678.  4-8
```

This example list uses single-digit section IDs to make it easier to draw; your actual list might contain larger numbers. Visually, these pairs of section assignments look like this:

Some of the pairs have noticed that one of their assignments **fully contains** the other. For example, `2-8` fully contains `3-7`, and `6-6` is fully contained by `4-6`. In pairs where one assignment fully contains the other, one Elf in the pair would be exclusively cleaning sections their partner will already be cleaning, so these seem like the most in need of reconsideration. In this example, there are `2` such pairs.

In how many assignment pairs does one range fully contain the other?

Advent of Code : Day 4

```
.234.....  2-4
.....678.  6-8

.23......  2-3
...45....  4-5

....567..  5-7
......789  7-9

.2345678.  2-8
..34567..  3-7

.....6...  6-6
...456...  4-6

.23456...  2-6
...45678.  4-8
```

This example list uses single-digit section IDs to make it easier to draw; your actual list might contain larger numbers. Visually, these pairs of section assignments look like this:

Some of the pairs have noticed that one of their assignments **fully contains** the other. For example, `2-8` fully contains `3-7`, and `6-6` is fully contained by `4-6`. In pairs where one assignment fully contains the other, one Elf in the pair would be exclusively cleaning sections their partner will already be cleaning, so these seem like the most in need of reconsideration. In this example, there are `2` such pairs.

In how many assignment pairs does one range fully contain the other?

Advent of Code : Day 4

```
using Input = std::integer_sequence<char
    ,'2','-','4',',','6','-','8','\n'
    ,'2','-','3',',','4','-','5','\n'
    ,'5','-','7',',','7','-','9','\n'
    ,'2','-','8',',','3','-','7','\n'
    ,'6','-','6',',','4','-','6','\n'
    ,'2','-','6',',','4','-','8','\n'>;
```

Advent of Code : Day 4

```
using Input = TypeList<
    TypeList<
        std::integral_constant<std::size_t, 2>,
        std::integral_constant<std::size_t, 4>,
        std::integral_constant<std::size_t, 6>,
        std::integral_constant<std::size_t, 8>>,
    TypeList<
        std::integral_constant<std::size_t, 2>,
        std::integral_constant<std::size_t, 3>,
        std::integral_constant<std::size_t, 4>,
        std::integral_constant<std::size_t, 5>>,
    TypeList<
        std::integral_constant<std::size_t, 5>,
        std::integral_constant<std::size_t, 7>,
        std::integral_constant<std::size_t, 7>,
        std::integral_constant<std::size_t, 9>>,
```

Advent of Code : Day 4

```
TypeList<
    std::integral_constant<std::size_t, 2>,
    std::integral_constant<std::size_t, 8>,
    std::integral_constant<std::size_t, 3>,
    std::integral_constant<std::size_t, 7>>,
TypeList<
    std::integral_constant<std::size_t, 6>,
    std::integral_constant<std::size_t, 6>,
    std::integral_constant<std::size_t, 4>,
    std::integral_constant<std::size_t, 6>>,
TypeList<
    std::integral_constant<std::size_t, 2>,
    std::integral_constant<std::size_t, 6>,
    std::integral_constant<std::size_t, 4>,
    std::integral_constant<std::size_t, 8>>>;
```

Advent of Code : Day 4

- Determine whether an entry is overlapping
- Count all the ones that overlap

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

Advent of Code : Day 4

- Determine whether an entry is overlapping
- Count all the ones that overlap

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

```
template <typename L>
using fn = mp_count_if<L, overlaps>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

```
template <typename L>
using fn = mp_count_if<L, overlaps>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

```
template <typename L>
using fn = mp_count_if<L, overlaps>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

```
template <typename L>
using fn = mp_count_if<L, overlaps>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

```
template <typename L>
using fn = mp_count_if<L, overlaps>;
```

Advent of Code : Day 4

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename L>
using overlaps = mp_or<
  contains<mp_at_c<L, 0>, mp_at_c<L, 1>, mp_at_c<L, 2>, mp_at_c<L, 3>>,
  contains<mp_at_c<L, 2>, mp_at_c<L, 3>, mp_at_c<L, 0>, mp_at_c<L, 1>>>;
```

```
template <typename L>
using fn = mp_count_if<L, overlaps>;
```

