ППОИС

Часть 1

Функциональный стиль программирования

Функциональное программирование (1)

- Функции являются объектами первого класса (First Class Object)
- Использование рекурсии в качестве основной структуры контроля потока управления
- Акцент на обработке списков
- Используются функции высшего порядка (High Order Functions)

Функциональное программирование (2)

- Функции являются “чистыми” (Pure Functions) – т.е. не имеют побочных эффектов
- Акцент на том, что должно быть вычислено, а не на том, как вычислять
- В функциональных языках не используются переменные (как именованные ячейки памяти)

Лямбда-выражения

Пример использования функторов

```
#include <iostream>
#include <algorithm>
#include <vector>

struct PrintFunctor {
    void operator()(int x) const {
        std::cout << x << std::endl;
    }
};

int main() {
    std::vector<int> v;
    v.push_back(1);
    v.push_back(2);
    std::for_each(v.begin(), v.end(), PrintFunctor());
}
```

Замена на лямбда-выражение

```
#include <iostream>
#include <algorithm>
#include <vector>

int main() {
    std::vector<int> v;
    v.push_back(1);
    v.push_back(2);
    std::for_each(v.begin(), v.end(), [] (int x) {
            std::cout << x << '\n';
        }
    );
}
```

Что такое лямбда-выражение

Выглядит как безымянная локальная функция, которые можно создавать внутри какого-либо выражения

На самом деле объект, у которого перегружен оператор () , т.е. функтор

Из чего состоит

```
lambda-expression ::=
                    ‘[’  [<список_захвата>]  ‘]’
                  [ ‘(’  <список_параметров> ‘)’  [‘mutable’ ] ]
                  [ ‘noexcept’ ]
                  [ ‘->’  <тип_возвращаемого_значения> ]
                    ‘{’  [<тело_лямбды>]  ‘}’
```

Список захвата

```
[]                      // без захвата переменных из внешней области видимости
[=]                     // все переменные захватываются по значению
[&]                     // все переменные захватываются по ссылке
[x, y]                  // захват x и y по значению
[&x, &y]                // захват x и y по ссылке
[in, &out]              // захват in по значению, а out — по ссылке
[=, &out1, &out2]       // захват всех переменных по значению, кроме out1 и out2,
                        // которые захватываются по ссылке
[&, x]                  // захват всех переменных по ссылке, кроме x…
```

Объекты, захваченные **значением** в лямбда, по умолчанию неизменяемы, но можно указать **mutable**

Список параметров

- почти такой же, как и в обычных функциях
- если пустой, то может быть опущен

```
auto call_foo  = [x](){ x.foo(); };

auto call_foo2 = [x]{ x.foo(); };
```

Тело функции

как и в обычных функциях

Возвращаемое значение (неявно и явно):

```
[](float f, int a) { return a*f; }

[](MyClass t) -> int { auto a = t.compute(); return a; }
```

Вызов и переиспользование

```
int multiplier = 5;
auto timesFive = [multiplier](int a) { return a * multiplier;
};
std::out << timesFive(2); // Prints 10


[] () { std::cout “Hello”; } ()


std::function<int(int, int)> gcd = [&](int a, int b){
    return b == 0 ? a : gcd(b, a%b);
};
```