



## C++ - Модуль 08

Шаблонные контейнеры, итераторы, алгоритмы

Резюме: Этот документ содержит упражнения Модуля 08 из модулей C++.

Версия : 7

# Содержание

I	Введение	2
II	Основные правила	3
III	Правила для конкретного модуля	5
IV	Упражнение 00: Легкий поиск	6
V	Упражнение 01: Размах	7
VI	Упражнение 02: Мутировавшая мерзость	9

# Глава I

## Введение

C++ — это язык программирования общего назначения, созданный Бьерном Страуструпом как расширение языка программирования C или «C с классами» (источник: [Википедия](#)).

Цель этих модулей — познакомить вас с объектно-ориентированным программированием. Это будет отправной точкой вашего путешествия по C++. Многие языки рекомендуются для изучения ООП. Мы решили выбрать C++, так как она является производным от вашего старого знакомого C. Поскольку это сложный язык, и для простоты ваш код будет соответствовать стандарту C++98.

Мы знаем, что современный C++ сильно отличается во многих аспектах. Так что, если вы хотите стать опытным разработчиком C++, вам решать, идти ли дальше после 42 Common Core!

# Глава II

## Основные правила

### Компиляция

- Скомпилируйте свой код с помощью C++ и флагов `-Wall -Wextra -Werror`
- Ваш код все равно должен компилироваться, если вы добавите флаг `-std=c++98`.

### Соглашения о форматировании и именованиях

- Каталог и упражнения будут называться следующим образом: `ex00`, `ex01`, ..., `exn`
- Назовите свои файлы, классы, функции, функции-члены и атрибуты, как требуется в рекомендациях.
- Пишите имена классов в формате `UpperCamelCase`. Файлы, содержащие код класса, будут всегда называться в соответствии с именем класса. Например: `ClassName.hpp/ClassName.h`, `ClassName.cpp` или `ClassName.tpp`. Затем, если у вас есть заголовочный файл, содержащий определение класса «BrickWall», обозначающего кирпичную стену, его имя будет `BrickWall.hpp`.
- Если не указано иное, каждое выходное сообщение должно заканчиваться символом новой строки. Символ отображается на стандартный вывод.
- Досвидания, Норминетт! В модулях C++ не применяется стиль кодирования. Вы можете следить за своим любимым. Но имейте в виду, что код, который не могут понять ваши коллеги-оценщики, — это код, который они не могут оценить. Старайтесь писать чистый и читаемый код.

### Разрешено/Запрещено

Вы больше не кодируете на C. Время C++! Следовательно:

- Вам разрешено использовать почти все из стандартной библиотеки. Таким образом, вместо того, чтобы придерживаться того, что вы уже знаете, было бы разумно использовать как можно больше C++-версий функций C, к которым вы привыкли.
- Однако вы не можете использовать никакую другую внешнюю библиотеку. Это означает, что C++11 (и производные формы) и библиотеки Boost запрещены. Также запрещены следующие функции: `*printf()`, `*alloc()` и `free()`. Если вы их используете, ваша оценка будет 0 и все.

- Обратите внимание, что если явно не указано иное, используемое пространство имен `<ns_name>` и ключевые слова друзей запрещены. В противном случае ваша оценка будет -42.
- Вам разрешено использовать STL только в Модуле 08. Это означает: никаких контейнеров (вектор/список/карта/и т. д.) и никаких алгоритмов (все, что требует включения заголовка `<algorithm>`) до тех пор. В противном случае ваша оценка будет -42.

#### Несколько требований к дизайну

- Утечка памяти происходит и в C++. Когда вы выделяете память (используя новый ключевое слово), вы должны избегать утечек памяти.
- От Модуля 02 до Модуля 08 ваши занятия должны быть оформлены в правовом стиле. Каноническая форма, за исключением случаев, когда прямо указано иное.
- Любая реализация функции, помещенная в заголовочный файл (кроме шаблонов функций), означает 0 для упражнения.
- Вы должны иметь возможность использовать каждый из ваших заголовков независимо от других. Таким образом, они должны включать все необходимые им зависимости. Однако вы должны избежать проблемы двойного включения, добавив защиту включения. В противном случае ваша оценка будет 0.

#### Прочтите меня

- При необходимости вы можете добавить несколько дополнительных файлов (например, для разделения кода). Поскольку эти назначения не проверяются программой, не стесняйтесь делать это, пока вы не дадите окончательные файлы.
- Иногда рекомендации к упражнению кажутся короткими, но примеры могут показать требования, которые явно не прописаны в инструкциях.
- Полностью прочтите каждый модуль перед началом! Действительно, сделайте это.
- Клянусь Одном, клянусь Твором! Используйте свой мозг !!!



Вам придется реализовать много классов. Это может показаться утомительным, если только вы не умеете писать сценарии в своем любимом текстовом редакторе.



Вам предоставляется определенная свобода для выполнения упражнений. Однако соблюдайте обязательные правила и не ленитесь. Вы будете упускать много полезной информации! Не стесняйтесь читать о теоретических концепциях.

# Глава III

## Правила для конкретного модуля

Вы заметите, что в этом модуле упражнения можно решать БЕЗ стандартных контейнеров и БЕЗ стандартных алгоритмов.


Однако именно их использование является целью данного Модуля. Вам разрешено использовать STL. Да, вы можете использовать Контейнеры (вектор/список/карту/и т. д.) и Алгоритмы (определенные в заголовке `<algorithm>`). Более того, вы должны использовать их как можно чаще. Таким образом, сделайте все возможное, чтобы применить их везде, где это уместно.

