



С++ - Модуль 05

Повторение и исключения

Резюме:

Этот документ содержит упражнения по созданию модуля из модулей 05С++.

Версия: 9

Содержание

I	Введение	2
II	Общие правила	3
III	Упражнение 00: Мамочка, когда я вырасту, я хочу стать бюрократом!	5
IV	Упражнение 01: Формируйтесь, личинки!	7
V	Упражнение 02: Нет, вам нужна форма 28В, а не 28С...	9
VI	Упражнение 03: По крайней мере, это лучше, чем приготовление кофе	11

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`, ..., `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++-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 08 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: Мамочка, когда я вырасту, я хочу стать бюрократом!

	Упражнение 00
Мамочка, когда я вырасту, я хочу стать бюрократом!	
Входящий каталог : <code>ex/00</code>	
Файлы для сдачи : <code>Makefile</code> , <code>main.cpp</code> , <code>Bureaucrat.h</code> , <code>hpp</code> , <code>Bureaucrat.cpp</code>	
Запрещенные функции : Нет	



Пожалуйста, обратите внимание, что классы-исключения не обязательно должны быть разработаны в православной канонической форме. Но все остальные классы должны.

Давайте спроектируем искусственный кошмар из офисов, коридоров, бланков и очередей.

Звучит забавно? Нет? Жаль.

Во-первых, начните с самого маленького винтика в этой огромной

бюрократической машине: **бюрократа**. **Бюрократ** должен иметь:

- Постоянное имя.
- И оценка, которая варьируется от (1максимально возможная оценка) до (150минимально возможная оценка).

Любая попытка создать Бюрократа, используя недопустимый класс, должна привести к ошибке:

либо исключение `Bureaucrat::GradeTooHighException`, либо исключение `Bureaucrat::GradeTooLowException`.

Вы предоставите геттеры для обоих этих атрибутов: `getName()` и `getGrade()`. Вставьте также две функции-члена для увеличения или уменьшения класса бюрократа. Если класс выходит за пределы диапазона, обе функции будут выбрасывать те же исключения, что и конструктор.



Помните. Поскольку оценка является 1 высшей и 150 низшей, повышение оценки должно 3 дать оценку 2 бюрократу.


Выброшенные исключения должны быть перехватываемыми с помощью блоков `try` и `catch`:

```
попробуйте
{
    /* делаем некоторые вещи с бюрократами */
}
catch (std::exception & e)
{
    /* обработка исключения */
}
```

Как обычно, проведите несколько тестов, чтобы доказать, что все работает так, как ожидалось.

Глава IV

Упражнение 01: Формируйтесь, личинки!

	Упражнение 01
Формируйтесь, личинки!	
Входящий каталог : ex/01	
Файлы для сдачи : Файлы из предыдущего упражнения + Form.{h, hpp}, Form.cpp	
Запрещенные функции : Нет	

Теперь, когда у вас есть бюрократы, давайте дадим им занятие. Что может быть лучше, чем заполнение стопки бланков?

Затем создадим класс **Form**. В нем есть:

- Постоянное имя.
- Булево значение, указывающее, подписано ли оно (при построении - нет).
- Для его подписания требуется постоянная оценка.
- И постоянная оценка, необходимая для

его выполнения. Все эти атрибуты

являются **частными**, а не защищенными.

Оценки **формы** подчиняются тем же правилам, которые применяются к Бюрократу. Таким образом, если оценка формы выходит за рамки, будут выброшены следующие исключения: `Form::GradeTooHighException` и `Form::GradeTooLowException`.

Как и раньше, напишите геттеры для всех атрибутов и перегрузку оператора `insertion (""),` который печатает всю информацию формы.

Добавьте также функцию-член `beSigned` к форме, которая принимает бюрократа в качестве параметра. Она изменяет статус формы на подписанный, если ранг бюрократа достаточно высок (выше или равен требуемому). Помните, что оценка 1 выше, чем оценка 2. Если оценка слишком низкая, выбросьте `Form::GradeTooLowException`.

Наконец, добавьте функцию-член `signForm` к Бюрократу. Если форма подписана, будет выведено что-то вроде:

<бюрократ> подписал <форму>

В противном случае будет выведено что-то вроде:


<Бюрократ> не смог подписать <форму> по <причине>.

