Data Structures

Today:

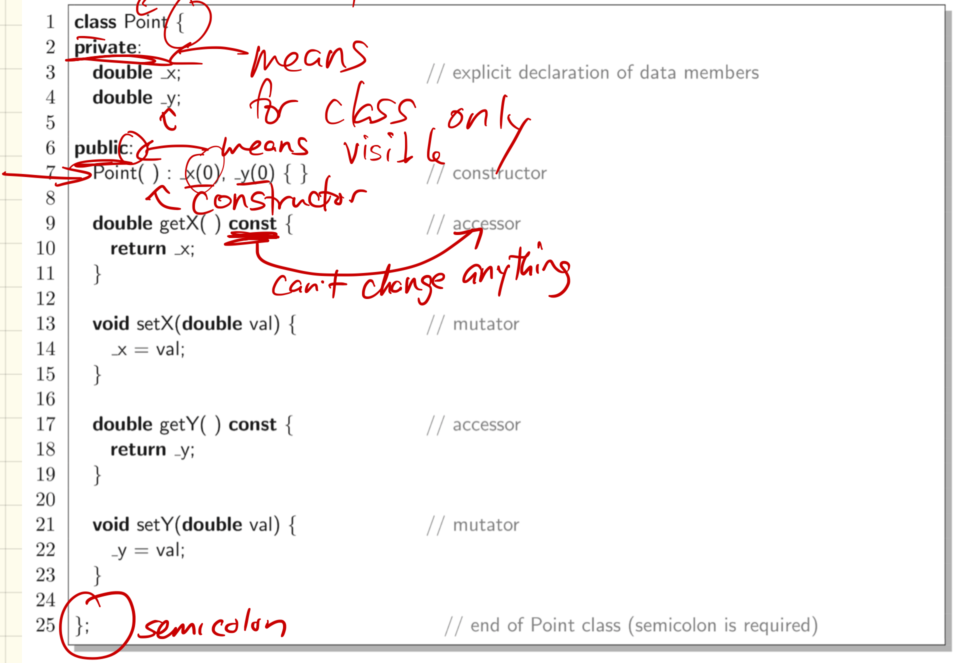
Classes

Variable Models

Announcements

- New office hours: 1-2pm on Friday (Wed. goes away, although I'm often in)
- Lab: due Friday (via git!) (make sure you pass judge program)
- Next HW: half written, half programming - up later today

Last time:



```
class Point {
private:
   double _x;                          // explicit declaration of data members
   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;
   }
};                                      // end of Point class (semicolon is required)
```

must capitlize

means for class only

means visible

constructor

can't change anything

semicolon

Figure 9: Implementation of a simple Point class.

Today: more...

Classes:

(1) Data + fcns: MUST be public, private, or protected

↑ more later

- Enforced by compiler!
- General convention: all data is private

(2) Constructor:

- name: Same as class (only 2 capitalized things)
- no return type (only time)
- Can initialize in list or in body:

```
Point(double initialX, double initialY):
        x(initialX), y(initialY) {}
      10.0                2.0
             ⇕ Same

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

More:

③ No self!

Just say x or y in class functions & will use class variables.

Note: can't use x & y as fcn variable

④ Accessor vs. mutator:

use const

A more complex one...

```
class Point {
private:
  double _x;
  double _y;

public:
  Point(double initialX=0.0, double initialY=0.0) : _x(initialX), _y(initialY) { }

  double getX( ) const { return _x; }        // same as simple Point class
  void setX(double val) { _x = val; }        // same as simple Point class
  double getY( ) const { return _y; }        // same as simple Point class
  void setY(double val) { _y = val; }        // same as simple Point class

  void scale(double factor) {
    _x *= factor;
    _y *= factor;
  }

  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)
```

2 slides ago

sets defaults if no input

in main: double d = mypoint.distance(otherpt);

object in Pt class

in main: mypoint + otherpoint;

mypoint.operator+(otherpt);

return

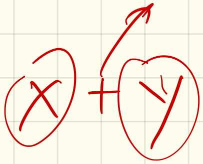
input - a point

Notes:

1) $\underline{x + other.x}$:

allowed only inside class, for when another object is an input

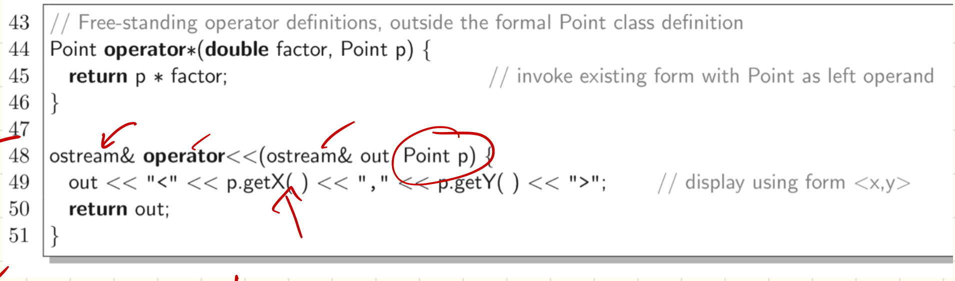
2) operator+ ;



3) two versions of operater*

Additional common functions, but after class :

}; //end of Point class



