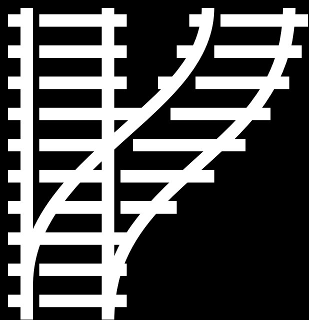


# Fully Functional C++ with Range-v3

Down-to-earth programming
Compact manual

**Published by**
Walletfox.com
Zurich, Switzerland

**1st Edition**
February 2020

# TABLE OF CONTENTS

1 Basics


| | |
|---|---|
| Functional programming toolbox | 6 |
| Ranges, views, actions | 7 |
| Views in detail | 8 |
| Range-v3 vs C++20 | 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 |
| Extension retrieval | 26 |
| Power and multiplication alternative | 26 |
| Luhn algorithm | 27 |
| Binary to decimal | 28 |


© Walletfox.com

# TABLE OF CONTENTS

**3 Example guide**


| | |
|---|---|
| Frequency count | 29 |
| Palindrome | 30 |
| Area of a polygon | 31 |
| Seven bridges of Königsberg | 32 |
| Fizz Buzz | 33 |
| Caesar cipher | 34 |
| CamelCase to snake_case | 35 |
| Multiplicative persistence | 36 |
| Matrix dimensions, rows, columns, transpose | 37 |
| Reverse Polish notation | 38 |
| Molecular weight (input, std::optional) | 40 |
| Planetary masses (input, class, std::optional) | 42 |
| Comprehensions - duplicates | 44 |
| Comprehensions - missing ranges | 44 |
| Comprehensions - Pythagorean triples | 45 |
| Comprehensions - matching positions | 45 |
| Comprehensions - squarions | 46 |


**4 Practice**


| | |
|---|---|
| Practice questions | 47 |
| Solutions - Generating sequences | 49 |
| Solutions - Transform | 50 |
| Solutions - Transform and accumulate | 50 |
| Solutions - Zip, transform and accumulate | 51 |
| Solutions - Drop, stride, cycle | 51 |
| Solutions - Group_by | 52 |
| Solutions - Range comprehensions | 54 |
| Solutions - User-defined types | 55 |


**5 Quick reference**


| | |
|---|---|
| Quick reference - Short list of range algorithms | 57 |
| Quick reference - Range access, iterators | 58 |
| Quick reference - Range inspection, other HOFs | 59 |
| Quick reference - Views | 60 |
| Quick reference - Actions | 69 |


© Walletfox.com

# QUICK START

## Use an online compiler

- To get started quickly, use an online compiler for C++ at **https://wandbox.org/.** Use at least **clang 9.0.0.** with **-std=c++2a** to compile the examples.
- 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.**

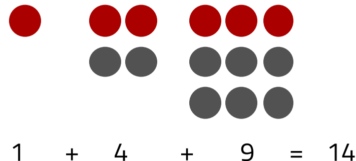
| | **Complete name** | **This manual** |
|---|---|---|
| **Algorithms** | ranges::accumulate | accumulate |
| **Views** | ranges::views::transform | views::transform |
| **Actions** | ranges::actions::transform | actions::transform |

## 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
}
```




© Walletfox.com

# FOLDING / ACCUMULATION

## AGGREGATE SUCCESSIVE ELEMENTS OF A RANGE

```
auto const v = std::vector{2,5,6};
auto val = accumulate(v,1,std::multiplies{}); // 2*5*6 = 60
```

## CALCULATE THE ARITHMETIC MEAN

```
auto const v = std::vector{5.2,3.6,4.1};
auto val = accumulate(v,0.0) / size(v); // 4.3
```

## CALCULATE THE WEIGHTED ARITHMETIC MEAN

```
auto const v1 = std::vector{8,5,5};
auto const v2 = std::vector{1,2,3};
auto ip = inner_product(v1,v2,0); // 8*1 + 5*2 + 5*3 = 33
auto sum = accumulate(v2,0); // 1 + 2 + 3 = 6
auto val = (double) ip / sum; // 5.5
```

## CALCULATE THE TOTAL PRICE OF A PRODUCT BASKET

```
struct Product {
    std::string name;
    double price;
};
auto v = std::vector<Product> {
          {"apple",1.48},{"bread",3.5},{"milk",1.69}};
auto val = accumulate(v,0.0,{},&Product::price); // 6.67
```

## CONCATENATE STRINGS USING AN UNDERSCORE

```
auto v = std::vector<std::string> {"we", "will", "see"};
auto val = accumulate(v | views::tail, front(v),
                      [](auto const& s1, auto const& s2){
                      return std::string{s1 + "_" + s2};});
// we_will_see
```

## CONVERT A VECTOR OF INTEGERS TO A NUMBER

```
auto const v = std::vector{1,7,5,6,9,3,5,8,4};
auto val = accumulate(v, 0LL, [](auto a, auto b) {
                             return a*10 + b;}); // 175693584
```

See The Reverse Polish Notation example for a more advanced use of accumulate.



© Walletfox.com

# 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

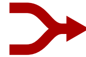
```
std::array<std::string,3> fizz {"","","Fizz"};
std::array<std::string,5> buzz {"","","","","Buzz"};
```

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...] */
```



© Walletfox.com