ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

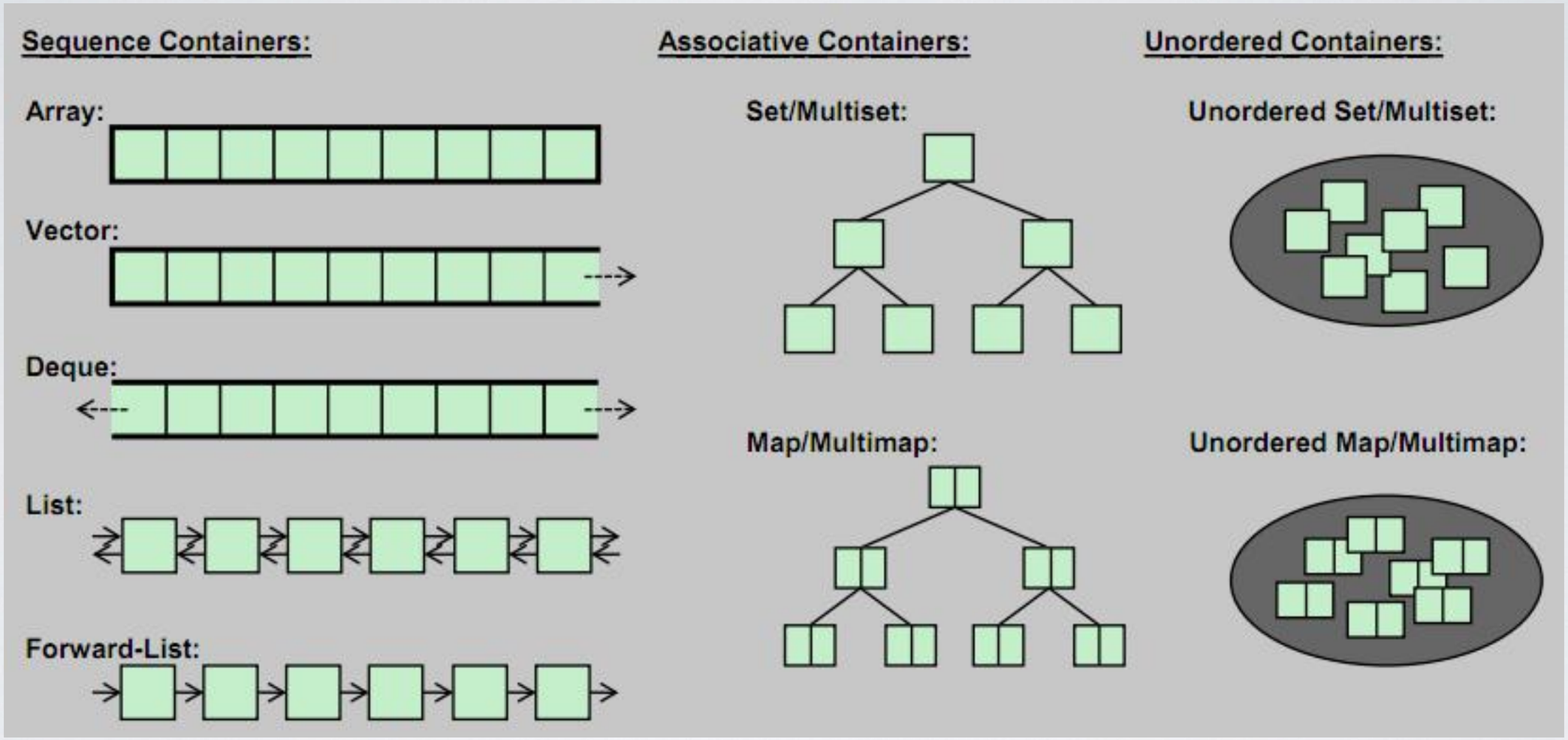
Лекция № | / **06**
18.03.2019 г.

CONTAINERS STL

CONTAINERS STL

Contiguous storage (непрерыв. хранилища)	*std::vector, std::deque, std::array.*
List storage (списки)	*std::list, std::forward_list.*
Search trees (деревья поиска)	*std::set, std::multiset.* *std::map, std::multimap.*
Hash tables (хеш-таблицы)	*std::unordered_set (_multiset),* *std::unordered_map (_multimap).*
Container adapters (адаптеры контейнеров)	*std::stack, std::queue,* *std::priority_queue.*

CONTAINERS STL



```cpp
#include <iostream>
#include <list>
#include <vector>
#include <algorithm>
#include <iterator>


int main()
{
    std::vector<int> v = { 1, 2, 3, 4 };
    std::list<int> l;

    std::copy(v.begin(), v.end(), std::front_inserter(l));

    for (auto x: l)
        std::cout << x << std::endl;  // 4… 3… 2… 1

    return 0;
}
```

Интерфейс std::list мало отличается от std::vector …

```cpp
#include <iostream>
#include <list>


int main() {
    std::list<int> l1 = { 1, 2, 3, 4 }, l2 = { 10, 20, 30 };

    auto it = l1.begin();
    ++it;       // указывает на «2»

    // Переносим элементы l2 в список l1
    l1.splice(it, l2);
    // l1: { 1, 10, 20, 30, 2, 3, 4}
    // l2: пуст

    l2.splice(l2.begin(), l1, it);
    // l1: { 1, 10, 20, 30, 3, 4}
    // l2: { 2 }, it недействителен

    it = l1.begin();
    std::advance(it, 3); // указывает теперь на «30»
    l1.splice(l1.begin(), l1, it, l1.end());
    // l1: { 30, 3, 4, 1, 10, 20 }

    for (auto x: l1) std::cout << x << std::endl;
    for (auto x: l2) std::cout << x << std::endl;
    return 0;
}
```