Advent of Code : Day 4 / Part 2

Advent of Code : Day 4 / Part 2

```
template <typename T1, typename T2, typename T3, typename T4>
using contains = mp_and<mp_ge<T1, T3>, mp_le<T2, T4>>;
```

```
template <typename T>
using overlaps = mp_and<
    mp_le<mp_at_c<T, 0>, mp_at_c<T, 3>>,
    mp_le<mp_at_c<T, 2>, mp_at_c<T, 1>>>;
```

```
template <typename L>
using fn = mp_count_if<L, overlaps>;
```

Advent of Code : Day 5

Advent of Code : Day 5

```
--- Day 5: Supply Stacks ---

The expedition can depart as soon as the final supplies have been
unloaded from the ships. Supplies are stored in stacks of marked crates,
but because the needed supplies are buried under many other crates, the
crates need to be rearranged.

The ship has a giant cargo crane capable of moving crates between stacks.
To ensure none of the crates get crushed or fall over, the crane operator
will rearrange them in a series of carefully-planned steps. After the
crates are rearranged, the desired crates will be at the top of each
stack.

The Elves don't want to interrupt the crane operator during this delicate
procedure, but they forgot to ask her which crate will end up where, and
they want to be ready to unload them as soon as possible so they can
embark.
```

Advent of Code : Day 5

--- Day 5: Supply Stacks ---

The expedition can depart as soon as the final supplies have been unloaded from the ships. Supplies are stored in stacks of marked crates, but because the needed supplies are buried under many other crates, the crates need to be rearranged.

The ship has a giant cargo crane capable of moving crates between stacks. To ensure none of the crates get crushed or fall over, the crane operator will rearrange them in a series of carefully-planned steps. After the crates are rearranged, the desired crates will be at the top of each stack.

The Elves don't want to interrupt the crane operator during this delicate procedure, but they forgot to ask her which crate will end up where, and they want to be ready to unload them as soon as possible so they can embark.

Advent of Code : Day 5

--- Day 5: Supply Stacks ---

The expedition can depart as soon as the final supplies have been unloaded from the ships. Supplies are stored in stacks of marked crates, but because the needed supplies are buried under many other crates, the crates need to be rearranged.

The ship has a giant cargo crane capable of moving crates between stacks. To ensure none of the crates get crushed or fall over, the crane operator will rearrange them in a series of carefully-planned steps. After the crates are rearranged, the desired crates will be at the top of each stack.

The Elves don't want to interrupt the crane operator during this delicate procedure, but they forgot to ask her which crate will end up where, and they want to be ready to unload them as soon as possible so they can embark.

Advent of Code : Day 5

They do, however, have a drawing of the starting stacks of crates and the rearrangement procedure (your puzzle input). For example:

```
    [D]
[N] [C]
[Z] [M] [P]
 1   2   3

move 1 from 2 to 1
move 3 from 1 to 3
move 2 from 2 to 1
move 1 from 1 to 2
```

In this example, there are three stacks of crates. Stack 1 contains two crates: crate `Z` is on the bottom, and crate `N` is on top. Stack 2 contains three crates; from bottom to top, they are crates `M`, `C`, and `D`. Finally, stack 3 contains a single crate, `P`.

Advent of Code : Day 5

They do, however, have a drawing of the starting stacks of crates and the rearrangement procedure (your puzzle input). For example:

```
    [D]
[N] [C]
[Z] [M] [P]
 1   2   3

move 1 from 2 to 1
move 3 from 1 to 3
move 2 from 2 to 1
move 1 from 1 to 2
```

In this example, there are three stacks of crates. Stack 1 contains two crates: crate `Z` is on the bottom, and crate `N` is on top. Stack 2 contains three crates; from bottom to top, they are crates `M`, `C`, and `D`. Finally, stack 3 contains a single crate, `P`.

