Lambda Expressions and Capture Solutions

Lambda Expressions and Variables

- What access to other variables does a lambda expression have by default?
  - A lambda expression has access to non-local variables
  - It also has access to static variables in the same scope
  - By default, lambda expressions have very limited access to local variables
- Write a simple program to demonstrate this access

Lambda capture

- What is meant by "variable capture" for a lambda expression?
  - Variable capture means that a local variable is made available for use in the body of the lambda expression
- What syntax is used for variable capture?
  - The captured variables are listed inside the [] of the lambda expression
- How is variable capture implemented?
  - The captured variable is stored in a private data member in the functor
  - The functor has a constructor which takes the captured variable by value
  - This copy of the captured variable is available for use in the functor's () operator

Variable capture example

- Write a program which creates a vector and calls find_if() to search for an element with a given property
- For the predicate function, use a lambda expression with hard-coded data
- Rewrite your program to use a local variable which is captured by the lambda expression
- Write an equivalent program which uses a functor instead of a lambda expression

Mutable lambda

- The following code attempts to find the index of the first match

```
int n{5}, idx{-1};
auto res2 = find_if(words.begin(), words.end(),
        [n, idx] (const string& str) { ++idx; return str.size() > n; }
 );
```

Mutable lambda

- Why does it not produce the desired results?
  - By default, captured variables are const inside the lambda body (they are implemented as const members of the functor)
  - Modifying a const gives a compile error
  - If the lambda is declared "mutable", the captured variable is no longer const (it is implemented as a non-const member of the functor)
  - The code will now compile
  - However, since the variable is captured by value, modifying the captured variable in the body only affects the lambda's copy of it, and not the original variable
  - When the lambda returns, the value of idx is still 0