Announcements

- HW due tomorrow
  (email will be hard for me tomorrow!)
- Dept. picnic next week, Wed. at 4pm
- HW2 posted later today
- Tutoring should start next week
Last time

Scoping Classes:

- syntax
- usage

class Name 
private:
public:
j
Inheritance

What is inheritance?

Create "child" class which steals the data & functions from "parent" class.

(a good way to be lazy)

```
FillableShape
  
    . . . Rectangle
    
    Circle
    
    Square
```
Example: Square class

class Square: public Rectangle {
    public:
        Square(double size=10, Point center=Point()); // 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.

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

protected:
Objects

In Python, variables are pointers to actual data.

- `a` = Point(0, 0)
- `b` = Point(3, 4)
- `a = b;`
C++: More versatile

C++ allows for 3 different types of variables:

1. Value - what you have seen so far
2. Reference
3. Pointer
Value Variables

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

Point \( a \):

\[
Point \ b(5,7); \quad a = b;
\]

More efficient (for both speed & space).
Now set $a = b$.

<table>
<thead>
<tr>
<th>a : Point</th>
<th>b : Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = 5.0$</td>
<td>$x = 5.0$</td>
</tr>
<tr>
<td>$y = 7.0$</td>
<td>$y = 7.0$</td>
</tr>
</tbody>
</table>

They stay separate!

*deep copy*
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!

So do changes to the point last?

**No**
2. Reference Variables

Syntax: \( \text{Point} & \ c(a) \)

- \( c \) 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 \( a \).
Example:

```c
int a;
a = 35;

int & b(a);

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

Table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Contents</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>a</td>
<td>140</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>141</td>
</tr>
</tbody>
</table>

The contents of variable `b` have been changed to `\times 11`.

```
142
143
144
145
146
147
148
149
```
Passing by reference

Reference variables aren't generally used in main.

Instead, primary purpose is in functions:

Ex:

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

In main:

```cpp
isOrigin(a)
```
Why pass by reference?

3 main reasons

1) Saves time (to copy)
2) Saves space
3) Allows changes to persist outside function
   "feature"
If we want the speed of passing by reference, but we don't want changes to variable, use const:

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

Const here means pt may not be changed.

Compiler will enforce that pt isn't changed inside the function.

Ex: `setX` in function would give an error.
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?

Goal is to change the output stream.
Pointer variables

Syntax: int * d;

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

Ex:

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

// give me address

d = &b;

// output is 8

cout << *d; // d = 8

// can't write d = b!
```

<table>
<thead>
<tr>
<th>variable</th>
<th>contents</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>281</td>
</tr>
<tr>
<td></td>
<td></td>
<td>282</td>
</tr>
<tr>
<td>b</td>
<td>6</td>
<td>283</td>
</tr>
<tr>
<td></td>
<td></td>
<td>284</td>
</tr>
<tr>
<td></td>
<td></td>
<td>285</td>
</tr>
<tr>
<td></td>
<td></td>
<td>286</td>
</tr>
<tr>
<td>x</td>
<td>5</td>
<td>287</td>
</tr>
</tbody>
</table>