Advent of Code : Day 5

Then, the rearrangement procedure is given. In each step of the procedure, a quantity of crates is moved from one stack to a different stack. In the first step of the above rearrangement procedure, one crate is moved from stack 2 to stack 1, resulting in this configuration:

```
[D]
[N] [C]
[Z] [M] [P]
 1   2   3
```

In the second step, three crates are moved from stack 1 to stack 3. Crates are moved one at a time, so the first crate to be moved (D) ends up below the second and third crates:

```
        [Z]
        [N]
    [C] [D]
    [M] [P]
 1   2   3
```

Advent of Code : Day 5

Then, the rearrangement procedure is given. In each step of the procedure, a quantity of crates is moved from one stack to a different stack. In the first step of the above rearrangement procedure, one crate is moved from stack 2 to stack 1, resulting in this configuration:

```
[D]
[N] [C]
[Z] [M] [P]
 1   2   3
```

In the second step, three crates are moved from stack 1 to stack 3. Crates are moved one at a time, so the first crate to be moved (D) ends up below the second and third crates:

```
        [Z]
        [N]
    [C] [D]
    [M] [P]
 1   2   3
```

Advent of Code : Day 5

Then, both crates are moved from stack 2 to stack 1. Again, because crates are moved one at a time, crate `C` ends up below crate `M`:

```
        [Z]
        [N]
[M]     [D]
[C]     [P]
 1   2   3
```

Finally, one crate is moved from stack 1 to stack 2:

```
        [Z]
        [N]
        [D]
[C] [M] [P]
 1   2   3
```

Advent of Code : Day 5

Then, both crates are moved from stack 2 to stack 1. Again, because crates are moved one at a time, crate C ends up below crate M:

```
        [Z]
        [N]
[M]     [D]
[C]     [P]
 1   2   3
```

Finally, one crate is moved from stack 1 to stack 2:

```
        [Z]
        [N]
        [D]
[C] [M] [P]
 1   2   3
```

Advent of Code : Day 5

The Elves just need to know which crate will end up on top of each stack; in this example, the top crates are `C` in stack 1, `M` in stack 2, and `Z` in stack 3, so you should combine these together and give the Elves the message `CMZ`.

After the rearrangement procedure completes, what crate ends up on top of each stack?

Advent of Code : Day 5

```
    [D]
[N] [C]
[Z] [M] [P]
 1   2   3

move 1 from 2 to 1
move 3 from 1 to 3
move 2 from 2 to 1
move 1 from 1 to 2
```

Advent of Code : Day 5

```
    [D]
[N] [C]
[Z] [M] [P]
 1   2   3
```

```
move 1 from 2 to 1
move 3 from 1 to 3
move 2 from 2 to 1
move 1 from 1 to 2
```

Advent of Code : Day 5

```
    [D]
[N] [C]
[Z] [M] [P]
 1   2   3

move 1 from 2 to 1
move 3 from 1 to 3
move 2 from 2 to 1
move 1 from 1 to 2
```

Advent of Code : Day 5

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
    Chars<>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','3',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','3'>,
    Chars<'m','o','v','e',' ','2',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','2'>>;
```

Advent of Code : Day 5

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
    Chars<>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','3',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','3'>,
    Chars<'m','o','v','e',' ','2',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','2'>>;
```

Advent of Code : Day 5

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
    Chars<>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','3',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','3'>,
    Chars<'m','o','v','e',' ','2',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','2'>>;
```

Advent of Code : Day 5

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
    Chars<>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','3',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','3'>,
    Chars<'m','o','v','e',' ','2',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','2'>>;
```

Advent of Code : Day 5

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
    Chars<>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','3',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','3'>,
    Chars<'m','o','v','e',' ','2',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','2'>>;
```

Advent of Code : Day 5

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move
- Apply each move in sequence to the list of stacks
- Take the top crate from each stack

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates
- Remove the spaces (' ') at the top of each stack
- Turn into a zero-indexed list of crates

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
```

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates
- Remove the spaces (' ') at the top of each stack
- Turn into a zero-indexed list of crates

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
```

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates
- Remove the spaces (' ') at the top of each stack
- Turn into a zero-indexed list of crates

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

```
static_assert(std::is_same_v<
    mp_list<foo<A, a>, foo<B, b>, foo<C, c>>,
    mp_transform<foo, mp_list<A, B, C>, mp_list<a, b, c>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

```
static_assert(std::is_same_v<
    mp_list<foo<A, a>, foo<B, b>, foo<C, c>>,
    mp_transform<foo, mp_list<A, B, C>, mp_list<a, b, c>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

