



## C++ - Модуль 01

Выделение памяти, указатели на  
элементы, ссылки, оператор switch

Резюме:

Этот документ содержит упражнения Модуля 01 из модулей C++.

Версия: 8

# Содержание

я	Введение	2
ил	Основные правила	3
III	Упражнение 00: Браииниииннннззз	5
IV	Упражнение 01: Моар мозг!	6
V	Упражнение 02: ПРИВЕТ, ЭТО МОЗГ	7
VI	Упражнение 03. Ненужное насилие	8
VII	Упражнение 04: Сед для неудачников	10
VIII	Упражнение 05: Карен 2.0	11
IX	Упражнение 06: Карен-фильтр	13

# 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 он 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 именовании

- Каталоги упражнений будут называться следующим образом: ex00, ex01, ..., exp
- Назовите свои файлы, классы, функции, функции-члены и атрибуты, как требуется в рекомендации.
- 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 и отображается на стандартный вывод.
- До свидания, 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 православном 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.

#### Прочти 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 сдаете обязательные файлы.
- Иногда рекомендации к упражнению кажутся короткими, но примеры могут показать требования, которые явно не прописаны в инструкциях.
- Полностью прочитайте каждый модуль перед началом! Действительно, сделай это.
- Клянусь Одним, клянусь Тором! Используй 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. Ты бы упускаю много полезной информации! Не стесняйтесь читать о теоретические концепции.

# Глава III

## Упражнение 00: Браииииииинннннззз

	Упражнение : 00
Браииииииинннннззз	
Каталог сдачи: ex00/ Файлы для	
сдачи: Makefile, main.cpp, Zombie.cpp, Zombie.{h, hpp}, newZombie.cpp, randomChump.cpp	
Запрещенные функции: нет	

Во-первых, реализуйте класс `Zombie`. Он имеет строковое имя частного атрибута.

Добавьте функцию-член `void` объявлять (`void`); к классу Зомби. Зомби объявить о себе следующим образом:

<имя>: Браииииииинннннззз...

Не печатайте угловые скобки (< и >). Для зомби по имени Фу сообщение было бы:

Фу: Браииииииинннннззз...

Затем реализуйте две следующие функции:


- `Zombie* newZombie( std::string name );`  
Он создает зомби, дает ему имя и возвращает его, чтобы вы могли использовать его вне функции. объем.
- `void randomChump( std::string name );`  
Он создает зомби, назовите его, и зомби объявит о себе.

Теперь, в чем суть упражнения? Вы должны определить, в каком случае зомби лучше размещать в стеке или куче.

Зомби должны быть уничтожены, когда они вам больше не нужны. Деструктор должен вывести сообщение с именем зомби в целях отладки.

# Глава IV

## Упражнение 01: Моар мозг!

	Упражнение : 01
Моар мозг!	
Каталог сдачи: ex01/ Файлы	
для сдачи: Makefile, main.cpp, Zombie.cpp, Zombie.{h, hpp}, ZombieHorde.cpp Запрещенные функции: нет	

Время создать орду зомби!

Реализуйте следующую функцию в соответствующем файле:

Зомби\*      `ZombieHorde( int N, std::string name );`


Он должен выделить N объектов-зомби за один раз. Затем он должен инициализировать зомби, дав каждому из них имя, переданное в качестве параметра. Функция возвращает указатель на первого зомби.

Реализуйте свои собственные тесты, чтобы убедиться, что ваша функция `zombieHorde()` работает должным образом. Попробуйте вызвать `анонс()` для каждого из зомби.

Не забудьте удалить всех зомби и проверить наличие утечек памяти.

# Глава V

## Упражнение 02: ПРИВЕТ, ЭТО МОЗГ

	Упражнение : 02
ПРИВЕТ ЭТО МОЗГ	
Каталог сдачи: ex02/	
Файлы для сдачи: Makefile, main.cpp	
Запрещенные функции: Нет	

Напишите программу, которая содержит:

- Строковая переменная, инициализированная как «ПРИВЕТ, ЭТО МОЗГ».
- `stringPTR`: указатель на строку.
- `stringREF`: ссылка на строку.

Ваша программа должна напечатать:

- Адрес памяти строковой переменной.
- Адрес памяти, удерживаемый `stringPTR`.
- Адрес памяти, удерживаемый `stringREF`.

А потом:


- Значение строковой переменной.
- Значение, на которое указывает `stringPTR`.
- Значение, на которое указывает `stringREF`.

Вот и все, никаких хитростей. Цель этого упражнения — демистифицировать ссылки, которые могут показаться совершенно новыми. Хотя есть небольшие отличия, это еще один синтаксис для того, что вы уже делаете: манипулирование адресами.



# Глава VI

## Упражнение 03. Ненужное насилие

	Упражнение : 03
Ненужное насилие	
Каталог сдачи: ex03/ Файлы для	
сдачи: Makefile, main.cpp, Weapon.cpp, Weapon.{h, hpp}, HumanA.cpp, HumanA.{h, hpp}, HumanB.cpp, HumanB.{h, л.с.}	
Запрещенные функции: нет	

Реализуйте класс оружия, который имеет:

- Частный тип атрибута, представляющий собой строку.
- Функция-член `getType()`, возвращающая константную ссылку на тип.
- Функция-член `setType()`, которая устанавливает тип, используя новый тип, переданный в качестве параметра.

Теперь создайте два класса: `HumanA` и `HumanB`. У них обоих есть оружие и имя. У них также есть функция `Attack()`, которая отображает (конечно, без угловых скобок):

<имя> атакует своим <типом оружия>

`HumanA` и `HumanB` почти одинаковы, за исключением этих двух крошечных деталей:

- В то время как `HumanA` принимает оружие в своем конструкторе, `HumanB` этого не делает.
- `HumanB` не всегда может иметь оружие, тогда как `HumanA` всегда будет вооружен.

