C++ Lambdas

Seonghyun Park

dd December, 2018

Outline

1. Function
   - Function Pointer
   - Type alias
   - Lamdas

2. Algorithm
   - Lambdas and algorithm

Today's Goal

- Learn how to treat functions as objects in C++ (function pointer, function object, lambdas)
  - Function pointers
  - Funcion objects
  - Lambdas
- Learn how to use C++ lambdas to impelemt algorithms efficiently

Function Pointer

- Code is also stored in the address space.
- In C, functions could be accessed using *function pointers*.
- Doesn't look so good.

```
int add(int x, int y) { return x + y; }

int (*fp)(int, int) = add;

fp(3, 4); // 7
```

Type Alias

- In C++, instead of using `typedef`, programmers can define type aliases with `using`
- Looks better, but still not pleasant

```
template <typenae T>
using BinaryOp<T> = T(*)(T, T);

BinaryOp<int> fp = add;
fp(3, 4); // 7
```

Function Objects

- More C++-ish way of defining function objects: *Function Objects*.
- The underlying idea is to overload call operator i.e. `operator()`.

```
template <typename T>
class Equal_to {
  T t;
  Equal_to(T t) : t(t) {}
  bool operator()(const T& that)
  {
    return t == that;
  };
};

Equal_to<char> equal_to_c{'c'};
```

Lambdas

- In C++11, anonymous functions can be declared without names

```
int x = 42;
std::string hello = "Hello, ";

[x](int z){ return z + x; }(17); // 59
[&hello](std::string name) { return hello + name; }("Ja
```

Algorithm Standard Library

- Lambdas are used with `algorithm` functions with iterators
- Each C++ STL usually come with iterators `.begin()` and `.end()`
- See `std for_each`, `std sort`, and so on

Algorithm Standard Library

```
std::vector<int> v = {1, 2, 3, 4, 5};

std::for_each(v.begin(), v.end(), [](int x){
  if (x < 3) { std::cout << x << ' '; }
});

// 1 2
```