```
static_assert(std::is_same_v<
    mp_list<foo<A, a>, foo<B, b>, foo<C, c>>,
    mp_transform<foo, mp_list<A, B, C>, mp_list<a, b, c>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

```
static_assert(std::is_same_v<
    mp_list<foo<A, a>, foo<B, b>, foo<C, c>>,
    mp_transform<foo, mp_list<A, B, C>, mp_list<a, b, c>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

```
static_assert(std::is_same_v<
    mp_list<foo<A, a>, foo<B, b>, foo<C, c>>,
    mp_transform<foo, mp_list<A, B, C>, mp_list<a, b, c>>>);
```

Advent of Code : Day 5 - Crates

mp_transform<F, L...>

```
template<template<class...> class F, class... L> using mp_transform = /*...*/;
```

`mp_transform<F, L1<T1…>, L2<T2…>, …, Ln<Tn…>>` applies `F` to each successive tuple of elements and returns `L1<F<T1, T2, …, Tn>…>`.

```
static_assert(std::is_same_v<
    mp_list<foo<A>, foo<B>, foo<C>>,
    mp_transform<foo, mp_list<A, B, C>>>);
```

```
static_assert(std::is_same_v<
    mp_list<foo<A, a>, foo<B, b>, foo<C, c>>,
    mp_transform<foo, mp_list<A, B, C>, mp_list<a, b, c>>>);
```

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates
- Remove the spaces (' ') at the top of each stack
- Turn into a zero-indexed list of crates

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
```

Advent of Code : Day 5 - Crates

- Get the crates part of the input

```
" [[ "
" NZ1"
" ]] "
"    "
"[[[ "
"DCM2"
"[[[ "
"    "
"  [ "
"  P3"
"  ] "
```

Advent of Code : Day 5 - Crates

- Get the crates part of the input

```
" [[ "
" NZ1"
" ]] "
"    "
"[[[ "
"DCM2"
"[[[ "
"    "
"  [ "
"  P3"
"  ] "
```

Advent of Code : Day 5 - Crates

mp_filter<P, L...>

```
template<template<class...> class P, class... L> using mp_filter = /*...*/;
```

`mp_filter<P, L1, L2, …, Ln>` removes the elements of the list `L1` for which `mp_to_bool<P<T1, T2, …, Tn>>` is `mp_false` and returns the result, where `Ti` are the corresponding elements of `Li`.

See also `mp_copy_if` and `mp_remove_if`, less general variants of `mp_filter` that only take a single list.

Advent of Code : Day 5 - Crates

mp_filter<P, L...>

```
template<template<class...> class P, class... L> using mp_filter = /*...*/;
```

`mp_filter<P, L1, L2, …, Ln>` removes the elements of the list `L1` for which `mp_to_bool<P<T1, T2, …, Tn>>` is `mp_false` and returns the result, where `Ti` are the corresponding elements of `Li`.

See also `mp_copy_if` and `mp_remove_if`, less general variants of `mp_filter` that only take a single list.

Advent of Code : Day 5 - Crates

mp_filter<P, L...>

```
template<template<class...> class P, class... L> using mp_filter = /*...*/;
```

`mp_filter<P, L1, L2, …, Ln>` removes the elements of the list `L1` for which `mp_to_bool<P<T1, T2, …, Tn>>` is `mp_false` and returns the result, where `Ti` are the corresponding elements of `Li`.

See also `mp_copy_if` and `mp_remove_if`, less general variants of `mp_filter` that only take a single list.

Advent of Code : Day 5 - Crates

mp_filter<P, L...>

```
template<template<class...> class P, class... L> using mp_filter = /*...*/;
```

`mp_filter<P, L1, L2, …, Ln>` removes the elements of the list `L1` for which `mp_to_bool<P<T1, T2, …, Tn>>` is `mp_false` and returns the result, where `Ti` are the corresponding elements of `Li`.

See also `mp_copy_if` and `mp_remove_if`, less general variants of `mp_filter` that only take a single list.

