Friend functions

Ch 11.2

Highlights

- friends

```cpp
class Point{
public:
    friend bool equals(Point first, Point second);
```

Review: private

Notice this line:

```
if(putin < barak)
```

Which runs…

```
if(feet > otherPerson.feet)
```

```
class height {
private:
    int inch;
    int feet;
```

putin's feet   barak's feet

This means putin is accessing barak's privates!

Private only means things NOT associated with the class (such as main) cannot use or access these variables/functions

Operator overloading

In operator overloading, the left variable "calls" the operator function on the right one

```
Point c = a+b;
```

function!

... is the same as ....

```
Point c = a.operator+(b);
```

Since the "operator+" function is inside the "Point" class, it can access all the private variables/functions
(see: pointReview.cpp)

friend functions

You can give a non-class function access to private variables by making it a <u>friend</u>

A friend function is not inside the class, but does have access to its private variables (friends don't mind sharing)

This allows you to give exceptions to the private rule for specific functions

friend functions

Instead of declaring a friend function at the top, do it inside the class:

```cpp
class Point{
public:
    friend bool equals(Point first, Point second);
```

The function description/implementation is identical to as if it was a non-friend:

```cpp
bool equals(Point first, Point second)
{
```

(See: pointFriends.cpp)

friend functions

How would you overload the << operator?
Would you use a friend?

What do you return?

Hint: cout is type "ostream"
Hint2: use call-by-reference

(See: pointFriendsOverload.cpp)

friend functions

How would you overload the << operator?
Would you use a friend?
 Yes, so you can put cout first
What do you return?
 ostream& so you can cout multiple things

How would cin work?
Any other case of when you can think you would need a friend with the point class?

friend functions

When would you want to use friend functions?

1. Typically when we want to involve two separate classes
(see: multiplePrivates.cpp)

2. When we care about the order of things... (as normal overloading needs your class to come first)