Если ваша реализация верна, выполнение следующего кода напечатает атаку «грубой дубинкой с шипами», а затем вторую атаку «какой-то другой тип дубины» для обоих тестовых случаев:

```
интервал основной
0 {
    {
        Оружейная дубинка = Оружие("грубая шипами");

        HumanA bob("Боб", дубина);
        боб.атака(); club.setType("какой
            другой тип клуба"); боб.атака(); } {

        Оружейная дубинка = Оружие("грубая шипами");

        ЧеловекБ Джим("Джим");
        джим.setWeapon (дубина);
        Джим.Атака();
        club.setType("какой другой тип клуба"); Джим.Атака();

    }

    вернуть 0;
}
```


Не забудьте проверить наличие утечек памяти.



Как вы думаете, в каком случае было бы лучше использовать указатель на Оружие? А отсылка к оружию? Почему? Подумайте об этом, прежде чем приступить к этому упражнению.

# Глава VII

## Упражнение 04: Сед для неудачников

	Упражнение : 04
Сед для неудачников	
Каталог сдачи: ex04/ Файлы для	
сдачи: Makefile, main.cpp, *.cpp, *.{h, hpp}	
Запрещенные функции: std::string::replace	

Создайте программу, которая принимает три параметра в следующем порядке: имя файла и две строки, s1 и s2.


Он откроет файл <filename> и скопирует его содержимое в новый файл. <имя файла>.replace, заменяя каждое вхождение s1 на s2.

Использование функций манипулирования файлами C запрещено и будет считаться мошенничеством. Разрешены все функции-члены класса std::string, кроме replace. Используйте их с умом!

Конечно, обрабатывать неожиданные входные данные и ошибки. Вы должны создать и сдать свой собственные тесты, чтобы убедиться, что ваша программа работает должным образом.

# Глава VIII

## Упражнение 05: Карен 2.0

	Упражнение : 05
Карен 2.0	
Каталог сдачи: ex05/ Файлы	
для сдачи: Makefile, main.cpp, Karen.{h, hpp}, Karen.cpp Запрещенные функции: нет	

Ты знаешь Карен? Мы все делаем, не так ли? Если вы этого не сделаете, найдите ниже тип комментирует Карен. Они классифицируются по уровням:

- Уровень "DEBUG" : сообщения отладки содержат контекстную информацию. В основном они используются для диагностики проблем.  
Пример: «Мне нравится иметь дополнительный бекон для моего гамбургера 7XL-двойной-сыр-тройной-рассол-специальный кетчуп. Я действительно люблю!»
- Уровень "INFO" : Эти сообщения содержат обширную информацию. Они полезны для отслеживание выполнения программы в производственной среде.  
Пример: «Я не могу поверить, что добавление дополнительного бекона стоит больше денег. Вы не положили достаточно бекона в мой бургер! Если бы вы это сделали, я бы не просил больше!»
- Уровень «ПРЕДУПРЕЖДЕНИЕ» . Предупреждающие сообщения указывают на возможную проблему в системе. Однако с этим можно справиться или проигнорировать.  
Пример: «Думаю, я заслуживаю бесплатного дополнительного бекона. Я прихожу сюда уже много лет, а вы начали работать здесь с прошлого месяца».
- Уровень "ОШИБКА" : Эти сообщения указывают на неустранимую ошибку. Обычно это критическая проблема, требующая ручного вмешательства.  
Пример: «Это неприемлемо! Я хочу поговорить с менеджером сейчас».

Вы собираетесь автоматизировать Карен. Это не составит труда, так как она всегда говорит одно и то же. Вы должны создать класс Карен со следующими закрытыми функциями-членами:

- аннулировать отладку (недействительно);
- недействительная информация( недействительная );
- недействительное предупреждение( недействительное );
- недействительная ошибка( недействительная );

У Карен также есть общедоступная функция-член, которая вызывает четыре указанные выше функции-члена. в зависимости от уровня, переданного в качестве параметра:


```
пустота пожаловаться( std::string level );
```

Цель этого упражнения — использовать указатели на функции-члены. Это не предложение. Карен приходится жаловаться, не используя лес if/else if/else. Она не думает дважды!

Создавайте и сдавайте тесты, чтобы показать, что Карен много жалуется. Вы можете использовать пример Комментарии.

# Глава IX

## Упражнение 06: Карен-фильтр

	Упражнение : 06
Карен-фильтр	
Каталог сдачи: ex06/ Файлы для	
сдачи: Makefile, main.cpp, Karen.{h, hpp}, Karen.cpp Запрещенные функции: нет	

Иногда ты не хочешь обращать внимание на все, что говорит Карен. Реализовать систему для фильтрации того, что говорит Карен, в зависимости от уровней журнала, которые вы хотите прослушать.

Создайте программу, которая принимает в качестве параметра один из четырех уровней. Он будет отображать все сообщения с этого уровня и выше. Например:

```
$> ./karenФильтр "ВНИМАНИЕ"
[ ПРЕДУПРЕЖДЕНИЕ ]
Я думаю, я заслуживаю того, чтобы получить немного дополнительного бекона бесплатно.
Я приезжаю уже много лет, а вы начали работать здесь с прошлого месяца.

[ ОШИБКА ]
Это неприемлемо, я хочу поговорить с менеджером сейчас.

$> ./karenFilter "Я не знаю, насколько я сегодня устал..."
[Возможно, жалуются на незначительные проблемы]
```

Хотя есть несколько способов справиться с Карен, один из самых эффективных — **ВЫКЛЮЧИТЬ** ее.

Дайте имя `karenFilter` вашему исполняемому файлу.