Перенос целого списка

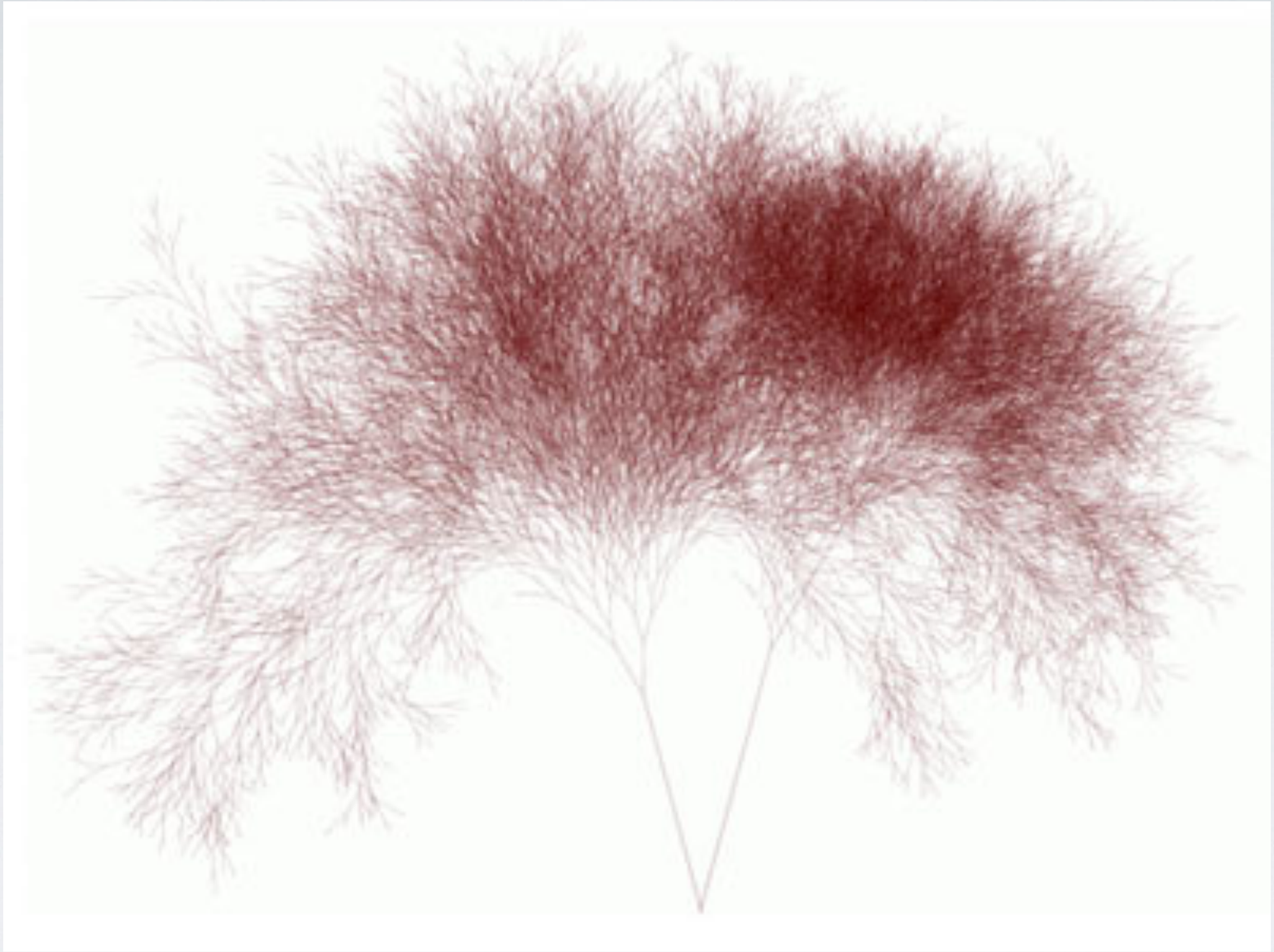
Перенос одного элемента

Перенос диапазона

std::list::splice

```cpp
#include <iostream>
#include <forward_list>

int main() {
    std::forward_list<int> first  = { 1, 2, 3 };
    std::forward_list<int> second = { 10, 20, 30 };

    auto it = first.begin();  // указывает на «1»

    first.splice_after(first.before_begin(), second);
                          // first: 10 20 30 1 2 3
                          // second: пуст
                          // "it" всё ещё указывает на «1»

    second.splice_after(second.before_begin(), first, first.begin(), it);
                          // first: 10 1 2 3
                          // second: 20 30

    first.splice_after(first.before_begin(), second, second.begin());
                          // first: 30 10 1 2 3
                          // second: 20

    std::cout << "first:";
    for (int x: first) std::cout << " " << x;
    std::cout << std::endl;

    std::cout << "second:";
    for (int x: second) std::cout << " " << x;
    std::cout << std::endl;

    return 0;
}
```

