Announcements

- HW due today
- Tutoring is now open for business
- Next assignment up soon
- Nice talk today at Math/CS club cryptography
  4pm in Ritter Lobby
Defining a class. Remember the `Point` class?

```cpp
class Point {
private:
    double x;
    double y;

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

// explicit declaration of data members

// end of Point class (semicolon is required)

```cpp
Point mypoint;  // data in the class
```

```cpp
cout << mypoint.x << endl;  // ERROR
mypoint. getX();
```
Classes - differences:

1. Data (public or private) is explicitly declared, not just used in constructor.

2. Constructor:
   - Name is always same as class
   - No return type
   - Can initialize variables in a list

\[
\text{Point}(\cdot) : \quad \begin{cases} 
  \text{int } x, y \Rightarrow x(0), -y(0) \\
  \text{int } x=0, \text{int } y=0 \Rightarrow -x(x), -y(y) \\
  \text{Point } (\cdot) \Rightarrow \begin{cases} 
    -x=0 \Rightarrow \text{?} \\
    -y=0 \Rightarrow \text{?}
  \end{cases}
\end{cases}
\]
A more complicated constructor:

```java
Point(double initialX=0.0, double initialY=0.0) { x(initialX), y(initialY) }
```

- Allows default parameters,
  but body is still empty.
Other things to note:

3. **No self**! Can just use -x or -y, or understood to be attributes of current object.

(Could use this, i.e. this.-x, if necessary.)

4. **Access control** - public versus private
   - main can't touch private variables
   - functions are often public

Can't make local variables w/same name:

Ex: no int -x; (inside Point function)
Other things to note (cont):

5 accessor versus mutator:

\[ x = 6 \]

difference?

```cpp
double getX() const {
    return x;
} // accessor

void setX(double val) {
    x = val;
} // mutator
```

Forced by compiler:

in `main`:

\[ \text{my point}.\text{getX}() = 5 \]

error
Robust Point class cont: might add some functionality:

doctor 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)
Things to note:
1) \(-x + \text{other} \neq x \leq \text{allowed if inside the class}

2) using operator + will be \(x + y\)
   (note const)
   \(z = x + y\)
   \(x = x + y\)

3) two versions of *
   in some instances have more than
   1 interpretation:
   \((3, 6) \star 2\)
   \((5, 2) \star (1, 3) = (6, 12)\)
   \((3, 6) \star 2\)
   \(5 \cdot 1 + 2 \cdot 3 = 11\)
Additional functions  
*(Not in class)*

```cpp
// Free-standing operator definitions, outside the formal Point class definition
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 outside of class?

C++ does not allow right operator to be instance of an object.
Inheritance

What is it?

A way to create a class that can steal another class' functions
(a way to be lazy)
Example: Square class

```cpp
#include "Rectangle.cpp"

class Square : public Rectangle {
public:
    Square(double size=10, Point center=Point(0)): 
        Rectangle(size, size, center) // parent constructor
    {
    }

    void setHeight(double h) { setSize(h); }
    void setWidth(double w) { setSize(w); }

    void setSize(double size) {
        Rectangle::setWidth(size); // make sure to invoke PARENT version
        Rectangle::setHeight(size); // make sure to invoke PARENT version
    }

double getSize() const { return getWidth(); }
}; // end of Square
```

Can't say: "height = 5"
Other issues:

A new type of data:
- We have seen `public` & `private`.
  Public is inherited and private is not.

But what about data which should be private, but also should be inherited?

Ex: `public`:
- `int height`
- `int width`

`protected`:
- `int height`
- `int width`

Allows any child class to gain access
Objects & Memory Management

In Python, variables were pointers to
data.

\[
\begin{align*}
\text{Point} & \quad \text{(x, y)} \\
-x & = 3.2 \\
y & = -5.8
\end{align*}
\]

\[
\begin{align*}
b & = a^\prime \\
b & = \text{Point} (3, 4) \\
a & = b^\prime
\end{align*}
\]

This gets erased (automatic garbage collection)
Example: Square class

class Square : public Rectangle {
public:
    Square(double size=10, Point center=Point()) :
        Rectangle(size, size, center)  // parent constructor
    {
    }

    void setHeight(double h) { setSize(h); }
    void setWidth(double w) { setSize(w); }

    void setSize(double size) {
        Rectangle::setWidth(size); // make sure to invoke PARENT version
        Rectangle::setHeight(size);  // make sure to invoke PARENT version
    }

    double getSize() const { return getWidth(); }
}; // end of Square
Other issues:
A new type of data: we have seen public & private. Public is inherited and private is not.

But what about data which should be private, but also should be inherited?

Ex: public:

```
int height;
int width;
```
Speeding up the Point class:

original:  

```cpp
double distance(Point other) const {
    // Implementation
}
```

faster:  

```cpp
double distance(const Point& other) const {
    // Implementation
}
```

Another:  

```cpp
Point operator+(const Point& other) const {
    return Point(_x + other._x, _y + other._y);
}
```

Note: Return type is still value. Why?