Шаблонные функции

vector

Лямбда-функции

Библиотека algoritm

Лекция #7

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

Шаблонные функции

Шаблонные функции

```
#include <iostream>

void my_swap(int & first, int & second)
{
        int temp(first);
        first = second;
        second = temp;
}
int main()
{
        int a = 5;
        int b = 10;
        std::cout << a << " " << b << std::endl;

        my_swap(a, b);

        std::cout << a << " " << b << std::endl;
}
```

Что делать, если нужна такая же функция для переменных типа double?

Использовать **шаблоны**

Шаблонные функции

Шабло́ны (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

```
template < параметры_шаблона > описание_функции
```

Шаблонные функции

- Все шаблоны функций начинаются со слова **template**, после которого идут угловые скобки, в которых перечисляется список параметров.
- Каждому параметру должно предшествовать зарезервированное слово **class** или **typename**.
  - `template <class T>`
  - `template <typename T>`
  - `template <typename T1, typename T2>`

Шаблонные функции. Пример 1. Часть 1

```
template < typename T >
void my_swap(T & first, T & second)
{
        T temp(first);
        first = second;
        second = temp;
}
```

typename в угловых скобках означает, что параметром шаблона будет тип данных.

T — имя параметра шаблона.

Вместо **typename** здесь можно использовать слово **class**:

template < class T >

В данном контексте ключевые слова **typename** и **class** эквивалентны.

Шаблонные функции. Пример 1. Часть 2

```
int main()
{
	int a = 5;
	int b = 10;
	std::cout << a << " " << b << std::endl;
	my_swap<int>(a, b);
	std::cout << a << " " << b << std::endl;

	double c = 77.89;
	double d = 54.22;
	std::cout << c << " " << d << std::endl;
	my_swap<double>(c, d);
	std::cout << c << " " << d << std::endl;
}
```

Шаблонные функции. Пример 1. Часть 3

- **Шаблон** — это лишь **макет**, по которому компилятор самостоятельно будет генерировать код.
- При виде такой конструкции: **my_swap<тип>** компилятор сам создаст функцию my_swap с необходимым типом. Это называется **инстанцирование** шаблона.
- То есть при виде my_swap<int> компилятор создаст функцию my_swap в которой **T** поменяет на **int**, а при виде my_swap<double> будет создана функция с типом double.
- Если где-то дальше компилятор опять встретит my_swap<int>, то он ничего генерировать не будет, т.к. код данной функции уже есть(шаблон с данным параметром уже инстанцирован).

Шаблонные функции. Пример 1. Часть 4

```
int main()
{
	int a = 5;
	int b = 10;
	std::cout << a << " " << b << std::endl;
//В ряде случаев компилятор может самостоятельно определить тип
	my_swap(a, b);
	std::cout << a << " " << b << std::endl;

	double c = 77.89;
	double d = 54.22;
	std::cout << c << " " << d << std::endl;
//В ряде случаев компилятор может самостоятельно определить тип
	my_swap(c, d);
	std::cout << c << " " << d << std::endl;
}
```

Шаблонные функции. Пример 2. Часть 1

```
#include <iostream>
using namespace std;
// перегрузка функции printArray для вывода массива на экран
void printArray(const int * arr, int count)
{
	for (int i = 0; i < count; i++)
		cout<<" "<< arr[i];
	cout << endl;
}

void printArray(const char * arr, int count)
{
	for (int i = 0; i < count; i++)
		cout << " " << arr[i];
	cout << endl;
}
```

Шаблонные функции. Пример 2. Часть 2

```
#include <iostream>
using namespace std;

// шаблон функции printArray
template <typename T>
void printArray(const T * arr, int count)
{
	for (int i = 0; i < count; i++)
		cout << " " << arr[i];
	cout << endl;

} // конец шаблона функции printArray
```

Шаблонные функции. Пример 2. Часть 3

```
int main()
{
        int ia[5] = { 1, 2, 3, 4, 5 };
        float fa[3] = {1.3,2.4,3.5};
        char ca[3] = { 'a','b','c' };

        printArray(ia,5);
        printArray(fa, 3);
        printArray(ca, 3);
return 0;
}
```