std::forward_list



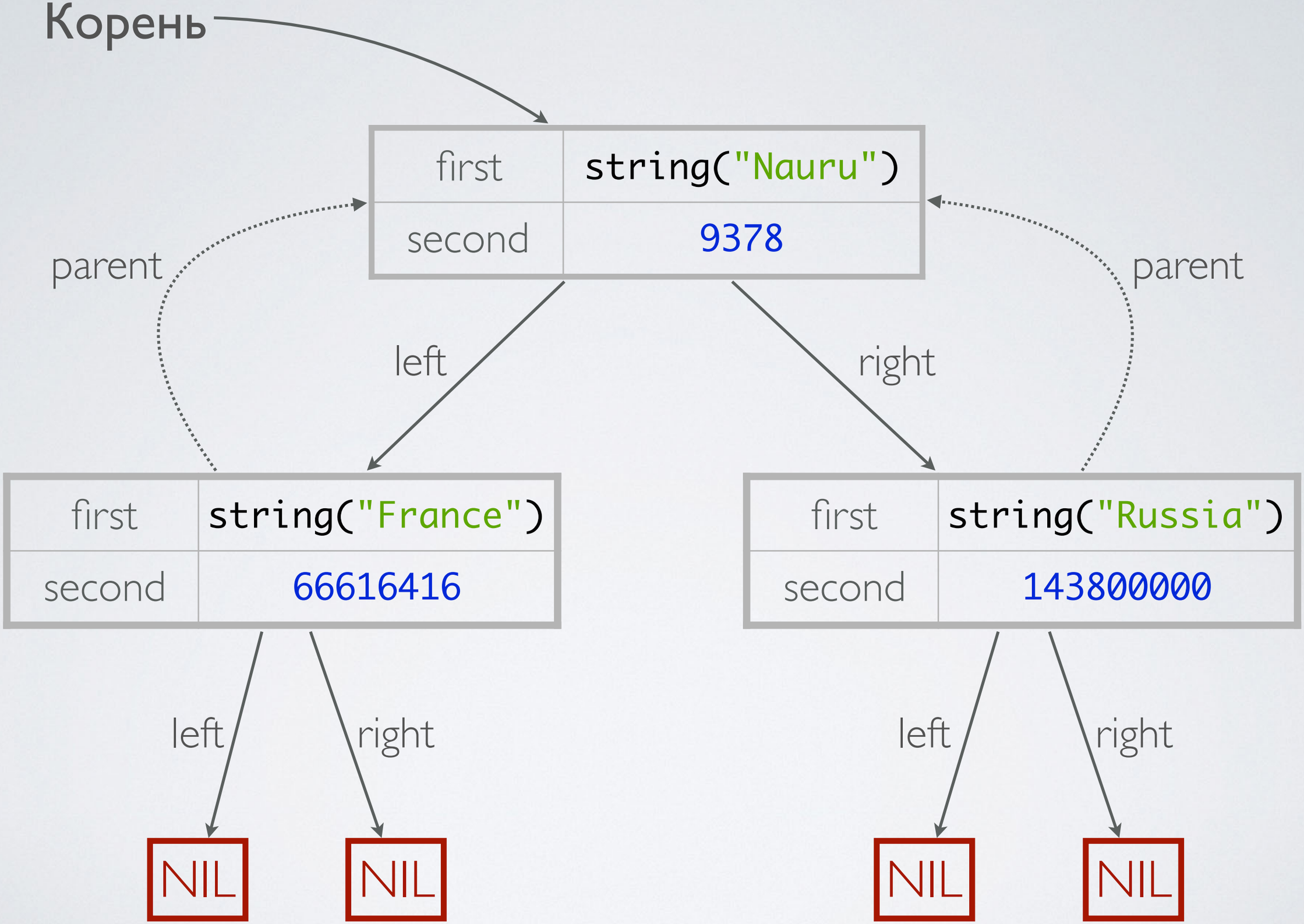
АССОЦИАТИВНЫЕ СТРУКТУРЫ

```cpp
#include <iostream>
#include <map>
#include <string>


int main() {
    std::map<std::string, int> population;

    population["Russia"] = 143800000;
    population["France"] = 66616416;
    population["Nauru"]  = 9378;

    std::string country;
    if (std::getline(std::cin, country)) {
        auto it = population.find(country);

        if (it == population.end())
            std::cout << "No data for country '" << country << "' found.\n" <<
                    "Meanwhile, Nauru population is " <<
                    population["Nauru"] << std::endl;
        else
            std::cout << it->first << " population is " << it->second
                      << std::endl;
    }

    return 0;
}
```

std::map

```
template<class T1, class T2>
struct pair {
    typedef T1 first_type;
    typedef T2 second_type;

    T1 first;
    T2 second;

    pair() : first(), second() {}

    pair(const T1 &a, const T2 &b)
      : first(a), second(b) {}

    // ...
};
```

Вспомогательный шаблонный класс std::pair


Корень
first
string("Nauru")
second
9378
parent
parent
left
right
first
string("France")
second
66616416
left
right
NIL
NIL
first
string("Russia")
second
143800000
left
right
NIL
NIL

```cpp
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<int, int> values = { { 1, 2 }, { 3, 4 }, { 0, 100 } };

    for (auto const &p: values) {
        cout << p.first << ": " << p.second << endl;
    }

    cout << "Lowest key: "  << values.begin()->first << endl;
    cout << "Highest key: " << values.rbegin()->first << endl;

    return 0;
}
```

