Лекция по C++

STL

Streams

```cpp
#include <iostream>

int main() {
    int a, b;
    std::cin >> a >> b;
    std::cout << a + b;
}
```

Streams

```cpp
#include <iostream>
#include <sstream>

int main() {
    int a;
    std::string s;
    std::stringstream ss;
    ss << 1 << 2 << "hsadkjf";
    ss >> a >> s; // a = 12, s = "hsadkjf"
}
```

Streams

```
#include <iostream>

int main() {
    ...
    if (a == 0) {
        std::cerr << "a = 0 !!!!!";
        return 0;
    }
}
```

Streams

Хотим выводить поля структуры в std:out? Или логировать в std::cerr?

```
#include <iostream>
#include <sstream>

struct MyStruct {
    int a, b;
    MyStruct(int a, int b) : a(a), b(b) {}
};
```

Streams

```
std::ostream& operator<<(std::ostream& s,
        const MyStruct& m) {
    s << m.a + m.b;
    return s;
}
int main() {
    MyStruct a(1, 2);
    std::cout << a;
    std::stringstream ss;
    ss << a;
    std::cerr << a;
}
```

Introduction to templates

```
int Min(int a, int b) {
    if (a < b) return a;
    return b;
}

int main() {
    auto res = Min(2, 3);
}
```

Если мы теперь хотим минимум для даблов?

Introduction to templates

```
double Min(double a, double b) {
    if (a < b) return a;
    return b;
}

int main() {
    auto res = Min(1.5, 3.5);
}
```

Для строк?...

Introduction to templates

```
template <class T>
T Min(T a, T b) {
    if (a < b) return a;
    return b;
}

int main() {
    auto res1 = Min(1, 2);
    auto res2 = Min<double>(10, 1.4);
    auto res3 = Min("asasa", "aaab");
}
```

Introduction to templates

```
struct MyStruct {
    int a, b;
    MyStruct(int a, int b) : a(a), b(b) {}
};
template <class T>
T Min(T a, T b) {
    return a < b ? a : b;
}
int main() {
    MyStruct a(1, 2), b(3, 1);
    auto res1 = Min(a, b);
}
```

Introduction to templates

```
struct MyStruct {
    int a, b;
    MyStruct(int a, int b) : a(a), b(b) {}
    bool operator<(const MyStruct& rhs) const {
        if (a == rhs.a) return b < rhs.b;
        return a < rhs.a;
    }
};
```

Template struct

```
template <typename T>
struct MyStruct {
    T a, b;
    MyStruct(T a, T b) : a(a), b(b) {}
    bool operator<(const MyStruct& rhs) const {
        if (a == rhs.a) return b < rhs.b;
        return a < rhs.a;
    }
};

int main() {
    MyStruct<int> a(1, 2), b(3, 1);
}
```

Template struct

```
template <typename T>
struct MyStruct {
    T a, b;
    MyStruct(T a, T b) : a(a), b(b) {}
    bool operator<(const MyStruct& rhs) const {
        if (a == rhs.a) return b < rhs.b;
        return a < rhs.a;
    }
};
int main() {
    MyStruct<int> a(1, 2);
    MyStruct<long double> b(3, 1);
    a < b;
}
```

Template struct

```
int main() {
    MyStruct<int> a(1, 2);
    MyStruct<long double> b(3, 1);
    a < b;
}
```

error: invalid operands to binary expression ('MyStruct<int>' and 'MyStruct<long double>')

Template struct

```
template <typename T>
struct MyStruct {
    T a, b;
    MyStruct(T a, T b) : a(a), b(b) {}
    template <class R>
    bool operator<(const MyStruct<R>& rhs) const {
        if (a == rhs.a) return b < rhs.b;
        return a < rhs.a;
    }
};
```

Stl containers: vector

```
#include <vector>
```

Complexity:
Добавление/удаление произвольного элемента - $O(n)$
Добавление/удаление в конец - $O(1)$
Доступ к произвольному элементу - $O(1)$
Поиск элемента - $O(n)$

Stl containers: vector

```cpp
#include <vector>

int main() {
    std::vector<int> a;
    a.push_back(1);
    a.push_back(2);
    std::vector<int> b(a.begin(), a.end());
    std::vector<int> c({1, 3, 4});
}
```

Stl containers: vector

```cpp
#include <vector>

int main() {
    std::vector<int> a({1, 2, 3, 4, 5});
    std::cout << a.front() << std::endl; // 1
    std::cout << a.back() << std::endl; // 5
    std::cout << *(a.begin()) << std::endl; // 1
    std::cout << *(a.begin() + 2) << std::endl; // 3
    std::cout << *(a.end()) << std::endl; // ub
}
```

