Announcements
- Program will be up soon
- A note for HW: start early!
class Point {
private:
  double _x;
  double _y; // explicit declaration of data members

public:
  Point( ) : _x(0), _y(0) { } // constructor

  double getX( ) const {
    return _x;
  }

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

  double getY( ) const {
    return _y;
  } // accessor

  void setY(double val) { 
    _y = val;
  } // mutator

3;
Inheritance

What is inheritance?

A “child” class can use data &
functions of a “parent” class.

Let’s us be lazy!
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. So far, have seen public and private. Main can see only in class.

What about data that main can't have, but child classes should?

protected:

    int _height;
    int _width;

    child classes can use these
Objects

In Python, variables are pointers to actual data.

```
b = a;
b = Point(3, 4);
a = b;
```
C++: More versatile

C++ allows for 3 different types of variables:

1. Value
2. Reference
3. Pointer
**Value Variables**

When a variable is created, a precise amount of memory is set aside.

Point \( a \):

Point \( b(5, 7) \):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( a ) : Point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x = 0.0 )</td>
<td>( y = 0.0 )</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( b ) : Point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x = 5.0 )</td>
<td>( y = 7.0 )</td>
<td></td>
</tr>
</tbody>
</table>

More efficient (for both speed & space).
Now set \( a = b \): setup

```
| a : Point |
|---|---|
|x = 12.0 |
| y = 7.0 |
```

```
| b : Point |
|---|---|
|x = 5.0 |
| y = 7.0 |
```

They stay separate!

With value variable, get deep copies by default:

\[ a.\text{set}X(12.0); \]
Functions: passing by value

```cpp
bool isOrigin(Point pt) {
  return pt.getX() == 0 && pt.getY() == 0;
}
```

When someone calls `isOrigin(myPoint)`, the value of `pt` is initialized as a new, separate variable.

Essentially, the line:
```
Point pt (myPoint);
```
is run at the beginning of the function.

Point `pt`;
`pt = myPoint;`
Reference Variables

Syntax: `Point & c(a)`;
- `c` is created as an alias for `a`;
- More like Python, but `c` is always the same as `a`.

Ex: `c = b;` will not make `c` point to `b`, but will actually change value of `a`.
<table>
<thead>
<tr>
<th>variable name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>140</td>
</tr>
<tr>
<td>b</td>
<td>141</td>
</tr>
<tr>
<td>c</td>
<td>142</td>
</tr>
<tr>
<td>d</td>
<td>143</td>
</tr>
<tr>
<td>e</td>
<td>144</td>
</tr>
<tr>
<td>f</td>
<td>145</td>
</tr>
<tr>
<td>g</td>
<td>146</td>
</tr>
<tr>
<td>h</td>
<td>147</td>
</tr>
<tr>
<td>i</td>
<td>148</td>
</tr>
<tr>
<td>j</td>
<td>149</td>
</tr>
</tbody>
</table>

Ex:
```c
int a;
int & b(a);
int c(7);
b = 63;
c = 11;
a = 50;
b = c;

int & d(c);
```

Why useful??
Passing by reference
Reference variables aren’t generally use in main.
Instead, primary purpose is in functions:

Ex:

```cpp
bool isOrigin(Point& pt) {
    return pt.getX() == 0 && pt.getY() == 0;
}
```

Point& pt (mypoint)

mypoint.x 141
mypoint.y 142

aliases the outside variable during the function.
Why pass by reference?

2 main reasons

① Changes made in the function will persist.

② Faster (no making a new copy)
Less space (no new copy)
If we want the speed of passing by reference, but we don't want changes to variable, use const:

```
bool isOrigin(const Point& pt) {
    return pt.getX() == 0 && pt.getY() == 0;
}
```

Const goes before variable!

Compiler will enforce that `pt` isn't changed inside the function.
Recall: Point output

```cpp
ostream& operator<<(ostream& out, Point p) {
    out << "\(" << p.getX() << ", " << p.getY() << "\)";
    return out;
}
```

Here, & is required since streams cannot be copied.

Note: don't use const. Why?

The whole point is to change the stream!
3) Pointer variables

Syntax: `int * d;`

`d` is created as a variable that stores a memory address.

**Ex:**

```
int b[8];
int* d;
```

Memory address of `b`:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Contents</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b</code></td>
<td>8</td>
<td>281</td>
</tr>
<tr>
<td></td>
<td></td>
<td>282</td>
</tr>
<tr>
<td></td>
<td></td>
<td>283</td>
</tr>
<tr>
<td></td>
<td></td>
<td>284</td>
</tr>
<tr>
<td></td>
<td></td>
<td>285</td>
</tr>
</tbody>
</table>

But `d` is **not** an `int`.
Can't write `d = b`!