



C++ - Модуль 04

Полиморфизм подтипов, абстрактные классы, интерфейсы

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

Версия : 10

Содержание

я	Введение	2
ил	Основные правила	3
III	Упражнение 00: Полиморфизм	5
IV	Упражнение 01: Я не хочу спалить мир	7
V	Упражнение 02. Абстрактный класс	9
VI	Упражнение 03: Интерфейс и реализация	10

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 она 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

- Каталог и упражнения 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 отображается на стандартный вывод.
- Досвидания, Норминетт! В модулях C++ не применяется стиль кодирования. Вы можете следовать 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 зависимости. Однако вы должны избежать проблемы двойного включения, добавив защиту от повторного включения. В противном случае ваша оценка будет 0.

Прочтите 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 создаете обязательные файлы.
- Иногда рекомендации к упражнению кажутся короткими, но примеры могут показать требования, которые явно не прописаны в инструкции.
- Полностью прочтите каждый модуль перед началом! Действительно, сделайте это.
- Клянусь Одном, клянусь Твором! Используйте 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. Вы будете упускать много полезной информации! Не стесняйтесь читать о теоретических концепциях.

Глава III

Упражнение 00: Полиморфизм

	Упражнение : 00
Полиморфизм	
Каталог с дачи: ex00/ Файлы	
для с дачи: Makefile, main.cpp, *.cpp, *.h, hpp}	
Запрещенные функции: нет	

Для каждого упражнения вы должны предоставить как можно более полные тесты. 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 под названием Animal. Имеет один защищенный атрибут:

- тип `std::string`;

Реализуйте класс `Dog`, который наследуется от `Animal`.

Реализуйте класс `Cat`, наследуемый от `Animal`.

Эти два производных класса должны устанавливать свое поле типа в зависимости от их имени. Затем тип собаки будет инициализирован как «собака», а тип кошки будет инициализирован как «кошка». Тип класса `Animal` можно оставить пустым или устанавливать значение по вашему выбору.

Каждое животное должно иметь возможность использовать функцию член: `makeSound()`.

Он печатает соответствующий звук (кошки не лают).

Запуск этого кода должен печатать определенные звуки классов Dog и Cat, а не животных.

```
интервал основной
0 {
    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; я ->с оздать звук(); и выведет
    j->с оздать звук(); мета->с оздать звук();

    ...


    вернуть 0;
}
```

Чтобы убедиться, что вы понимаете, как это работает, реализуйте класс WrongCat, наследуемый от класса WrongAnimal. Если вы замените Animal и Cat неправильными в приведенном выше коде, WrongCat должен вывести звук WrongAnimal.

Внедрите и дайте больше тестов, чем указано выше.

Глава IV

Упражнение 01: Я не хочу поджигать мир

	Упражнение : 01
Я не хочу поджигать мир	
Каталог с дачи: ex01/ Файлы	
для с дачи: файлы из предыдущего упражнения + *.cpp, *.{h, hpp}	
Запрещенные функции: нет	

Конструкторы и деструкторы каждого класса должны обрабатывать определенные сообщения .

Реализуйте класс `Мозг` . Он содержит массив из 100 `std::string`, называемых идеями .
 Таким образом, у Собаки и Кошки будет собственный атрибут `Мозг *`.
 После построения Собака и Кошка создадут свой `Мозг` , используя `new Brain()`; При уничтожении Собака и Кошка удалят свой `Мозг` .

Вашей основной функцией создайте и заполните массив объектов `Animal` . Половина из них будет объектами `Dog` , а другая половина — объектами `Cat` . В конце выполнения вашей программы выполните цикл по этому массиву и удалите каждое животное. Вы должны удалить непосредственно собак и кошек как животных . Соответствующие деструкторы должны вызываться в ожидаемом порядке .

Не забудьте проверить наличие утечек памяти .

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


```
интервалосной
0 {
    const Animal* j = new Dog(); const
    Animal* i = new Cat();

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


    ...

    вернуть 0;
}
```

Внедрите и добавьте больше тестов, чем указано выше.

Глава V

Упражнение 02. Абстрактный класс

	Упражнение : 02
Абстрактный класс	
Каталог с дачи: ex02/ Файлы	
для с дачи: файлы из предыдущего упражнения + *.cpp, *.{h, hpp}	
Запрещенные функции: нет	


В конце концов, создание объектов Animal не имеет смысла. Это правда, они не издаю ни звука!

Чтобы избежать возможных ошибок, класс Animal по умолчанию не должен создавать экземпляры. Исправьте класс Animal, чтобы никто не мог создать его экземпляр. Все должно работать как прежде.

