

# Проектирование больших систем на C++

Коноводов В. А.

кафедра математической кибернетики ВМК  
vkonovodov@gmail.com

Лекции 1-2  
07.09.2018, 14.09.2018

# Вывод типов шаблонов

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

---

В коде

```
int x = 0;  
f(x);
```

тип `T` выводится как `int`.

# Вывод типов шаблонов

**Случай 1.** Тип параметра — указатель или ссылка:

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

Ссылочная часть игнорируется.

---

```
int x = 1;  
const int cx = x;  
const int& rx = x;  
f(x);  
f(rx);  
f(cx);
```

# Вывод типов шаблонов

**Случай 2.** Передача по значению:

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

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

---

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

```
const char* const ptr = "abcd";  
f(ptr);
```

# Автоматический вывод типа переменной

```
auto x = 0;  
auto y = x, *z = &x;  
auto& ref = x;  
const auto & xcref = x;  
auto* p = &x;  
const auto* cp = p;  
const auto pc = &x ;
```

# Автоматический вывод типа переменной

```
auto x = 0;  
auto y = x, *z = &x; // int * y  
auto& ref = x; // int& ref  
const auto & xcref = x; // const int&  
auto* p = &x; // int* p;  
const auto* cp = p; // const int * p;  
const auto pc = &x ; // int * const pc;
```

auto предполагает список `initializer_list` в фигурных скобках, вывод шаблона – нет.

# Ключевое слово `decltype`

```
decltype(1 + 2) x = 1; // int
decltype(x) y = x; // int
decltype(1,x) z = x; // ?
```

`decltype` можно использовать везде, где можно использовать тип.

# Круглые и фигурные скобки

До C++11 невозможно было создать контейнер содержащим определенный набор значений.

```
std::vector<int> v {1, 3, 5};
```

Запрет сужающей инициализации:

```
double a, b;  
int c{a + b}; // error!  
int d = a + b; // ok  
int e(a + b); // ok
```



# Круглые и фигурные скобки

Что будет напечатано?

```
class T {
    public:
        T(int a, bool b)
            { std::cout << "int, bool" << std::endl;}
        T(int a, double b)
            { std::cout << "int, double" << std::endl;}
        T(std::initializer_list<long double> l)
            {std::cout << "init list" << std::endl; }
};

int main() {
    T t1(10, true);
    T t2 {10, true};
    T t3(10, 0.2);
    T t4 {10, 0.2};
    return 0;
}
```

# Круглые и фигурные скобки

Что будет напечатано?

```
class T {
    public:
        T(int a, bool b)
            { std::cout << "int, bool" << std::endl;}
        T(int a, double b)
            { std::cout << "int, double" << std::endl;}
        T(std::initializer_list<long double> l)
            {std::cout << "init list" << std::endl; }
};

int main() {
    T t1(10, true); // int, bool
    T t2 {10, true}; // init list
    T t3(10, 0.2); // int, double
    T t4 {10, 0.2}; // init list
    return 0;
}
```

## range-based циклы

```
for (T var : container) {  
    // ...  
}
```

```
using TElem = std::pair<int, int>;  
using TCont = std::vector<TElem>;  
TCont container;  
for (auto x : container) {  
    // x - копия элемента в контейнере  
}  
for (auto& x : container) {  
    // x - ссылка на элемент в контейнере  
}
```

# 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; // ок  
    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 инстанцированных шаблонов.

# 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!
```

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;
```

```
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;  
}
```

# Другие особенности C++11

- ▶ перемещение, move-конструкторы и rvalue-ссылки;
- ▶ lambda-выражения;
- ▶ контейнеры: tuple, array, unordered\_set, unordered\_map, ...;
- ▶ умные указатели unique\_ptr, shared\_ptr, ...;
- ▶ ...