В противном случае вы получите очень плохую оценку, даже если ваш код работает должным образом. Пожалуйста, не ленитесь.

Вы можете определить свои шаблоны в файлах заголовков, как обычно. Или, если вы хотите, вы можете написать свои объявления шаблонов в файлах заголовков и написать их реализации в файлах `.hpp`. В любом случае файлы заголовков являются обязательными, а файлы `.hpp` — необязательными.

# Глава IV

## Упражнение 00: Легкий поиск

	Упражнение : 00
Легко найти	
Каталог с дачи: ex00/ Файлы для	
с дачи: Makefile, main.cpp, easyfind.{h, hpp} и необязательный файл: easyfind.tpp	
Запрещенные функции: нет	

Первое простое упражнение — это то, с чего начать с правой ноги.

Напишите шаблон функции `easyfind`, который принимает тип `T`. Он принимает два параметра. Первый имеет тип `T`, а второй — целое число.

Предполагая, что `T` является контейнером целых чисел, эта функция должна найти первое вхождение второго параметра в первый параметр.

Если вхождение не найдено, вы можете либо создать исключение, либо вернуть значение ошибки. на ваш выбор. Если вам нужно вдохновение, проанализируйте, как ведут себя стандартные контейнеры.


Конечно, внедрите и сдайте свои собственные тесты, чтобы убедиться, что все работает так, как ожидалось.



Вам не нужно обрабатывать ассоциативные контейнеры.

# Глава V

## Упражнение 01: размах

	Упражнение : 01
Ок вагывать	
Каталог с дачи: ex01/	
Файлы для с дачи: Makefile, main.cpp, Span.{h, hpp}, Span.cpp	
Запрещенные функц ии: нет	

Разработайте класс `Span`, который может хранить максимум  $N$  целых чисел.  $N$  — переменная типа `int` без знака и будет единственным параметром, передаваемым конструктору.

Этот класс будет иметь функцию-член с именем `addNumber()` для добавления одного числа в `Span`. Он будет использоваться для его заполнения. Любая попытка добавить новый элемент, если уже сохранено  $N$  элементов, должна вызвать исключение.

Затем реализуйте две функции-члена: `shortestSpan()` и `longestSpan()`.

Они соответственно найдут кратчайший промежуток или самый длинный промежуток (или расстояние, если хотите) между всеми сохраненными числами и вернут его. Если числане сохранены или есть только одно, диапазон не может быть найден. Таким образом, сгенерируйте исключение.

Конечно, вы будете писать свои собственные тесты, и они будут намного более тщательными, чем приведенные ниже. Проверьте свой `Span` хотя бы с минимум 10 000 номеров. Больше было бы еще лучше.



Запуск этого кода:

```
интервалосновой
0 {
    Диапазон sp = Диапазон(5);

    sp.addNumber(6);
    sp.addNumber(3);
    sp.addNumber(17);
    sp.addNumber(9);
    sp.addNumber(11);

    std::cout << sp.shortestSpan() << std::endl; std::cout <<
    sp.longestSpan() << std::endl;

    вернуть 0;
}
```

Должен выводиться :

```
$> ./ex01 2
```

```
14
$>
```


И последнее, но не менее важное: было бы замечательно заполнить ваш Span, используя итераторов. Делать тысячу вызовов addNumber() так раздражает. Реализуйте функциючлен, чтобы добавить много номеров в ваш Span за один вызов.



Если вы не имеете понятия, изучите Контейнеры. Некоторые функции-члены принимают ряд итераторов, чтобы добавить последовательность элементов в контейнер.

# Глава VI

## Упражнение 02: Мутировавшая мерзость

	Упражнение : 02
Мутировавшая мерзость	
Каталог с дачи: ex02/ Файлы	
для с дачи: Makefile, main.cpp, MutantStack.{h, hpp} и необязательный файл: MutantStack.tpp	
Запрещенные функции: нет	

Теперь пора перейти к более серьезным вещам. Давайте разработаем что-нибудь странное.

Контейнер `std::stack` очень удобен. К сожалению, это один из немногих контейнеров STL, который НЕ является итерируемым. Это очень плохо.

Но зачем нам это принимать? Конечно, если мы позволим себе зарезать исходный стек для создания недостающих функций.

Чтобы исправить эту несправедливость, вы должны сделать контейнер `std::stack` итерируемым.

Напишите класс `MutantStack`. Он будет реализован в терминах `std::stack`. Он будет предлагать все свои функции-члены, а также дополнительную функцию-итераторы.

Конечно, вы будете писать и сдавать свои собственные тесты, чтобы убедиться, что все работает как положено.

Найдите тестовый пример ниже.

```

интервал основной
0 {
    Стек мутантов<int>      стек;

    mstack.push(5);
    mstack.push(17);

    std::cout << mstack.top() << std::endl;

    mstack.pop();

    std::cout << mstack.size() << std::endl;

    mstack.push(3);
    mstack.push(5);
    mstack.push(737); //
    [...] mstack.push(0);

    MutantStack<int>::iterator it = mstack.begin();
    MutantStack<int>::iterator ite = mstack.end();

    ++э то;
    --Эт э в
    то время как (э то != ite)
    {
        std::cout << *it << std::endl; ++э то;
    }
    std::stack<int> s(mstack); вернуть
    0; }

```

Если вы запустите его в первый раз с своим MutantStack и во второй раз замените MutantStack, например, на std::list, два вывода должны быть одинаковыми. Конечно, при тестировании другого контейнера обновите приведенный ниже код с соответствующими функциями-членами (push() может стать push\_back()).