Обход дерева

```cpp
#include <iostream>
#include <map>

using namespace std;

void f(const map<int, int> &m) {
    // ERROR: operator[] is not const!
    // cout << m[0] << endl;

    cout << m.at(0) << endl;
}

int main() {
    map<int, int> values = { { 1, 2 }, { 3, 4 }, { 0, 100 } };

    f(values);  // 100

    return 0;
}
```

operator[] и const. at()

```cpp
#include <iostream>
#include <fstream>
#include <map>
#include <cctype>

using namespace std;

string next_word(istream &is);

int main(int argc, char **argv) {
    if (argc != 2)
        return 1;

    ifstream ifs(argv[1]);

    if (!ifs)
        return 2;

    string s;
    map<string, int> counters;

    while (!(s = next_word(ifs)).empty())
        ++counters[s];

    for (auto it = counters.begin(); it != counters.end(); ++it)
        cout << it->first << ": " << it->second << endl;
}
```

Подсчёт слов

```
string next_word(istream &is) {
    // пропускаем пробелы
    while (is.good()) {
        char c = is.get();

        if (!is.good())
            break;

        if (!isspace(c)) {
            is.unget();
            break;
        }
    }

    string result;

    if (!is)
        return result;  // пустая строка

    while (is.good()) {
        char c = is.get();

        if (!is.good() || isspace(c))
            break;

        result.push_back(c);
    }

    return result;
}
```

Функция next_word()

```cpp
#include <iostream>
#include <string>
#include <map>

using namespace std;

struct Point {
    Point(double _x = 0, double _y = 0) : x(_x), y(_y) {}

    double x, y;
};

inline ostream &operator<<(ostream &os, const Point &p) {
    return os << "[" << p.x << ", " << p.y << "]";
}

int main() {
    map<string, Point> places;

    places.insert(make_pair("Bottom Left", Point(0, 0)));
    places.insert({ "Top Left", Point(0, 100) });

    // pair<iterator, bool>
    auto res = places.insert({ "Top Left", Point(-1, 100) });

    if (res.second)
        cout << "New element was inserted";
    else
        cout << "Old element was updated";

    cout << " (" << res.first->first << "): " << res.first->second << endl;

    return 0;
}
```

std::map::insert

```cpp
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<string, int> marks = {
        { "Vasya", 2 },
        { "Kolya", 3 },
        { "Petya", 4 },
        { "Sasha", 5 },
        { "Artem", 2 }
    };

    // Удаляем двоечников
    for (auto it = marks.begin(); it != marks.end(); ) {
        if (it->second < 3)
            it = marks.erase(it);
        else
            ++it;
    }

    for (const auto &p: marks)
        cout << p.first << ": " << p.second << endl;

    return 0;
}
```

std::map::erase

```cpp
#include <iostream>
#include <map>
#include <string>

using namespace std;

struct Date {
    int y, m, d;

    Date(int _y = 0, int _m = 0, int _d = 0) :
        y(_y), m(_m), d(_d) {}
};

map<Date, string> birthdays;

int main() {

    birthdays.insert({ Date(), "Haha" });

    // SYNTAX ERROR:
    // ......
    //  invalid operands to binary expression ('const Date' and 'const Date')
    //    {return __x < __y;}

    return 0;
}
```

Собственный класс в качестве ключа

```cpp
#include <iostream>
#include <map>
#include <string>

using namespace std;

struct Date {
    int y, m, d;

    Date(int _y = 0, int _m = 0, int _d = 0) :
        y(_y), m(_m), d(_d) {}
};

inline bool operator<(const Date &d1, const Date &d2) {
    return (d1.y < d2.y) || ((d1.y == d2.y) &&
        (d1.m < d2.m || (d1.m == d2.m && d1.d < d2.d)));
}

inline ostream &operator<<(ostream &os, const Date &date) {
    return os << date.y << '-' << date.m << '-' << date.d;
}

map<Date, string> birthdays;

int main() {
    birthdays.insert({ Date(1980, 7, 15), "Oleg" });
    birthdays.insert({ Date(1914, 10, 6), "Thor Heyerdahl" });
    birthdays.insert({ Date(1830, 8, 18), "Franz Joseph I." });

    for (const auto &p: birthdays)
        cout << p.first << ": " << p.second << endl;

    return 0;
}
```

Реализация operator<

```cpp
#include <map>
#include <vector>
#include <iostream>

using namespace std;
int main() {
    vector<Person> people = {
        { "Egor",    { 2013,  9, 13 } },
        { "Oleg",    { 1980,  7, 15 } },
        { "Denis",   { 1980,  7, 15 } },
        { "Tanya",   { 1982,  1,  5 } },
        { "Evgeniy", { 1982, 11,  3 } }
    };

    multimap<int, const Person *> people_by_year;

    for (const Person &p: people)
        people_by_year.insert({ p.dob.y, &p });

    auto born_1982 = people_by_year.equal_range(1982);

    cout << "Born in 1982:" << endl;
    for (auto it = born_1982.first; it != born_1982.second; ++it)
        cout << it->second->name << endl;

    return 0;
}
```

```cpp
struct Date {
    int y, m, d;

    Date(int _y = 0, int _m = 0, int _d = 0) :
        y(_y), m(_m), d(_d) {}
};

struct Person {
    string name;
    Date dob;
};
```

