Объектно-ориентированное программирование

Петрусевич Денис Андреевич

Доцент Кафедры Проблем управления

petrusevich@mirea.ru

Тема 5. Функторы

Функтор – объект, который можно использовать как функцию (объект класса, в котором переопределен operator())

Чаще всего, функторы передаются как параметры другой функции

Аналогично в функции можно передавать указатели на другие функции (обычно достаточно передать адрес функции)

Тема 5. Функции как объекты

Переопределение операции сравнения элементов в функции сортировки

```
//c1<c2?
bool compare_chars(char c1, char c2)
{ }
…
list<char> lst;
lst.sort(compare_chars);
```

Тема 5. Функции как объекты

Применение функции к элементам списка

```
char CaesarForward(char c)
{…}
…
transform(lst.begin(), lst.end(), lst.begin(), CaesarForward);
```

Тема 5. Функции как объекты

Подсчет числа элементов, для которых функция glas() вернула true

```
char glas(char c)
{…}
…
int n2 = count_if(lst.begin(), lst.end(), glas);
```

Тема 5. Функторы

Функтор – объект класса, где переопределена операция operator()

```
class EvenOddFunctor {
…
void operator()(int x) {…}
…
};
…
EvenOddFunctor evenodd;
evenodd = for_each(my_list, my_list + 10, evenodd);
```

Тема 5. Лямбды

Анонимная функция внутри другой функции

```
[ captureClause ] ( параметры ) -> возвращаемыйТип
   {
   //тело функции
   }
```

Поля «captureClause», «параметры» могут быть пустыми

Нет возвращаемого типа => используется вывод типа с помощью auto

Не требуется создавать отдельную функцию для задачи (обычно очень простая)

Тема 5. Лямбды

Поле captureClause – для доступа к переменным из окружающей области видимости из лямбды (указать, какие переменные захватить)

Для каждой используемой переменной создаётся копия с тем же именем, которую можно использовать в лямбда-функции

По умолчанию изменить значения переменных в основной функции нельзя

Тема 5. Лямбды

Использование лямбд в алгоритмах

```
int main()
{
struct
{
   void operator()(int x) const { std::cout << x << '\n'; }
} someInstance;
…
std::vector<int> v; …
std::for_each(v.begin(), v.end(), someInstance);
std::for_each(v.begin(), v.end(), [] (int x) { std::cout << x
   << '\n'; } ); }
```

Тема 5. Лямбды

Захват переменных

Работает только для автоматических переменных. Нельзя захватить глобальные, статические переменные

```
int main()
{
    int x = 10;

    [x]{ cout << x; }(); //захвачен x, выводим значение на экран

}
```

Тема 5. Лямбды

Захват всех переменных по значению

```
int main()
{
    int x = 10;

    [=]{ cout << x; }(); //захвачен x, выводим значение на
   экран

}
```

Тема 5. Лямбды

Захват полей класса

```
class MyClass{
    const int x = 100;
public:
    void show(){
        [this]{ cout << x; }();      //захват this
    }
};
int main() {
    MyClass m;
    m.show();
}
```

Тема 5. Лямбды

Как изменить внешнюю для лямбды переменную?

- mutable
- Захват по ссылке

Тема 5. Лямбды

Захват всех значений по ссылке

```
int x = 10;
double y = 5.5;

[&](){ x++, y = y + 10; }();          //захват всех значений
по ссылке из текущей области видимости

cout << "x == " << x << '\n';          //11
cout << "y == " << y << '\n';          //15.5
```

Тема 5. Лямбды

Захват по ссылке

```
int ammo{ 10 };
auto shoot
{
    [&ammo]()
    { // переменная ammo захватывается по ссылке
      --ammo;
      cout << "Pew! " << ammo << " shot(s) left.\n";
    }
};
```

Тема 5. Лямбды

Использование ключевого слова mutable

```
int ammo{ 10 };
auto shoot
{
    // Добавляем ключевое слово mutable
    [ammo]() mutable {
    --ammo;
    cout << "Pew! " << ammo << " shot(s) left.\n";
  }
};
```

Тема 5. Лямбды

Передача параметров в лямбды – с помощью круглых скобок

Первые скобки – для списка формальных параметров

Вторые скобки – для передаваемых значений

```
[](const int x){ cout << "x == " << x; }(10);
[](const int x, const double d){ cout << "x == " << x << "\nd
   == " << d; }(10, 20.3);
```

Тема 5. Лямбды

Лямбда-функцию можно сохранить в переменную типа auto и затем вызвать

```
auto lambda_fun2 = []{ return 200; };
cout << lambda_fun2();//выведет на экран 200
```

Тема 5. Лямбды

Ещё один пример захвата по ссылке

```
int x = 1, y = 1;
auto foo = [&x, &y]() { ++x; ++y; };
foo();
std::cout << x << " " << y << std::endl;
```

