



С++ - Модуль 06

Касты С++

Резюме:

*Этот документ содержит упражнения модуля 06 из модулей
С++.*

Версия: 5

Содержание

I	Введение	2
II	Общие правила	3
III	Дополнительное правило	5
IV	Упражнение 00: Преобразование скалярных типов	6
V	Упражнение 01: Сериализация	8
VI	Упражнение 02: Определение вещественного типа	9

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

Глава III

Дополнительно

е правило


Следующее правило применяется ко всему модулю и не является обязательным.

В каждом упражнении преобразование типа должно быть решено с помощью одного конкретного типа приведения.

Ваш выбор будет проверен во время защиты.

Глава IV

Упражнение 00: Преобразование скалярных типов

	Упражнени е 00
Преобразование скалярных типов	
Входящий каталог : <i>ex00/</i>	
Файлы для сдачи : <i>Makefile, *.cpp, *.{h, hpp}</i>	
Разрешенные функции : Любая функция для преобразования строки в <i>int</i> , <i>float</i> или <i>double</i> . Это поможет, но не сделает всю работу.	

Напишите программу, которая принимает в качестве параметра строковое представление литерала C++ в его наиболее распространенной форме. Этот литерал должен принадлежать к одному из следующих скалярных типов: *char*, *int*, *float* или *double*. За исключением параметров *char*, будет использоваться только десятичная нотация.

Примеры литералов символов: *'c'*, *'a'*, ...

Чтобы упростить работу, обратите внимание, что не отображаемые символы не должны использоваться в качестве входных данных. Если преобразование в *char* не является отображаемым, печатается информационное сообщение.

Примеры литералов *int*: *0*, *-42*, *42...*

Примеры литералов *float*: *0.0f*, *-4.2f*, *4.2f...*

Вы должны также обрабатывать эти псевдобуквы (вы знаете, для науки): *-inff*, *+inff* и *nanf*.

Примеры двойных литералов: *0.0*, *-4.2*, *4.2...*

Вам также придется работать с этими псевдобуквами (ну, вы знаете, для развлечения): *-inf*, *+inf* и *nan*.


Сначала необходимо определить тип литерала, переданного в качестве параметра, преобразовать его из строки в его фактический тип, затем **явно** преобразовать его в три других типа данных. И наконец, вывести результаты, как показано ниже.

Если преобразование не имеет смысла или переполняется, выведите сообщение, информирующее пользователя о том, что преобразование типа невозможно. Включите любой заголовок, необходимый для обработки числовых ограничений и специальных значений.

```
./convert 0
char: Не отображается
int: 0
float: 0.0f
double: 0.0
./convert nan
char:
невозможно int:
невозможно
float: nanf
double: nan
./convert 42.0f
char: '42'
int: 42
float: 42.0f
double: 42.0
```


Глава V

Упражнение 01: Сериализация

	Упражнение :
	01
	Сериализа

Реализуйте следующие функции:

`uintptr_t serialize(Data* ptr);`

Он принимает указатель и преобразует его в беззнаковый целочисленный тип `uintptr_t`.

`Data* deserialize(uintptr_t raw);`

Он принимает беззнаковый целочисленный параметр и преобразует его в указатель на `Data`.

Напишите программу для проверки того, что ваши функции работают так, как ожидалось.


Вы должны создать непустую (это означает, что у нее есть члены данных) структуру данных.

Используйте `serialize()` для адреса объекта `Data` и передайте его возвращаемое значение в `deserialize()`. Затем убедитесь, что возвращаемое значение функции `deserialize()` равно исходному указателю.

Не забудьте сдать файлы своей структуры данных.

Глава VI

Упражнение 02: Определение вещественного типа

	Упражнение : 02
Определите реальный тип	
Входящий каталог : ex02/	
Файлы для сдачи : Makefile, *.cpp, *.{h, hpp}	
Запрещенные функции : std::typeinfo	

Реализуйте класс **Base**, который имеет только публичный виртуальный деструктор. Создайте три пустых класса **A**, **B** и **C**, которые публично наследуются от **Base**.



Эти четыре класса не обязательно должны быть оформлены в православной канонической форме.

Реализуйте следующие функции:

Base * generate(void);

Он случайным образом инстанцирует **A**, **B** или **C** и возвращает экземпляр в виде указателя **Base**. Не стесняйтесь использовать для реализации случайного выбора все, что вам нравится.

void identify(Base* p);

Он печатает фактический тип объекта, на который указывает **p**: "**A**", "**B**" или "**C**".

void identify(Base& p);

Она печатает фактический тип объекта, на который указывает **p**: "**A**", "**B**" или "**C**". Использование указателя внутри этой функции запрещено.

Включение заголовка **typeinfo** запрещено.

Напишите программу для проверки того, что все работает так, как ожидалось.