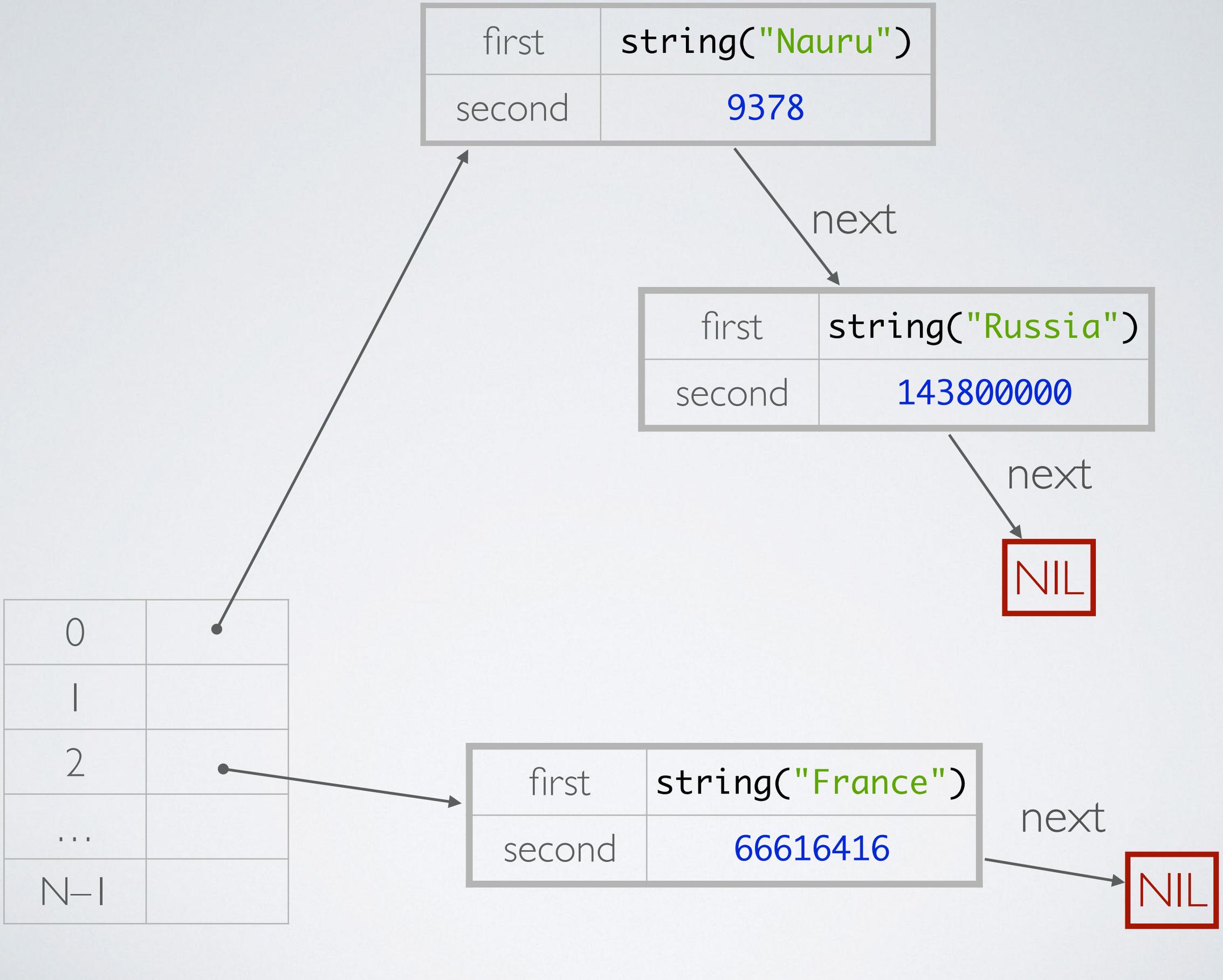
std::multimap

```cpp
#include <set>
#include <iostream>

const char *names[] = { "Vasya", "Kolya", "Vasya", "Vasya", "Petya" };

using namespace std;

int main() {
    set<string> unique_names;

    for (auto name: names)
        unique_names.insert(name);

    for (auto name: unique_names)
        cout << name << endl;          // Kolya… Petya… Vasya
}
```

std::set

```cpp
#include <iostream>
#include <unordered_map>
#include <string>

using namespace std;

int main() {
    unordered_map<string, int> population;

    population["Russia"] = 143800000;
    population["France"] = 66616416;
    population["Nauru"]  = 9378;

    string country;
    if (getline(cin, country)) {
        auto it = population.find(country);

        if (it == population.end())
            cout << "No data for country '" << country << "' found.\n" <<
                    "Meanwhile, Nauru population is " <<
                    population["Nauru"] << endl;
        else
            cout << it->first << " population is " << it->second << endl;
    }

    return 0;
}
```

std::unordered_map


first
string("Nauru")
second
9378
next
first
string("Russia")
second
143800000
next
NIL
0
1
2
…
N–1
first
string("France")
second
66616416
next
NIL

```cpp
#include <iostream>
#include <unordered_map>
#include <string>

using namespace std;

inline bool operator==(const Date &d1, const Date &d2) {
    return d1.y == d2.y && d1.m == d2.m && d1.d == d2.d;
}

template<>
struct hash<Date> {
    size_t operator()(const Date &d) const {
        return (d.y << 10) | (d.m << 5) | d.d;
    }
};
```

```cpp
struct Date {
    int y, m, d;

    Date(int _y = 0, int _m = 0, int _d = 0) :
        y(_y), m(_m), d(_d) {}
};
```

```cpp
inline ostream &operator<<(ostream &os, const Date &date) {
    return os << date.y << '-' << date.m << '-' << date.d;
}

unordered_map<Date, string> birthdays;

int main() {
    birthdays.insert({ Date(1980, 7, 15), "Oleg" });
    birthdays.insert({ Date(1914, 10, 6), "Thor Heyerdahl" });
    birthdays.insert({ Date(1830, 8, 18), "Franz Joseph I." });

    for (const auto &p: birthdays)
        cout << p.first << ": " << p.second << endl;

    return 0;
}
```

