Проектирование больших систем на C++

Коноводов В. А.

кафедра математической кибернетики ВМК

Лекция 10
20.11.2018

Вопрос

Что напечатает программа?

```
void f(unsigned i) {
    std::cout << "f(int)" << std::endl;
}

template <typename T>
void f(const T& i) {
    std::cout << "template f" << std::endl;
}

int main() {
    f(3);
}
```

Вопрос

Что напечатает программа?

```
void f(unsigned i) {
    std::cout << "f(int)" << std::endl;
}

template <typename T>
void f(const T& i) {
    std::cout << "template f" << std::endl;
}

int main() {
    f(3);
}
```

template f

SFINAE: Substitution Failure Is Not An Error

А если так?

```
unsigned f(unsigned i) {
    std::cout << "f(int)" << std::endl;
    return i + 1;
}

template <typename T>
typename T::value_type f(const T& i) {
    std::cout << "template f" << std::endl;
    return i + 1;
}

int main() {
    f(3);
}
```

SFINAE: Substitution Failure Is Not An Error

А если так?

```
unsigned f(unsigned i) {
    std::cout << "f(int)" << std::endl;
    return i + 1;
}

template <typename T>
typename T::value_type f(const T& i) {
    std::cout << "template f" << std::endl;
    return i + 1;
}

int main() {
    f(3);
}
```

`f(int)`

Подстановка `int::value_type f(const int i)` невалидна.

std::enable_if

std::enable_if: если передано true, то в структуре присутствует тип type (второй параметр), если передано false, то никакого type нет.

```
template<bool B, class T = void >
struct enable_if;
```

C++14 добавляет алиас:

```
template <bool B, typename T = void>
using enable_if_t = typename enable_if<B, T>::type;
```

std::enable_if: пример

Использование в возвращаемом значении:

```
template <typename T>
std::enable_if_t<std::is_floating_point<T>::value, T> Do(const T& x) {
    std::cout << "float type" << std::endl;
    return x;
}

template <typename T>
std::enable_if_t<std::is_integral<T>::value, T> Do(const T& x) {
    std::cout << "integral type" << std::endl;
    return x;
}
```

std::enable_if: пример

В классах:

```
template<typename T, typename = void>
class A;

template<class T>
class A<T, std::enable_if_t<std::is_integral<T>::value>> {
};
```

- `std::enable_if_t<std::is_integral<int>::value>` → `void`
- `std::enable_if_t<std::is_integral<float>::value>` → ошибка компиляции

Перегрузка и универсальные ссылки – 2

```
#include <string>

class A {
  private:
    std::string text;
  public:
    template <typename T>
    explicit A(T&& str) : text(std::forward<T>(str)) {}
    explicit A(int x) : text(std::to_string(x)) {}
};

int main() {
    A x("123");
    auto copyX(x); // error!
}
```

Перегрузка и универсальные ссылки – 2

Хочется написать так в шаблонный конструктор:

```
template <typename T, typename = std::enable_if_t<CONDITION>>
explicit A(T&& str) : text(std::forward<T>(str)) {}
```

`CONDITION` — тип `T` не является классом `A`.

Перегрузка и универсальные ссылки – 2

Хочется написать так в шаблонный конструктор:

```
template <typename T, typename = std::enable_if_t<CONDITION>>
explicit A(T&& str) : text(std::forward<T>(str)) {}
```

CONDITION — тип T не является классом A.

$$\downarrow\downarrow\downarrow$$

```
!std::is_same<A, T>::value
```

Перегрузка и универсальные ссылки – 2

Хочется написать так в шаблонный конструктор:

```
template <typename T, typename = std::enable_if_t<CONDITION>>
explicit A(T&& str) : text(std::forward<T>(str)) {}
```

`CONDITION` — тип `T` не является классом `A`.

$$\downarrow\downarrow\downarrow$$

`!std::is_same<A, T>::value`

Тип `T` может быть выведен как lvalue-ссылка `A&`, а это не `A`.

