59 Lambdas in C++

1. lambda用来做什么

*lambda*本质上是我们定义一种叫做匿名函数的方式，用这种方法不需要实际创建一个函数，就像是一个快速的一次性函数，我们更想将它视作一个变量而不是像一个正式的函数那样，在我们编译的代码中作为一个符号存在。

只要你有一个函数指针，你都可以在C++中使用lambda，这就是它的工作原理，所以lambda是我们不需要通过函数定义就可以定义一个函数的方法。
lambda的用法是，在我们会设置函数指针指向函数的任何地方，我们都可以将它设置为lambda。

lambda是一个指定一个函数未来想要运行的代码的很好的方法。

如果我们想把外部变量放到lambda函数内部的指令中呢?
和我们创建自己的函数其实一样，都是有两个方法：值传递和引用传递，这也就是捕获这一块的东西，`[]`就是我们打算如何传递变量。（`[=]`,传递所有变量，通过值传递；`[&]`传递所有变量，通过引用传递）
还可以只传入单独的变量，`[a]`通过值传递传入a,`[&a]`通过引用传递。

```C++
#include <iostream>
#include <vector>
#include <functional>

void ForEach(const std::vector<int>& values,const std::function<void(int)>& func)
{
    for (int value : values)
        func(value);
}

int main()
{
    std::vector<int> values = { 1,5,4,2,3 };
    int a = 5;
    auto lambda = [=](int value) {std::cout << a << std::endl; };
    ForEach(values,lambda);

    std::cin.get();
}
```

尽管这里在lambda中值传递并复制了a，但是不允许我们在其中对a重新赋值，不像普通函数中一样。要修复这个问题，只需要在前面加入`mutable`（这些都可以在cppreference.com中了解到）

```C++
auto lambda = [=](int value) mutable { a = 6; std::cout << a << std::endl; };

// 6 6 6 6 6
```

2. 为什么可能想要使用lambda

```cpp
#include <algorithm>

auto it = std::find_if(values.begin(), values.end(), [](int value) {return value > 3;
});

std::cout << *it << std::endl;
```

`std::find_if` 函数在 `values` 容器中查找满足 lambda 表达式的条件的第一个元素。`values.begin()` 和 `values.end()` 分别表示容器的起始和结束迭代器，这个函数将在容器的所有元素范围内查找。
`[](int value) {return value > 3; }` 是一个匿名的 lambda 表达式，它表示查找条件：元素的值大于 3。
`std::find_if` 会返回一个迭代器指向第一个满足条件的元素，如果没有找到符合条件的元素，将返回 `values.end()` 的迭代器。

相当于：

```cpp
void ForEach(const std::vector<int>& values,const std::function<void(int)>& func)
{
    for (int value : values)
        if(value > 3)    // 添加bool判断
            func(value);
}
```

但是我们可以使用lambda，通过指定这个布尔局域快速地实现。