ФТД и замыкания в ОИЯП (C++)

C++ - вариации на тему...

C++11:

- модуль <functional>
- анонимные функции

ФТД и замыкания в ОИЯП (C++)

Какие функциональные объекты (потенциальные значения) есть в C++ (C)?

- функции
- указатели на функции-члены
- функторы

ФТД и замыкания в ОИЯП (C++)

Какие функциональные объекты (потенциальные значения) есть в C++ (C)?

- функции

```
int add (int x, int y) { return x + y;}
```

ФТД и замыкания в ОИЯП (C++)

Какие функциональные объекты (потенциальные значения) есть в C++ (C)?

- указатели на функции-члены

```
class X {
        int _bias;
        X() : _bias(42) { }
....
        int add (int x, int y) { return x + y + _bias; }
        static int Add(int x, int y) { return x + y; }
....
};
```

ФТД и замыкания в ОИЯП (C++)

Какие функциональные объекты (потенциальные значения) есть в C++ (C)?

- функторы

```
class add_t {
        int _bias;
public:
        add_t (int b = 0) : _bias(b) {}
        int ()(int x, int y) { return x + y + _bias; }
};
```

ФТД и замыкания в ОИЯП (C++)

Старый стиль:

- указатель на функцию (C)

```
int (*func)(int, int y)  = add;
int a = func(5,2); // 7
func = X::Add;
a = func(5,2); // 7
```

ФТД и замыкания в ОИЯП (C++)

Старый стиль:

- указатель на функцию (C)
- указатель на функцию-член

```
int (X::*func) (int, int) = &X::add;
X x; X * pX = &x;
x.*func(2,5); // 49!!!
pX->*func(1,2); // 45
```

ФТД и замыкания в ОИЯП (C++)

Старый стиль:

- функтор

```
add_t add;
int a = add(2,5); // 7
add_t add42(42);
a = add42(2,5); // 49
```

!!! Ближе всего к понятию "функциональное значение"

Захватывает значения, в т.ч. и ссылки (в конструкторе)

Чего не хватает - (полу)автоматической поддержки замыканий.

ФТД и замыкания в ОИЯП (C++)

модуль <functional> - "новый стиль":

- указатель на функцию

int (*func)(int, int y) <=> function <int(int, int)> func = add;

Вызов: func(2,5);

- указатель на функцию-член

int (X::*func) (int, int) <=> function <int(X*, int, int)> func = &X::add;

Вызов: func(&x,2,5); func(pX, 2,5);

- функтор

add_t add <=> function <int(int, int)> func = add;

Вызов: func(2,7);

ФТД и замыкания в ОИЯП (C++)

std::function и функторы

Было - функтор без изменения состояния (add_t)

Функторы с изменяемым состоянием:

```
class adder {
        int _sum;
public:
        adder(int identity = 0) : _ sum(identity) {}
        int () (int x) { return _sum += x; }
};
adder a(42);
std::function <int(int)> a1 = a;
std::function <int(int)> a2 = a;
a1(10); // 52
a2(20); //62
```

ФТД и замыкания в ОИЯП (C++)

std::function и функторы с изменяемым состоянием:

```
class adder {
        int _sum;
public:
        adder(int identity) : _ sum(identity) {}
        int () (int x) { return _sum += x; }
};
adder a(42);
std::function <int(int)> a1 = a;
std::function <int(int)> a2 = a;
a1(10); // 52
a2(20); //62
// по значению
```

ФТД и замыкания в ОИЯП (C++)

std::function и функторы с изменяемым состоянием:

```
adder a;
std::function <int(int)> a1 = a;
std::function <int(int)> a2 = a;
std::function <int(int)> ra1 = std::ref(a);
std::function <int(int)> ra2 = std::ref(a);
a1(1); // 1
a2(2); // 2
ra1(1); // 1
ra2(2); // 3
```

ФТД и замыкания в ОИЯП (C++)

std::function - ФТД, обобщающий разновидности прежних указательных типов

std::function <int(int)> f;

Начальное значение - есть - фактически - nullptr?

f(1); // throws std::bad_function_call

Поэтому есть неявное преобразование в bool:

```
if (f)
        a = f(1);
```

ФТД и замыкания в ОИЯП (C++)

std::function - "каррирование" (currying) - функция std::bind

std::bind - связывает функцию с аргументами, возвращает функцию с меньшим числом аргументов

```
int sub(int x, int y) { return x -y; }
std::function<int(int, int)> f0  = std::bind(sub,_1,_2);
std::function<int()> f2 = bind(sub,10,20);
std::function<int(int)> f1 = bind(sub,10, _1);
```

ФТД и замыкания в ОИЯП (C++)

std::function - "каррирование" (currying) - функция std::bind

std::bind - связывает функцию с аргументами, возвращает функцию с меньшим числом аргументов

```
int sub(int x, int y) { return x -y; }
std::function<int(int, int)> f0  = sub;
std::function<int(int,int)> f2 = bind(sub,_1,_2);
std::function<int(int)> f1 = bind(sub,10, _1);
f0(10,20); // -10
f2(10,20); // -10    = sub(10,20)
f1(20); // -10   = sub(10, x) при x = 20
```

а наоборот?

