Упражнения и задачи

Решения задач, отмеченных звездочкой ⋆, можно присылать в любом виде (файл с кодом, ссылка на github, ...) на почту `vkonovodov@gmail.com`. Бонусные баллы за решения задач ⋆ могут быть поставлены нескольким первым приславшим правильное решение.

C++11/14.

(1) ⋆ Реализуйте класс `ComplexNumber` так, чтобы компилировался следующий код:

```
constexpr ComplexNumber Conjugate(const ComplexNumber& x) {
    ComplexNumber res;
    res.SetRe(x.GetRe());
    res.SetIm(-x.GetIm());
    return res;
}

int main() {
    constexpr ComplexNumber a(1, 2);
    constexpr ComplexNumber b(1, -2);
    constexpr auto c = Conjugate(a);
    static_assert(b == c, "failed");
}
```

(2) ⋆ Напишите функцию Compose, которая строит функтор-суперпозицию двух заданных функциональных объектов (принимающих на вход одну переменную). Пример: перевести в массив `double` одним вызовом `std::transform` массив `std::string`, в котором записаны дробные числа:

```
const char* f2(const std::string& str) {
    return str.c_str();
}

int main() {
    std::string s[] = {"1.2", "2.343", "3.2"};
    std::vector<double> d(3);
    auto f1 = atof;
    std::transform(s,
                   s + 3,
                   d.begin(),
                   Compose(f1, f2));
    // ...
}
```

Постарайтесь использовать возможности C++11/14.

Распределение памяти и умные указатели.

(3) Напишите аллокатор, который выполняет стандартное распределение памяти с помощью `operator new` и логирует сообщения о работе каждой функции. Проверьте его работу с несколькими стандартными контейнерами. Попробуйте модифицировать аллокатор так, чтобы он по возможности выделял блоки памяти из внутреннего статического буфера.

(4) ⋆ Реализуйте указатель `TSharedPtr` или `TIntrusivePtr`, создав базовый класс `TBasePtr`, в котором определите операторы `->`, `*`, `==`, `!=`, `bool`. В основном классе должны быть определены основные копирующие и перемещающие операции, методы `UseCount`, `Get`, а также конструктор от обычного указателя.

Продемонстрируйте работу умного указателя с каким-нибудь счетчиком ссылок.

(5) ⋆ В следующей программе не вызываются перегруженные `operator new` и `operator delete`.

```
#include <iostream>

class A {
  public:
    // ...

    static void * operator new(size_t size) {
        std::cout << "operator new!" << std::endl;
        return ::operator new(size);
    }

    static void operator delete(void *p, size_t size) {
        std::cout << "operator delete!" << std::endl;
        return ::operator delete(p);
    }
};

int main() {
    auto sp = std::make_shared<A>();
}
```

Используя функцию `std::allocate_shared` и свой аллокатор, реализуйте создание `std::shared_ptr`, при котором вызывается перегруженный `operator new`, но при этом не используется конструкция `new A`.

Паттерны проектирования.

(6) ⋆ Спроектируйте и реализуйте иерархию классов для генерации случайных чисел, а также фабрику для создания экземпляра класса генератора с заданными параметрами.

```
class TRandomNumberGenerator {
  public:
    virtual -TRandomNumberGenerator();
    virtual double Generate() const = 0;
};
```

Необходимо поддержать три типа распределения: Пуассона, Бернулли и геометрическое. Тип распределения задаётся строковым параметром `type`, который принимает значения из множества `poisson`, `bernoulli`, `geometric`. При создании генератора передаются также параметры распределений.

(7) Используя идиому Type Erasure, напишите класс `Any`, сохраняющий объекты любого типа, и обеспечивающий к ним доступ:

```
Any a(5);
a.get<int>(); // 5
a.get<std::string>(); // error
```

Метапрограммирование.

(8) Напишите шаблон для вычисления биномиального коэффициента C_n^k на этапе компиляции.

(9) ⋆ Определите структуру

```
template <char ...c> struct TString { };
```

и необходимые операторы пользовательских литералов так, чтобы компилировался код:

```
constexpr auto hello = "hello"_s + " world"_s;
static_assert(hello == "hello world"_s);
```