

COMP26020

Programming Languages and Paradigms

Lecture 38: Functional Capabilities in C++

Pavlos Petoumenos
pavlos.petoumenos@manchester.ac.uk

Last Week

Generic programming

One generic implementation → an infinite number of concrete versions

No overhead, higher abstraction, less effort

C++ Standard Library

Containers, Iterators, Algorithms

This Week

Functional C++

Modern C++

Part 1 Epilogue

What is functional programming?

Functional Programming

Part II is all about FP

From next week's video:

★ Treat functions like any other data

- define without naming
- functions as inputs/outputs of other functions!

★ Reason about them like mathematical functions

- If $fx = gx$ for all x , then $f = g$
- Let $x = \underline{f\ 10}$ in $\underline{x + x + x} = \underline{f\ 10 + f\ 10 + f\ 10}$

C++ is not a functional language!

Treat functions like data 

Reason about them 

But it does have some functional capabilities...

Standard Library Algorithms

`transform()` : apply function on range and store in another range

`copy_if()` : copy elements in range to another range if they make the predicate true

`reduce()` : Reduce the range into a single value by applying a function on every element

Standard Library Algorithms

`transform()` : apply function on range and store in another range

`copy_if()` : copy elements from one range to another range if the predicate returns true

`reduce()` : reduce a range of elements into a single value by applying a binary function on every element

Functions that take functions as inputs

Standard Library Algorithms

FP equivalents:

`transform()` \rightarrow `map`

`copy_if()` \rightarrow `filter`

`reduce()` \rightarrow `fold`

Anonymous Functions

Old C++ did not have any

Closest thing: function objects (Functors)

- Awkward and verbose

- Hindered the use of the library algorithms

- Assume they never existed

Lambda expressions



Anonymous function objects

[capture] (parameters) -> return_type { function_body }

```
int max(int x, int y) { return x > y ? x : y; }
```

```
[] (int x, int y) -> int { return x > y ? x : y; }
```

```
[] (int x, int y) { return x > y ? x : y; }
```

```
[] (auto x, auto y) { return x > y ? x : y; }
```



**Generic
Lambda**



Lambda expressions

Can be passed as arguments

Can be returned from functions

Can be stored

First class citizens

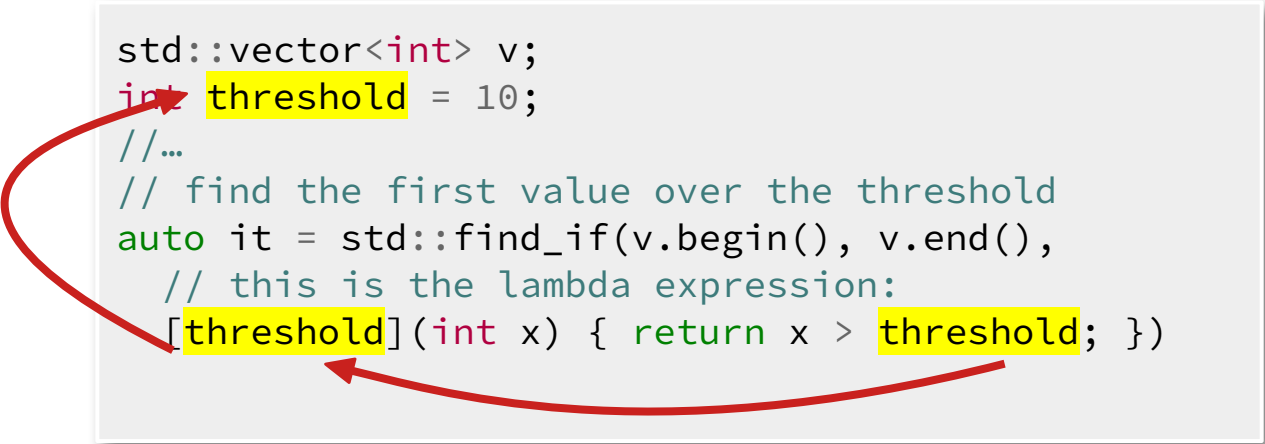
**So, what's going on
with those [] ?**

Lambda capture (Closure)

[capture] (parameters) -> return_type { function_body }

Captures part of the environment

Binds names outside the lambda to names inside



```
std::vector<int> v;  
int threshold = 10;  
//...  
// find the first value over the threshold  
auto it = std::find_if(v.begin(), v.end(),  
    // this is the lambda expression:  
    [threshold](int x) { return x > threshold; });
```

The image shows a C++ code snippet with two red arrows illustrating lambda capture. The first arrow starts at the `threshold` variable in the lambda expression `[threshold]` and points to the `threshold` variable defined in the global scope. The second arrow starts at the `threshold` variable in the lambda body `return x > threshold;` and also points to the same `threshold` variable in the global scope. This demonstrates how the lambda captures the `threshold` variable from its enclosing environment.

Lambda capture

Pass information to a function without using more args

```
std::find_if(it1, it2, predicate); // predicate → unary operator
std::for_each(it1, it2, Fn); // Fn → unary operator
...

std::sort(it1, it2, sortFn); // sortFn → binary operator
std::reduce(it1, it2, reduceFn); // reduceFn → binary operator
...
```

```
auto it = std::find_if(v.begin(), v.end(),
    // this is the lambda expression:
    [threshold](int x)
    { return x > threshold; })
```

Lambda capture

[] → Capture nothing

[var] → Capture var by value

[&var] → Capture var by reference

[=] → Capture everything by value

[&] → Capture everything by reference

[&, var] → Capture by reference, apart from var

Recap

Algorithms that operate on functions

transform \rightarrow map

reduce \rightarrow fold

copy_if \rightarrow filter

Lambda functions

Anonymous

[] \rightarrow Capture list: pass extra context without arguments

Up Next

Range Views