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
```

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates
- Remove the spaces (' ') at the top of each stack
- Turn into a zero-indexed list of crates

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates
- Remove the spaces (' ') at the top of each stack

```
"  NZ1"
"DCM2"
"   P3"
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
template <typename L> using remove_spaces =
    mp_transform_q<mp_bind_front<mp_filter_q, mp_quote<is_alpha>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
template <typename L> using remove_spaces =
    mp_transform_q<mp_bind_front<mp_filter_q, mp_quote<is_alpha>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
template <typename L> using remove_spaces =
    mp_transform_q<mp_bind_front<mp_filter_q, mp_quote<is_alpha>>, L>;
```

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
template <typename L> using remove_spaces =
    mp_transform_q<mp_bind_front<mp_filter_q, mp_quote<is_alpha>>, L>;
```

Advent of Code : Day 5 - Crates

- Get the crates part of the input
- Convert from integer sequence to list of types - list of lists
- Transpose the list so we have a list for each column
- Remove the divider columns so we are left with only stacks of crates
- Remove the spaces (' ') at the top of each stack
- Turn into a zero-indexed list of crates

Advent of Code : Day 5 - Crates

```
class Crates {
template <typename L>
using get_crate_input = mp_first<mp_split<L, Chars<>>>;
template <typename L>
using from_sequence = mp_transform<mp_from_sequence, L>;
template <typename L> using transpose =
    mp_apply_q<mp_bind_front<mp_transform_q, mp_quote<mp_list>>, L>;
template <typename L> using remove_dividers =
    mp_filter_q<mp_bind_back<mp_count_if_q, mp_quote<is_digit>>, L>;
template <typename L> using remove_spaces =
    mp_transform_q<mp_bind_front<mp_filter_q, mp_quote<is_alpha>>, L>;
template <typename L>
using listify = mp_push_front<L, mp_list<>>;
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move
- Apply each move in sequence to the list of stacks
- Take the top crate from each stack

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,
            from_sequence,
            transpose,
            only_columns,
            remove_spaces,
            listify>,
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,      | get_crate_input
            from_sequence,        | from_sequence
            transpose,            | transpose
            only_columns,         | only_columns
            remove_spaces,        | remove_spaces
            listify>,             | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
public:
    template <typename T>
    using fn = mp_invoke_q<
        mp_compose<
            get_crate_input,        | get_crate_input
            from_sequence,          | from_sequence
            transpose,              | transpose
            only_columns,           | only_columns
            remove_spaces,          | remove_spaces
            listify>,               | listify
        T>;
};
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
    Chars<>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','3',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','3'>,
    Chars<'m','o','v','e',' ','2',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','2'>>;
```

Advent of Code : Day 5 - Crates

- Parse the crates diagram into an indexable list of stacks

```
mp_list<
    mp_list<>,
    mp_list<
        std::integral_constant<char, 'N'>,
        std::integral_constant<char, 'Z'>>,
    mp_list<
        std::integral_constant<char, 'D'>,
        std::integral_constant<char, 'C'>,
        std::integral_constant<char, 'M'>>,
    mp_list<
        std::integral_constant<char, 'P'>>>
```

Advent of Code : Day 5

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move
- Apply each move in sequence to the list of stacks
- Take the top crate from each stack

Advent of Code : Day 5 - Moves

```
using Input = TypeList<
    Chars<' ',' ',' ',' ','[','D',']',' ',' ',' ',' '>,
    Chars<'[','N',']',' ','[','C',']',' ',' ',' ',' '>,
    Chars<'[','Z',']',' ','[','M',']',' ','[','P',']'>,
    Chars<' ','1',' ',' ',' ','2',' ',' ',' ','3',' '>,
    Chars<>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','3',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','3'>,
    Chars<'m','o','v','e',' ','2',' ','f','r','o','m',' ','2',
          ' ','t','o',' ','1'>,
    Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','1',
          ' ','t','o',' ','2'>>;
```

Advent of Code : Day 5 - Moves

- Convert from integral sequence to list of integral constant types
- Split on a SPACE
- Remove all the words
- Convert the remaining lists into a list of three integral constants

```
Chars<'m','o','v','e',' ','1',' ','f','r','o','m',' ','2',
      ' ','t','o',' ','1'>,
```

Advent of Code : Day 5 - Moves

- Convert from integral sequence to list of integral constant types
- Split on a SPACE
- Remove all the words
- Convert the remaining lists into a list of three integral constants

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
```

Advent of Code : Day 5 - Moves

- Convert from integral sequence to list of integral constant types
- Split on a SPACE
- Remove all the words
- Convert the remaining lists into a list of three integral constants

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
```

