λ-function

The Anonymous Function

Mohammed Yaseen Mowzer

February 28, 2014

Properties

- Closures (variable capture)
- First-Class
- Used with higher-order functions (e.g. for_each)
- Popular in functional programming languages

Implementations of λ

- Python

```
lambda x: x * x
```

- Lisp/Scheme

```
(lambda (x y) (* x y))
```

- Haskell

```
\x y -> x * y
```

- JavaScript

```
function (x, y) { return x * y;}
```

- C++11

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

Common Patterns
Pseudocode

- First-class (can be assigned to a variable)

  ```
  fn = lambda x, y: x * y
  fn(3, 4) #-> 12
  ```

- Closures

  ```
  def make_counter() {
  c = 0
  return lambda: c += 1
  }
  counter = make_counter()
  counter() #-> 1
  counter() #-> 2
  ```

- Higher-order functions (can be passed into a higher order function)

  ```
  map((lambda x: x + 1), [1, 2, 4]) #-> [2, 3, 5]
  ```

Syntax (C++11)

[*capture*] (*parameters*) -> *return_type* {*body*}

Minimum working example;

```
#include <iostream>
using namespace std;
int main()
{
cout « [] (int x, int y) {return x * y;}(3, 5);
}
```

Output: 15
Return value is implied

First-class

- Easiest way

```
auto func = [] (int x, int y) {cout << x * y;};
func(6, 7); //--> 42
```

- Other way

```
#include <functional>
using namespace std;

function<void (int, int)>
func = [] (int x, int y) {cout << x * y;};
func(6, 7); //--> 42
```

C++11 Closure

Copy by Reference

```
#include <iostream>
#include <functional>
using namespace std;

int main()
{
  int x = 3;
  auto func = [&] (int y) {return x++ * y;};
  cout << func(5) << endl; // 15
  cout << func(5) << endl; // 20
  cout << x << endl; // 5
}
```

C++11 Closure

Copy by Value

```
#include <iostream>
#include <functional>
using namespace std;

int main()
{
  int x = 3;
  auto func = [=] (int y) {
    return x++ * y; // compile error: x is read-only
  };
  cout << func(5) << endl;
  cout << func(5) << endl;
  cout << x << endl;
}
```

C++11 Closure

Copy by Value

```
#include <iostream>
#include <functional>
using namespace std;

int main()
{
  int x = 3;
  auto func = [=] (int y) {
    return x * y;
  };
  cout << func(5) << endl; // 15
  ++x;
  cout << func(5) << endl; // 15
  ++x;
  cout << x << endl; // 5
}
```

C++11 Closure

Copy by value and reference

```
#include <iostream>
#include <functional>
using namespace std;

int main()
{
  int x = 3;
  int y = 5
  auto func = [&x, y] () {
    return x++ * y;
  };
  cout << func() << endl; // 15
  cout << func() << endl; // 20
  cout << x << endl; // 5
}
```

List of capture commands

[]	Capture nothing
[&]	Capture all variables by reference
[=]	Capture all variables by value
[&foo]	Capture foo by reference, don't capture anything else
[foo]	Capture foo by value, don't capture anything else
[=, &foo]	Capture all variables by value except foo, capture it by reference
[&, foo]	Capture all variables by reference except foo, capture it by value
[this]	Capture the this pointer of the enclosing class.

Higher Order Functions

for_each the better for loop and other new functions

```
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- for_each

```
for_each(v.begin(), v.end(), [] (int a) {
  cout << a << ' ';
});
```

- all_of: Returns true if every elements satisfies the condition (similar any_of)

```
all_of(v.begin(), v.end(), [] (int a) { return a >
     0; }); // true
```

- find_if: returns an iterator to the first element that satisfies the predicate

```
find_if(v.begin(), v.end(), [] (int a) { return a
     % 4 == 3; }); // 3
```

Higher Order Functions

Filtering

```
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

remove_if: removes all elements satisfying the predicate and returns an iterator to the new last element

```
auto end = remove_if(v.begin(), v.end(), [] (int a) {
    return a % 2 == 1;
});

for_each(v.begin(), end, [] (int a) {
    cout << a << ' ';
});

// {2, 4, 6, 8, 10}
```

Higher Order Functions

Mod filter

```
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

auto mod_filter = [] (vector<int> &v, int r, int n) {
  auto end = remove_if(v.begin(), v.end(),
    [=] (int a) {
      return !(a % n == r);
  });
  while(end < v.end()) {
    v.pop_back();
  }
};

mod_filter(v, 1, 3);

for_each(v.begin(), v.end(), [] (int a) {
  cout << a << ' ';
});

// {1, 4, 7, 10}
```

Higher Order Functions

Templates

```
struct point {
  int x;
  int y;
};
vector<point> ps = {{1, 3}, {4, 5}, {1, 2}, {5, 2},
    {4, 8}};

sort(ps.begin(), ps.end(), [] (point a, point b) {
  return abs(a.x * a.x - b.x * b.x) <
         abs(a.y * a.y - b.y * b.y);});

for_each(ps.begin(), ps.end(), [] (point a) {
  cout << a.x << ' ' << a.y << endl;});
```