CodInClub
Powered By BridgeLabz

Wk5 /S3/ Lecture # : DSOOPS-22

Topics Covered

- What is a Lambda Function?
- Basic Syntax and Use
- Lambda with Parameters
- Capturing Variables (by value & by reference—including [=], [&])
- Passing Lambdas to Functions

What is a Lambda Function?

A lambda function is a small, quick, unnamed function you can write right where you use it—no need for a full separate definition. Lambdas help keep your code concise and are especially handy for short tasks, like sorting or filtering.

Example:

```
auto sayHello = []() { std::cout << "Hello!\n"; };
sayHello(); // Output: Hello!
```

Basic Lambda Syntax

```
auto myLambda = []() { /* code here */ };
myLambda(); // Call it
```

- The brackets `[]` are the *capture list*—explained more below.

Lambda with Parameters

Lambdas can take arguments and return results like regular functions.

```
auto add = [](int a, int b) { return a + b; };
std::cout << add(2, 3) << std::endl; // Output: 5
```

Capturing Variables (The Capture List: [ ], [=], [&])

Suppose you want your lambda to use variables from the surrounding code. You use the capture list (inside the `[]`) to control this:

- Capture by Value:
  If you write `[=]`, you are letting the lambda use all surrounding variables, but only as local copies. So, even if you change variables inside the lambda, the originals stay the same outside of it.

```
int a = 5, b = 7;
auto sum = [=]() { std::cout << a + b << std::endl; }; // a and b copied
sum(); // Output: 12
// a and b are NOT changed outside lambda
```

- Capture by Reference:
  Writing `[&]` means the lambda can use and change all local variables directly ("by reference").

```
int count = 0;
auto inc = [&]() { count++; }; // count can be changed by lambda
inc();
std::cout << count << std::endl; // Output: 1
// count IS changed outside lambda
```

- Custom Capture:
  You can also do things like `[a, &b]` to capture a by value and b by reference, or `[=, &c]` to capture all by value except c, which is by reference.

Passing Lambdas to Functions

You can use lambdas directly as function arguments (great for things like applying a function to many elements):

```
#include <functional>
void runTwice(std::function<void()> func) {
    func();
    func();
}

int main() {
    auto greet = []() { std::cout << "Hi!\n"; };
    runTwice(greet); // Output: Hi! Hi!
    return 0;
}
```

Practice Problems and Activities

Easy 1

What will be the output? Why?

```
auto greet = []() { std::cout << "Welcome!\n"; };
greet();
```

Tip: Look at how this lambda is being called.

Easy 2

Write a lambda that squares a number and prints the result. Call it with 6.

Medium

Use a lambda and a loop to print only even numbers from this array:

```
int arr[] = {4, 7, 10, 13};
```

Hint: Try `[](int x) { /* ... */ }` inside your loop.

Hard

Given `int sum = 100;`, write a lambda that takes an integer x, adds it to sum, and updates the original sum variable. Show sum is updated after calling the lambda.
(Hint: Use `[&]` to capture sum by reference.)

Wrap-Up & Key Takeaways

- Lambdas are one-off, inline functions for quick jobs.
- Use the capture list `[ ]` to access outside variables:
  - `[=]`: all by value (copies, originals unchanged)
  - `[&]`: all by reference (originals can be modified)
  - Can mix as needed for more control.
- Lambdas can be passed to other functions for tasks like filtering, sorting, or multiple execution.
- Writing and using simple lambdas helps practice functional, modern C++ code!