Advent of Code : Day 5 - Moves

- Convert from integral sequence to list of integral constant types
- Split on a SPACE
- Remove all the words
- Convert the remaining lists into a list of three integral constants

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
```

Advent of Code : Day 5 - Moves

- Convert from integral sequence to list of integral constant types
- Split on a SPACE
- Remove all the words
- Convert the remaining lists into a list of three integral constants

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

```
class Moves
{
template <typename L>
using get_moves_input = mp_second<mp_split<L, Chars<>>>;
template <typename L>
using split_on_space = mp_split<L, mp_constant<' '>>;
template <typename L> using remove_words =
    mp_remove_if_q<L, mp_bind_back<mp_count_if_q, mp_quote<is_alpha>>>;
template <typename StateT, typename T>
using from_digits_ = mp_constant<StateT::value * 10 + (T::value - '0')>;
template <typename L>
using from_digits = mp_fold<L, mp_int<0>, from_digits_>;
template <typename L>
using as_ints = mp_transform<from_digits, L>;
```

Advent of Code : Day 5 - Moves

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move
- Apply each move in sequence to the list of stacks
- Take the top crate from each stack

Advent of Code : Day 5 - Moves

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move

```
public:
    template <typename T>
    using fn = mp_transform_q<
        mp_compose<
            mp_from_sequence,
            split_on_space,
            remove_words,
            as_ints>,
        get_moves_input<T>>;
};
```

Advent of Code : Day 5 - Moves

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move

```
public:
    template <typename T>
    using fn = mp_transform_q<
        mp_compose<
            mp_from_sequence,          mp_from_sequence
            split_on_space,          | split_on_space
            remove_words,            | remove_words
            as_ints>,                | as_ints
        get_moves_input<T>>;
};
```

Advent of Code : Day 5 - Moves

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move

```
TypeList<mp_list<std::integral_constant<int, 1>,
                 std::integral_constant<int, 2>,
                 std::integral_constant<int, 1>>,
         mp_list<std::integral_constant<int, 3>,
                 std::integral_constant<int, 1>,
                 std::integral_constant<int, 3>>,
         mp_list<std::integral_constant<int, 2>,
                 std::integral_constant<int, 2>,
                 std::integral_constant<int, 1>>,
         mp_list<std::integral_constant<int, 1>,
                 std::integral_constant<int, 1>,
                 std::integral_constant<int, 2>>>
```

Advent of Code : Day 5

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move
- Apply each move in sequence to the list of stacks
- Take the top crate from each stack

Advent of Code : Day 5 - Crane Simulation

```
template <typename L>
using simulate_crane = mp_fold<
    mp_invoke_q<Moves, L>,
    mp_invoke_q<Crates, L>,
    do_move>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename L>
using simulate_crane = mp_fold<
    mp_invoke_q<Moves, L>,
    mp_invoke_q<Crates, L>,
    do_move>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename L>
using simulate_crane = mp_fold<
    mp_invoke_q<Moves, L>,
    mp_invoke_q<Crates, L>,
    do_move>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename L>
using simulate_crane = mp_fold<
    mp_invoke_q<Moves, L>,
    mp_invoke_q<Crates, L>,
    do_move>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename L>
using simulate_crane = mp_fold<
    mp_invoke_q<Moves, L>,
    mp_invoke_q<Crates, L>,
    do_move>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
mp_list<
    mp_list<>,
    mp_list<std::integral_constant<char, 'C'>>,
    mp_list<std::integral_constant<char, 'M'>>,
    mp_list<std::integral_constant<char, 'Z'>,
            std::integral_constant<char, 'N'>,
            std::integral_constant<char, 'D'>,
            std::integral_constant<char, 'P'>>>
```

Advent of Code : Day 5

- Parse the crates diagram into an indexable list of stacks
- Parse the moves in to a list of types, one for each move
- Apply each move in sequence to the list of stacks
- Take the top crate from each stack

Advent of Code : Day 5

```
public:
    template <typename L>
    using fn = mp_transform<
        mp_first,
        mp_pop_front<simulate_crane<L>>>;
};
```

Advent of Code : Day 5