```
// 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;
}
```

-> use cout

Why? so we can call 6 * (2,3)

-> cout << mypt << endl;

> <2,4>

Finally:

.h vs. .cpp files:

So far, just used cpp.
The .h extension is just for classes

Idea:

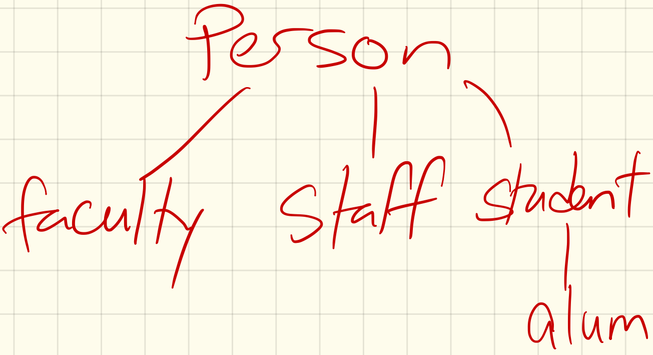
- Seperate classes from main, which might need many of them.
- Then import all needed .h files into one cpp file that has the main

Inheritance

What is it?

Class is a "subset" of another - can steal all fcns + data

Ex: Any of graphics objects in Python



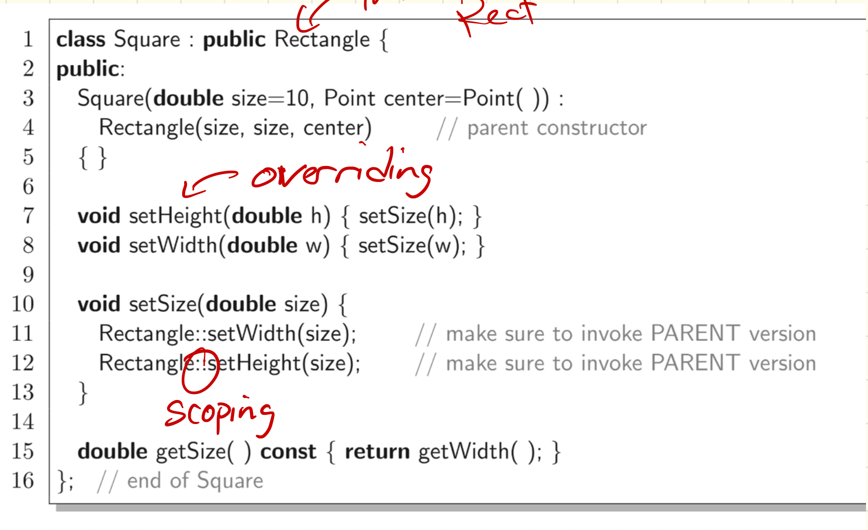
Code example:

Suppose we make a Rectangle class:

- two private variables (height & width) & center, a Point
- functions to reset each

Square class:

inherit from Rect



```
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
```

overriding

scoping

And protected data:

- Public
- Private:
- Protected:

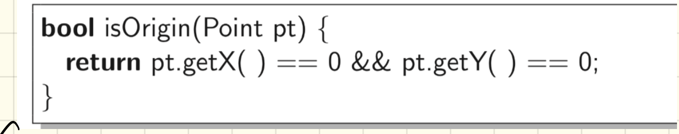
↳ for inheritance (+ friend class)

Not public, but only children + friend classes can see it.

More on variables

In Python, variables were just identifiers for some underlying object.

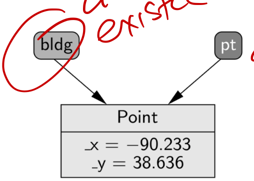
This had implications when passing variables to functions:



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

↳ So if you do:

```
if (isOrigin(bldg))
  <code>
```



already existed

new identifier in fcn

Figure 14: An example of parameter passing in Python.

in lists - meant
had shallow copies

C++: Much more versatile:

3 parameter types

① Value

② Reference

③ Pointer

So far, you've been using value - easiest.

Reference + Pointer require looking at memory more carefully...

① Value Variables

When a variable is created, a precise amount of memory is allocated:

```
Point a;
Point b(5,7);
```

Memory:

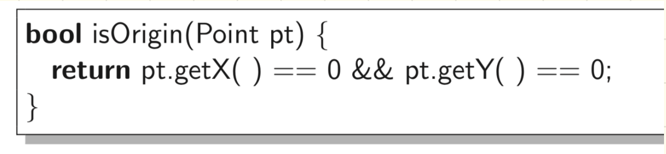
labels	content	addresses (hex #s)
		867
		868
		869
		870
		871
		872
		873
		⋮
		1011
		1012
		1014
		1015
		⋮

Now:

```
a=b ;
```

What happens?

Functions + passing by value:



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

When someone calls

isOrigin(mypoint);

The (local) variable pt is
  created as a new, separate
  variable

Essentially, compiler inserts
  Point pt(mypoint);
as first line of the function.

So - What if we change pt?

② Reference variables

Syntax:

```
Point& c(a);
```

What it does: