

Peter the Great
Saint-Petersburg Polytechnic University

Наука Программирования

Занятие №7
«Variadic Templates. LSP»

Осенний семестр 2018
Преподаватель: асс. каф. Чуканов
В.С

19.11.18

Содержание

- Variadic templates
- Perfect Forwarding
- LSP

Variadic Templates

- Объявление переменного числа типов
 - `template <typename ...Args>`
- Объявление набора параметров типов, заданных `Args`
 - `double f(Args... args)`
- Распаковка кортежа параметров
 - `g(args...);`

Распаковка кортежа

```
template<class ...Us> void f(Us... pargs) {}  
template<class ...Ts> void g(Ts... args) {  
    f(&args...); // "&args..." is a pack expansion  
    // "&args" is its pattern  
}  
g(1, 0.2, "a"); // Ts... args expand to int E1,  
double E2, const char* E3  
// &args... expands to &E1, &E2, &E3  
// Us... pargs expand to int* E1, double* E2,  
const char** E3
```

Распаковка кортежа (2)

```
f(&args...); // expands to f(&E1, &E2, &E3)
f(n, ++args...); // expands to f(n, ++E1, ++E2,
++E3);
f(++args..., n); // expands to f(++E1, ++E2,
++E3, n);
f(const_cast<const Args*>(&args)...);
// f(const_cast<const E1*>(&X1),
const_cast<const E2*>(&X2), const_cast<const
E3*>(&X3))
f(h(args...) + args...); // expands to
// f(h(E1,E2,E3) + E1, h(E1,E2,E3) + E2,
h(E1,E2,E3) + E3)
```

Распаковка кортежа (3)

```
(const args&..) // -> (const T1& arg1, const T2&
arg2, ...)
```

```
((f(args) + g(args))...) // -> (f(arg1) +
g(arg1), f(arg2) + g(arg2), ...)
```

```
(f(args...) + g(args...)) // -> (f(arg1,
arg2, ...) + g(arg1, arg2, ...))
```

Использование лямбда

- Список захвата лямбда-функции может принимать кортеж типов

```
template<class ...Args>
void f(Args... args) {
    auto lm = [&, args...]{ return g(args...); };
    lm();
}
```

Пример: Сумма

```
template <typename T>
double sum(T t) {
    return t;
}
```

```
// Рекурсия: поэлементная развертка кортежа
template <typename T, typename... Rest>
double sum(T t, Rest... rest) {
    return t + sum(rest...);
}
```


Пример №2

```
template <typename T>
T square(T t) { return t * t; }

// Our base case just returns the value.
template <typename T>
double power_sum(T t) { return t; }

// Our new recursive case.
template <typename T, typename... Rest>
double power_sum(T t, Rest... rest) {
    return t + power_sum(square(rest)...);
}
```

```
power_sum(2.0, 4.0, 6.0);
```

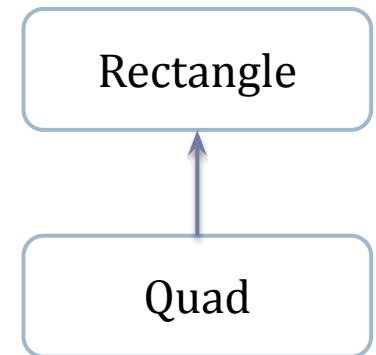
Принцип LSP

- ▣ Принцип подстановки Барбары Лисков
 - ▶ Liskov substitution principle, 1987
 - ▶ Если $\forall x \in Base\ g(x) = true$ то $\forall y \in Derived\ g(y) = true$
 - ▶ Всякий объект базового класса должен быть заменяем наследником с сохранением корректности семантики оперирующего кода

```
class Rectangle
{
public:
    void SetWidth(int w) { w_ = w; }
    void SetHeight(int h) { h_ = h; }
protected:
    int w_;
    int h_;
};
```

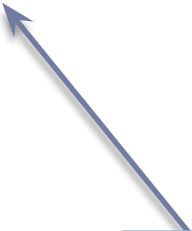
LSP: Пример

- Геометрические фигуры: прямоугольник, квадрат
 - Квадрат – более «специализированное» определение прямоугольника
 - Методы `Set/GetWidth`, `Set/GetHeight`, `GetArea()`
 - Квадрат $\rightarrow w == h$?



LSP: Реализация Rectangle (1)

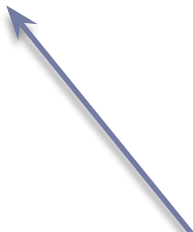
```
class Rectangle
{
public:
    void SetWidth(int w) { w_ = w; }
    void SetHeight(int h) { h_ = h; }
    int GetArea() { return w_ * h_; }
protected:
    int w_;
    int h_;
};
```



Невиртуальные
методы не будут
корректно работать
для Quad

LSP: Реализация Rectangle (2)

```
class Rectangle
{
public:
    virtual void SetWidth(int w) { w_ = w; }
    virtual void SetHeight(int h) { h_ = h; }
    int GetArea() { return w_ * h_; }
protected:
    int w_;
    int h_;
};
```



С точки зрения семантики класса эти методы НЕ ДОЛЖНЫ БЫТЬ виртуальными

LSP: Реализация Quad

```
class Quad : public Rectangle
{
public:
    void SetWidth(int w) { w_ = w; h_ = w; }
    void SetHeight(int w) { w_ = w; h_ = w; }
};
```



Семантика quad
соблюдена

Нарушение LSP

- ❑ Ошибки проектирования иерархии
 - ❑ При реализации Quad потребовалось сделать изменения в базовом Rectangle (+ virtual)
 - ❑ Семантика методов базового класса поменялась
 - ❑ Ранее разделенные методы стали зависимы
 - ❑ Quad не является Rectangle в данном контексте

```
void g(Rectangle *p)
{
    p->SetHeight(5);
    p->SetWidth(4);
    assert(p->GetArea() == 20);
}
```

Заключение

- Variadic templates

- Мощный инструмент создания прототипов для кортежей типов

- LSP

- Подстановка наследника вместо базового не должна менять семантику оперирующего кода