Задание хеш-функции

```
Buckets: 12853
Load factor: 0.778028
First non-empty bucket: 1, size = 8. Keys:
1 1 1 1 1 1 1 1
```

```cpp
#include <unordered_map>
#include <iostream>
#include <random>

using namespace std;

default_random_engine generator;
uniform_int_distribution<int> distribution(1, 1000);

int main() {
    unordered_multimap<int, int> values;

    for (int i = 0; i < 10000; ++i)
        values.insert({ distribution(generator), distribution(generator) });

    cout << "Buckets: " << values.bucket_count() << endl;
    cout << "Load factor: " << values.load_factor() << endl;

    size_t bucket = 0;

    while (values.bucket_size(bucket) == 0)
        ++bucket;

    cout << "First non-empty bucket: " << bucket << ", size = "
         << values.bucket_size(bucket) << ". Keys:" << endl;

    for (auto it = values.begin(bucket); it != values.end(bucket); ++it)
        cout << it->first << ' ';

    cout << endl;

    return 0;
}
```

std::unordered_multimap

```cpp
#include <unordered_set>
#include <iostream>

using namespace std;

int main() {
    unordered_set<string> elements;
    string input;

    while (getline(cin, input)) {
        if (input.empty())
            continue;

        switch (input[0]) {
        case '+':
            elements.insert(input.substr(1));
            break;

        case '-':
            elements.erase(input.substr(1));
            break;

        case '.':
            for (const auto &el: elements)
                cout << "> " << el << endl;
            break;

        default:
            cout << "Enter +something, -something, or '.'" << endl;
        }
    }

    return 0;
}
```

```
l
Enter +something, -something, or '.'
+abc
+abc
+def
+def
.
> def
> abc
-def
.
> abc
```

std::unordered_set

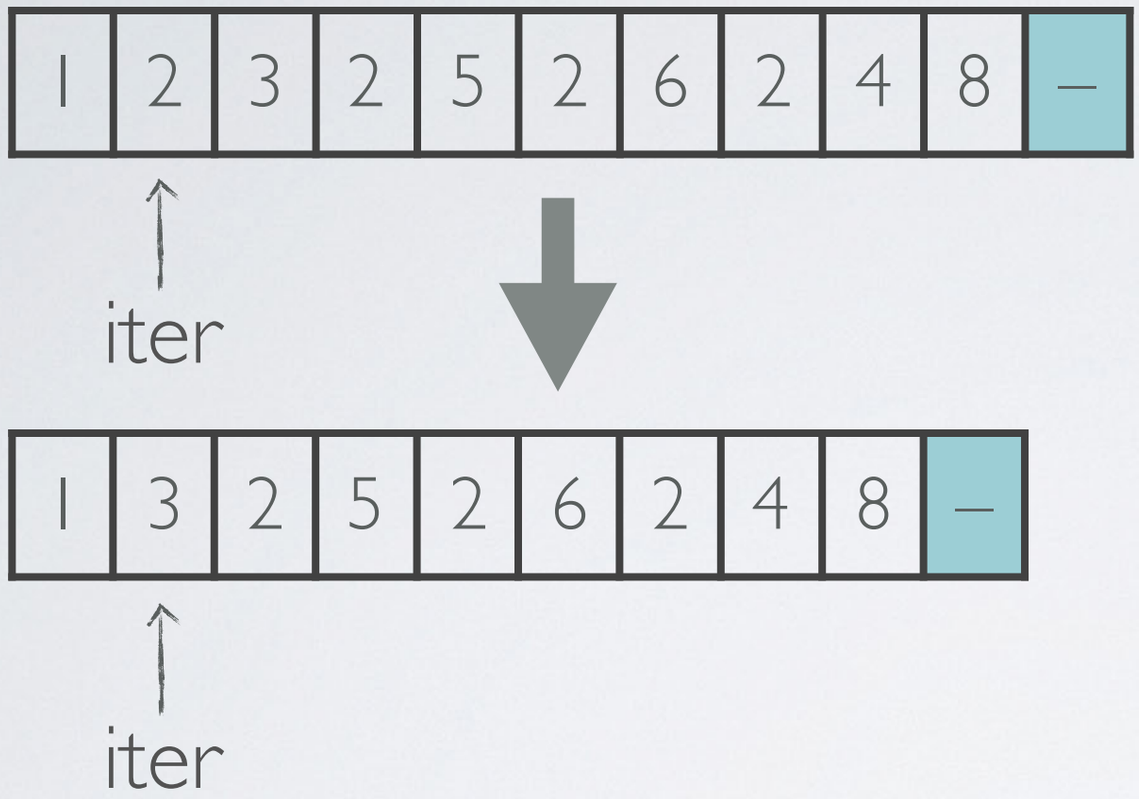
ERASE-REMOVE IDIOM ON STD::VECTOR

Bad code!

