ECE326

PROGRAMMING LANGUAGES

Lecture 36 : Lambdas, Generators and Coroutines

Kuei (Jack) Sun

ECE

University of Toronto

Fall 2019

Closure

- Function that can *capture* enclosing environment
- Capture
  - The use of outer variables in a closure
- Heavy use of type inference to make code succinct
- Rust

```
let closure_annotated = |i: i32| -> i32 { i + 1 };
let closure_inferred = |i | i + 1 ;
```

- C++11

```
auto func = [] { cout << "Hello world"; };
auto printi = [](int val) { cout << val; };
```

No parameter closures can omit parantheses ()

Capture

- In C++, captures can be specified in detail

`[]`	Capture nothing
`[&]`	Capture any referenced variable by reference
`[=]`	Capture any referenced variable by copy
`[=, &foo]`	Capture any ref'ed variable by copy, but capture *foo* by reference
`[bar]`	Capture *bar* by making a copy. don't copy anything else
`[this]`	Capture the this pointer of the enclosing class

- Closures are implemented as *functors*
  - Recall that functor is a class that overloads the call operator
  - All captures are stored as member variables

Anonymous Function

- In Rust and C++, closures are also anonymous
  - Name of the closure is unspecified
- In Python, anonymous functions are called lambdas
  - Closure can be named or nameless in Python

```
>> func = lambda x: x + 1
>> func(2)
3
>> full_name = lambda first, last: "%s %s"%(first, last)
>> full_name("Hello", "World")
'Hello World'
>> list(map(lambda x: x*x, range(1, 5)))
[1, 4, 9, 16]
```

Anonymous Function

- Other names
  - Lambda expression
  - Function literals
- Main uses
  - As closure
  - Pass to higher order functions
  - Allows function code to be physically closer to usage

```
def square(x):
        return x*x
… many lines later …
map(square, range(1, 5))
```

```
map(lambda x: x*x, range(1, 5))
```