Википедия
).

Цель этих модулей - познакомить вас с **объектно-ориентированным программированием**. Это станет отправной точкой вашего путешествия по C ++. Многие языки рекомендуются для изучения ООП. Мы решили выбрать C ++, поскольку он является производным от вашего старого друга C.
Потому что 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 дальше после 42 Common
Core!
2

Глава II

Общие 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, ...
, exn

•
Назовите свои файлы, классы, функции, функции-члены и атрибуты в соответствии с требованиями
руководства.

•
Запишите имена классов в формате верхнего регистра. Файлы, содержащие код класса
, всегда будут называться в соответствии с именем класса. Например:
classNam e.hpp/classNam e.h, ClassNam e.cpp , или classNam e.tpp. Затем, если у вас
есть заголовочный файл, содержащий определение класса "BrickWall", обозначающего кирпичную стену, его имя будет BrickWall.hpp.

•
Если не указано иное, все выходные сообщения должны заканчиваться символом новой строки и отображаться в стандартном выводе.

•
Прощай, Норма! В модулях C++ не применяется никакой стиль кодирования. Вы можете следить за своим любимым. Но имейте в виду, что код, который ваши коллеги-оценщики не могут понять, - это код, который они не могут оценить. Сделайте все возможное, чтобы написать чистый и читаемый
код.

Разрешено / Запрещено

Вы больше не пишете на языке Си. Пора переходить на C++! Поэтому:

•

Вам разрешено использовать практически 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 какую-либо другую внешнюю библиотеку. Это означает, что C ++ 11 (и производные
формы) и библиотеки Boost запрещены. Следующие функции также запрещены
: *printf(), *alloc() и free(). Если вы их используете, ваша оценка будет равна 0, и
все.

•

Обратите внимание, что, если явно не указано иное, использование пространства имен <ns_name> и
ключевых слов friend запрещено. 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 тех пор. В противном случае ваша оценка будет равна -42.

Несколько требований к дизайну

•

Утечка памяти происходит и в C ++. Когда вы выделяете память (используя
ключевое слово new), вы должны избегать **утечек памяти**.

•

С модуля 02 по модуль 08 ваши классы должны быть оформлены в **ортодоксальной канонической форме, за исключением случаев, когда явно 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 им зависимости. Однако вы должны избежать
проблемы двойного включения, добавив **защитные элементы включения**. В противном случае ваша оценка будет
равна 0.

Прочти меня

•

Вы можете добавить несколько дополнительных файлов, если вам нужно (например, для разделения вашего кода). Поскольку эти
назначения не проверяются программой, не стесняйтесь делать это, пока вы включаете обязательные файлы.

•

Иногда рекомендации по выполнению упражнения выглядят короткими, но примеры могут показать
требования, которые явно не прописаны в инструкциях.
•

Полностью прочитайте каждый модуль перед началом работы! Действительно, сделай это.

•

Клянусь Одином, клянусь Тором! Используй свой мозг!!!

Вам придется реализовать множество классов.
Это может показаться утомительным,
если только вы не умеете писать сценарий в своем любимом текстовом редакторе.
Вам предоставляется определенная свобода для выполнения упражнений.
Однако соблюдайте обязательные правила и не ленитесь.
Вы бы
пропустили много полезной информации!
Не стесняйтесь читать о

теоретических концепциях.
4

Глава III

Упражнение 00: BraiiiiiiinnnzzzZ

Упражнение : 00
Брейиииииинннзззззз
Каталог для сдачи в аренду: *ex*
00
/
Файлы, которые нужно сдать :
Makefile, main.cpp , Зомби.{h, hpp}, Zombie.cpp , newZombie.cpp , randomChump.cpp
Запрещенные функции :
Нет
Во-первых, реализуйте класс Zombie. Он имеет строковое частное имя атрибута.
Добавьте функцию-член void announcement(void); к классу Zombie. Зомби объявляют о себе следующим образом:
<имя>: BraiiiiiiinnnzzzZ...
Не печатайте угловые скобки (< и >). Для зомби по имени Foo сообщение будет следующим:
Фу: Брейиииииинннзззз3...
Затем реализуйте две следующие функции:

•
Zombie* newZombie(std::string name);
Он создает зомби, присваивает ему имя и возвращает его, чтобы вы могли использовать его вне области действия функции
.