Перегрузка и универсальные ссылки – 2

Хочется написать так в шаблонный конструктор:

```
template <typename T, typename = std::enable_if_t<CONDITION>>
explicit A(T&& str) : text(std::forward<T>(str)) {}
```

CONDITION — тип T не является классом A.

$$\downarrow\downarrow\downarrow$$

`!std::is_same<A, T>::value`

Тип T может быть выведен как lvalue-ссылка A&, а это не A.

$$\downarrow\downarrow\downarrow$$

`!std::is_same<A, std::decay_t<T>>::value`

Перегрузка и универсальные ссылки – 2

Всё ок, но есть проблема:

```
class B: public A {
public:
    B(const B& other) : A(other) {...}
    // ...
};
```

Перегрузка и универсальные ссылки – 2

Всё ок, но есть проблема:

```
class B: public A {
public:
    B(const B& other) : A(other) {...}
    // ...
};
```

Воспольуемся этим:

```
std::is_base_of<T1,T2>::value; // истинно, если
                               // T2 - производный от T1

std::is_base_of<int,int>::value; // false
std::is_base_of<A,A>::value;     // true
```

Перегрузка и универсальные ссылки – 2

```
template <
    typename T,
    typename = std::enable_if_t<
        !std::is_base_of<A, std::decay_t<T>>::value
    >
>
explicit A(T&& str) : text(std::forward<T>(str)) {}
```

Но есть еще вторая перегрузка, которую нам нужно вызывать для целочисленных типов:

```
explicit A(int x) : text(std::to_string(x)) {}
```

Перегрузка и универсальные ссылки – 2

Отключаем шаблонный конструктор для обработки целочисленных аргументов:

```
class A {
  private:
    std::string text;
  public:
    template <
        typename T,
        typename = std::enable_if_t<
            !std::is_base_of<A, std::decay_t<T>>::value
            &&
            !std::is_integral<std::remove_reference_t<T>>::value
        >
    >
    explicit A(T&& str) : text(std::forward<T>(str)) {}
    explicit A(int x) : text(std::to_string(x)) {}
};
```

Замечание

`is_arithmetic<T>`: если T является арифметическим типом (целочисленным или в формате с плавающей запятой), то имеется константа-член `value`, которая будет равна `true`. Для всех остальных типов `value` будет равна `false`.

```
std::is_arithmetic<int*>::value;        // false
std::is_arithmetic<int const>::value;   // true
std::is_arithmetic<int&>::value;        // false
```

Пример с закрытым конструктором

```
class C {
  public:
    C() = delete;
    int f() { return 5; }
};

int main() {
    decltype(C().f()) a = 5; // не компилируется
}
```

Пример с закрытым конструктором

```
class C {
  public:
    C() = delete;
    int f() { return 5; }
};

int main() {
    decltype(std::declval<C>().f()) a = 5;  // ok!
}
```

Пример с закрытым конструктором

```
class C {
  public:
    C() = delete;
    int f() { return 5; }
};

int main() {
    decltype(std::declval<C>().f()) a = 5;  // ok!
}
```

Происходит только вывод типа. Работает во время компиляции и только в таких контекстах.

Пример с закрытым конструктором

```
class C {
  public:
    C() = delete;
    int f() { return 5; }
};

int main() {
    decltype(std::declval<C>().f()) a = 5;  // ok!
}
```

Происходит только вывод типа. Работает во время компиляции и только в таких контекстах.

```
template<typename T, typename U>
using TSum = decltype(std::declval<T>() + std::declval<U>());
```

Есть ли в классе метод?

```
template<class...> using dummy = void;

template <class T, typename = void>
struct does_have_super_func : public std::false_type {};

template <class T>
struct does_have_super_func<
    T,
    dummy<decltype(std::declval<T>().super_func())>
> : public std::true_type {};

struct A { int super_func(); };
struct B { int func(); };

int main() {
    std::cout << does_have_super_func<A>::value << std::endl;
    std::cout << does_have_super_func<B>::value << std::endl;
    std::cout << does_have_super_func<int>::value << std::endl;
}
```

C++17: constexpr if вместо SFINAE

```
template<class TIt>
void Sort(TIt first, TIt last) {
    using T = std::iterator_traits<TIt>::value_type;
    if constexpr (std::is_integral_v<T>) {
        RadixSort(first, last);
    } else {
        QuickSort(first, last);
    }
}
```

Readability, Correctness, Efficiency

- Хорошо ли написан код?
- Корректно ли написан код?
- Эффективно ли работает код?