Stl containers: vector

```cpp
#include <vector>

int main() {
    std::vector<int> c(100);
    // vector of size 100, filled with 0
    std::vector<int> a(100, 1);
    // vector of size 100, filled with 1
    std::vector<std::vector<double>> b(10,
        std::vector<double>(5, 0.5));
    // vector of doubles of size 10 x 5, filled with 0.5
}
```

Stl containers: vector

```cpp
#include <vector>

int main() {
    std::vector<int> a({1, 2, 3, 4, 5});
    a.resize(3); // a = [1, 2, 3]
    a.resize(5, 1); // a = [1, 2, 3, 1, 1]
    a.assign(5, 1); // a = [1, 1, 1, 1, 1]
    a.clear();
    assert(a.empty());
}
```

Stl containers: vector

```cpp
#include <vector>

int main() {
    std::vector<MyStruct> a; // ok
    a.resize(3); // a = [1, 2, 3]
    a.resize(5, 1); // a = [1, 2, 3, 1, 1]
    a.assign(5, 1); // a = [1, 1, 1, 1, 1]
    a.clear();
    assert(a.empty());
}
```

Stl containers: vector

```
struct MyStruct {
    int a, b;
    MyStruct(int a, int b) : a(a), b(b) {}
    bool operator<(const MyStruct& rhs) const {
        if (a == rhs.a) return b < rhs.b;
        return a < rhs.a;
    }
};
int main() {
    std::vector<MyStruct> a; // ok
    std::vector<MyStruct> b(100);
        // error: no default constructor
    std::vector<MyStruct> c(100, MyStruct(0, 0)); // ok
}
```

Stl containers: vector

```
int main() {
    std::vector<std::pair<int, int>> b;
    b.push_back(std::make_pair(1, 2));
    b.emplace_back(1, 2);

    std::vector<MyStruct> a;
    a.push_back(MyStruct(1, 2));
    a.emplace_back(1, 2);
}
```

Stl containers: set, map

```
#include <set>
#include <map>
```

Set/map - красно-черное дерево
Добавление/удаление элемента - $O(logn)$
Поиск элемента - $O(logn)$
Хранятся в порядке сортировки

Stl containers: set, map

Set - множество уникальных элементов.

```cpp
#include <set>

int main() {
    std::set<int> a({3, 1, 1, 6});
    for (auto el : a) std::cout << el << " "; // 1 3 6

    a.erase(2); // ничего не произошло
    a.erase(a.begin()); // a = {3, 6}

    a.insert(2); // a = {2, 3, 6}
    a.insert(2); // ничего не произошло: a = {2, 3, 6}
}
```

Stl containers: set, map

```cpp
#include <set>

int main() {
    std::vector<int> b({1, 2, 5, 7, 0});
    std::set<int> c(b.begin(), b.end());

    std::set<int> a({3, 1, 1, 6});
    for (auto el : a) std::cout << el << " ";
    // если вам нужен итератор:
    for (auto it = a.begin(); it != a.end(); ++it) {
        std::cout << *it << " ";
    }
}
```

Stl containers: set, map

lower_bound - первый элемент, не меньше X
upper_bound - первый элемент, больше X

```cpp
#include <set>

int main() {
    std::set<int> a({1, 3, 5, 7, 9});
    auto b = a.lower_bound(4);
    std::cout << *b; // 5
}
```

Stl containers: set, map

set $<$ *T* $>$ - у *T* должен быть оператор $<$

```
struct MyStruct {
    int a, b;
    MyStruct(int a, int b) : a(a), b(b) {}
    bool operator<(const MyStruct& rhs) const {
        if (a == rhs.a) return b < rhs.b;
        return a < rhs.a;
    }
};

int main() {
    std::set<MyStruct> a; // ok
}
```

Stl containers: set, map

map - контейнер с ключами и значениями

```
#include <map>

int main() {
    std::map<std::string, int> a({
        {"aba", 2},
        {"aac", 17},
        {"za", 1},
    });
    for (auto el : a) {
        std::cout << el.first << " " <<
            el.second << std::endl;
    }
}
```

Stl containers: set, map

```cpp
int main() {
    std::map<std::string, int> a;
    a["aa"] = 1;
    a["ss"] = 2;
    a["ss"] = 3;
    // a : {"aa" : 1, "ss" : 3}

    std::map<std::string, int> b;
    b.emplace("aa", 1);
    b.emplace("ss", 2);
    b.emplace("ss", 3);
    // b : {"aa" : 1, "ss" : 2}
}
```

