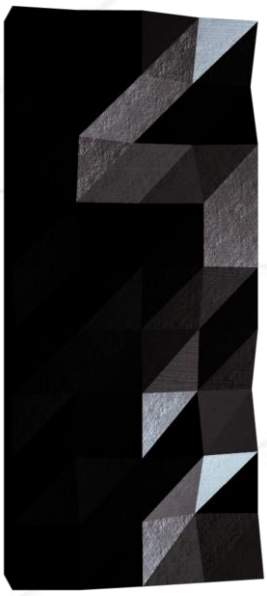

C++ - Модуль 04

Полиморфизм подтипов, абстрактные классы, интерфейсы

Резюме:
Этот документ содержит упражнения модуля 04 из модулей C++.

Версия: 10

Содержание

I	**Введение**	**2**
II	**Общие правила**	**3**
III	**Упражнение 00: Полиморфизм**	**5**
IV	**Упражнение 01: Я не хочу поджигать мир**	**7**
V	**Упражнение 02: Абстрактный класс**	**9**
VI	**Упражнение 03: Интерфейс и подведение итогов**	**10**

Глава I Введение

C++ - это язык программирования общего назначения, созданный Бьярном Струструпом как продолжение 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++, поскольку он является производным от вашего старого друга C. Поскольку это сложный язык, и для того, чтобы все было просто, ваш код будет соответствовать стандарту C++98.

Мы понимаем, что современный C++ во многих аспектах сильно отличается. Поэтому, если вы хотите стать квалифицированным разработчиком C++, 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, ... , exn
- 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++-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

Глава III

Упражнение 00: Полиморфизм

	Упражнени е : 00
Полиморфизм	
Входящий каталог : *ex00/*	
Файлы для сдачи : Makefile, main.cpp, *.cpp, *.{h, hpp}	
Запрещенные функции : Нет	

Для каждого упражнения вы должны предоставить **наиболее полные тесты,** которые вы можете.
Конструкторы и деструкторы каждого класса должны отображать определенные сообщения. Не используйте одно и то же сообщение для всех классов.

Начните с реализации простого базового класса **Animal**. Он имеет один защищенный атрибут:

- std::string type;

Реализуйте класс **Dog,** который наследуется от Animal.
Реализуйте класс **Cat**, который наследуется от Animal.

Эти два производных класса должны установить свое поле типа в зависимости от своего имени. Тогда тип класса Dog будет инициализирован в "Dog", а тип класса Cat будет инициализирован в "Cat". Тип класса Animal может быть оставлен пустым или установлен в значение по вашему выбору.

Каждое животное должно уметь использовать функцию-член:
makeSound()
Он напечатает соответствующий звук (кошки не лают).

Выполнение этого кода должно вывести специфические звуки классов Dog и Cat, а не Animal.

```
int main()
{
    const Animal* meta = new Animal();
    const Animal* j = new Dog();
    const Animal* i = new Cat();

    std::cout << j->getType() << " " <<
    std::endl; std::cout << i->getType() << " "
    << std::endl; i->makeSound(); //выдаст
    звук кошки!
    j-
    >makeSound
    (); meta-
    >makeSound();
    …

    вернуть 0;
```

Чтобы убедиться, что вы поняли, как это работает, реализуйте класс **WrongCat**, который наследуется от класса **WrongAnimal**. Если вы замените в приведенном выше коде слова Animal и Cat на неправильные, то WrongCat должен выводить звук WrongAnimal.

Выполните и сдайте больше тестов, чем приведенные выше.

Глава IV

Упражнение 01: Я не хочу поджигать мир

	Упражнени е : 01
Я не хочу поджигать мир.	
Входящий каталог : *ex01/*	
Файлы для сдачи : Файлы из предыдущего упражнения + *.cpp, *.{h, hpp}	
Запрещенные функции : Нет	

Конструкторы и деструкторы каждого класса должны отображать определенные сообщения.

Реализуйте класс **Brain**. Он содержит массив из 100 строк std::string, называемых идеями.
Таким образом, Собака и Кошка будут иметь частный атрибут Brain*.
После создания, Собака и Кошка создадут свой Мозг, используя new Brain();
После уничтожения Пес и Кот удалят свой мозг.

В своей главной функции создайте и заполните массив объектов **Animal**. Половина массива будет состоять из объектов **Dog**, а другая половина - из объектов **Cat**. В конце выполнения программы выполните цикл по этому массиву и удалите всех животных. Вы должны удалить непосредственно собак и кошек как Животных. Соответствующие деструкторы должны быть вызваны в ожидаемом порядке.

Не забудьте проверить **утечку памяти**.

Копия Собаки или Кошки не должна быть поверхностной. Таким образом, вы должны проверить, что ваши копии являются глубокими копиями!

```
int main()
{
    const Animal* j = new Dog();
    const Animal* i = new Cat();


    delete j;//не должно создавать утечку
    удалить i;
    ...

    вернуть 0;
}
```

Выполните и сдайте больше тестов, чем приведенные выше.

Глава V

Упражнение 02: Абстрактный класс

	Упражнени е : 02
Абстрактный класс	
Входящий каталог : *ex02/*	
Файлы для сдачи : Файлы из предыдущего упражнения + *.cpp, *.{h, hpp}	
Запрещенные функции : Нет	

Создание объектов-животных не имеет смысла, в конце концов. Это правда, они не издают никаких звуков!

