601.220 Intermediate Programming

C++ classes

C++: non-object-oriented programming

We already saw `structs`, which bring together several variables that collectively describe one "thing":

```
struct rectangle {
    double width;
    double height;
};
```

We might additionally define some functions that do things with rectangles, like print them or calculate their area:

C++: non-object-oriented programming

```cpp
// rectangle1.cpp:
#include <iostream>

using std::cout;
using std::endl;

struct rectangle {
    double width;
    double height;
};

void print_rectangle(struct rectangle r) {
    cout << "width=" << r.width
         << ", height=" << r.height << endl;
}

double area(struct rectangle r) {
    return r.width * r.height;
}

int main() {
    rectangle r = {30.0, 40.0};
    print_rectangle(r);
    cout << "area=" << area(r) << endl;
    return 0;
}
```

C++: non-object-oriented programming

```
$ g++ -c rectangle1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o rectangle1 rectangle1.o
$ ./rectangle1
width=30, height=40
area=1200
```

C++: Object-oriented programming

As good Java or Python programmers, though, we prefer to have the related functionality (`print_rectangle`, `area`) be *part of the object* (the `struct` in this case)

No simple way to do this in C. But in C++...

C++: Object-oriented programming

```
#include <iostream>

using std::cout;
using std::endl;

class Rectangle {

public:
    double width;
    double height;

    void print() const {
        cout << "width=" << width << ", height=" << height << endl;
    }

    double area() const {
        return width * height;
    }
};
```

C++: Object-oriented programming

```
//continuation of file on previous slide

int main() {
    Rectangle r = {30.0, 40.0};
    r.print();
    cout << "area=" << r.area() << endl;
    return 0;
}
```

C++: Object-oriented programming

- A class definition is like a *blueprint* defining a type
- Objects of that type are created from that *blueprint*

- Once we define a class, we have one blueprint from which we can create 0 or more objects
- Each of the objects is an *instance* of the class and has its own copies of all instance variables

C++: Object-oriented programming

```
void print() const { ...
```

- The use of `const` as modifier in method header indicates that the function will not modify any member fields
  - the rectangle object on which we call `print` will not be modified by the call

C++ classes

Basic principles for writing C++ classes

- Class definition goes in a `.h` file
- Functions can be declared *and defined* inside `class { ... };`
- Only define member function inside the class definition if it's *very* short
  - this is called "in-lining" the function definition
- Otherwise, put a prototype in the class definition and define the member function in a `.cpp` file
  - you'll need to qualify the function with the class scope such as `Classname::function() { }` in the .cpp file

C++ classes

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
class Rectangle {
    ...
    double area() const {
        // short definition inside class
        return width * height;
    }
};
#endif
```

C++ classes

Or, in `Rectangle.h`:

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
class Rectangle {
    ...
    double area() const;
    ...
};
#endif
```

Later, in `Rectangle.cpp`:

```
#include "Rectangle.h"

double Rectangle::area() const {  // def'n outside class
    return width * height;
}
```

C++ classes

- Fields and member functions can be `public` or `private`
  - (or `protected`, discussed later)
- We use `public:` and `private:` to divide class definition into sections according to whether members are `public` or `private`
- Everything is `private` by default

C++ classes

- A public field or member function can be accessed freely by any code with access to the class definition (code that includes the .h file)
- A private field or member function can be accessed from other member functions in the class, but *not* by a user of the class

C++ classes

```
class Rectangle {
public:
    double area() const {
        // definition inside class
        return width * height;  // OK
    }
    ...
private:
    double width, height;
};
```

C++ classes

```
class Rectangle {
    ...
private:
    double width, height;
};

int main() {
    Rectangle r;
    std::cout << r.width << std::endl; // not OK!
    return 0;
}
```

C++ classes

- Don't try to initialize class fields immediately when they are declared

```
class Rectangle {
    ...

    double width = 10;   // NO!
};
```

- This kind of initialization is only allowed for static fields

C++ classes

```
// rectangle3.h:

class Rectangle {
public:
    double area() const {
        return width * height;
    }

private:
    double width, height;
};
```