



# 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

Functional programming toolbox	6
Ranges, views, actions	7
Views in detail	8
Range-v3 vs C++Ranges	9
Quick start	10
Avoiding pitfalls	11

2

Drill

Higher-order functions	12
Sequence generation	13
Mapping / projection	14
Folding / accumulation	15
Filtering	16
Lookup	16
Selection	17
Zipping	18
Many-to-one	18
One-to-many	19
Yield, yield_if, yield_from	20
Input extraction	21
Materialization	21
Printing / output	22

3

Example  
guide

Hamming distance	23
Approximation of $\pi$	23
Digits in a factorial	24
Approximation of e	24
Fibonacci sequence	25
Triangular sequence	25
Star numbers	26
Arithmetic sequence 385	26
Least common multiple	27
Meeting overlaps	27
Decimal to binary	28
Continued fraction	28
Extension retrieval	29
Power and multiplication alternative	29
Luhn algorithm	30
Binary to decimal	31

# TABLE OF CONTENTS



3

Example  
guide

Frequency count	32
Palindrome	33
Area of a polygon	34
Seven bridges of Königsberg	35
Fizz Buzz	36
Caesar cipher	37
CamelCase to snake_case	38
Multiplicative persistence	39
Balanced parentheses	40
Runsums	41
Rail fence cipher	42
Non-uniform chunks	43
Convert seconds to compound duration	44
Linear convolution 1D	45
Reverse Polish notation	46
Molecular weight (input, std::optional)	48
Planetary masses (input, class, std::optional)	50
Matrix dimensions, rows, columns, transpose	52
Comprehensions - duplicates	53
Comprehensions - missing ranges	53
Comprehensions - Pythagorean triples	54
Comprehensions - matching positions	54
Comprehensions - squarions	55

4

Practice

Practice questions	56
Solutions - Generating sequences	58
Solutions - Transform / accumulate	59
Solutions - Transform / enumerate / zip / accumulate	59
Solutions - Drop / stride / cycle	60
Solutions - Cycle / sliding	61
Solutions - Chunk / Chunk_by	62
Solutions - Range comprehensions	64
Solutions - User-defined types	66

5

Quick  
reference

Quick reference - Short list of range algorithms	68
Quick reference - Range access, iterators	69
Quick reference - Range inspection, other HOFs	70
Quick reference - Views	71



## 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
<b>Algorithms</b>	ranges::accumulate	accumulate
<b>Views</b>	ranges::views::transform	views::transform
<b>Actions</b>	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
}
```

$$1 + 4 + 9 = 14$$



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...] */
```



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]]
```

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