C++

Fall 2022

compscicenter.ru

Башарин Егор

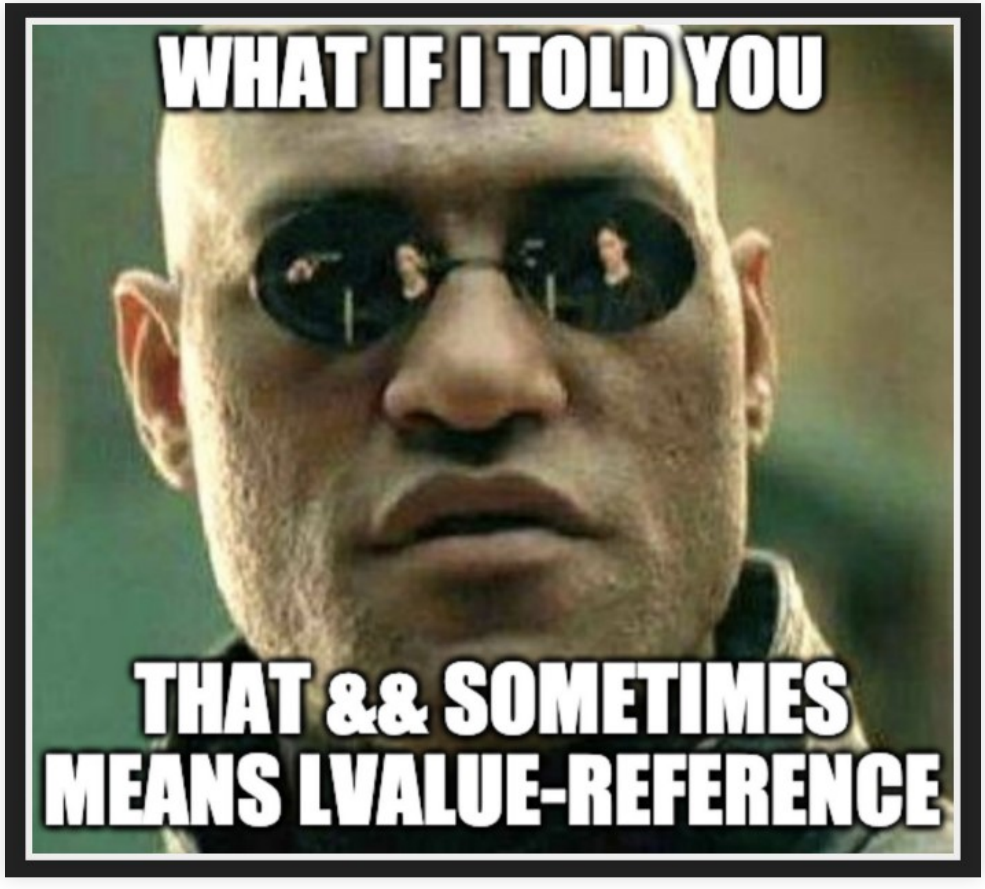
eaniconer@gmail.com
https://t.me/egorbasharin

Лекция X

Universal references. Perfect forwarding

Section 1

Universal references



defining universal references

```
template <class T>
void f(T&& t) { }       // t -- universal reference

struct S {
    template <class T>
    void g(T&& t) { }  // t -- universal reference
};
```

Параметр шаблонной функции, объявленный как `rvalue`-ссылка на параметр шаблона **этой функции**.

```
Для примера выше:
* `t` -- параметр шаблонной функции
* `T&& t` -- объявление параметра функции
* `T` -- параметр шаблона
* `T&&` -- rvalue-ссылка на параметр шаблона
```

usage example

```
template <class T>
void f(T&& t) { }

// 1.
f(10);              // rvalue -> [T = int] -> [T&& = int&&

// 2.
int i = 10;
f(i);               // lvalue -> [T = int&] -> [T&& = int&

// 3.
f(std::move(i));  // rvalue -> [T = int] -> [T&& = int&&
```

https://cppinsights.io/s/e652eac4

```
Первая стрелка показывает результат вывода типа
Вторая стрелка показывает результат схлопывания ссылок
```

Reference collapsing

Схлопывание ссылок

Проблема ссылки на ссылку при использовании псевдонимов типов и параметров шаблона.

	`[T = U&]`	`[T = U&&]`
T&	U&	U&
T&&	U&	U&&

https://cppinsights.io/s/9814f738

Finding universal reference

```
void f(int&& r) {  }                  // (1)

template <class T>
void g(T&& t) { }                     // (2)


template <class T>
void h(const T&& t) { }               // (3)


template <class T>
void p(std::vector<T>&& v) { } // (4)
```

Finding universal reference

```
template <class T>
struct S {

    void f(T&& t) {}    // (1)

    template <class U>
    void g(U&& u) {}    // (2)

};
```

unveil `std::move`

Чтобы понять, как работает `std::move`, напишем свой:

Заготовка: Click me

Решение: Click me

Section 2

perfect-forwarding

Example

```
struct Object {};
int processImpl(const Object&) { return 1; }
int processImpl(Object&&) { return 2; }

template <class T>
int process(T&& t) {
    return processImpl(t);
}

int main() {
    Object obj;
    assert(process(obj) == 1);
    assert(process(std::move(obj)) == 2);
    assert(process(Object{}) == 2);
}
```

Example

```
...
template <class T>
int process(T&& t) {
    return processImpl(t);
}
...
```

Всегда будет выбираться перегрузка с параметром lvalue-ссылкой, так как выражение `t` имеет категорию `lvalue`

Как починить?

Fixing...

```
...
template <class T>
int process(T&& t) {
    return processImpl(t);
}
...
```

Что хотелось бы сделать:

- `T` — ссылка:
  - `lvalue`: `processImpl(t)`
  - `rvalue`: `processImpl(std::move(t))`
- `T` — не ссылка: `processImpl(std::move(t))`

Fixed

`std::forward` from `<utility>`

```
...
template <class T>
int process(T&& t) {
    return processImpl(std::forward<T>(t));
}
...
```

unveil `std::forward`

Чтобы понять, как работает `std::forward`, напишем свой:

Заготовка: Click me

Решение: Click me