



# С++ - Модуль 03

## Наследова ние

*Резюме:*

*Этот документ содержит упражнения по созданию модуля из модулей 03С++.*

*Версия: 6*

# Содержание

<b>I</b>	<b>Введение</b>	<b>2</b>
<b>II</b>	<b>Общие правила</b>	<b>3</b>
<b>III</b>	<b>Упражнение 00: Аааа... ОТКРЫТО!</b>	<b>5</b>
<b>IV</b>	<b>Упражнение 01: Сирена, любовь моя!</b>	<b>7</b>
<b>V</b>	<b>Упражнение 02: Повторяющаяся работа</b>	<b>8</b>
<b>VI</b>	<b>Упражнение 03: Теперь это странно!</b>	<b>9</b>

# 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.



## аниемодулей

- 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 только08 в модуле.** Это означает: никаких **контейнеров** (вектор/лист/мап/ 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 C++. 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 (кроме шаблонов функций), означает 0упражнение.
- 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 следующей 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.
- Одним, 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 теоретических концепциях.

## Глава III

# Упражнение 00: Аааа... ОТКРЫТО!

	Упражнение 00
	Аааа... ОТКРЫТО!
Входящий каталог : <i>ex/00</i>	
Файлы для сдачи : <i>Makefile, main.cpp, ClapTrap.cpp, ClapTrap.{h, hpp}</i>	
Запрещенные функции : Нет	

Во-первых, вы должны реализовать класс! Как оригинально!

Он будет называться **ClapTrap** и будет иметь следующие приватные атрибуты, инициализированные значениями, указанными в скобках:

- Имя, которое передается в качестве параметра в конструктор
- Очки попадания (10), представляют здоровье ловушки.
- Очки энергии (10)
- Урон от атаки (0)

Добавьте следующие публичные функции-члены, чтобы ClapTrap выглядела более реалистично:

- `void attack(const std::string& target);`
- `void takeDamage(unsigned int amount);`
- `void beRepaired(unsigned int amount);`

Когда Хлопушка атакует, она заставляет цель потерять <урон от атаки> хит-пойнтов. Когда Хлопушка восстанавливает себя, она получает обратно <количество> хит-пойнтов. Атака и ремонт стоят по 1 очку энергии. Конечно, ClapTrap не может ничего сделать, если у него не осталось ни хит-пойнтов, ни очков энергии.





аниемодулей

Во всех этих функциях-членах необходимо вывести сообщение, описывающее происходящее. Например, функция `attack()` может вывести что-то вроде (конечно, без угловых скобок):

Хлопушка <имя> атакует <цель>, нанося <дамаж> пунктов урона!


Конструкторы и деструкторы должны также отображать сообщение, чтобы ваши коллеги-оценщики могли легко увидеть, что они были вызваны.

Проводите и сдавайте собственные тесты, чтобы убедиться, что ваш код работает так, как ожидается.



## Глава IV

### Упражнение 01: Сирена, любовь моя!

	Упражнение 01
	Сирена, любовь моя!
	Входящий каталог : ex/01
	Файлы для сдачи : Файлы из предыдущего упражнения + ScavTrap.cpp, ScavTrap.{h, hpp}
	Запрещенные функции : Нет

Поскольку ловушек ClapTrap никогда не бывает достаточно, сейчас вы создадите производного робота. Он будет называться **ScavTrap** и унаследует конструкторы и деструктор от Clap-ловушки. Однако его конструкторы, деструктор и атака() будут печатать другие сообщения. В конце концов, ловушки ClapTrap осознают свою индивидуальность.

Обратите внимание, что правильное построение/разрушение цепочки должно быть показано в ваших тестах. Когда создается ScavTrap, программа начинает с построения ClapTrap. Разрушение происходит в обратном порядке. Почему?

**ScavTrap** будет использовать атрибуты ClapTrap (в последствии обновляя ClapTrap) и должен их инициализировать:

- Имя, которое передается в качестве параметра в конструктор
- Очки попадания (100), представляют здоровье ловушки.
- Очки энергии (50)
- Урон от атаки (20)

У ScavTrap также будет своя специальная емкость:


```
void guardGate();
```

Эта функция члена отобразит сообщение о том, что ScavTrap сейчас находится в режиме Gate keeper.

Не забывайте добавлять в свою программу дополнительные тесты.

## Глава V

# Упражнение 02: Повторяющаяся работа

	Упражнение 02
	Повторяющаяся работа
	Входящий каталог : ex/02
	Файлы для сдачи : Файлы из предыдущих упражнений + FragTrap.cpp, FragTrap.{h, hpp}
	Запрещенные функции : Нет

Создание хлопушек, вероятно, начинает действовать вам на нервы.

Теперь реализуйте класс **FragTrap**, который наследуется от ClapTrap. Он очень похож на ScavTrap. Однако сообщения о его создании и уничтожении должны быть разными. Правильное построение/разрушение цепочки должно быть показано в ваших тестах. Когда создается FragTrap, программа начинает с построения ClapTrap. Разрушение происходит в обратном порядке.

То же самое для атрибутов, но в этот раз с другими значениями:

- Имя, которое передается в качестве параметра в конструктор
- Очки попадания (100), представляют здоровье ловушки.
- Очки энергии (100)
- Урон от атаки (30)

FragTrap также имеет специальную емкость:

```
void HighFivesGuys(void);
```


Эта функция-член выводит на стандартный вывод положительный запрос "Дай пять".

Опять же, добавьте больше тестов в вашу программу.



## Глава VI

### Упражнение 03: Теперь это странно!

	Упражнение 03
	Теперь это странно!
	Входящий каталог : <code>ex/03</code>
	Файлы для сдачи : Файлы из предыдущих упражнений + <code>DiamondTrap.cpp</code> , <code>DiamondTrap.{h, hpp}</code>
	Запрещенные функции : Нет

В этом упражнении вы создадите монстра: ловушку `ClapTrap`, которая является наполовину `FragTrap`, наполовину `ScavTrap`. Он будет называться **DiamondTrap** и будет наследоваться как от `FragTrap`, так и от `ScavTrap`. Это так рискованно!

Класс `DiamondTrap` будет иметь атрибут `name private`. Дайте этому атрибуту точно такое же имя переменной (здесь не идет речь об имени робота), как и в базовом классе `ClapTrap`.

Чтобы было понятнее, вот два примера.  
Если переменной `ClapTrap` является имя, присвойте имя имени переменной `DiamondTrap`.  
Если переменная `ClapTrap` имеет имя `_name`, присвойте имя `_name` переменной `DiamondTrap`.

Его атрибуты и функции-члены будут взяты из любого из его родительских классов:

- Имя, которое передается в качестве параметра в конструктор
- `ClapTrap::name` (параметр конструктора + суффикс `"_clap_name"`)
- Хит-пойнты (`FragTrap`)
- Энергетические пункты (`ScavTrap`)
- Урон от атаки (`FragTrap`)



- attack() (Scavtrap)

аниемодулей

В дополнение к специальным функциям обоих родительских классов, DiamondTrap будет обладать собственными специальными возможностями:

```
void whoAmI();
```

Эта функция-член будет отображать как свое имя, так и имя ClapTrap.

Конечно, подобъект ClapTrap объекта Diamondtrap будет создан один раз, и только один раз. Да, есть одна хитрость.

Опять же, добавьте больше тестов в вашу программу.



Знаете ли вы флаги компилятора -Wshadow и -Wno-shadow?