•
void randomChump(std::string name);
Он создает зомби, называет его, и зомби объявляет о себе.
Итак, в чем собственно смысл этого упражнения? Вы должны определить, в каком случае лучше размещать зомби в стеке или куче.
Зомби должны быть уничтожены, когда они вам больше не понадобятся. Деструктор должен напечатать сообщение с именем зомби для целей отладки.
5

Глава IV

Упражнение 01: Моар брейнз!

Упражнение : 01
Моар брейнз!
Каталог для сдачи в аренду: *ex*
01
/
Файлы, которые нужно сдать :
Makefile, main.cpp , Зомби.{h, hpp}, Zombie.cpp , zombieHorde.cpp

Запрещенные функции :

`Нет`

Время создать **орду зомби**!

Реализуйте следующую функцию в соответствующем файле:

Зомби*

zombieHorde(int N, std::имя строки);

Он должен выделить N объектов-зомби за одно выделение. Затем он должен инициализировать зомби, присвоив 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 функцию announce() для каждого из зомби.
Не забудьте удалить всех зомби и проверить**, нет ли утечек памяти**.
6

Глава V

Упражнение 02: ПРИВЕТ, ЭТО МОЗГ

Упражнение : 02
ПРИВЕТ, ЭТО МОЗГ
Каталог для сдачи в аренду: *ex*
`02`
`/`
Файлы, которые нужно сдать :
`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 печатать:
•
Адрес памяти строковой переменной.
•
Адрес памяти, хранящийся в stringPTR.
•
Адрес памяти, хранящийся в stringREF.
А потом:
•
Значение строковой переменной.
•
Значение, на которое указывает stringPTR.
•
Значение, на которое указывает stringREF.

Вот и все, никаких фокусов. Цель этого упражнения - развеять мистику ссылок, которые могут показаться совершенно новыми. Хотя есть некоторые небольшие различия, это еще один синтаксис для того, что вы уже делаете: манипулирование адресами.
7

Глава VI

Упражнение 03: Ненужное насилие

Упражнение : 03
Ненужное насилие

Каталог для сдачи в аренду: *ex*
03
/
Файлы, которые нужно сдать :
Makefile, main.cpp , Оружие.{h, ГЭС}, Weapon.cpp , ХуманА.{h, ГЭС}, HumanA.cpp , HumanB.{h, hpp}, HumanB.cpp
Запрещенные функции :
Нет
Реализовать класс оружия, который имеет:
•
Частный тип атрибута, представляющий собой строку.
•
Функция-член GetType(), которая возвращает постоянную ссылку на type.
•
Функция-член setType(), которая задает тип, используя новый, переданный в качестве параметра.

Теперь создайте два класса: **HumanA** и **HumanB**. У них обоих есть Оружие и имя. У них также есть функция-член 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 использует Оружие в своем конструкторе, HumanB этого не делает.
•
У **человека не всегда может** быть оружие, в то **время как человек всегда** будет вооружен.

Если ваша реализация верна, выполнение следующего кода выведет атаку с использованием "грубой шипастой дубинки", а затем вторую атаку с использованием "дубинки другого типа" для обоих
тестовых случаев:

```
int
main
()
{
{
Оружейная
дубинка = Оружие (
"грубая шипастая дубинка")
);
ХуманА боб (
"Bob"
                          , клуб);
bob.attack();
club.setType(
"какой-то другой тип клуба"
);
    bob.attack();
}
{
Оружейная
дубинка = Оружие (
"грубая шипастая дубинка")
);
HumanB Джим (
"Джим"
                        );
```

```
jim.setWeapon(дубинка);
jim.attack();
club.setType(
"какой-то другой тип дубинки"
);
    джим.атака();
}
```
Возврат
```
0
;
}
```
Не забудьте проверить**, нет ли утечек памяти**.
```
В каком случае, по вашему мнению, было бы лучше использовать указатель на
Оружие?
И ссылка на Оружие?
Почему?
Подумайте об этом, прежде
чем приступать к этому упражнению.
```