Stl containers: set, map

[] вызывают дефолтный конструктор!

```
int main() {
    std::map<std::string, int> a;
    std::cout << a.size() << std::endl; // 0
    if (a["aa"] == 3) {
        // nothing
    }
    std::cout << a.size() << std::endl; // 1
}
```

Stl containers: set, map

```
std::map<std::string, int> a;
std::cout << a.size() << std::endl; // 0
if (a.count("aa")) {
    a.at("aa") = 1;
}
std::cout << a.size() << std::endl; // 0
```

Stl containers: set, map

Мультимножества - *multimap*, *multiset*

```
std::multiset<int> a{1, 2, 2, 5};
```

Initializer list

```
std::vector<int> a({1, 2, 2, 5});
    // {..} - список инициализации
// можно создать его явно
std::initializer_list<int> b{1, 2, 3};
// Потом от него можно создать какой-нибудь контейнер
std::vector<int> c(b);
// Больше он ничего не умеет :)
```

Stl containers: unordered_set, unordered_map

```
#include <unordered_set>
#include <unordered_map>
```

unordered_set/*unordered_map* - хеш-таблица
Добавление/удаление элемента - $O(1)$
Поиск элемента - $O(1)$

Stl containers: unordered_set, unordered_map

Unordered структуры умеют почти то же самое, что и *set*/*map*

```
int main() {
    std::unordered_set<int> b{1, 2, 3};
    std::cout << b.count(2) << std::endl;
    b.insert(17);

    // you will never need that
    b.rehash(100);
    std::cout << b.load_factor() << std::endl;
}
```

Stl containers: unordered_set, unordered_map

unordered_set от своей структуры

```
template<>
struct std::hash<MyStruct> {
    size_t operator()(const MyStruct& a) const {
        return a.a * 41 + a.b;
    }
};

int main() {
    std::unordered_set<MyStruct> a;
}
```

Stl containers: list, forward_list

```cpp
#include <list>
#include <forward_list>
int main() {
    std::list<int> l({1, 2, 3, 4, 5});
    l.pop_back();
    l.pop_front();
    l.push_back(17);
    l.push_front(7);
    for (auto el : l) std::cout << el << " ";
    // 7 2 3 4 17

    std::forward_list<int> fl({1, 2, 3, 4, 5});
    fl.pop_front();
    fl.push_front(7);
    for (auto el : fl) std::cout << el << " ";
    // 7 2 3 4 5
}
```

Stl containers: queue, stack, deque

```
#include <stack>
#include <queue>
```

queue - добавление в конец, удаление из начала за O(1)
stack - добавление в конец, удаление с конца за O(1)
deque - добавление/удаление в конец, добавление/удаление в начало, random access доступ за O(1)

Stl containers: queue, stack, deque

```
int main() {
    std::queue<int> q;
    q.push(2);
    int a = q.front();
    q.pop();

    std::stack<int> s;
    s.push(2);
    int b = s.top();
    s.pop();
}
```

Stl containers: queue, stack, deque

```
int main() {
    std::deque<int> d;
    d.push_back(2);
    d.push_front(2);

    int a = d.front();
    int b = d.back();
    int c = d[0];

    d.pop_back();
    d.pop_front();
}
```

```
#include <priority_queue>
```

priority_queue - куча
Добавление за $O(logn)$
Получить максимум/минимум за $O(1)$

```cpp
int main() {
    std::priority_queue<int> p;
    p.emplace(3);
    p.emplace(1);
    p.emplace(7);
    std::cout << p.top(); // 7
}
```

```cpp
int main() {
    std::priority_queue<int, std::vector<int>,
        std::greater<int>> p;
    p.emplace(3);
    p.emplace(1);
    p.emplace(7);
    std::cout << p.top() << std::endl; // 1

    std::vector<int> a({1, 2, 3});
    std::priority_queue<int, std::vector<int>,
        std::less<int>> q(std::less<int>(), a);
    std::cout << q.top() << std::endl; // 3
    return 0;
}
```

priority_queue - контейнер-адаптер. Ей можно передать другой контейнер, на котором она построится (в данном случае вектор)

