



C++ - Модуль 01

Распределение памяти, указатели на
члены, ссылки, оператор switch

Резюме:

*Этот документ содержит упражнения модуля 01 из модулей
C++.*

Версия: 9

Содержание

I	Введение	2
II	Общие правила	3
III	Упражнение 00: Браиииииииннннзззззззз	5
IV	Упражнение 01: Больше мозгов!	6
V	Упражнение 02: ЗДРАВСТВУЙТЕ, ЭТО МОЗГ	7
VI	Упражнение 03: Ненужное насилие	8
VII	Упражнение 04: Сед для неудачников	10
VIII	Упражнение 05: Harl 2.0	11
IX	Упражнение 06: Фильтр Харла	13

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 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 (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 теоретических концепциях.

Упражнение 00:

Браиининннн333333

	Упражнение : 00
Braiiiiiiiiinnnnzzzz	
Входящий каталог : ex00/	
Файлы для сдачи : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp	
Запрещенные функции : Нет	

Добавьте функцию-член `void announce(void);` в класс

объявляют о себе следующим образом:

Не печатайте угловые скобки (< и >). Для зомби с именем Foo сообщение будет выглядеть следующим образом:

Foo: Браииниииннннннннннн33333...

Затем реализуйте две следующие функции:


- `Zombie* newZombie(std::string name);`
Он создает зомби, присваивает ему имя и возвращает его, чтобы вы могли использовать его за пределами области действия функции.
- `void randomChump(std::string name);`
Он создает зомби, дает ему имя, и зомби объявляет о себе.

Итак, в чем, собственно, смысл упражнения? Вы должны определить, в каком случае лучше выделять зомби на стеке или куче.

Зомби должны быть уничтожены, когда они вам больше не нужны. Деструктор должен выводить сообщение с именем зомби для целей отладки.

Глава IV

Упражнение 01: Больше мозгов!

	Упражнение : 01 Больше мозгов!
Входящий каталог : <i>ex01/</i>	
Файлы для сдачи : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp	
Запрещенные функции : Нет	

Пора создать **орду зомби**!

Реализуйте следующую функцию в соответствующем файле:

```
зомби*    zombieHorde( int N, std::string name );
```

Он должен выделить N объектов зомби за одно распределение. Затем она должна инициализировать зомби, присвоив каждому из них имя, переданное в качестве параметра. Функция возвращает указатель на первого зомби.


Проведите собственные тесты, чтобы убедиться, что ваша функция `zombieHorde()` работает так, как ожидается.

Попробуйте вызвать функцию `announce()` для каждого из зомби.

Не забудьте удалить всех зомби и проверить на **утечку памяти**.

Глава V

Упражнение 02: ПРИВЕТ ЭТО МОЗГ

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

Напишите программу, которая содержит:

- Строковая переменная, инициализированная в "HI THIS IS BRAIN".
- stringPTR: указатель на строку.
- stringREF: Ссылка на строку.

Ваша программа должна

напечатать:

- Адрес памяти строковой переменной.
- Адрес памяти, хранящийся в stringPTR.
- Адрес памяти, хранящийся в


stringREF. А затем:

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

Вот и все, никаких хитростей. Целью этого упражнения является демистификация ссылок, которые могут показаться совершенно новыми. Хотя есть некоторые небольшие различия, это другой синтаксис для того, что вы уже делаете: работа с адресами.

Глава VI

Упражнение 03: Ненужное насилие

	Упражнение : 03
Ненужное насилие	
Входящий каталог : <code>ex03/</code>	
Файлы для сдачи : <code>Makefile</code> , <code>main.cpp</code> , <code>Weapon.{h, hpp}</code> , <code>Weapon.cpp</code> , <code>HumanA.{h, hpp}</code> , <code>HumanA.cpp</code> , <code>HumanB.{h, hpp}</code> , <code>HumanB.cpp</code>	
Запрещенные функции : Нет	

Реализуйте класс оружия, который имеет:

- Частный тип атрибута, который представляет собой строку.
- Функция-член `getType()`, которая возвращает `const`-ссылку на тип.
- Функция-член `setType()`, которая устанавливает тип, используя новый тип, переданный в качестве параметра.

Теперь создайте два класса: **HumanA** и **HumanB**. Они оба имеют оружие и имя. У них также есть функция-член `attack()`, которая отображает (конечно, без угловых скобок):

<имя> атакует своим <типом оружия>

`HumanA` и `HumanB` практически одинаковы, за исключением этих двух мелких деталей:

- В то время как `HumanA` принимает `Weapon` в своем конструкторе, `HumanB` не принимает.
- Человек **не всегда** может иметь оружие, в то время как человек **всегда** будет вооружен.

Если ваша реализация верна, то выполнение следующего кода выведет атаку с "грубой шипованной дубиной", а затем вторую атаку с "каким-то другим типом дубины" для обоих тестовых случаев:

```
int main()
{
    Weapon club = Weapon("грубая шипованная дубина");
```



Как вы думаете, в каком случае лучше использовать указатель на Weapon?

