



С++ - Модуль 08

Шаблонные контейнеры, итераторы, алгоритмы

Резюме:

*Этот документ содержит упражнения модуля 08 из модулей
С++.*

Версия: 7

Содержание

I	Введение	2
II	Общие правила	3
III	Правила, относящиеся к конкретному модулю	5
IV	Упражнение 00: Легкая находка	6
V	Упражнение 01: Размах	7
VI	Упражнение 02: Мутировавшая мерзость	9

Глава I Введение

С++ - это язык программирования общего назначения, созданный Бьярном Струstrupом как продолжение языка программирования С, или "С с классами" (источник: [Википедия](#)).

Цель этих модулей - познакомить вас с **объектно-ориентированным программированием**. Это будет отправной точкой вашего путешествия по С++. Многие языки рекомендуются для изучения ООП. Мы решили выбрать С++, поскольку он является производным от вашего старого друга С. Поскольку это сложный язык, и для того, чтобы все было просто, ваш код будет соответствовать стандарту С++98.

Мы понимаем, что современный С++ во многих аспектах сильно отличается. Поэтому, если вы хотите стать квалифицированным разработчиком С++, вам предстоит пройти дальше 42 Common Core!

Глава II Общие правила

Компиляция

- Скомпилируйте ваш код с помощью `c++` и флагов `-Wall -Wextra -Werror`
- Ваш код будет компилироваться, если вы добавите флаг `-std=c++98`

Форматирование и соглашения об именовании

- Каталоги упражнений будут называться так: `ex00`, `ex01`, ... , `exp`
- Назовите свои файлы, классы, функции, функции-члены и атрибуты в соответствии с требованиями руководства.
- Записывайте имена классов в формате **UpperCamelCase**. Файлы, содержащие код класса, всегда будут именоваться в соответствии с именем класса. Например: `ClassName.hpp/ClassName.h`, `ClassName.cpp` или `ClassName.tpp`. Тогда, 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 завершаться символом новой строки и выводиться на стандартный вывод.
- *До свидания, Норминет!* В модулях C++ нет принудительного стиля кодирования. Вы можете следовать своему любимому стилю. Но имейте в виду, что код, который ваши коллеги-оценщики не могут понять, они не могут оценить. Делайте все возможное, чтобы писать чистый и читабельный код.

Разрешено/Запрещено

Вы больше не кодируете на C. Пора переходить на C++! Поэтому:

- 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++-шных версий функций языка C, к которым вы привыкли.
- Однако вы не можете использовать никакие другие внешние библиотеки. Это означает, что библиотеки C++11 (и производные формы) и 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-друзья запрещены. В противном случае ваша оценка будет равна -42.
- **Вам разрешено использовать STL только в модуле 08.** Это означает: никаких **контейнеров** (вектор/список/карта/и так далее) и никаких **алгоритмов** (все, что требует включения заголовка `<algorithm>`) до этого момента. В противном случае ваша оценка будет -42.

Несколько требований к дизайну

- Утечка памяти происходит и в С++. Когда вы выделяете память (с помощью функции `new` ключевое слово), вы должны избегать **утечек памяти**.
- С модуля 02 по модуль 08 ваши занятия должны быть построены в **православной канонической форме, за исключением случаев, когда прямо указано иное**.
- Любая реализация функции, помещенная в заголовочный файл (за исключением 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 зависимости. Однако вы должны избегать проблемы двойного включения, добавляя **защитные элементы include**. В противном случае ваша оценка будет равна 0.

Читать

- Вы можете добавить несколько дополнительных файлов, если это необходимо (например, для разделения вашего кода). Поскольку эти задания не проверяются программой, не стесняйтесь делать это, если вы сдаете обязательные файлы.
- Иногда указания к упражнению выглядят кратко, но на примерах можно увидеть требования, которые не прописаны в инструкциях в явном виде.
- Перед началом работы полностью прочитайте каждый модуль! Действительно, сделайте это.
- Одином, Тором! Используйте свой мозг!!!



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



Вам предоставляется определенная свобода в выполнении упражнений. Однако соблюдайте обязательные правила и не ленитесь. Иначе вы пропустите много полезной информации! Не стесняйтесь читать о

Глава III

Правила, специфичные для конкретного модуля

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


Однако их использование как раз и является целью данного модуля. Вам разрешено использовать STL. Да, вы можете использовать **контейнеры** (vector/list/map/ и так далее) и **алгоритмы** (определенные в заголовке <algorithm>). Более того, вы должны использовать их как можно больше. Таким образом, делайте все возможное, чтобы применять их везде, где это уместно.

