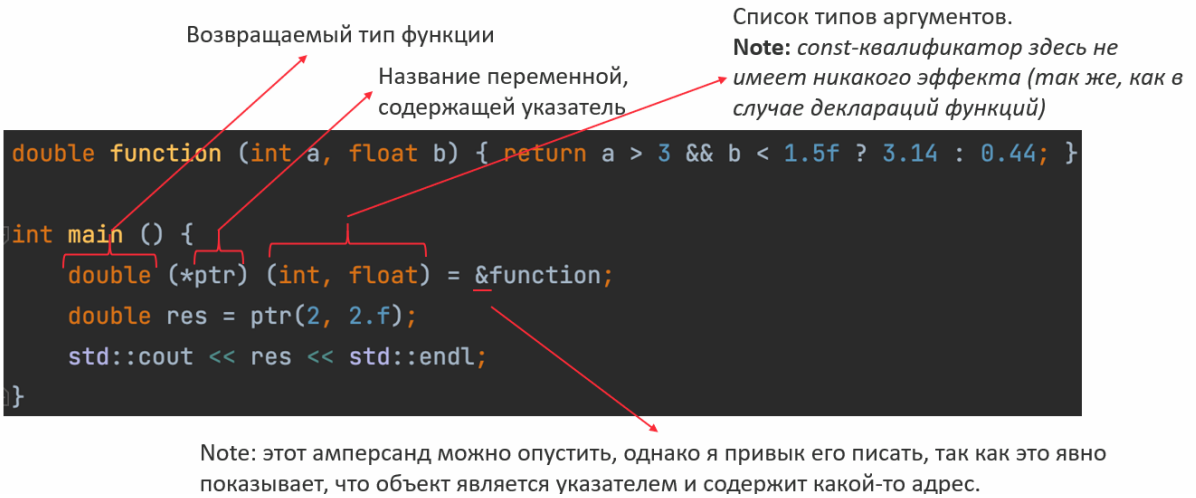
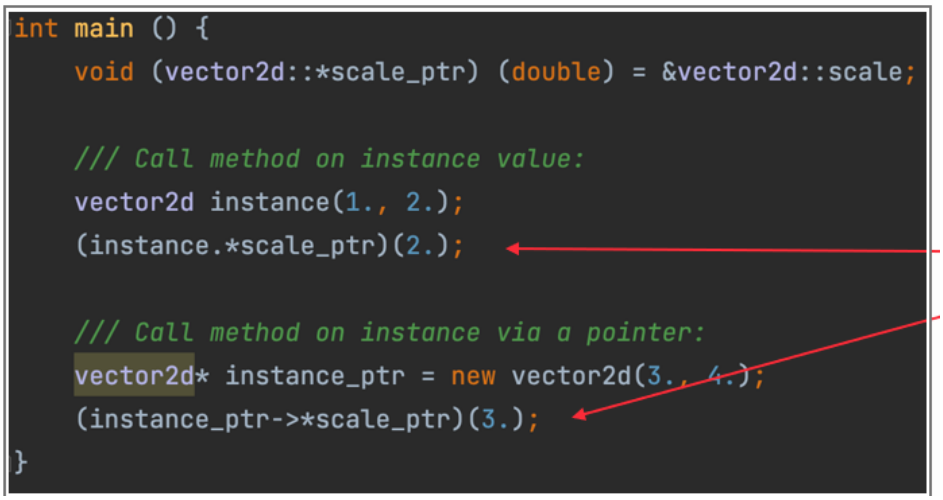
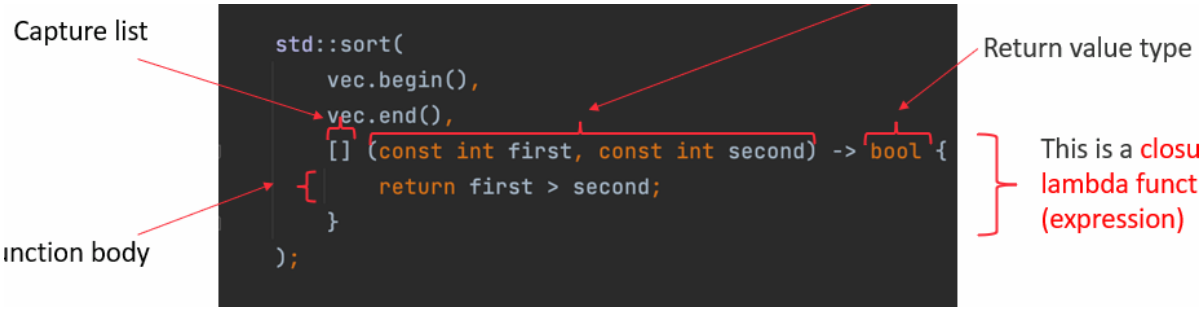
- *Функтор (функциональная сущность) это просто объект, у которого определен оператор **operator()** Нахуя такие нужны? Как минимум для инкапсуляции связанных с ними методов*
- *Указатели на функцию желательно наглядно типизировать передавая в другие функции, а не писать auto в аргументах*



- *Можно создавать также указатели на методы классов и вытаскивать их из определённой сущности класса. Зачем? Кто бы знал…*



- *С помощью оператора :: доступа к пространству имён объекта можно также вытащить указатель на поле &my_class::x*
- *Лямбда выражения позволяют создавать функции без названия (но можно дать им название, сохранив в переменную типа auto)*



- *В листе захвата перед лямбдой можно давать лямбде доступ к переменным из внешнего пространства имён (и ссылкам на них) [&variable], также переприсваивая их [name_in_lambda = variable*2].*
- *При захвате по не по ссылке всё равно можно изменять переменный написав ключевое слово mutable*

```
int num = 0;
std::function<void()> func = [num] () mutable -> void {
    /// Incrementing the local copy
    std::cout << ++ num << std::endl;
};
```

- *Если нужно захватить константную ссылку, то нужно использовать [&name_in_lambda = std::as_const(variable)]*
- *Можно захватить всё пространство имён через [=] (как переменные) или [&] (как ссылки), но так лучше не делать, так как можно сломать много объектов в изначальном пространстве имён, передав лямбду в другое пространство (например функцию).*
- *В аргументах лямбды можно использовать auto. Это по сути делает её шаблоном, но неявным, поэтому использовать это нужно только когда вы точно не будете передавать неподходящие типы в неё.*

НЕ ОК, подразумевалось использование template

```
auto multiply = [] (auto val) -> decltype(val) { return val * 2; };
std::cout << multiply(2) << " " << multiply(3.14) << std::endl;
```

- *Или, начиная с 20ых плюсов, использовать явные шаблоны*

```
auto forward_into_receiver =
    [&receiver = std::as_const(receiver)] <typename... args> (args&&... values) {
        receiver(std::forward<args>(values)...);
    };
```

- ***std::function** создан для того, чтобы хранить и передавать лямбды или **любые** другие функциональные сущности в другие функции и методы и явно объявлять тип значений, которые они принимают и возвращают.*

```
void receive_and_call (const std::function<void(int, float, double)>& f) {
    f(4, 2.31f, 3.14);
}
```

std::bind
std::mem_fn
std::reference_wrapper
std::invoke
Кратко описаны в презентации