ФТД и замыкания в ОИЯП (C++)

std::function - "каррирование" (currying) - функция std::bind

std::bind - связывает функцию с аргументами, возвращает функцию с меньшим числом аргументов

```
int sub(int x, int y) { return x -y; }
std::function<int(int)> foo = bind(sub,10,_1);
std::function<int(int)> bar = bind(sub,_1,20);
foo(20); // -10    = sub(10, x) при x = 20
bar(10); // -10   = sub(x, 20) при x = 10
std::function<int(int,int)> f = bind(sub,_1,_2); // "масло масляное"
std::function<int(int,int)> ff = bind(sub,_2,_1); // "масло не масляное"
f (x, y) <=> ff(y,x ) <=> sub(x,y)
```

ФТД и замыкания в ОИЯП (C++)

std::function - "каррирование" (currying) - функция std::bind

std::bind - связывает функцию с аргументами, возвращает функцию с меньшим числом аргументов

T f (T1 arg1, T2 arg2, ...)

std::bind(f, ...)

Что такое f? - все, что было (выше) - function <T(T1, T2, ...)>

Реализация bind() :

std::bind(f, _1, std::ref(a1),_2, a2,a3,....)(x,y) => { T2& _a1 = a1; T4 _a2 = a2;

return f(x, a1,y, a2,a3,....);} // _1,_2 заменяются на параметры (x,y)

Как передаются параметры? - по значению... (то есть внутри конкретизации bind запоминаются значения a1,a2,a3 ...)

ФТД и замыкания в ОИЯП (C++)

std::function - "каррирование" (currying) - функция std::bind

std::bind - связывает функцию с аргументами, возвращает функцию с меньшим числом аргументов

Как передаются параметры? - по значению...

```
int a = 1;
std::function <int(int)> sub_a = std::bind(sub, _1, a); // связывание - здесь!!
cout << sub_a(2) << ',' << sub_a(10) ; //1,9
a = 5;
cout << sub_a(2) << ',' << sub_a(10) ; //1,9
```

ФТД и замыкания в ОИЯП (C++)

Как передаются параметры? - по значению...

А по ссылке?

```
int a = 1;
std::function <int(int)> sub_a = std::bind(sub, _1, std::ref(a));
// связывание - здесь - но по ссылке
cout << sub_a(2) << ',' << sub_a(10) ; //1,9
a = 5;
cout << sub_a(2) << ',' << sub_a(10) ; // -3, 5
```

То же самое относится и к ссылкам на функциональный (первый) аргумент....

ФТД и замыкания в ОИЯП (C++)

Как передаются параметры? - по значению...

А по ссылке?

То же самое относится и к ссылкам на функциональный (первый) аргумент....

```
adder a;
std::function <int(int)> add_a = std::bind(a, _1);  // связывание - по значению
cout << add_a(2) << ',' << add_a(10) ; //2,10
std::function <int(int)> add_refa = std::bind(std::ref(a), _1);  // связывание - по ссылке
cout << add_refa(2) << ',' << add_refa(10) << ',' <<  add_refa(1) ; // 2, 12, 13
```

ФТД и замыкания в ОИЯП (C++)

В предыдущих сериях: пока только явное указание "захваченных" значений и переменных:

- функторы - в конструкторе
- bind(....) - все, что не есть _1, _2, _3, ...

Ну и напоследок:

Обобщение - анонимные функции - "захватывают" почти все (но надо тоже указать...)

