Проектирование больших систем на C++

Коноводов В. А.

кафедра математической кибернетики ВМК

Лекция 2
15.09.2017

Пример с теста

Задача. Напишите код, который удалит из вектора `int`-ов элементы, равные 0.

Пример с теста

Задача. Напишите код, который удалит из вектора `int`-ов элементы, равные 0.

Чем плохо решать задачу так:

```
for (auto i = v.begin(); i < v.end(); ++i) {
    if ((*i) == 0) v.erase(i);
}
```

Пример с теста

Задача. Напишите код, который удалит из вектора `int`-ов элементы, равные 0.

Чем плохо решать задачу так:

```
for (auto i = v.begin(); i < v.end(); ++i) {
    if ((*i) == 0) v.erase(i);
}
```

Вариант решения:

```
v.erase(
    std::remove(v.begin(), v.end(), 0),
    v.end()
);
```

Еще один пример с теста

Задача. Что напечатает программа:

```
class Base {
  public:
    void f() { std::cout << "Base :: f" << std::endl; }
    virtual ~Base() {}
};

class Derived: public Base {
  public:
    void f() { std::cout << "Derived :: f" << std::endl; }
};

int main() {
    Base* b = new Derived;
    b->f();
    delete b;
}
```

Еще один пример с теста, усложним

```
class Base {
  public:
    Base() { f(); }
    virtual void f(int a = 1) {cout << "Base::f " << a << endl;}
    ~Base() { cout << "~Base" << endl; }
};

class Derived: public Base {
  public:
    Derived() { f(); }
    virtual void f(int b = 2) { cout << "Derived::f " << b << endl;}
    ~Derived() { cout << "~Derived" << endl; }
};

int main() {
    Base* b = new Derived;
    b->f();
    delete b;
}
```

Вывод типов шаблонов (в прошлый раз)

Случай 1. Тип параметра — указатель или ссылка:

```
template <typename T>
void f(T& param);
```

Ссылочная часть игнорируется.

Вывод типов шаблонов (в прошлый раз)

Случай 1. Тип параметра — указатель или ссылка:

```
template <typename T>
void f(T& param);
```

Ссылочная часть игнорируется.

Случай 2. Передача по значению:

```
template <typename T>
void f(T param);
```

Ссылочная часть игнорируется. Если после этого остается const, то он тоже игнорируется.

Ключевое слово override

Где ошибки?

```
struct A {
    virtual void foo();
    void bar();
};

struct B : A {
    void foo() const override;
    void foo() override;
    void bar() override;
};
```

Ключевое слово override

Где ошибки?

```
struct A {
    virtual void foo();
    void bar();
};

struct B : A {
    void foo() const override; // ошибка, не совпадают сигнатуры
    void foo() override;  // ok
    void bar() override;  // ошибка, A::bar не виртуальна
};
```

Ключевое слово override

Перекрытие функций:

- Функция базового класса должна быть виртуальной
- В базовом и производном классе должны совпадать:
  - имена функций,
  - типы параметров,
  - константность,
  - возвращаемые типы и спецификации исключений.

Ключевое слово final

Препятствие перекрытия

```
struct A {
    virtual void foo() final; // запрет переопределения функции
    void bar() final; // ошибка, так как функция не виртуальна
};
struct B final : A { // от структуры B нельзя отнаследоваться
    void foo(); // ошибка: A::foo - final
};
struct C : B { // ошибка, B - final
};
```

Удаленные функции

Некопируемый класс:

```
template <typename T>
class TConfig {
  private:
    TConfig(const TConfig&);
    TConfig& operator=(const TConfig&);
};
```

Удаленные функции

Некопируемый класс:

```
template <typename T>
class TConfig {
  public:
    TConfig(const TConfig&) = delete;
    TConfig& operator=(const TConfig&) = delete;
};
```

Удаленные функции

- Удаленной может быть любая функция (не только функции–члены класса).
- Полезное применение — предотвращение использования ненужных инстанциирований шаблонов.

Псевдонимы

typedef:

```
typedef
    std::shared_ptr<std::map<std::string, std::string> >
    TMyPtr;
typedef bool (*FPtr)(int, int);
```

Псевдонимы

typedef:

```
typedef
    std::shared_ptr<std::map<std::string, std::string> >
    TMyPtr;
typedef bool (*FPtr)(int, int);
```

using (C++11):

```
using TMyPtr =
    std::shared_ptr<std::map<std::string, std::string> >;
using FPtr = bool (*)(int, int);
```

В чем отличие typedef от using?

Псевдонимы

typedef:

```
typedef
    std::shared_ptr<std::map<std::string, std::string> >
    TMyPtr;
typedef bool (*FPtr)(int, int);
```

using (C++11):

```
using TMyPtr =
    std::shared_ptr<std::map<std::string, std::string> >;
using FPtr = bool (*)(int, int);
```

В чем отличие typedef от using?

Объявление псевдонимов поддерживает шаблонизацию.

scoped enumerations

```
enum Color {black, white, blue};
bool white; // error!
```

scoped enumerations

```
enum Color {black, white, blue};
bool white; // error!
```

C++11:

```
enum class Color { red, green = 20, blue };
Color r = Color::blue;
switch (r) {
    case Color::red;  // ...
    case Color::green;  // ...
    case Color::blue;  // ...
}
int n = r; // ошибка
int n = static_cast<int>(r);
```

scoped enumerations

```
enum Color {black, white, blue};
bool white; // error!
```

C++11:

```
enum class Color { red, green = 20, blue };
Color r = Color::blue;
switch (r) {
    case Color::red;  // ...
    case Color::green;  // ...
    case Color::blue;  // ...
}
int n = r; // ошибка
int n = static_cast<int>(r);
```

Базовый тип — int.

constexpr

```
const int a = 10;
const int b = std::numeric_limits<int>::max();
const int c = INT_MAX;
```

constexpr

```
const int a = 10;
const int b = std::numeric_limits<int>::max();
const int c = INT_MAX;


int a;
const int b = a; // ok
constexpr auto s = a; // error
```

constexpr

```
const int a = 10;
const int b = std::numeric_limits<int>::max();
const int c = INT_MAX;
```

```
int a;
const int b = a; // ok
constexpr auto s = a; // error
```

```
constexpr int f() {return 1024;}
```

constexpr-функция должна состоять из одного return (C++11), возвращать константу или вызывать такую же функцию. Вычисление должно производиться во время компиляции (с аргументами, значения которых известны во время компиляции).

Пример: проверка простоты числа в compile-time

```
constexpr bool is_div(int a, int b) {
  return (b == 1) || (a % b != 0 && is_div(a, b - 1) );
}

constexpr bool is_prime(int number) {
  return number != 1 && is_div(number, number / 2);
}

int main() {
  static_assert(is_prime(29) , " 29 is not prime");
  static_assert(is_prime(36) , " 36 is prime");
  return 0;
}
```

Задача

Написать **constexpr**-функцию `Size`, которая позволит сделать так:

```
int a[5];
double b[Size(a)];
```

Другие особенности C++11

- перемещение, move-конструкторы и rvalue-ссылки;
- lambda-выражения;
- новые контейнеры: tuple, array, unordered_set, unordered_map, ...;
- умные указатели unique_ptr, shared_ptr, ...;
- ...

Задача

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");
}
```