Containers-adapters

priority_queue
stack
queue

Stl algorithms

```cpp
#include <algorithm>
int main() {
    std::vector<int> a({1, 2, 4, 0, 7, 2});
    std::sort(a.begin(), a.end()); // 0 1 2 2 4 7
    std::sort(a.rbegin(), a.rend()); // 7 4 2 2 1 0

    std::vector<int> b({5, 2, 6, 3, 1, 0, 0, 8, 7});
    auto it = b.begin();
    std::nth_element(b.begin(), it + 5, b.end());
    for (auto el : b) std::cout << el << " ";
        // 1 2 0 3 0 !5! 6 8 7
}
```

rbegin, rend - reverse iterators

Lambda функции

```
struct MyStruct {
    int a, b;
    MyStruct(int a, int b) : a(a), b(b) {}
};
int main() {
    std::vector<MyStruct> a;
    for (int i = 0; i < 100; ++i) a.emplace_back(
        rand() % 100, rand() % 100);

    auto my_less = [] (const MyStruct& lhs,
            const MyStruct& rhs) -> bool {
        if (lhs.a == rhs.a) return lhs.b > rhs.b;
        return lhs.a < rhs.a;
    };
    std::sort(a.begin(), a.end(), my_less);
}
```

Lambda функции

```
[/* замыкание */] (/* аргументы */) -> return_type {
    /* тело */
};
```

Lambda функции

```
int main() {
    auto my_min = [] (int a, int b) {
        if (a < b) return a;
        return b;
    };
    std::cout << my_min(2, 5);
}
```

Lambda функции

```
int main() {
    int t = 4;
    auto add_t = [] (int a) {
        return a + t;
    };
    std::cout << add_t(7);
}
```

error: variable 't' cannot be implicitly captured in a lambda with no capture-default specified

Lambda функции

```
int main() {
    int t = 4;
    auto add_t = [t] (int a) {
        return a + t;
    };
    std::cout << add_t(7);
}
```

Lambda функции

```
int main() {
    int t = 4, q = 4;
    auto inc = [&t, q] () {
        ++t;
        ++q;
    };
    std::cout << t << " " << q;
}
```

error: cannot assign to a variable captured by copy in a non-mutable lambda

Lambda функции

```
int main() {
    int t = 4, q = 4;
    auto inc = [&t, q] () {
        ++t;
        ++q;
    };
    inc();
    std::cout << t << " " << q;
}
```

error: cannot assign to a variable captured by copy in a non-mutable lambda

Lambda функции

```
int main() {
    int t = 4, q = 4;
    auto inc = [&t, q] () mutable {
        ++t;
        ++q;
    };
    inc();
    std::cout << t << " " << q; // 5 4
}
```

Lambda функции

```
int main() {
    MyStruct some_long_struct_name(1, 2);
    auto print = [&a = some_long_struct_name] () {
        std::cout << a.a << " " << a.b;
    };
    print();
}
```

Lambda функции

```
int main() {
    auto a = [=](){};
    auto b = [&](){};
    [](){}();

    [](){ std::cout << ":)"; }(); // :)
}
```

std::function

```
template<>
struct std::less<int> {
    bool operator()(int a, int b) {
        return a < b;
    }
};

int main() {
    auto my_less = [](int a, int b){ return a < b; };
    bool(*p)(int, int);
    std::function<bool(int, int)> func1 = my_less;
    std::function<bool(int, int)> func2 = std::less<int>();
    std::function<bool(int, int)> func3 = p;
}
```

Stl algorithms

```cpp
int main() {
    std::vector<int> a({5, 2, 6, 3});
    auto even = [](int num) {
        return num % 2 == 0;
    };

    auto it = std::find_if(a.begin(), a.end(), even);
    std::cout << *it << std::endl; // 2

    it = std::find_if_not(a.begin(), a.end(), even);
    std::cout << *it << std::endl; // 5
}
```

Exceptions

```
int main() {
    int a;
    std::cin >> a;
    try {
        if (a == 0) {
            throw std::runtime_error("zero!");
        }
    } catch (std::runtime_error& e) {
        std::cout << e.what();
        return 0;
    }
}
```

Exceptions

```
int main() {
    int a;
    std::cin >> a;
    try {
        if (a == 0) throw std::logic_error("zero!");
    }
    catch (std::runtime_error& e) {
        std::cout << e.what();
        return 0;
    }
    catch (std::logic_error& e) {
        std::cout << "logic error";
        return 0;
    }
}
```

Exceptions

```
class MyException : public std::exception {
    std::string what_str;
public:
    MyException(std::string what_str) : what_str(what_str) {
    const char* what() const noexcept override {
        return what_str.c_str();
    }
};
```

Exceptions

```
int main() {
    int a;
    std::cin >> a;
    try {
        if (a == 0) throw MyException("zero!");
    }
    catch (std::exception& e) {
        std::cout << e.what();
        return 0;
    }
}
```