Announcements

- HW1 due Sat.
- Look for HW2 on website soon.
- Lab tomorrow (posted on lab page)
  - don't forget to email pre-lab before class!!
A note on variable scopes:

```cpp
int main () {
    int a;
    if (a > 0) {
        int b = 12; // b is destroyed
    } else {
        int b = 16; // b is destroyed
    }
    cout << "a is " << a << endl;
    cout << "b is " << b << endl; // compiler error
    // a is destroyed
}
```
for loops:

```cpp
for (int i = 0; i < val; i++) {
    // i is destroyed
    cout << i << error
}
```
Arrays as inputs to functions

Example: Write a function to specify if sum of values in an array is even.

```cpp
bool evenSum(int anArray[][], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum = sum + anArray[i][i];

    return (sum % 2 == 0);
}
```
Note: an array actually makes a (the array) a pointer.
(More on these later...)

Doesn't copy whole array but can pretend that it does - just use it like an array.

To call: int main() {
  // create & put values in myArray
  if (evenSum(myArray, length)) {
    cout << "The sum is even" << endl;
  }
}
Classes

What is a class?

- data field: a collection of data
- store arbitrary collections of data
- along with allowed operations

Ex: records for people
Creating an instance of a class

Example:

```cpp
string s;
string greeting("Hello");
```

Why?

Never:

```cpp
string s();
```

Why?

Declares a function named `s` with no inputs which does nothing.

Never:

```cpp
string("Hello") greeting;
```

Why?

Compiler hates it.
Example:

```cpp
class Point {
private:
    double x;
    double y; // explicit declaration of data members

public:
    Point( ) : x(0), y(0) {} // constructor
    double getX( ) const {
        return x;
    } // accessor
    void setX(double val) {
        x = val;
    } // mutator
    double getY( ) const {
        return y;
    } // accessor
    void setY(double val) {
        y = val;
    } // mutator
};
```
Classes:

1) Data—public or private—is explicitly declared, not just used in constructor. This is done inside the class, but not inside a function.

Why?

Scope would only be that function.
2. Constructor Function

- name: same as class

- no return type (only one!)

- can initialize variables via a list

\[
\text{Point( ) : } \{ x(0), y(0) \} \quad \Leftrightarrow \quad \begin{cases} x = 0; \\ y = 0; \end{cases}
\]

\[
\text{Point(double initialX=0.0, double initialY=0.0) : } \{ x(\text{initialX}), y(\text{initialY}) \} \}
\]
Other differences

3) No `self`! Can just use `-x` or `-y` & it immediately scopes to the class attributes.

(There is a "this", but its usage is a bit more complex.)

4) Access control - public versus private.

```plaintext
Point myPoint;
myPoint.x = 2; // compiler error
```
Accessor versus mutator

double getX() const { return x; }
void setX(double val) { x = val; }

can enforce this in C++.
Robust point class: add functionality

```cpp
double dist = pt1.distance(pt2);

Point p3 = pt1 + pt2;
Point p3 = pt1.operator+(pt2);
Point p3 = pt1 * pt2;
Point p3 = pt1 * 3.0;
```

```cpp
double distance(Point other) const {
  double dx = x - other.x;
  double dy = y - other.y;
  return sqrt(dx * dx + dy * dy);  // sqrt imported from cmath library
}

void normalize() {
  double mag = distance(Point());  // measure distance to the origin
  if (mag > 0)
    scale(1/mag);
}

Point operator+(Point other) const {
  return Point(x + other.x, y + other.y);
}

Point operator*(double factor) const {
  return Point(x * factor, y * factor);
}

double operator*(Point other) const {
  return x * other.x + y * other.y;
};  // end of Point class (semicolon is required)
```
Important things

1) $x + \text{other} \_ x \leftarrow \text{allowed only inside the class}

2) using operator +

3) two versions of $\_ \_ \_ \text{can't use } \$\_ \_\_ \_ \_ \text{ Instance since return types were different}
Additional functions

```cpp
Point operator*(double factor, Point p) {
    return p * factor; // invoke existing form with Point as left operand
}

ostream& operator<<(ostream& out, Point p) {
    out << "(" << p.getX() << ", " << p.getY() << ");"; // display using form <x,y>
    return out;
}
```

Why? overloading print

cout << pt;