Шаблонные функции. Пример 3

```
#include <iostream>
using namespace std;

template <typename T1, typename T2 >
auto sumTwo(const T1 &x, const T2 &y)
{
            return x + y;
}

int main()
{
       cout << sumTwo(2, 3) << endl;     //5
       cout << sumTwo(2.5, 3.5) << endl;//6
       cout << sumTwo(2, 3.5) << endl;  //5.5
       cout << sumTwo('a','b') << endl; //195
return 0;
}
```

Шаблонные функции. Пример 3. Часть 2

```
#include <iostream>
using namespace std;

template <class T1, class T2 >
auto sumTwo(const T1 &x, const T2 &y)
{
            return x + y;
}

int main()
{
       cout << sumTwo(2, 3) << endl;     //5
       cout << sumTwo(2.5, 3.5) << endl;//6
       cout << sumTwo(2, 3.5) << endl;  //5.5
       cout << sumTwo('a','b') << endl; //195
return 0;
}
```

Библиотека vector

Библиотека vector

Вектор (**vector**) почти то же самое, что массив, к его элементам можно обращаться также как к элементам массива.

Преимущества векторов

- Не нужно специально следить за размером, так как вектор свой размер знает.
- Чтобы поменять размер достаточно удалить или добавить элементы во время выполнения

Чтобы использовать вектор в программе, нужно подключить библиотеку **vector** и указать используемое пространство имен **std**

Библиотека vector. Функции для работы с памятью

- v.size() - размер вектора,
- v.max_size() - максимальный размер вектора,
- v.reserve(n) - установить минимальный размер выделенной памяти на вектор,
- v.resize(n) - установить размер n для вектора,
- v.capacity() - количество свободной памяти, выделенной под вектор.

Библиотека vector. Работа с элементами

- v.push_back(e) - добавить элемент в конец
- v.pop_back() - удалить последний элемент
- v.front() - первый элемент вектора
- v.back() - последний элемент вектора
- v.insert(i, e) - вставка элемента в позицию i,
- v.erase(myV2) - удаление последовательности элементов
- v[i] или v.at(i) - доступ к элементу i вектора.

Библиотека vector. Функции для работы указателями

- v.begin() - указатель на начало вектора
- v.end() - указатель на конец вектора
- v.rbegin() - реверсивный указатель на конец вектора
- v.rend() - реверсивный указатель на начало вектора

Функции для работы с объектами

- v.clear() - очистка вектора
- v1.swap(v2) – обмен содержимым

Библиотека vector. Инициализация нулями

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
// вектор из 10 элементов, которые инициализируются нулями
	vector<int> myVector(10);

// вывод элементов вектора на экран
	for (int i = 0; i < myVector.size(); i++)
		cout << myVector[i] << ' ';

return 0;
}
```

Библиотека vector. Инициализация одинаковыми значениями

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
// вектор из 10 элементов, которые инициализируются 7
	vector<int> myVector(10,7);

// вывод элементов вектора на экран
	for (int i = 0; i < myVector.size(); i++)
		cout << myVector[i] << ' ';

return 0;
}
```

Библиотека vector. reserve – выделение памяти

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
// объявление вектора
	vector<int> myVector;
// выделение памяти под 10 элементов
	myVector.reserve(10);

// вывод элементов вектора на экран
for (int i = 0; i < myVector.size(); i++)
	cout << myVector[i] << ' ';

return 0;
}
```

Библиотека vector. Копирование вектора

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
vector<int> myVector1(10);

// Входной массив:
for (int i = 0; i < myVector1.size(); i++) {
                myVector1[i] = i;
}

// Скопированный массив:
      vector<int> myVector2(myVector1);

return 0;
}
```

Библиотека vector. push_back – добавление в конец

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        // вектор из 3-х  нулевых элементов
        vector<int> v(3);

        // добавление в конец
        v.push_back(10);
        v.push_back(20);
        v.push_back(30);


        for (int i = 0; i < v.size(); i++)
                        cout << v[i] << ' ';

return 0;
}
```

Библиотека vector. front() – первый; back() - последний

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector<int> v;
        // добавление в конец
        v.push_back(10);
        v.push_back(20);
        v.push_back(30);

        // первый и последний элементы
        cout << v.front() << ' '<<v.back();

return 0;
}
```