В противном случае вы получите очень плохую оценку, даже если ваш код работает так, как ожидалось. Пожалуйста, не ленитесь.

Вы можете определять свои шаблоны в заголовочных файлах, как обычно. Или, если вы хотите, вы можете написать объявления своих шаблонов в заголовочных файлах и написать их реализацию в файлах .hpp. В любом случае, заголовочные файлы являются обязательными, а файлы .hpp - необязательными.

Глава IV

Упражнение 00: Легкая находка

	Упражнение : 00
	Легко найти
	Входящий каталог : <code>ex00/</code>
	Файлы для сдачи : <code>Makefile</code> , <code>main.cpp</code> , <code>easyfind.{h, hpp}</code> и дополнительный файл: <code>easyfind.tpp</code>
	Запрещенные функции : Нет

Первое легкое упражнение - это способ начать с правильной ноги.

Напишите шаблон функции `easyfind`, которая принимает тип `T`. Она принимает два параметра.

Первый имеет тип `T`, а второй - целое число.

Предполагая, что `T` - это контейнер **целых чисел**, эта функция должна найти первое вхождение второго параметра в первый параметр.

Если вхождение не найдено, вы можете либо выбросить исключение, либо вернуть значение ошибки по вашему выбору. Если вам нужно немного вдохновения, проанализируйте, как ведут себя стандартные контейнеры.

Конечно, проведите и сдайте собственные тесты, чтобы убедиться, что все работает так, как предполагается.




Вам не нужно работать с ассоциативными контейнерами.

Глава V

Упражнение 01:

Размах

	Упражнение : 01
	Пролет
	Входящий каталог : <i>ex01/</i>
	Файлы для сдачи : <i>Makefile, main.cpp, Span.{h, hpp}, Span.cpp</i>
	Запрещенные функции : Нет

Разработайте класс **Span**, который может хранить максимум *N* целых чисел. *N* - это переменная `unsigned int`, которая будет единственным параметром, передаваемым в конструктор.

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

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

Они, соответственно, найдут самый короткий или самый длинный промежуток (или расстояние, если хотите) между всеми сохраненными числами и вернут его. Если в памяти нет ни одного числа или только одно, пролет не может быть найден. Таким образом, будет выброшено исключение.

Конечно, вы напишете свои собственные тесты, и они будут гораздо более тщательными, чем приведенные ниже. Протестируйте свой `Span` как минимум на 10 000 числах. Больше будет даже лучше.

Выполняем этот код:

```
int main()
{
    Span sp = Span(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 выводить:

```
$> ./ex01
2
14
$>
```

И последнее, но не менее важное: было бы замечательно заполнять Span, используя **диапазон итераторов**. Выполнение тысячи вызовов addNumber() так раздражает. Реализуйте функцию-член для добавления многих чисел в Span за один вызов.



Если у вас нет ни малейшего представления, изучите
Контейнеры. Некоторые
функции-члены принимают диапазон итераторов, чтобы
добавить произвольное количество элементов в контейнер

Глава VI

Упражнение 02: Мутировавшая мерзость

	Упражнение : 02
	Мутировавшая мерзость
	Входящий каталог : <i>ex02/</i>
	Файлы для сдачи : <i>Makefile, main.cpp, MutantStack.{h, hpp}</i> и дополнительный файл: <i>MutantStack.tpp</i>
	Запрещенные функции : Нет

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

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

Но почему мы должны соглашаться на это? Особенно, если мы можем взять на себя смелость испортить исходный стек, чтобы создать недостающие функции.

Чтобы исправить эту несправедливость, необходимо сделать контейнер `std::stack` итерабельным.

Напишите класс **MutantStack**. Он будет реализован в терминах стека `std::stack`.

Он будет предлагать все свои функции-члены, плюс дополнительную возможность: **итераторы**.

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

Ниже приведен пример тестирования.

```
int main()
{
    MutantStack<int>    mstack;

    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();

    ++ит;
    -ит;
    while (it != ite)
    {
        std::cout << *it << std::endl;
        ++ит;
    }
    std::stack<int> s(mstack);
    вернуть 0;
}
```

Если вы запустите его в первый раз с вашим MutantStack, а во второй раз замените MutantStack, например, на std::list, два результата должны быть одинаковыми. Конечно, при тестировании другого контейнера обновите приведенный ниже код, добавив соответствующие функции-члены (push() может стать push_back()).

