Functional Programming in C++

David Letscher

Saint Louis University

Programming Languages
What is needed to incorporate functional programming?

Compiler support
Tail optimization to avoid issues with stack recursion depth limits.

Language features
- First-class functions
- Higher-order functions
- Currying and binding
- Immutable data
- Pure functions
- Lazy evaluation
- Functors, monads, ...
### Function pointers in C

```c
#include <stdio.h>

double cm_to_inches(double cm) {
    return cm / 2.54;
}

double apply(double (*f)(double), double x) {
    return f(x);
}

int main(void) {
    double (*func1)(double) = cm_to_inches;
    double meter_in_inches = cm_to_inches(100);

    double meter_in_inches2 = apply(cm_to_inches, 100);
}
```
First-class functions in C++

Function objects

class square {
public:
    double operator() (double x) {
        return x*x;
    }
};

Can be passed as parameters to other functions, methods, ...
First-class functions in C++-11

Language enhancements

- Lambda functions
- auto keyword
- std::function
- std::bind
Lambda functions

```cpp
[](double x, double y) { return x + y; }
```

Return type is deduced by the compiler, if possible.
Lambda functions

[] (double x, double y) { return x + y; }

Return type is deduced by the compiler, if possible.

Return type specified

[] (double x, double y) -> double { return x + y; }

What are function types?

```cpp
class Function
{
    private:
        int m_age;
    public:
        Function(int age)
            : m_age(age)
        {
        }
        int get_age()
        {
            return m_age;
        }
};

auto add = [] (double x, double y)
    -> double {
        return x + y;
    }
add(2,3);
```
What are function types?

```cpp
auto add = [] (double x, double y) -> double {
    return x + y;
};
add(2,3);
```

What type is `add`?

```cpp
std::function<int(int, int)>
```

Example: performing arithmetic

```cpp
map<
class char,
function<
class double(class double, class double)>>
functionTable;

functionTable['+'] =
    [](class double x, class double y) {
        return x + y;
    };

functionTable['−'] =
    [](class double x, class double y) {
        return x − y;
    };

functionTable['∗'] =
    [](class double x, class double y) {
        return x ∗ y;
    };

functionTable['/'] =
    [](class double x, class double y) {
        return x / y;
    };

functionTable['^'] = std::pow;
```
Example: performing arithmetic

```cpp
cout << functionTable['*'](3.0, 4.5) << endl;
cout << functionTable['^'](3.0, 4.5) << endl;
```

Imagine parsing a string, tokenizing it and using the function table to perform the calculations. Avoids lots of cases.
Higher-order functions

Three common patterns:

**Map**  Apply a function to all elements of a container.  
  `map` in Haskell

**Filter**  Remove elements of a container not meeting a condition.  
  `filter` in Haskell

**Reduce**  Accumulate values from a container.  
  `foldl`, `foldr` in Haskell
Map in C++

Uses `std::transform`.


Squaring all entries in a list

```cpp
vector<int> numbers = {0, 1, 2, 3, 4, 5};

auto square = [] (int n) { return n * n; }

transform(numbers.begin(), numbers.end(), numbers.begin(), square);
```

Result:   {0, 1, 4, 9, 16, 25}
Filter in C++

Uses `std::remove_if`.

http://en.cppreference.com/w/cpp/algorithm/remove

Remove the odd numbers

```cpp
vector<int> numbers = {0, 1, 2, 3, 4, 5};

remove_if(numbers.begin(), numbers.end(), [](int n) { return n % 2 == 1; });
```

Result: `{0, 2, 4}`.
Reduce in C++

Uses `std::accumulate`.


```cpp
vector<int> numbers = {0, 1, 2, 3, 4, 5};

int sum = accumulate(numbers.begin(), numbers.end(), 0, [](int x, int y) {
    return x+y;
});

Result: 15.
```
Function binding in C++


```cpp
int foo(string s, int n, list<int> l);

auto f1 = std::bind(foo, "Hello", _1, _2);
auto f2 = std::bind(foo, _2, _3, _1);
```
Pure functions, immutable data
Lazy evaluation
Compile time programming