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

Глава VI

Упражнение 03. Интерфейс и резюме

	Упражнение : 03
Интерфейс и резюме	
Каталог с дачи: ex03/ Файлы	
для с дачи: Makefile, main.cpp, *.cpp, *.h, hpp}	
Запрещенные функции: нет	

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

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

```

класс AMateria
{
    защищено [...]

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

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

    виртуальный AMateria* clone() const = 0;
    использование виртуальной функции (ICharacter& target);
};
  
```

Реализуйте конкретные классы `Material Ice` и `Cure`. Используйте их имена в нижнем регистре («лед» для льда, «лечение» для лечения), чтобы установить их типы. Конечно, их функция-член `clone()` вернет новый экземпляр того же типа (т.е. если вы клонируете Ледяную Материю, вы получите новую Ледяную Материю).

Функция-член `use(ICharacter&)` будет отображать:

- Лед: `"* стреляет ледяной стрелой в <имя> *"`
- Лечение: `"* лечит раны <имя> *"`

`<имя>` — это имя Персонажа, переданное в качестве параметра. Не печатайте угловые скобки (`<` и `>`).



При назначении Материи другому копирование типа не делается.

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

```
class ICharacter {
public:
    virtual ~ICharacter() {} virtual
    std::string const & getName() const = 0; оборудование
    виртуальной пус тоты (AMateria* m) = 0; виртуальная
    пус тоты unequip(int idx) = 0; использование виртуальной
    пус тоты (int idx, ICharacter& target) = 0;
};
```

В инвентаре Персонажа 4 слота, что означает максимум 4 Материи.

Инвентарь пуст при строительстве. Они экипируют Материю в первый свободный слот, который они найдут. Это означает, что в таком порядке: от слота 0 до слота 3. В случае, если они попытаются добавить Материю полный инвентарь или использовать/снять несуществующую Материю, ничего не делайте (но тем не менее, ошибки запрещены). Функция-член `unequip()` НЕ ДОЛЖНА удалять Материю.



Распорядитесь Материями, оставленными вашим персонажем на полу, как вам угодно. Сохраняйте адрес перед вызовом `unequip()` или чего-то еще, но не забывайте, что вы должны избегать утечек памяти.

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



Инвентарь вашего персонажа может поддерживать любой тип АМатерия .

У вашего Персонажа должен быть конструктор, принимающий его имя в качестве параметра. Любая копия (используя конструктор копирования или оператор присваивания копии) персонажа должна быть глубокой. Во время копирования Материи Персонажа должны быть удалены, прежде чем новые будут добавлены в его инвентарь. Разумеется, Материи должны быть удалены при уничтожении Персонажа.

Напишите конкретный класс MaterialSource, который будет реализовывать следующий интерфейс:

```
класс IMaterialSource
{
    общественно: виртуальный ~IMaterialSource()
    {} виртуальный недействительный
    LearnMaterial(AMateria*) = 0; виртуальная AMateria* createMaterial(std::string const & type) = 0;
};
```

- выучитьМатерию(AMateria*)

Копирует Материю переданную в качестве параметра, и сохраняя ее в памяти, чтобы ее можно было клонировать позже. Как и Персонаж, Источник Материи может знать не более 4 Материй. Они не обязательно уникальны.

- createMaterial(std::string const &)

Возвращает новую Материю. Последний является копией Материи, ранее изученной Материей. -Источником, тип которой равен типу, переданному в качестве параметра. Возвращает 0, если тип неизвестен.

В двух словах, ваш MaterialSource должен уметь изучать «шаблоны» Материй, чтобы создавать их по мере необходимости. Затем вы можете генерировать новую Материю, используя только строку, идентифицирующую ее тип.

Запуск этого кода:

```
интервал ос новной
0 {
    IMateriaSource* src = новый MateriaSource(); src-
    >learnMateria (новый лед()); src->learnMateria (новое
    лечение ());

    ICharacter* me = новый персонаж("я");

    AMateria* tmp; tmp
    = src-> createMateria («лед»); я -> оборудовать
    (tmp); tmp = src-> createMateria («лечить»);
    я -> оборудовать (tmp);

    ICharacter* bob = новый символ ("bob");

    я ->использовать(0, *bob);
    я ->использовать(1, *bob);

    удалить боб;
    удалить меня ;
    удалить источник;

    вернуть 0;
}
```

Должен выводиться я :

```
$> clang++ -W -Wall -Werror *.cpp $> ./a.out |
cat -e * с треля ет ледя ной с трелой в боба
*$ * лечит раны боба*$
```

Как обычно, внедрите и **сдайте** больше тестов, чем указано выше.



Вы можете пройти этот модуль, не выполняя упражнение 03.