Внедрите и сдайте тесты, чтобы убедиться, что все работает так,

как ожидается.

Глава V

Упражнение 02: Нет, вам нужна форма 28В, а не 28С...

	Упражнение 02
Нет, вам нужна форма 28В, а не 28С...	
Входящий каталог : ex/02	
Файлы для сдачи : Файлы из предыдущих упражнений + ShrubberyCreationForm.[{h, hpp},cpp], RobotomyRequestForm.[{h, hpp},cpp], PresidentialPardonForm.[{h, hpp},cpp].	
Запрещенные функции : Нет	

Поскольку у вас теперь есть основные формы, пришло время сделать еще несколько, которые действительно что-то делают.

Во всех случаях базовый класс `Form` должен быть абстрактным классом. Помните, что атрибуты формы должны оставаться приватными и что они находятся в базовом классе.

Добавьте следующие конкретные классы:

- **КустарникСозданиеФормы**: Требуемые оценки: исполнение 145, знака 137
Создает файл `<target>_shrubbery` в рабочем каталоге и записывает в него деревья ASCII.
- **RobotomyRequestForm**: Требуемые оценки: знак exes 72,45
Издает несколько звуков сверления. Затем сообщает, что `<цель>` была успешно роботомирована в 50% случаев. В противном случае сообщает, что роботомия не удалась.
- **PresidentialPardonForm**: Требуемые оценки: sign exes 25,
Сообщает 5, что `<цель>` была помилована Зафодом Библброксом.

Все они принимают только один параметр в своем конструкторе: цель формы. Например, "home", если вы хотите посадить дома кустарник.

Теперь добавьте функцию-член `execute(Bureaucrat const & executor) const` к базовой форме и реализуйте функцию для выполнения действия формы из конкретных классов. Вы должны проверить, что форма подписана и что класс бюрократа, пытающегося выполнить форму, достаточно высок. В противном случае выбросьте соответствующее исключение.

Хотите ли вы проверять требования в каждом конкретном классе или в базовом классе (затем вызывать другую функцию для выполнения формы) - решать вам. Однако один способ красивее другого.

Наконец, добавьте функцию-член `executeForm(Form const & form)` в бюрократ. Она должна попытаться выполнить форму. Если попытка успешна, выведите что-то вроде:


<бюрократ> выполнил <форму>

Если нет, выведите явное сообщение об ошибке.

Проводите и сдавайте тесты, чтобы убедиться, что все работает так, как ожидается.

Глава VI

Упражнение 03: По крайней мере, это лучше, чем приготовление кофе

	Упражнение 03
По крайней мере, это лучше, чем приготовление кофе	
Входящий каталог : <code>ex/03</code>	
Файлы для сдачи : Файлы из предыдущих упражнений + <code>Intern.{h, hpp}</code> , <code>Intern.cpp</code>	
Запрещенные функции : Нет	

Поскольку заполнение форм и так раздражает, было бы жестоко просить наших бюрократов делать это целый день. К счастью, интерны существуют. В этом упражнении вы должны реализовать класс **Intern**. У стажера нет ни имени, ни оценки, ни уникальных характеристик. Единственное, о чем заботятся бюрократы, - это чтобы он выполнял свою работу.

Однако у интерна есть одна важная способность: функция `makeForm()`. Она принимает две строки. Первая - имя формы, вторая - цель формы. Она возвращает указатель на **объект Form** (имя которого равно имени, переданному в качестве параметра), цель которого будет инициализирована вторым параметром.

Будет выведено что-то вроде:

Стажер создает `<form>`

Если имя формы, переданное в качестве параметра, не существует, выведите явное сообщение об ошибке.

Вы должны избегать нечитаемых и некрасивых решений, таких как использование леса if/elseif/else. Такие вещи не будут приняты в процессе оценки. Вы больше не в Piscine (бассейне). Как обычно, вы должны проверить, что все работает так, как ожидается.

Например, приведенный ниже код создает **RobotomyRequestForm**, нацеленную на "Ben- der":

```
{
    Intern someRandomIntern;
    Form*rrf  ;

    rrf = someRandomIntern. makeForm("запрос на робототехнику", "Бендер");
}
```