

Lambda Functions Part Four Solutions

- What is meant by variable capture in a lambda expression?
 - A variable that is “captured” from the enclosing scope can be used in the body of a lambda expression.
 - If the variable is captured by value, the body will get its own copy of the original variable
 - If the variable is captured by reference, the body will get a reference to the original variable

- What options are available to a programmer who wants to be able to alter a captured variable?
 - Declare the lambda expression “mutable”. This allows the body to change its copy of the variable, which will not alter the original
 - Capture the variable by reference. This allows the body to modify the original variable

- Write a simple program which shows the difference between default capture, mutable capture and capture by reference

- What considerations should a programmer bear in mind when capturing a variable by reference?
 - As with any use of references, there is the danger that the reference could “dangle”
 - This means that the reference variable can still be used after the original variable it refers to has ceased to exist
 - This can occur when lambda expressions are stored and used in a different scope from the one they were created in

- What is meant by implicit capture?
 - All the variables in scope will be captured
- What is the syntax for performing an implicit capture?
 - [=] or [&], depending on whether by value or by reference
- If we are performing an implicit capture by reference, how can we prevent the lambda function from modifying an important variable?
 - Capture that variable separately, e.g. [&, =a]

- Create a class which defines a lambda function in one of its member functions. The lambda function should modify a data member of the class
- Write a simple program to test your code

- Consider the code on the next slide
- After executing this code, which of the following gives the resulting values of x, y and z be?

x = 42, y = 99, z = 0

x = 42, y = 99, z = 141

x = 42, y = 99, z = 145

x = 43, y = 100, z = 143

x = 44, y = 101, z = 145

x = 44, y = 101, z = 141

- This code creates a lambda expression, stores it in the variable `l` and evaluates it twice

```
int x{42}, y{99}, int z{0};
```

```
auto l = [=,&z]() mutable { ++x; ++y; z = x + y; };
```

```
l();
```

```
l();
```

- Write a program to check your answer. Explain the results
 - The lambda expression is mutable. This means that variables inside the lambda expression can be modified.
 - x and y are captured by value. This means that the x and y variables inside the lambda expression are copies of the x and y variables in the outer scope.
 - z is captured by reference. This means that the z inside the lambda expression is a reference to the outer z variable.
 - Modifications in the lambda expression will affect the outer z variable, but not x and y.