Тема 5. Лямбды

```
auto lambda_fun1 = [](){};      //Запомнили лямбда-функцию в
   объект
auto lambda_fun2 = [](){ return 200; };
auto hello_world1 = [](){ cout << "hello"; };
auto hello_world2 = [](const char* S){ cout << S; };

//вызываем лямбда-функции с помощью объектов-хранителей
lambda_fun1();          //пустая, ничего не произойдёт
lambda_fun2();          //ничего не произойдёт
cout << lambda_fun2();//выведет на экран 200
hello_world1();       //выведет на экран "hello"
hello_world2("hi-hi-hi");       //выведет на экран "hi-hi-hi"
```

Тема 5. Лямбды

Схожим образом можно использовать указатель на функцию

```
void foo(double (*ptr)())
{cout << ptr() << '\n';}
int main()
{
    auto lambda_fun = [](()->double{ … };
   foo(lambda_fun);
}
```

Тема 5. Лямбды

Второй подход к использованию указателя на лямбду

```
void foo(function<double()> ptr)
{cout << ptr() << '\n';}

int main()
{
    auto lambda_fun = [](()->double
     {
         return 200.5;
     };
foo(lambda_fun);
}
```

Тема 5. Лямбды

У лямбды два параметра. Их типы указаны в скобках (по аналогии с заданием указателя на функцию)

```
void foo(function<double(int, string)> ptr)
{      cout << ptr(0, "") << '\n';}
int main()
{
    auto lambda_fun = [](int x, string S)->double
    {
      return 200.5;
    };
foo(lambda_fun);
}
```

Тема 5. Лямбды

Можно использовать шаблоны функций

```
template <typename T>
void foo(T ptr)
{cout << ptr(0, "") << '\n';}

int main()
{
    auto lambda_fun = [](int x, string S)->double
    {
      return 200.5;
    };
foo(lambda_fun);}
```

Тема 5. Лямбды

```
[] // 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, &y] // захват всех переменных по ссылке, кроме x…
```

Тема 5. Ключевое слово auto

Сообщение компилятору определить тип переменной по инициализирующему значению

```
auto x = 4.0; //double
auto y = 3+4; //целое
```

Можно применять, когда тип имеет сложный вид (шаблоны, указатели на функции и т.п.)

Тема 5. Ключевое слово auto

```
auto var = some_expression;
```

Тип some_expression	Тип var
T*, const T*	T*, const T*
T, const T, T&, const T&	T

```
auto& var = some_expression;
```

Тип some_expression	Тип var
T&	T&
const T&	const T&

Тема 5. Ключевое слово auto

Не следует использовать как тип параметров функции, но можно использовать как тип результата

```
auto foo(auto v1, auto v2) -> decltype(v1+v2) ;
int foo(auto v1, bool v2);
foo(“C++ is cool?”, true);
```

Есть шаблоны функций. `auto` здесь не нужно

Тема 5. Ключевое слово auto

Плохо (тип x?):

```
void mySwap(auto a, auto b)
{
    auto x = a;
    a = b;
    b = x;
}
```

Хорошо:

```
auto subtract(int a, int b)
{
    return a - b;
}
```

Тема 5. Ключевое слово auto

Перебор элементов в контейнере

```
deque<double> dqDoubleData(10, 0.1);
for (auto iter = dqDoubleData.begin(); iter !=
   dqDoubleData.end(); ++iter)
{ /* ... */ }
for (auto elem : dqDoubleData) // COPIES elements, not much
   better than the previous examples
{ /* ... */ }
```

Тема 5. Ключевое слово auto

Перебор элементов в контейнере

```
deque<double> dqDoubleData(10, 0.1);
for (auto& elem : dqDoubleData) // observes and/or modifies
   elements IN-PLACE
 { /* ... */ }
for (const auto& elem : dqDoubleData) // observes elements IN-
   PLACE
{ /* ... */ }
```

Литература

Лямбды и функторы:

https://habr.com/ru/post/66021/

https://habr.com/ru/company/otus/blog/455978/

https://habr.com/ru/company/otus/blog/444524/

http://ci-plus-plus-snachala.ru/?p=11

https://ravesli.com/lyambda-zahvaty-v-s/

https://ravesli.com/lyambda-vyrazheniya-anonimnye-funktsii-v-s/

auto/decltype:

https://docs.microsoft.com/ru-ru/cpp/cpp/auto-cpp?view=vs-2019

https://docs.microsoft.com/ru-ru/cpp/cpp/decltype-cpp?view=vs-2019

https://habr.com/ru/post/206458/

https://ravesli.com/urok-62-klyuchevoe-slovo-auto-vyvod-tipov/

Спасибо за внимание!

Петрусевич Денис Андреевич

Доцент Кафедры Проблем управления

petrusevich@mirea.ru