Readability, Correctness, Efficiency

- Хорошо ли написан код?
- Корректно ли написан код?
- Эффективно ли работает код?
- Тестирование
- Формальная верификация (автоматическая/полу-автоматическая)
- ...

Тестирование

- Цель тестирования кода?

Тестирование

- Цель тестирования кода?
- Как искать баги в коде и упростить себе жизнь?

Тестирование

- Цель тестирования кода?
- Как искать баги в коде и упростить себе жизнь?
  - Хороший дизайн кода и code review

Тестирование

- Цель тестирования кода?
- Как искать баги в коде и упростить себе жизнь?
  - Хороший дизайн кода и code review
  - unit-тестирование

Тестирование

- Цель тестирования кода?
- Как искать баги в коде и упростить себе жизнь?
  - Хороший дизайн кода и code review
  - unit-тестирование
  - Интеграционное тестирование

Тестирование

- Цель тестирования кода?
- Как искать баги в коде и упростить себе жизнь?
  - Хороший дизайн кода и code review
  - unit-тестирование
  - Интеграционное тестирование
  - UI-тесты

Тестирование

- Цель тестирования кода?
- Как искать баги в коде и упростить себе жизнь?
  - Хороший дизайн кода и code review
  - unit-тестирование
  - Интеграционное тестирование
  - UI-тесты
  - Мониторинги

Тестирование

- Цель тестирования кода?
- Как искать баги в коде и упростить себе жизнь?
  - Хороший дизайн кода и code review
  - unit-тестирование
  - Интеграционное тестирование
  - UI-тесты
  - Мониторинги
  - Acceptance тестирование

Тестирование

Unit-тесты (модульное тестирование):

- Быстрые
- Полные для своего компонента

Тестирование

Unit-тесты (модульное тестирование):

- Быстрые
- Полные для своего компонента

Интеграционные тесты:

- Медленные
- Проверить всё невозможно
- Могут чаще флапать

Unit-тестирование

Задача: добавляем private-метод в класс с 100500 методами и полями. Нужно протестировать этот метод.

Behavior and State Based Testing

- Тесты, проверяющие что метод или функция отработали корректно, проверяют состояния объекта после вызова.
- Тесты на корректность самого взаимодействия объектов с другими объектами (а не результата).

gtest/gmock

Google C++ Testing Framework.

- Открытая библиотека для unit-тестирования.
- Ключевое понятие — assert
- Результат выполнения:
  - success
  - nonfatal failure
  - fatal failure
- Тест — набор assert'ов
- Набор тестов — тестовая программа или testing suite

gtest

```
ASSERT_TRUE(condition);
ASSERT_FALSE(condition);
ASSERT_EQ(val1, val2);
ASSERT_NE(val1, val2);
ASSERT_LT(val1, val2);
ASSERT_LE(val1, val2);
ASSERT_GT(val1, val2);
ASSERT_GE(val1, val2);
ASSERT_FLOAT_EQ(val1, val2);
ASSERT_DOUBLE_EQ(val1, val2);
ASSERT_NEAR(val1, val2, abs_error);

// strings
ASSERT_STREQ(str1, str2);
ASSERT_STRNE(str1, str2);
ASSERT_STRCASEEQ(str1, str2);
ASSERT_STRCASENE(str1, str2);
```

gtest

```
// exceptions
ASSERT_THROW(statement, exception_type);
ASSERT_ANY_THROW(statement);
ASSERT_NO_THROW(statement);
```

gtest

```
// exceptions
ASSERT_THROW(statement, exception_type);
ASSERT_ANY_THROW(statement);
ASSERT_NO_THROW(statement);
```

Можно добавлять message:

```
ASSERT_EQ(1, 2) << "i dont know why, but 1 != 2";
```

gtest

```
// exceptions
ASSERT_THROW(statement, exception_type);
ASSERT_ANY_THROW(statement);
ASSERT_NO_THROW(statement);
```

Можно добавлять message:

```
ASSERT_EQ(1, 2) << "i dont know why, but 1 != 2";
```

- TEST — макрос для определения теста
- RUN_ALL_TESTS() — запуск всех тестов