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: обобщение понятия функтора

Общий вид:

[capture](arg_list)->ret_type { func_body }

это анонимная функция: ret_type ??? (arg_list) {func_body}

Примеры:

```
cout << []() -> int { return 42;} (); // 42
cout << [](int x) -> int { return x + 42;} (10); // 52
int b = 20;
cout << [b](int x) -> int { return x + b;} (10); // 30
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: обобщение понятия функтора
Общий вид:
[capture](arg_list)->ret_type { func_body }
ret_type - выводим <=> можно опускать
Примеры:

```
cout << []() { return 42;} (); // 42
cout << [](int x) { return x + 42;} (10); // 52
int b = 20;
cout << [b](int x) { return x + b;} (10); // 30
cout << [](int x) { return x/2.0; } (3); // 1.5 или около того....
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: обобщение понятия функтора

Общий вид:

[capture](arg_list)->ret_type { func_body }

В capture - как по значению, так и по ссылке

Примеры:

```
int b = 20;
function <int(int)> add_refb = [&b](int x) { return x + b;} ;
function <int(int)> add_b = [b](int x) { return x + b;};
cout << add_b(10); // 30
cout << add_refb(10); // 30
int b = -5;
cout << add_b(10); // 30
cout << add_refb(10); // 5
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: обобщение понятия функтора

Общий вид:

[capture](arg_list)->ret_type { func_body }

В capture - как по значению, так и по ссылке

Примеры:

```
int b = 20;
function <int(int)> add_refb = [b](int x) -> { return x + b;} ;
function <int(int)> add_b = [&b](int x) -> { return x + b;};
cout << add_b(10); // 30
cout << add_refb(10); // 30
int b = -5;
cout << add_b(10); // 30
cout << add_refb(10); // 5
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: обобщение понятия функтора

```
int sum = 0;
std::vector<int> v; ....
std::for_each(v.begin(), v.end(), [&sum](int x) { s += x;});
std::vector<int> squares;
transform (v.begin(), v.end(), squares.begin(), [](int x) { return x*x;});
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: замыкания

[&]() {} // весь контекст по ссылке

[=]() {} // весь контекст по значению

[ident]() {} // ident по значению

[&ident]() {} // &ident

[&x, =]() {} // комбинация

[&, x]() {} // комбинация

[&x, i, j, &s] () {} // комбинация

А что есть контекст? - тело объемлющей функции... => нельзя глобальные переменные...

НО: ***ОЧЕНЬ*** важный частный случай:

[this] () { ... видимость как в функции-члене ...}

[=] () {} // если эта лямбда - внутри функции-члена класса X, то она имеет доступ ко всем членам класса X

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: замыкания

[this] () { ... если эта лямбда - внутри функции-члена класса X, то она имеет доступ ко всем членам класса X... }

// анонимная функция-член класса

А эта функция - еще и имеет доступ (по значению) к локальным переменным объемлющей функции-члена класса

[=] () {...если эта лямбда - внутри f() - функции-члена класса X, то она имеет доступ ко всем членам класса X, также ко всем локальным переменным функции f.... }

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: замыкания

[this] () { ... видимость как в функции-члене ...}

[=] () {} // если эта лямбда - внутри функции-члена класса X, то она имеет доступ ко всем членам класса X

```
class X {
                        int _i;
public:
                        X() : _i(0) {}
                        std::function<void(void)>  foo() {
                                        return [this]() { ++_i; };
                        }
                        void print() { cout << _i << endl;}
};
X& x = *new X();
x.print(); // 0
x.foo()();
x.print(); // 1
x.foo()();
x.print(); // 2
delete &x;
auto ff = x             std::vector<int> v; ....
.foo();
ff(); ff(); // _i .....
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: замыкания

```
X& x = *new X();
x.print(); // 0
x.foo()(); x.print(); // 1
x.foo()(); x.print(); // 2
auto ff = x.foo();
// если в этом месте сделать - delete &x; // ff ->this - повисла
ff(); ff(); // _i .....
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: обобщение понятия функтора

Осторожно со ссылками!

```
int sum = 0;
std::vector<int> v; ....
std::for_each(v.begin(), v.end(), [&sum](int x) { s += x;});
```

Что будет, если функция "покинет" ОВ?

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: осторожно со ссылками!

```
std::vector<int> v;
std::function<int (int)> f () {
        int sum = 0;
        return std::for_each(v.begin(), v.end(), [&sum](int x) { return sum += x;});
}
auto ff = f();
int a = ff(0);
// undefined behavior - ссылка на sum -"повисает"
```

ФТД и замыкания в ОИЯП (C++)

Лямбда-функции в C++: замыкания

Насколько это важно?

Меняет подход (можно сказать, и парадигму) с чистого ООП на элементы функционального (не заменяя, а дополняя)

Упрощенный пример:

class MarvelGrid {

// умеет распознавать что-то особенное и уведомлять об этом владельца (то есть объект визуального класса, внутри которого располагается экземпляр класса MarvelGrid)

};

Как уведомлять - это главный вопрос.

ФТД и замыкания в ОИЯП (C++)

Упрощенный пример:

