‘Functional’ C++

- Lambdas and Closures
- Map, Filter, and Reduce
- Partially Applied Functions

Lambdas: Anonymous Functions

- Lambdas are functions that don't have a name.
- They are called 'first-class functions' because they can be assigned to variables, passed as arguments, etc.
- A lambda looks like `[] (arguments) { function body }`.
- You can assign them to variables of type `function` (although I usually just use `auto`).

Examples: `hello.cpp`, `sort.cpp`

Hello World: `hello.cpp`

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

int main() {
  function<void()> hello = [] () {
    cout << "Hello World" << endl;
  };
  hello();

  [] () {
    string name;
    cout << "Who are you? ";
    cin >> name;
    cout << "Goodbye " << name << endl;
  }();

  return 0;
```

Closures: They're Just Lambdas

- Closures let you capture, or 'close over' variables that are in scope when the lambda is declared.
- You can capture by value by using `[=]` or by reference, using `[&]`.

Examples: `hello-closure.cpp`, `class-closure.cpp`

Closing over `name`: `hello-closure.cpp`

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

int main() {
  string name = "bob";

  auto get_name = [&] () {
    cout << "Who are you? ";
    cin >> name;
  };

  get_name();
  cout << "Goodbye " << name << endl;

  return 0;
}
```

Map, err, transform

- `transform(start, end, destination, func(element))` loops over a vector, applying a function to each element.
- Each element gets replaced by the return value of the function.

Transform docs

transform() : transform.cpp

```
#include<iostream>
#include<vector>
#include<algorithm>
#include"print_vector.h"
using namespace std;

int main() {
  vector<int> n = {1,2,3,4,5,6};
  print_vector(n);

  transform(
      n.begin(), // Start
      n.end(),   // End
      n.begin(), // Destination
      [] (int x) {return x*x;}
    );
  print_vector(n);
```

Filter, err, remove_if

- `remove_if(start,end,predicate)` loops over a vector and 'removes' items if `predicate(item)` returns true.
- It really returns an iterator to the new end of the vector.
- To actually shrink the vector, call `vector.erase(remove_if_return, vector.end()`.

Remove_if docs

remove_if : remove-if.cpp

```
#include<iostream>
#include<vector>
#include<algorithm>
#include"print_vector.h"
using namespace std;

int main() {
  vector<int> n = {1,2,3,4,5,6,7,8,9};
  print_vector(n);

  auto end = remove_if(
      n.begin(),
      n.end(),
      [] (int x) { return x % 2 == 0; }
    );

  n.erase(end,n.end());
```

Reduce, err, accumulate

- `accumulate(start, end, intitial, func(accumulator, element))` loops over a vector and, well, accumulates each element into a single value.
- It starts with some user-specified intital value.
- `accumulate` is really a general case of `transform` and `remove_if`.

Example: `accumulate.cpp`
Accumulate docs

Partial Function Application

- What if you've got a function that you want to use for, say, `transform`, but it takes a second argument?
- You can 'partially apply' that function with `bind()`!
- `bind(func, arg1, arg2, _1)` returns a function that takes one argument.

Example: `bind.cpp`
Bind docs