Библиотека vector. insert - вставка

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector<int> v;
        // добавление в конец
        v.push_back(10);

        v.insert(v.begin(), 1024);

        // 1024 10
        cout << v.front() << ' '<<v.back();

return 0;
}
```

Библиотека vector. end() – указатель на конец

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector<int> v;
        // добавление в конец
        v.push_back(10);

        v.insert(v.end(), 1024);

        // 10 1024
        cout << v.front() << ' '<<v.back();

return 0;
}
```

Библиотека vector. begin() – указатель на начало

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector<int> v;
        // добавление в конец
        v.push_back(10);
        v.push_back(20);

        v.insert(v.begin()+1, 1024);

        // 10 20
        cout << v.front() << ' '<<v.back();

return 0;
}
```

Библиотека vector. rbegin() - реверсивный указатель на начало

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector<int> v(3);
        v.push_back(10);
        v.push_back(20);
        v.push_back(30);

        cout << *(v.rbegin()) ;   //30
        cout << *(v.rbegin() +1 );//20
        cout << *(v.rbegin() + 2);//10

return 0;
}
```

Библиотека vector. rend() - реверсивный указатель на конец

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
	vector<int> v(3);
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);

	cout << *(v.end()-1); //30
	cout << *(v.end()-2); //20
	cout << *(v.end()-3); //10

return 0;
}
```

Библиотека vector. pop_back – удаление последнего

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        // вектор из 3-х  нулевых элементов
        vector<int> v(3);

        // добавили 2 элемента в конец
        v.push_back(100);
        v.push_back(-55);
        // удалили последний элемент
        v.pop_back();

        // 0 0 0 100
        for (int i = 0; i < v.size(); i++)
                cout << v[i] << ' ';

return 0;
}
```

Библиотека vector. clear – удаление всех элементов

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        // 10 10 10
        vector<int> v(3,10);

        //удаляет все элементы коллекции
        v.clear();

        // v - пустой
        cout << v.size() <<endl; // 0


return 0;
}
```

Библиотека vector. erase – удаление элемента

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{       vector<int> v;
        v.push_back(10);
        v.push_back(20);
        v.push_back(30);

        //удаляет указанный элемент
        v.erase(v.begin() + 1);

        // 10 30
        for (int i = 0; i < v.size(); i++)
                cout << v.at(i) << ' '; // 10
return 0;
}
```

Библиотека vector. swap – обмен векторов

```
#include <vector>
#include <iostream>
using namespace std;
int main()
{       vector <int> v1, v2;
        //v1 = 1 2 3, всего 3 элемента
        v1.push_back(1); v1.push_back(2); v1.push_back(3);

        //v2 = 10 20, всего 2 элемента
        v2.push_back(10); v2.push_back(20); //2

        v1.swap(v2);

        cout << v1.size() << endl; //2
        cout << v2.size() << endl; //3


}
```

Библиотека vector. Итераторы

```
#include <iostream>
#include <iterator>
#include <vector>
using namespace std;
int main()
{
        vector<int> v;
        v.push_back(10);
        v.push_back(20);
        v.push_back(30);
        //  копировать от, до, куда
        copy(v.begin(), //  итератор начала массива
             v.end(),   //  итератор конца массива
             //итератор потока вывода
             ostream_iterator<int>(cout, " "));

return 0;
}
```

Лямбда-функции или лямбда-выражения

Лямбда-функции

Лямбда-функции появились в C++11. Они представляют собой анонимные функции, которые можно определить в любом месте программы

```
#include <iostream>
int main() {

        // лямбда-функция
        auto myLambda = [](){
        std::cout << "Hello, lambda!" << std::endl;
        };

        // вызов лямбда-функции
        myLambda();

return 0;
}
```

Лямбда-функции

Лямбда-выражение , иногда также называемое *лямбда-* функцией или (строго говоря, но разговорно) как *лямбда*, является упрощенной нотацией для определения и использования

анонимного функционального объекта

```
int main()
{
//функция
	auto lambda = [](auto x, auto y)
		{return x + y; };

//вызов лямбда функции
	cout << lambda(5,6) << endl;

return 0;
}
```

