Overloading Function Call

In the standard library, functions are often passed as arguments to algorithms. It is possible to pass a pointer to a function, but it is often better to pass a *function object*, i.e., an object of a class that overloads the function call operator, `operator()`. It is better because calls can be inlined. Example:

```
struct Sin {
    double operator()(double x) const {
        return std::sin(x);
    }
};

void f(Sin sin_obj) {
    double y = sin_obj(1.0); // calls Sin::operator()(double)
}
```

See next slide for another example.

Functions With State

With function objects, something which is used exactly as a function can have state that is saved between function calls.

```
class Accumulator {
public:
    Accumulator() : sum() {}
    double get_sum() const { return sum; }
    void operator()(double x) { sum += x; }
private:
    double sum;
};

double f(const vector<double>& v) {
    Accumulator accum;
    for (double x : v) {
        accum(x); // looks like a normal function call
    }
    return accum.get_sum();
}
```