Глава VII

Упражнение 04: Сэд - для неудачников

Упражнение : 04

Сэд - это для неудачников

Каталог для сдачи в аренду: *ex*

```
04
/
```

Файлы, которые нужно сдать :

```
Makefile, main.cpp , *.cpp, *.{h, hpp}
```

Запрещенные функции :

```
std::строка::заменить
```

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

Конечно, обрабатывайте неожиданные входные данные и ошибки. Вы должны создать и включить свои собственные тесты, чтобы убедиться, что ваша программа работает должным образом.

Глава VIII

Упражнение 05: Харл 2.0

Упражнение : 05

Харл 2.0

Каталог для сдачи в аренду: *ex*

```
05
/
```

Файлы, которые нужно сдать :
`Makefile, main.cpp , Харл.{h, гэс}, Harl.cpp`
Запрещенные функции :
`Нет`
Ты знаешь Харла? Мы все так делаем, не так ли? В случае, если вы этого не сделаете, ознакомьтесь ниже с
комментариями, которые делает Харл. Они классифицируются по уровням:
•
Уровень "**DEBUG**": сообщения отладки содержат контекстную информацию. В основном они используются для диагностики проблем.
Пример: *"Я люблю есть дополнительный бекон для моего бургера 7XL с двойным сыром, тройным маринадом и специальным
кетчупом. Я действительно хочу!"*
•
Уровень "**ИНФОРМАЦИЯ**": 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. Ты не положил
достаточно бекона в мой бургер! Если бы ты это сделал, я бы не просил большего!"*
•
Уровень "**ПРЕДУПРЕЖДЕНИЕ**": предупреждающие сообщения указывают на потенциальную проблему в системе.
Однако с этим можно справиться или проигнорировать.
Пример: *"Я думаю, что заслуживаю получить немного дополнительного бекона бесплатно. Я прихожу сюда уже
много лет, в то время как ты начал работать здесь с прошлого месяца ".*
•
Уровень "**ОШИБКА**": эти сообщения указывают на то, что произошла неустранимая ошибка.
Обычно это критическая проблема, требующая ручного вмешательства.
Пример: *"Это неприемлемо! Я хочу поговорить с менеджером прямо сейчас ".*
11
C++ - Модуль 01
Выделение памяти, указатели на элементы,
ссылки, оператор switch
Вы собираетесь автоматизировать Harl. Это будет нетрудно, так как в нем всегда говорится
одно и то же. Вам необходимо создать класс Harl со следующими закрытыми функциями-членами:
•
пустая отладка (void);
•
недействительная информация (void);
•
недействительное предупреждение (void);
•
ошибка аннулирования (void);
Harl также имеет общедоступную функцию-член, которая вызывает четыре функции-члена, описанные 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 включите тесты, чтобы показать, что Харл часто жалуется. Вы можете использовать примеры
комментариев.

Глава IX

Упражнение 06: Фильтр Харла

Упражнение : 06
Фильтр Harl
Каталог для сдачи в аренду: *ex*
`06`
`/`
Файлы, которые нужно сдать :
`Makefile, main.cpp , Харл.{h, гэс}, Harl.cpp`
Запрещенные функции :
`Нет`
Иногда не хочется обращать внимания на все, что говорит Харл. Внедрите
систему фильтрации того, что говорит Harl, 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:

```
$> ./harlFilter "ПРЕДУПРЕЖДЕНИЕ"
[ ПРЕДУПРЕЖДЕНИЕ ]
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.
[ОШИБКА]
Это неприемлемо, я хочу поговорить с менеджером прямо сейчас.
$> ./harlFilter "Я не уверен, насколько я устал сегодня ..."
[ Вероятно, жалуется на незначительные проблемы ]
```

Хотя есть несколько способов справиться с Harl, один из наиболее эффективных - отключить его.
Присвойте вашему исполняемому файлу имя harlFilter.
Вы должны использовать и, возможно, обнаружить оператор switch в этом упражнении.
`Вы можете пройти этот модуль, не выполняя упражнение 06.`