Лямбда-функции

```
int main()
{
//лямбда функция с двумя аргументами, возвращает bool
	auto LambdaAB = [](int a, int b) -> bool
	{
		return a>b;
	};

//вызов лямбда функции
	cout << LambdaAB(5, 2) << endl;

 return 0;
}
```

Лямбда-функции

```
int main()
{
//лямбда функция с одним аргументом, возвращает int
	auto LambdaABS = [](int a)->int
		{
		return (a>0) ? (a) : (-a);
		};


//вызов лямбда функции
	cout << LambdaABS(-5) << endl;

 return 0;
}
```

Лямбда-функции

```
int main()
{
//функция, возвращающая случайное значение от 0.0 до 1.0
	auto LambdaRand = []()->float
	{
		return float(rand()) / RAND_MAX;
	};

//вызов лямбда функции
	cout << LambdaRand() << endl;


return 0;
}
```

Лямбда-функции

```
#include <iostream>
int main() {

        // лямбда-функция
        auto square = [](int x) { return x * x; };

        // вызов лямбда-функции
        std::cout << square(16) << std::endl;

return 0;
}
```

Лямбда-функции

- **Лямбда-выражение** всегда начинается с **[]** (скобки могут быть непустыми), затем идет необязательный список параметров, а затем непосредственно тело функции.
- По умолчанию **лямбда** возвращает **void.**
- При наличии одного **return** в лямбда-выражении, компилятор вычисляет тип возвращаемого значения самостоятельно.

```
[](int i) { return i % 2 == 0; }
```

Лямбда-функции

- Если же в лямбда-выражении присутствует **if** или **switch** (или другие сложные конструкции), то нужно указать тип возвращаемого значения явно:

```
[](int _n) -> double
{
        if (_n < 5)
                return _n + 1.0;
        else if (_n % 2 == 0)
                return _n / 2.0;
        else
                return _n * _n;
}
```

Лямбда-функции. Примеры

```
[](const int x){ return x >= 0 && x % 7 == 0;}
```

```
[](const int x, const int y)
                { return std::abs(x) < std::abs(y); }
```

```
[](int x) {std::cout << ++x << " "; }
```

Что возвращают эти функции?

Лямбда-функции. Примеры

```
[k, m](const int x) { return x % k == 0 && x > m; }
```

```
[&num_positives, &num_evens](const int x)
{
        if (x >= 0) num_positives++;
        if (x % 2 == 0) num_evens++;
}
```

Что возвращают эти функции?

Когда использовать лямбда-функции

- Лямбда-функции особенно полезны, когда мы хотим передать операцию в качестве аргумента алгоритму.
- Лямбда-выражение - это способ оптимизации кода и способ сделать его более привлекательным.

Когда использовать лямбда-функции

- Один из лучших примеров правильного использования лямбда-функций связан с библиотекой алгоритмов **stl**.
- Большинство функций этой библиотеки принимают аргумент-предикат.

Предикат в программировании — выражение, использующее одну или более величину с результатом булева типа.

algorithm

Библиотека algorithm. std::replace

```
void replace(ForwardIt first, ForwardIt last,
        const T& old_value, const T& new_value);
```

Заменяет все элементы в диапазоне [**first**, **last**) на **new_value**

Библиотека algorithm. Пример1. std::replace. Часть 1

```
#include <iostream>
#include <string>
#include <array>
#include <algorithm>
using namespace std;

int main() {

        string s("0*0*0*");

        replace(s.begin(), s.end(), '0', '*');

        cout << s<<endl; // *****
```

Библиотека algorithm. Пример1. std::replace. Часть 2

```
// целочисленный массив из 5 элементов
array<int, 5> ar{ 0, 7, 0, 7, 0 };


// заменили все 0 на 3
replace(ar.begin(), ar.end(), 0, 3);

for (int a : ar) {
cout << a << " "; // 3 7 3 7 3
}


cout << '\n';
return 0;
}
```

Библиотека algorithm. std::replace_if

```
void replace_if(ForwardIt first, ForwardIt last,
                UnaryPredicate p, const T& new_value);
```

Заменяет все элементы в диапазоне [**first, last**], **удовлетворяющие определенному условию**, на **new_value**.