```
public:
    template <typename L>
    using fn = mp_transform<
        mp_first,
        mp_pop_front<simulate_crane<L>>>;
};
```

Advent of Code : Day 5

```
public:
    template <typename L>
    using fn = mp_transform<
        mp_first,
        mp_pop_front<simulate_crane<L>>>;
};
```

Advent of Code : Day 5

```
public:
    template <typename L>
    using fn = mp_transform<
        mp_first,
        mp_pop_front<simulate_crane<L>>>;
};
```

Advent of Code : Day 5

```
public:
    template <typename L>
    using fn = mp_transform<
        mp_first,
        mp_pop_front<simulate_crane<L>>>;
};
```

Advent of Code : Day 5

```
public:
    template <typename L>
    using fn = mp_transform<
        mp_first,
        mp_pop_front<simulate_crane<L>>>;
};
```

Advent of Code : Day 5

```
public:
    template <typename L>
    using fn = mp_transform<
        mp_first,
        mp_pop_front<simulate_crane<L>>>;
};
```

```
mp_list<
    std::integral_constant<char, 'C'>,
    std::integral_constant<char, 'M'>,
    std::integral_constant<char, 'Z'>>
```

Advent of Code : Day 5 - Part 2

--- Part Two ---

As you watch the crane operator expertly rearrange the crates, you notice the process isn't following your prediction.

Some mud was covering the writing on the side of the crane, and you quickly wipe it away. The crane isn't a CrateMover 9000 - it's a CrateMover 9001.

The CrateMover 9001 is notable for many new and exciting features: air conditioning, leather seats, an extra cup holder, and the ability to pick up and move multiple crates at once.

Again considering the example above, the crates begin in the same configuration:

```
    [D]
[N] [C]
[Z] [M] [P]
 1   2   3
```

Advent of Code : Day 5 - Part 2

--- Part Two ---

As you watch the crane operator expertly rearrange the crates, you notice the process isn't following your prediction.

Some mud was covering the writing on the side of the crane, and you quickly wipe it away. The crane isn't a CrateMover 9000 - it's a CrateMover 9001.

The CrateMover 9001 is notable for many new and exciting features: air conditioning, leather seats, an extra cup holder, and the ability to pick up and move multiple crates at once.

Again considering the example above, the crates begin in the same configuration:

```
    [D]
[N] [C]
[Z] [M] [P]
 1   2   3
```

Advent of Code : Day 5 - Part 2

Moving a single crate from stack 2 to stack 1 behaves the same as before:

```
[D]
[N] [C]
[Z] [M] [P]
 1   2   3
```

However, the action of moving three crates from stack 1 to stack 3 means that those three moved crates stay in the same order, resulting in this new configuration:

```
        [D]
        [N]
    [C] [Z]
    [M] [P]
 1   2   3
```

Next, as both crates are moved from stack 2 to stack 1, they retain their order as well:

Advent of Code : Day 5 - Part 2

```
        [D]
        [N]
[C]     [Z]
[M]     [P]
 1   2   3
```

Finally, a single crate is still moved from stack 1 to stack 2, but now it's crate `C` that gets moved:

```
        [D]
        [N]
        [Z]
[M] [C] [P]
 1   2   3
```

In this example, the CrateMover 9001 has put the crates in a totally different order: `MCD`.

Before the rearrangement process finishes, update your simulation so that the Elves know where they should stand to be ready to unload the final supplies. After the rearrangement procedure completes, what crate ends up on top of each stack?

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_reverse<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_identity_t<mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename HowQ, typename CratesT, typename MoveT,
          typename FromT = mp_at<CratesT, from_ndx<MoveT>>,
          typename ToT = mp_at<CratesT, to_ndx<MoveT>>>
using do_move = mp_replace_at<
    mp_replace_at<
        CratesT,
        from_ndx<MoveT>,
        mp_drop<FromT, how_many<MoveT>>>,
    to_ndx<MoveT>,
    mp_append<mp_invoke_q<HowQ, mp_take<FromT, how_many<MoveT>>>, ToT>>;
```

Advent of Code : Day 5 - Crane Simulation

```
template <typename L, class Q>
using simulate_crane = mp_fold_q<
    mp_invoke_q<Moves, L>,
    mp_invoke_q<Crates, L>,
    mp_bind_front<do_move, Q>>;
```