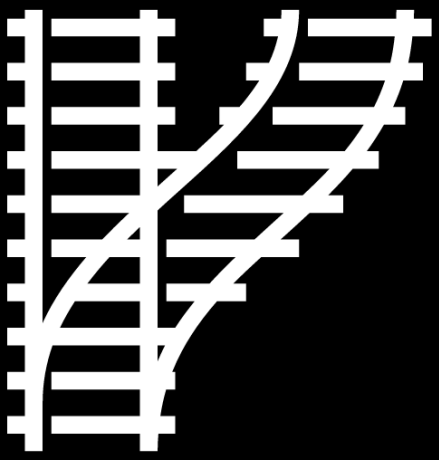


Fully Functional C++ with Range-v3

Down-to-earth programming
Compact manual

Published by
Walletfox.com
Zurich, Switzerland

3rd Edition
April 2023

TABLE OF CONTENTS

1 Basics

2 Drill

3 Example guide

TABLE OF CONTENTS

3 Example guide

4 Practice

5 Quick reference

QUICK START

Use an online compiler

- To get started quickly, use an online compiler for C++ at **https://godbolt.org/.** Use **x86-64 gcc 12.2** with **-std=c++2a** and **range-v3** enabled to compile.
- Remember to **#include <range/v3/all.hpp>.** Alternatively, you may include specific headers, e.g. **#include <range/v3/view/remove_if.hpp>** .

Work with the namespace ranges

- Make ranges accessible with '**using namespace ranges**; '. Throughout this manual, we use this declaration in order to shorten code examples. To prevent name collision, **refrain from writing 'using namespace std;**.' Should you need a function from the std namespace, write explicitly **std::name_of_the_function.**

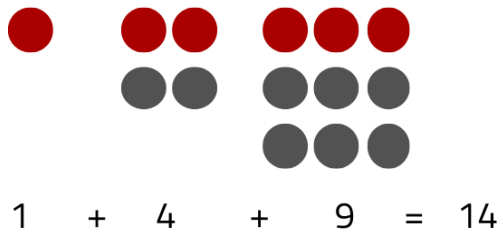
	Complete name	This manual
Algorithms	ranges::accumulate	accumulate
Views	ranges::views::transform	views::transform
Actions	ranges::actions::transform	actions::transform

Your first code

```
#include <range/v3/all.hpp>
#include <iostream>

using namespace ranges;

int main(){
    // your code goes here
    auto r_int = views::iota(1); // [1,2,3...]
    auto rng = r_int | views::transform([](int x){return x*x;}); // [1,4,9...]
    auto sum = accumulate(rng | views::take(3),0); // 1 + 4 + 9 = 14
    std::cout << sum; // 14
}
```



FIZZ BUZZ

1

GENERATE A SEQUENCE WITH:

- `"Fizz"` if the number is divisible by 3
- `"Buzz"` if the number is divisible by 5
- `"FizzBuzz"` if it is divisible both by 3 and 5
- Otherwise print **the number itself**

2

CREATE BASIC FIZZ AND BUZZ SEQUENCES

```
std::array<std::string,3> fizz {"","","Fizz"};
std::array<std::string,5> buzz {"","","","","Buzz"};
```

3

CYCLE THE SEQUENCES

```
auto r_fizzes = fizz | views::cycle;
// [ , ,Fizz, , ,Fizz...]
auto r_buzzes = buzz | views::cycle;
// [ , , , ,Buzz, , , , ,Buzz...]
```

4

COMBINE FIZZES AND BUZZES

```
auto r_fizzbuzz = views::zip_with(std::plus{},
                                  r_fizzes, r_buzzes);
/* [ , ,Fizz, ,Buzz,Fizz, , ,Fizz,Buzz, ,
   Fizz, , ,FizzBuzz, , ,Fizz...] */
```

5

CREATE AN INFINITE SEQUENCE OF INTEGERS AND CONVERT THEM TO STRINGS

```
auto r_int_str = views::iota(1) |
                 views::transform([](int x){
          return std::to_string(x);}); // [1,2,3...]
```

6

PICK AN INTEGER OR FIZZBUZZ BASED ON LEXICOGRAPHICAL ORDER

```
auto rng = views::zip_with(
      [](auto a, auto b){return std::max(a,b);},
                          r_fizzbuzz, r_int_str);
/* [1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,
   11,Fizz,13,14,FizzBuzz,16,17...] */
```

RUNSUMS

1 IDENTIFY ALL CONSECUTIVE NUMBERS THAT SUM UP TO NUMBER 15

```
auto n = 15; // [7,8],[4,5,6],[1,2,3,4,5]
```

2 GENERATE TRIANGULAR NUMBERS

```
auto r_tri = views::iota(1) | views::partial_sum;
// [1,3,6...]
```

3 REPEAT THE TARGET NUMBER N

```
auto r_n = views::repeat(n); // [15,15,15...]
```

4 SUBTRACT THE TRIANGULAR SEQUENCE FROM THE REPEATED NUMBER SEQUENCE

```
auto r_min =  views::zip_with(std::minus{},r_n,r_tri);
// [15 - 1 = 14, 15 - 3 = 12, 15 - 6 = 9...]
```

5 TAKE ELEMENTS GREATER THAN ZERO

```
auto r_minp = r_min | views::take_while([](int x){
                        return x > 0; }); // [14,12,9,5]
```

6 PAIR SUBTRACTED NUMBERS WITH INTEGER SEQUENCE 2,3,4...

```
auto r_pair = views::zip(r_minp, views::iota(2));
// (14:2)(12:3)(9:4)(5:5)
```

7 KEEP ONLY MUTUALLY DIVISIBLE PAIRS

```
auto r_ok = r_pair | views::filter([](auto const& p){
                        return (p.first % p.second == 0);
}); // (14:2)(12:3)(5:5)
```

8 GET RUNSUMS STARTING FROM QUOTIENT k

```
auto rng = r_ok | views::transform([](auto const& p){
                auto k = p.first / p.second;
                return views::iota(k, k + p.second);
}); // [[7,8],[4,5,6],[1,2,3,4,5]]
```

n + (n + 1)	2n + 1	(15 - 1) / 2 = 7	[7,8]
n + (n + 1) + (n + 2)	3n + 3	(15 - 3) / 3 = 4	[4,5,6]
n + (n + 1) + (n + 2) + (n + 3)	4n + 6	(15 - 6) / 4 = -	[]
n + (n + 1) + (n + 2) + (n + 3) + (n + 4)	5n + 10	(15 - 10) / 5 = 1	[1,2,3,4,5]