Okay, any ideas on how to make women more interested in us?

Make more exceptions?
Redefine our methods?
Stop treating them like objects?

Classes
Ch 10.1 - 10.3
Highlights

- public/private
- constructor
- operator overloading

```cpp
class myName
{
public:
    myName();
    myName(int x);
    char takeThis();

private:
    char itsASecretToEveryone;
};
```

- friend functions

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

Arrays group together similar data types (any amount you want)

Classes (and structs) group together dissimilar types that are logically similar
A class is a new type that you create (much like int, double, ...)

An instance of date class

```java
int main ()
{
    int x;
    date today;
    date yesterday;
}
```

Another instance

Blueprint for all objects

```java
class date
{
    public:
    int day;
    int month;
    int year;
    void print();
};
```
public vs private

class date
{
    private:
    int day;
    int month;
    int year;
    public:
    void print();
    void setDate(int day, int month, int year);
};
public vs private
Creating interfaces with public allows users to not worry about the private implementation

So... more work for you (programmer) less work for everyone else
public vs private

The **public** keyword allows anyone anywhere to access the variable/method

The **private** keyword only allows access by/in the class where the variable/method is defined
(i.e. only variables of this type can access this within itself)
public vs private

All variables should be **private**

While this means you need methods to set variables, users do not need to know how the class works

This allows an easier interface for the user (also easier to modify/update code)

(See: datePrivate.cpp)
The idea is: if the stuff underneath changes, it will not effect how you use it

For example, you change from a normal engine to a hybrid engine... but you still fill it up the same way
public vs private

An important point: private just means only “date” things can modify the private variables of a “date” object

However, two different “date” objects can access each other's privates

(see: privateDates.cpp)
The date class has two functions: `setDate()` and `print()`.

As we need to run `setDate()` on a variable before it is useful anyways.

In fact, such a thing exists and is called a **constructor** (run every time you create a variable).
Constructors

The class name and the constructor must be identical (constructors also have no return type)

```cpp
class date
{
private:
    int day;
    int month;
    int year;
public:
    date(int day, int month, int year);
    // ^^ constructor has same name as class
    void print();
};
```

(See: dateConstructor.cpp)
Constructors

If you don't put a constructor, C++ will make a default constructor for you (no arguments)

date(); ← default constructor
date(int day, int month, int year);

To use the default constructor say this:
date never; .... or ... date never = date();

... not this:
date notWhatYouWant();
// ^ function declaration
Constructors

If you declared constructors you must use one of those

Only if you declare no constructors, does C++ make one for you (the default)

Note: our dateConstructor.cpp has no way to change the value of the date after it is created (thus gives control over how to use class)
TL;DR Constructors

Constructors are functions, but with a few special properties:

(1) They have no return type
(2) They must have the same name as the class they are constructing
(3) If you want to make an instance of a class you MUST run a constructor (and if you ever run a constructor, you are making an object)
Just as writing very long main() functions can start to get confusing...

... writing very long .cpp files can also get confusing.

Classes are a good way to split up code among different files.
#include

You can `#include` your class back in at the top or link to it at compile time.

You have to be careful as `#include` basically copies/pastes text for you.

Will not compile if class declared twice (used in two different classes you `#include`).
```cpp
#include date.cpp
#include date.hpp
#include runDate.cpp

class date {
    int month;
    int day;
    int year;

public:
    date(int m, int d, int y);
    void showDate();

private:
    int day;
    int month;
    int year;
};

int main ()
{
    date today = date(3, 20, 2017);
    today.showDate();
}

void date::showDate()
{
    cout << month << "/" << day << "/" << year;
}

void date::date(int m, int d, int y)
{
    day = d;
    month = m;
    year = y;
}

Then compile with: g++ runDate.cpp date.cpp
```
To get around this, you can use compiler commands in your file:

```
#include

#define DATE

class date
{
  public:
    int day;
    int month;
    int year;
    void print() const;
};

#endif
```

“if not defined”

“define”

This ensures you only have declarations once.

(See: dateClass.hpp, dateClass.cpp, runDate.cpp)
Operator Overload
Ch 11.1
Basic point class

Suppose we wanted to make a simple class to represent an (x,y) coordinate point

```cpp
class Point{
    private:
        int x;
        int y;
    public:
        Point();
        Point(int startX, int startY);
        void showPoint();
};
```

(See: pointClass.cpp)
Basic point class

Now let's extend the class and make a function that can add two \((x,y)\) coordinates together (like vectors)

With two ints?

With another point?

(See: pointClassAdd.cpp)
Operator overloading

We can overload the `+` operator to allow easy addition of points.

This is nothing more than a “fancy” function.

(See: pointOverload.cpp)
Operator overloading

When overload operators in this fashion, the computer will convert a statement such as:

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

... into ...

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

... where the left side of the operator is the “calling” class and the right side is a argument function!
Operator overloading

You cannot change the number of parts to an operator ("+" only gets 2, "!" only gets 1)

Cannot create “new” operators (can only overload existing ones)

Cannot change order of precedence ("*" is always before "+")

Operator "=" is special... save for later
Terrible units

Let's make a class that stores people's heights using the terrible imperial units!

(see: heights.cpp)
Terrible units

Write the following operators to compare two different heights:

<
==
>

(see: heightsCompare.cpp)
Operator overloading

Long list of operators you can overload:

( ) // this is normal overloading
+, -, *, /, %
!, <, >, ==, !=, <=, >=, ||, &&
// should be able to do anything above here
<<, >>, [ ]
=, +=, -=, *=, /=, %=, ++ (before/after), --(b/a)
^, &, |, ~, (comma), ->*, ->
^=, &=, |=, <<=, >>=

<<=, >>=
Operator overloading

Functions define a general procedure (or code block) to run on some inputs

Constructors are nothing but “special” functions that initialize class variables

Operator overloading is a special function that is disguised as a symbol
Friend functions

Ch 11.2
Review: private

Notice this line:

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

Which runs...

```plaintext
if(feet > otherPerson.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.
friend functions

You can give a non-class function access to private variables by making it a friend.

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:

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

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

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)