```
class MarvelGrid {
// умеет распознавать что-то особенное и уведомлять об этом владельца (то есть объект визуального класса, внутри которого располагается экземпляр класса MarvelGrid)
};
class Wnd1 {
   ... MarvelGrid _grid;
};
class Wnd2 {
   .... MarvelGrid _grid;
}; // и так далее
```

Как _grid уведомляет своего владельца?

ФТД и замыкания в ОИЯП (C++)

Один из вариантов ООП-подхода: ввести (абстрактный) базовый класс, который декларирует реакции на уведомление

```
class WndWithMarvelGrid {
        friend class MarvelGrid;
public:
         WndWithMarvelGrid() : _grid(this) { .... }
protected:
        virtual void OnMarvelEvent()  {};
        MarvelGrid _grid;
};
class Wnd1 : public WndWithMarvelGrid {
protected:
        void OnMarvelEvent() override;
};
class Wnd2: public WndWithMarvelGrid {
protected:
        void OnMarvelEvent() override;
};
```

ФТД и замыкания в ОИЯП (C++)

Один из вариантов ООП-подхода: ввести (абстрактный) базовый класс, который декларирует реакции на уведомление, и добавить в класс MarvelGrid владельца, который умеет реагировать на события:

```
class MarvelGrid {
        WndWithMarvelGrid * _Parent;
public:
        MarvelGrid(WndWithMarvelGrid * pParent) : _Parent(pParent) {}
protected:
        void FireMarvelEvent() { _Parent -> OnMarvelEvent(); }
};
```

ФТД и замыкания в ОИЯП (C++)

Проблема: появляется новый класс-владелец.

Как его добавлять?

```
class AnotherWnd: public WndWithMarvelGrid {
protected:
        void OnMarvelEvent() override;
};
```

Real life: WndWithMarvelGrid - много всего....

ФТД и замыкания в ОИЯП (C++)

```
class WndWithMarvelGrid : public SomeWindow {
        friend class MarvelGrid;
public:
         WndWithMarvelGrid() : _grid(this) { .... }
        // много всего другого
protected:
        virtual void OnMarvelEvent() = 0;
        MarvelGrid _grid;
};
```

ФТД и замыкания в ОИЯП (C++)

Новый класс-владелец.

class AnotherWnd: public SomeOtherWindow, public WndWithMarvelGrid { // конфликт SomeOtherWindow и SomeWindow

protected:

	void OnMarvelEvent() override;

};

Real life: WndWithMarvelGrid и AnotherWnd - много всего....

ФТД и замыкания в ОИЯП (C++)

Что делать?

ФП

```
class MarvelGrid {
        std::function<void(void)> _OnEvent;
public:
        MarvelGrid(std::function<void(void)> _OnEvent) : _OnEvent(OnEvent) {}
        void FireMarvelEvent() { if (_OnEvent) _OnEvent(); }
};
```

ФТД и замыкания в ОИЯП (C++)

Новый класс-владелец.

```
class AnotherWnd: ......{
        AnotherWnd() : m_Grid([this](){ OnMarvelEvent();}) {
        }
protected:
        MarvelGrid m_Grid;
        void OnMarvelEvent();
};
```

ФТД и замыкания в ОИЯП

Python - замыкания

```
def inc1(): # замыкание
        n = 0
        def f():
                nonlocal n
                n += 1
                return n
        return f
g = inc1()
g()
g()
```

ФТД и замыкания в ОИЯП

Динамические ЯП: Python, JavaScript

```
def incN(n): # замыкание
        def f(x):
                return x + n
        return f
a = incN(0)
a(1)
a(10)
b = incN(5)
b(0)
b(10)
```

ФТД и замыкания в ОИЯП

Python - замыкания и анонимные функции

```
def add(x,y):
        return x + y

def add1(x): # не замыкание
        return add(x, 1)

def add1_1(): # замыкание
        def adder1(x):
                return add(x, 1)
        return adder1

def add1_2():
        return LAMBDA(x): add(x, 1)

inc1 = add1_1()
inc1(5)
```

ФТД и замыкания в ОИЯП

JavaScript - замыкания и анонимные функции

```
function add(x, y) {return x+ y}
function add1(){
        return function (x) { return add(x, 1) ;}
}
```

или вот так (JavaScript-2015):

```
function add1(){       return x=>add(x, 1) }
```

Пример: сумматор

```
function adder(init, N) { var sum = init; return () => sum += N; }
f = adder(0,10)
f(); f(); f(); // 10,20,30,40, ......
```