Почему?

И ссылке на Weapon?

Подумайте об этом,

ПРОСЬБА НЕ ПЕРЕКЛЮЧАТЬСЯ НА ЭТОМУ УПРАЖНЕНИИ

```
Weapon club = Weapon("грубая шипованная дубина");

HumanB jim("Jim"); jim.
setWeapon(club); jim.attack();
club.setType("какой-то другой тип клуба");
jim.attack();
}

return 0;
}
```


Глава VII

Упражнение 04: Сед для неудачников

	Упражнение : 04
	Сед - для неудачников
	Входящий каталог : <code>ex04/</code>
	Файлы для сдачи : <code>Makefile, main.cpp, *.cpp, *.{h, hpp}</code>
	Запрещенные функции : <code>std::string::replace</code>

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 `<filename>.replace`, заменяя каждое вхождение `s1` на `s2`.


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

Конечно, обрабатывайте непредвиденные входные данные и ошибки. Вы должны создавать и сдавать собственные тесты, чтобы убедиться, что ваша программа работает так, как ожидается.

Глава VIII

Упражнение 05:

Harl 2.0

	Упражнение : 05
	Harl 2.0
	Входящий каталог : <code>ex05/</code>
	Файлы для сдачи : <code>Makefile</code> , <code>main.cpp</code> , <code>Harl.{h, hpp}</code> , <code>Harl.cpp</code>
	Запрещенные функции : Нет

Вы знаете Харла? Мы все знаем, не так ли? Если вы не знаете, найдите ниже, какие комментарии делает Харл. Они классифицированы по уровням:

- уровень **"DEBUG"**: Отладочные сообщения содержат контекстную информацию. В основном они используются для диагностики проблем.
Пример: *"Я люблю, когда у меня есть дополнительный бекон для моего бургера 7XL с двойным сыром и тройным огурцом и специальным кетчупом. Правда!"*
- уровень **"INFO"**: Эти сообщения содержат обширную информацию. Они полезны для отслеживания выполнения программы в производственной среде.
Пример: *"Я не могу поверить, что добавление дополнительного бекона стоит больше денег. Вы не положили в мой бургер достаточно бекона! Если бы вы положили, я бы не просил добавки!"*
- Уровень **"ПРЕДУПРЕЖДЕНИЕ"**: Предупреждающие сообщения указывают на потенциальную проблему в системе. Однако ее можно решить или проигнорировать.
Пример: *"Я думаю, что заслуживаю того, чтобы бесплатно получить немного дополнительного бекона. Я хожу сюда уже много лет, в то время как вы начали работать здесь в прошлом месяце".*
- уровень **"ERROR"**: Эти сообщения указывают на возникновение неустранимой ошибки. Обычно это критическая проблема, требующая ручного вмешательства.
Пример: *"Это неприемлемо! Я хочу поговорить с менеджером".*

Вы собираетесь автоматизировать Harl. Это будет несложно, поскольку он всегда говорит одно и то же. Вы должны создать класс **Harl** со следующими частными функциями-членами:

- `void debug(void);`
- `void info(void);`
- `void warning(void);`
- `void error(void);`

Harl также имеет публичную функцию-член, которая вызывает четыре функции-члена выше в зависимости от уровня, переданного в качестве параметра:


```
void    complain( std::string level );
```

Цель этого упражнения - использовать **указатели на функции-члены**. Это не предложение. Harl должен жаловаться, не используя лес `if/else if/else`. Он не думает дважды!

Создайте и сдайте тесты, показывающие, что Harl часто жалуется. Вы можете использовать комментарии к примерам.

Глава IX

Упражнение 06: Фильтр Харла

	Упражнение : 06 Фильтр Harl
Входящий каталог : <i>ex06/</i>	
Файлы для сдачи : Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp	
Запрещенные функции : Нет	

Иногда вы не хотите обращать внимание на все, что говорит Harl. Внедрите систему фильтрации того, что говорит Харл, в зависимости от уровня журнала, который вы хотите прослушивать.

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

```
$> ./harlFilter "WARNING"
[ WARNING ]
Я думаю, что заслуживаю получить немного дополнительного бекона бесплатно.
Я прихожу сюда уже много лет, в то время как вы начали работать здесь с прошлого месяца.

[ ERROR ]
Это неприемлемо, я хочу поговорить с менеджером.

$> ./harlFilter "Я не уверен, насколько я сегодня устал..." [ Возможно, жалуется на несущественные
```

Хотя существует несколько способов борьбы с Harl, один из самых эффективных - ОТКЛЮЧИТЬ его.

Дайте имя harlFilter вашему исполняемому файлу.

В этом упражнении вы должны использовать и, возможно, обнаружить оператор switch.



Вы можете пройти этот модуль без выполнения упражнения 06.

