«Концепты» на C++

Ненормальное программирование*, C++*

Всем доброго времени суток.

Придумано и написано под влиянием некоторых публикаций Страуструпа на тему концептов в C++.
Захотелось мне однажды необычного — сделать так, чтобы нешаблонные функции/методы на C++ могли принимать в качестве аргумента любой объект, имеющий определенный набор методов, примерно так:

```
void fn(VectorWrapper<int> x)
{
    for (size_t i = 0; i < x.size(); ++i)
    {
        doSomething(x[i]);
    }
}

::std::vector<int> sv;
QList<int> qv;
OtherSuperVector<int> ov;

fn(sv);
fn(qv);
fn(ov);
```

Причем сделать это не используя наследование от базового класса.
Как это можно сделать, читайте под катом.

Основная трудность, с которой я столкнулся — создание типа VectorWrapper, который имел бы только один шаблонный аргуент (тип хранимого значения), но при этом мог быть создан из чего-угодно, имеющего определенный набор методов. В моем примере это operator[] и size(). После некоторого количества времени раздумий родилась примерно такая конструкция, которая использует возможности стандарта C++11.

```
template <typename T>
class VectorWrapper
{
public:

    template <typename C>
    VectorWrapper(C& container) :
    _getter([&container](size_t i) -> T&
    {
            return container[i];
    }),
```

```
    _sizeGetter([&container]() -> size_t
    {
            return container.size();
    })
    {
    }

    T& operator[](size_t i)
    {
        return _getter(i);
    }

    size_t size()
    {
        return _sizeGetter();
    }

private:
    ::std::function<T&(size_t) > _getter;
    ::std::function<size_t() > _sizeGetter;
};
```

В итоге, при создании объекта этого класса, лямбдами захватывается переданный в конструктор объект, а методы самого класса просто вызывают сохраненные лямбды, дергающие, в свою очередь, методы захваченного объекта.
Теперь в этот враппер можно завернуть все, что угодно, имеющее методы size() и operator[].

Не знаю, можно ли это использовать где-то в реальной жизни, свою проблему, которую я хотел решить таким способом, я решил раньше, чем придумал все это безобразие. Так же есть подозрение, что если повсеместно использовать подобные классы, можно сильно ухудшить производительность.

Ну и чисто из любопытства вопрос хабражителям — можно ли сотворить подобное, не прибегая к помощи лямбд и C++11?