Чтобы избежать возможных ошибок, класс Animal по умолчанию не должен быть инстанцируемым.
Исправьте класс Animal так, чтобы никто не мог его инстанцировать. Все должно работать как прежде.

Если вы хотите, вы можете обновить имя класса, добавив к Animal префикс A.

Глава VI

Упражнение 03: Интерфейс и подведение итогов

	Упражнени е : 03
Интерфейс и обзор	
Входящий каталог : *ex03/*	
Файлы для сдачи : Makefile, main.cpp, *.cpp, *.{h, hpp}	
Запрещенные функции : Нет	

Интерфейсы не существуют в C++98 (даже в C++20). Однако чистые абстрактные классы принято называть интерфейсами. Таким образом, в этом последнем упражнении давайте попробуем реализовать интерфейсы, чтобы убедиться, что вы усвоили этот модуль.

Завершите определение следующего класса **AMateria** и реализуйте необходимые функции-члены.

```
класс AMateria
{
    защищенный:
        [...]

    общественность:
        AMateria(std::string const & type);
        [...].

        std::string const & getType() const; //Возвращает тип материи

        virtual AMateria* clone() const = 0;
        virtual void use(ICharacter& target);
};
```

Реализуйте конкретные классы Materias **Ice** и **Cure**. Используйте их имена в нижнем регистре ("ice" для Ice, "cure" для Cure) для задания их типов. Конечно, их функция-член clone() будет возвращать новый экземпляр того же типа (то есть, если вы клонируете Ice Materia, вы получите новую Ice Materia).

Выводится функция-член use(ICharacter&):

- Лед: "* выстреливает ледяной болт в <имя> *".
- Cure: "* исцеляет раны <имя>".

<имя> - это имя символа, переданного в качестве параметра. Не печатайте угловые скобки (< и >).

При назначении Материи другому, копирование типа не имеет смысла.

Напишите конкретный класс **Character**, который будет реализовывать следующий интерфейс:

```
класс ICharacter
{
    общественность:
            virtual ~ICharacter() {}
            virtual std::string const & getName() const = 0; virtual
            void equip(AMateria* m) = 0;
            virtual void unequip(int idx) = 0;
            virtual void use(int idx, ICharacter& target) = 0;
};
```

Персонаж имеет в инвентаре 4 слота, что означает максимум 4 материи. При строительстве инвентарь пуст. Персонаж экипирует материями первый попавшийся пустой слот. То есть в таком порядке: от слота 0 до слота 3. В случае, если они пытаются добавить Материю в полный инвентарь или использовать/экипировать несуществующую Материю, ничего не делайте (но все же ошибки запрещены). Функция unequip() НЕ должна удалять Materia!

Обращайтесь с материями, которые ваш персонаж оставил на полу, как вам удобно. Сохраняйте адреса перед вызовом unequip() или чего-либо еще, но не забывайте, что вы должны избегать утечек памяти.

Функция-член use(int, ICharacter&) должна будет использовать Materia в слоте[idx] и передать целевой параметр в функцию AMateria::use.

Инвентарь вашего персонажа сможет поддерживать любой тип AMateria.

Ваш **символ** должен иметь конструктор, принимающий его имя в качестве параметра. Любая копия (с помощью конструктора копирования или оператора присвоения копий) персонажа должна быть **глубокой**. Во время копирования материя персонажа должна быть удалена, прежде чем новая будет добавлена в его инвентарь. Разумеется, материя должна быть удалена при уничтожении персонажа.

Напишите конкретный класс **MateriaSource**, который будет реализовывать следующий интерфейс:

```
класс IMateriaSource
{
    общественность:
            virtual ~IMateriaSource() {}
            virtual void learnMateria(AMateria*) = 0;
            virtual AMateria* createMateria(std::string const & type) = 0;
};
```

- learnMateria(AMateria*)
  Копирует Материю, переданную в качестве параметра, и сохраняет ее в памяти, чтобы ее можно было клонировать позже. Как и персонаж, **источник MateriaSource** может знать не более 4 Materia. Они не обязательно уникальны.
- createMateria(std::string const &)
  Возвращает новую материю. Последняя является копией Materia, ранее изученной **источником MateriaSource,** тип которой равен типу, переданному в качестве параметра. Возвращает 0, если тип неизвестен.

В двух словах, ваш **MateriaSource** должен уметь изучать "шаблоны" Materia, чтобы создавать их по мере необходимости. Затем вы сможете генерировать новую материю, используя только строку, определяющую ее тип.

Выполняем этот код:

```
int main()
{
    IMateriaSource* src = new MateriaSource();
    src->learnMateria(new Ice());
    src->learnMateria(new Cure());

    ICharacter* me = new Character("me");

    AMateria* tmp;
    tmp = src->createMateria("ice");
    me->equip(tmp);
    tmp = src->createMateria("cure");
    me->equip(tmp);

    ICharacter* bob = new Character("bob");

    me->use(0, *bob);
    me->use(1, *bob);

    удалить
    боба;
    удалить
    меня;
    удалить
    src;

    вернуть 0;
```

Должен выводить:

```
$> clang++ -W -Wall -Werror *.cpp
$> ./a.out | cat -e
* стреляет ледяным болтом в Боба *$
* лечит раны Боба *$
```

Как обычно, выполните и сдайте больше тестов, чем приведенные выше.

Вы можете пройти этот модуль без выполнения упражнения 03.