Библиотека algorithm. Пример1. std::replace_if. Часть 1

```
#include <iostream>     // std::cout
#include <algorithm>    // std::replace_if
#include <vector>       // std::vector

bool IsOdd(int i) { return ((i % 2) == 1); }

int main() {

        std::vector<int> myvector;

        // set some values:
        // 1 2 3 4 5 6 7 8 9
        for (int i = 1; i<10; i++) myvector.push_back(i);
```

Библиотека algorithm. Пример1. std::replace_if. Часть 2

```
// set some values:
for (int i = 1; i<10; i++) myvector.push_back(i);
// 1 2 3 4 5 6 7 8 9


std::replace_if( myvector.begin(),
                 myvector.end(),
                 [](int x) {return (x % 2!=0) ;},
                 0); // 0 2 0 4 0 6 0 8 0
```

Библиотека algorithm. Пример1. std::replace_if. Часть 3

```
std::cout << "myvector contains:";

        for ( std::vector<int>::iterator it = myvector.begin();
              it != myvector.end(); ++it)
                std::cout << ' ' << *it;

std::cout << '\n';

return 0;
}
```

Библиотека algorithm. Пример1. std::replace_if. Часть 4

```
#include <iostream>     // std::cout
#include <algorithm>    // std::replace_if
#include <vector>       // std::vector

bool IsOdd(int i) { return ((i % 2) == 1); }

int main() {
std::vector<int> myvector;

// set some values:
for (int i = 1; i<10; i++) myvector.push_back(i);
// 1 2 3 4 5 6 7 8 9

std::replace_if(myvector.begin(), myvector.end(), IsOdd,  0);
// 0 2 0 4 0 6 0 8 0
```

Библиотека algorithm. Пример. std::copy

```
#include <iostream>     // std::cout
#include <algorithm>    // std::copy
#include <vector>       // std::vector

int main() {
int myints[] = { 10,20,30,40,50,60,70 };
std::vector<int> myvector(7);

std::copy(myints, myints + 7, myvector.begin());

std::cout << "myvector contains:";
for (std::vector<int>::iterator it = myvector.begin();
     it != myvector.end(); ++it)
     std::cout << ' ' << *it;

std::cout << '\n';

return 0;
}
```

Библиотека algorithm. Пример. std::copy_if. Часть 1

```
#include <iostream>     // std::cout
#include <algorithm>    // std::copy_if, std::distance
#include <vector>       // std::vector

int main() {

        std::vector<int> foo = { 25,15,5,-5,-15 };

        std::vector<int> bar(foo.size());
```

Библиотека algorithm. Пример. std::copy_if. Часть 2

```
// copy only positive numbers:

auto it = std::copy_if(foo.begin(),
                       foo.end(),
                       bar.begin(),
                       [](int i) {return !(i<0); });

// shrink container to new size
bar.resize(std::distance(bar.begin(), it));
```

Библиотека algorithm. Пример. std::copy_if. Часть 3

```
        std::cout << "bar contains:";
        for (int& x : bar) std::cout << ' ' << x;
        std::cout << '\n';

return 0;
}
```

Библиотека algorithm. Пример. std::remove

```
#include <iostream>     // std::cout
#include <algorithm>    // std::remove
int main() {

int myints[] = { 10,20,30,30,20,10,10,20 };// 10 20 30 30 20 10 10 20

// bounds of range:
int* pbegin = myints;                                  // ^
int* pend = myints + sizeof(myints)/sizeof(int); // ^                  ^

pend = std::remove(pbegin, pend, 20); // 10 30 30 10 10 ?  ?  ?
                                      // ^              ^
std::cout << "range contains:";
for (int* p = pbegin; p != pend; ++p)
std::cout << ' ' << *p;
std::cout << '\n';

return 0;
}
```

Функторы для STL

Функторы

```
#include <iostream>
#include <algorithm>//for_each
#include <numeric>//iota
using namespace std;
// функтор
bool fn(const int x)
{
	cout << x << " ";
	return 0;
}
void main()
{
	int arr[10];
	// заполнение массива
	std::iota(std::begin(arr), std::end(arr), 0);

	// печать массива
	std::for_each(std::begin(arr), std::end(arr), fn);
}
```

Спасибо за внимание!