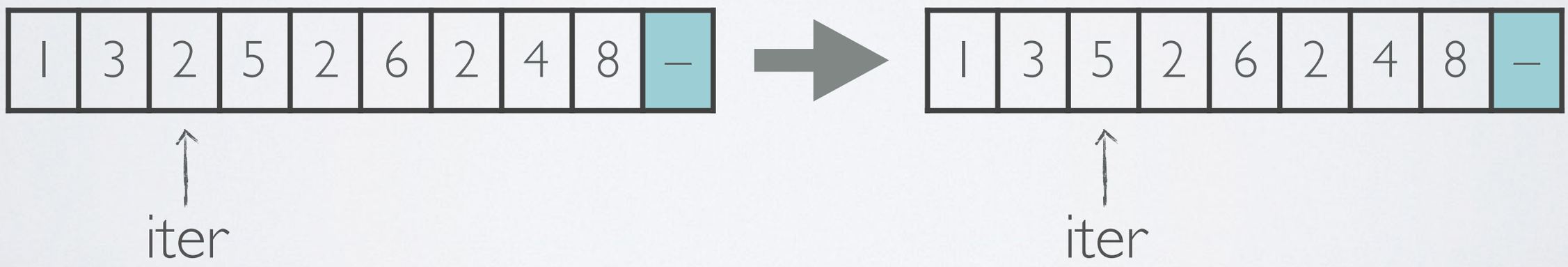
```
#include <iostream>
#include <vector>
#include <algorithm>

int main()
{
   std::vector<int> v{ 1, 2, 3, 2, 5, 2, 6, 2, 4, 8 };
   for (std::vector<int>::iterator iter = v.begin();
             iter !=v.end(); ) {
      if (*iter == 2) {
         iter = v.erase(iter);
         continue;
      }
      ++iter;
   }

   return 0;
}
```

Complexity of erase: O(n)

ERASE OPERATION



iter = v.erase(iter);



iter = v.erase(iter);

ERASE-REMOVE IDIOM ON STD::VECTOR

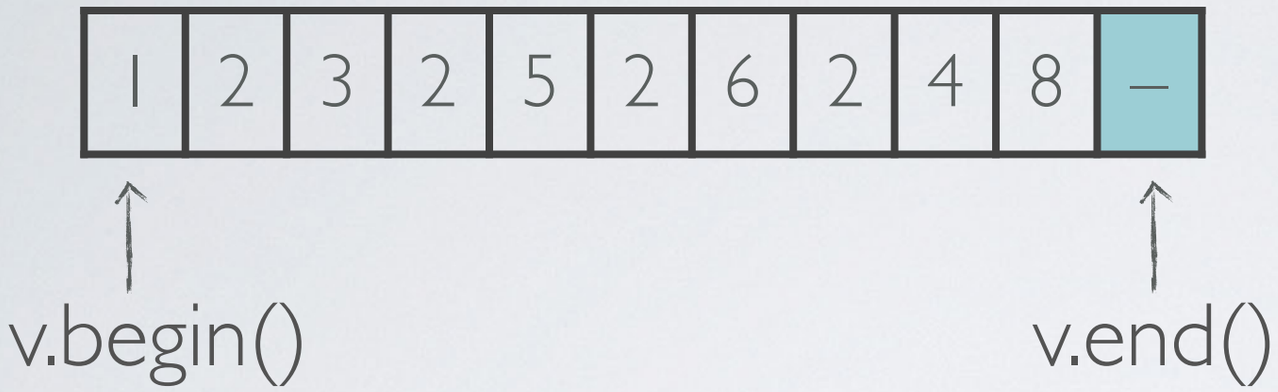
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

int main()
{
    std::vector<int> v{ 1, 2, 3, 2, 5, 2, 6, 2, 4, 8 };

    const auto new_end = std::remove(v.begin(), v.end(), 2);
    v.erase(new_end, v.end());

    ...

