Проектирование больших систем на C++

Коноводов В. А.

кафедра математической кибернетики ВМК
vkonovodov@gmail.com

Лекция 1
11.09.2019

О чем курс

- Синтаксис C++11/14/17 и новые особенности
- Семантика перемещения, умные указатели
- Паттерны и идиомы в проектировании на C++
- Обработка ошибок, тестирование, отладка
- Оптимизация программ на C++
- Шаблоны и метапрограммирование
- Многопоточность

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; // 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;
```

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.

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.

```
auto cmp = [](char const* p1, char const* p2) {
                    return strcmp(p1, p2) < 0;
                }
std::map<char const*, unsigned, decltype(cmp)> MyMap(cmp);
```

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;  // 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.