

# Algorithms and Iterators Workshop Solutions

## Vector of random numbers

```
#include <random>
#include <algorithm>

static std::mt19937 mt;                // Unseeded
// static std::random_device rd;
// static std::mt19937 mt(rd());      // Seeded

namespace exercises {
    int get() {
        std::uniform_int_distribution<int> ud(0, 1000);
        return ud(mt);
    }
}
```

## Vector of random numbers contd

```
int main() {  
    std::vector<int> vec(10);  
    std::generate_n(vec.begin(), 10, exercises::get);  
  
    for (auto el: vec)  
        std::cout << el << ", ";  
}
```

## Exercises 3-5

```
int main() {  
    std::vector<int> vec(10);  
    std::generate_n(vec.begin(), 10, exercises::get);  
  
    for (auto el: vec)  
        std::cout << el << ", ";  
  
    auto max = std::max_element(vec.begin(), vec.end());  
    std::cout << "\nMax element is " << *max << "\n";  
    std::cout << "Index of max element is " << max - vec.begin() << "\n";  
    std::cout << "Sum is " << std::accumulate(vec.cbegin(), vec.cend(), 0) << "\n";  
}
```

## Exercise 6

// ...

```
auto max = std::max_element(vec.cbegin(), vec.cend());  
int maxel = *max;
```

```
double arr[10];  
std::transform(vec.cbegin(), vec.cend(), std::begin(arr),  
               [maxel] (int i) { return (double)i/maxel;});  
std::cout << "\nNormalized vector: ";  
for (auto el: arr)  
    std::cout << el << ", ";
```

## Exercises 7-9

```
int n = std::count_if(vec.cbegin(), vec.cend(), [] (int i) { return i%2;});
std::cout << "\nVector contains " << n << " odd elements\n";

sort(vec.begin(), vec.end());
auto it = std::upper_bound(vec.cbegin(), vec.cend(), 455);
std::cout << "\nVector contains " << vec.end() - it << " elements > 455\n";

std::list<double> ld;
std::copy_if(vec.cbegin(), vec.cend(), back_inserter(ld),
             [] (int i) { return i%2; });
for (auto el: ld)
    std::cout << el << ", ";
```

## Exercises 10-11

```
// ...
```

```
std::sort(vec.begin(), vec.end(), [] (int l, int r) { return r < l; } );
```

```
std::cout << "\nSorted vector: ";
```

```
print(vec.cbegin(), vec.cend());
```

```
// ...
```

```
// Generic sort requires random-access iterators, which std::list does not have
```

```
// Use member function instead
```

```
ld.sort([] (double l, double r) { return r < l; } );
```

```
std::cout << "\nSorted list: ";
```

```
print(ld.cbegin(), ld.cend());
```

## Exercises 12-13

```
std::shuffle(vec.begin() + 1, vec.end() - 1, mt);
std::cout << "\nShuffled vector: ";
print(vec.cbegin(), vec.cend());

//...

auto last = std::remove_if(vec.begin(), vec.end(),
                           [](int i) { return i%2; });

std::cout << "\nVector after remove: ";
print(vec.cbegin(), vec.cend());
vec.erase(last, vec.end());
std::cout << "\nVector after erase: ";
print(vec.cbegin(), vec.cend());
```



## Exercise 14

```
std::ofstream ofile("test.txt");
```

```
// Make sure there is at least one element in vec before we iterate over it
```

```
if (ofile && vec.cbegin() != vec.cend()) {  
    std::ostream_iterator<int> out(ofile, ",");  
    std::copy(vec.cbegin(), vec.cend() - 1, out);  
    std::ostream_iterator<int> out2(ofile);  
    std::copy(vec.cend() - 1, vec.cend(), out2);  
}
```

## Exercise 15

```
int main() {  
    std::ifstream wif("words.txt");  
  
    if (wif) {  
        std::istream_iterator<std::string> iit(wif);  
        std::istream_iterator<std::string> eof;  
        std::set<std::string> words(iit, eof);  
  
        for (auto s: words)  
            std::cout << s << ", ";  
    }  
}
```

## Exercise 16

```
int main() {  
    std::ifstream wif("words.txt");  
  
    if (wif) {  
        std::istream_iterator<std::string> iit(wif);  
        std::istream_iterator<std::string> eof;  
        std::vector<std::string> words(iit, eof);  
  
        std::cout << words.size() << " words in file\n";  
    }  
}
```

## Exercise 17

```
int main() {  
    std::ifstream wif("words.txt");  
  
    if (wif) {  
        std::string line;  
        std::vector<std::string> lines;  
  
        while (getline(wif, line))  
            lines.push_back(line);  
  
        std::cout << "File contains " << lines.size() << " lines\n";  
    }  
}
```

## Exercise 18

// Add to exercise 17 solution

```
int count{0};  
std::accumulate(lines.begin(), lines.end(), count,  
                [&count] (int i, std::string s) { return count += s.size(); } );  
  
std::cout << "File contains " << count << " characters\n";
```

## Exercise 19

```
int main() {  
    std::ifstream wif("words.txt");  
    std::ifstream wif2("words2.txt");  
  
    if (wif && wif2) {  
        std::istream_iterator<std::string> iit(wif);  
        std::istream_iterator<std::string> eof;  
        std::set<std::string> words(iit, eof);  
    }  
}
```

## Exercise 19 contd

```
std::istream_iterator<std::string> iit2(wif2);
std::set<std::string> words2(iit2, eof);
std::ostream_iterator<std::string> out(std::cout, ", ");

std::set_intersection(words.begin(), words.end(),
                      words2.begin(), words2.end(), out);
}
}
```

## Exercise 20

```
#include <iostream>
#include <numeric>
#include <vector>

int main() {
    std::vector<int> numbers(6);
    std::iota(numbers.begin(), numbers.end(), 1);
    std::cout << std::accumulate(numbers.begin(), numbers.end(), 1,
        [] (int a, int b) { return a*b; }
    );
}
```



## Exercise 21

```
#include <iostream>
#include <string>
#include <algorithm>

int main() {
    std::string s{ "abc" };
    do {
        std::cout << s << "\n";
    } while (std::next_permutation(s.begin(), s.end()));
}
```