    return 0;
}
```

Good code!

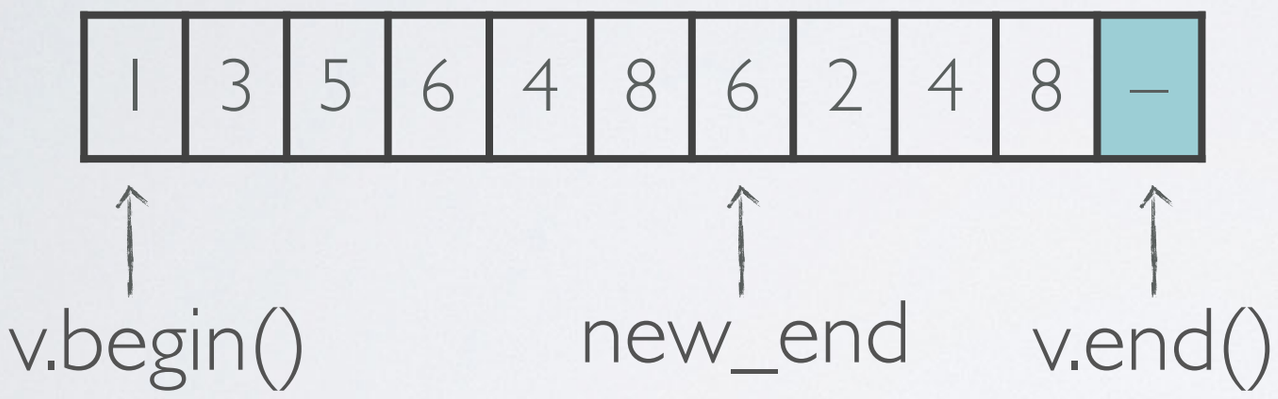
Console:

1, 3, 5, 6, 4, 8

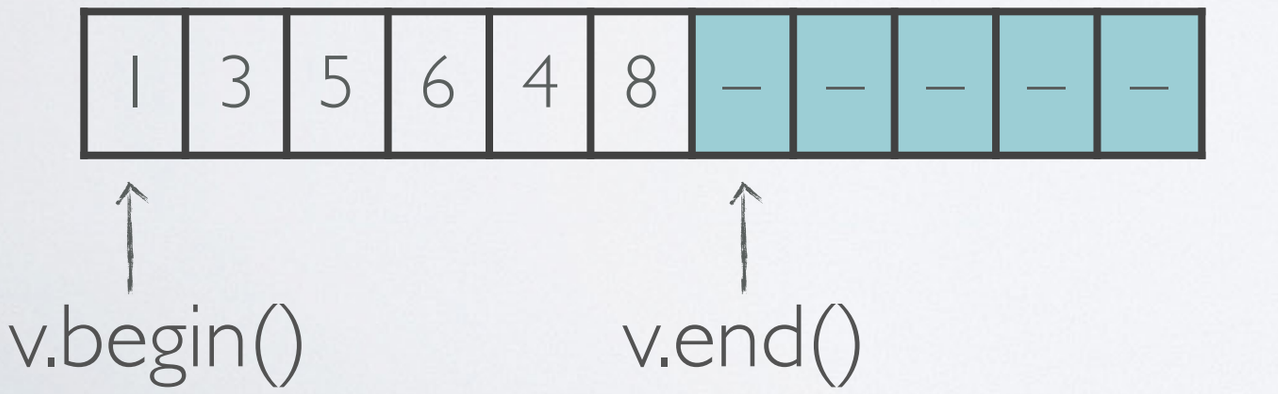
ERASE-REMOVE IDIOM ON STD::VECTOR



Initial state



new_end = remove(begin, end, 2);



v.erase(new_end, end);

ERASE-REMOVE IDIOM ON STD::VECTOR

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

bool odd(int i) { return i % 2 != 0; }

int main()
{
   std::vector<int> v{ 1, 2, 3, 2, 5, 2, 6, 2, 4, 8 };

   const auto new_end = std::remove_if(v.begin(), v.end(), odd);
   v.erase(new_end, v.end());

   ...

   return 0;
}
```

Console:

2, 2, 2, 6, 2, 4, 8

FILTERING DUPLICATES

```
#include <iostream>
#include <set>
#include <string>
#include <iterator>

int main()
{
   std::set<std::string> set;

   for(std::string str; cin >> str; set.insert(str)){}

   ...

   return 0;
}
```

FILTERING DUPLICATES

```
#include <iostream>
#include <set>
#include <string>
#include <iterator>

int main()
{
   std::set<std::string> set;
   std::istream_iterator<std::string> it {std::cin};
   std::istream_iterator<std::string> end;

   std::copy(it, end, std::inserter(set, set.end()));

   ...

   return 0;
}
```

FAST OR SAFE WAY TO ACCESS STD::ARRAY

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

int main()
{
   const size_t container_size {1000};
   std::array<int, container_size> arr;

   std::iota(std::begin(arr), std::end(arr), 0);

   std::cout << "Out of range element value: "
             << arr[container_size + 10] << "\n";
   ...

   return 0;
}
```


No bounds checking.

FAST OR SAFE WAY TO ACCESS STD::ARRAY

```cpp
int main()
{
   const size_t container_size {1000};
   std::array<int, container_size> arr;
   ...

   try{
      std::cout << "Out of range element value: "
                << arr.at(container_size + 10) << "\n";
   } catch (const std::out_of_range &e) {
      std::cout << "Ooops, out of range access detected: "
                << e.what() << "\n";
   }

   return 0;
}
```

Bounds checking.

WORD FREQUENCY COUNTER

```
std::string filter_punctuation(const std::string &s)
{
   const char* forbidden{ ".,:; " };
   const size_t idx_start(s.find_first_not_of(forbidden));
   const size_t idx_end(s.find_last_not_of(forbidden));

   return s.substr(idx_start, idx_end - idx_start + 1);
}
```

WORD FREQUENCY COUNTER

```
int main()
{
   std::map<std::string, size_t> words;
   int max_word_len{ 0 };

   std::string s;
   while (std::cin >> s) {
      auto filtered(filter_punctuation(s));
      max_word_len = std::max<int>(max_word_len, filtered.length());
      ++words[filtered];
   }
   ...
}
```

WORD FREQUENCY COUNTER

```
int main()
{
   ...
   std::vector<std::pair<std::string, size_t>> word_counts;
   word_counts.reserve(words.size());

   std::move(std::begin(words), std::end(words),
                                std::back_inserter(word_counts));
   ...
}
```

WORD FREQUENCY COUNTER

```
bool isGreater(const std::pair<std::string, size_t>& a,
               const std::pair<std::string, size_t>& b){
   return a.second > b.second;
}
```

```
int main()
{
   ...

   std::sort(word_counts.begin(), word_counts.end(), isGreater);
   ...
}
```

WORD FREQUENCY COUNTER

```
int main()
{
   ...

   std::cout << "# " << std::setw(max_word_len) << "<WORD>"
             << " #<COUNT>\n";

   for (const auto &[word, count] : word_counts) {
      std::cout << std::setw(max_word_len + 2) << word << " #"
                << count << '\n';
   }
}
```

Structured bindings (since C++17)

КОНЕЦ ШЕСТОЙ ЛЕКЦИИ

```
lections.insert({
    "Lection 6",
    "Лекция № 6. Стандартная библиотека C++. Часть 2"
});
```