CSCI 2100: More C++

Classes
Variable Models

Recap

- HW1 due today
- Lab 2 - due Sunday
- HW2: Two problems
  - 1st: pen + paper
  - 2nd: on Zybook

already posted

on my page

Compiling on Hopper or in Lab:

- Go to terminal/console ← ssh to hopper
- Edit .cpp/.h file
- at prompt:

```
> g++ main.cpp
or

g++ main.cpp Class.cpp
     -Wall  -o a.out
```

main

class file

turn on messages

gives output nam (optional)

then:

```
./a.out
```

Command Line Tips

In general, 5 or 6 commands will go far!

- ls
- cp -v sourcefile destfile
- mkdir name
- rmdir name
- cd directory
  - ↳ variants

-v : verify

- mv sourcefile destfile
- rm file

↳ Careful!

-i to be careful (interactive mode)

I'll post some tutorials

Others

← text editors

- emacs, vi or nano (kate)
- g++
- make (later)
- man command

manuel pages

```
> man ls
```

Also:

- CS page has info on connecting (Dennis + I can also help!)
- Many, many resources online

Bit vise or putty on windows

A few tricks

- Hit up arrow : gives last command, which you can then edit
- Tab will auto complete file names
- On lab or nomachine, & gives prompt back

  ie `> kate myfile &`

- `.` is current directory

  `..` is parent (up one level)

  `~/` is home

  `/` is root

Ex:

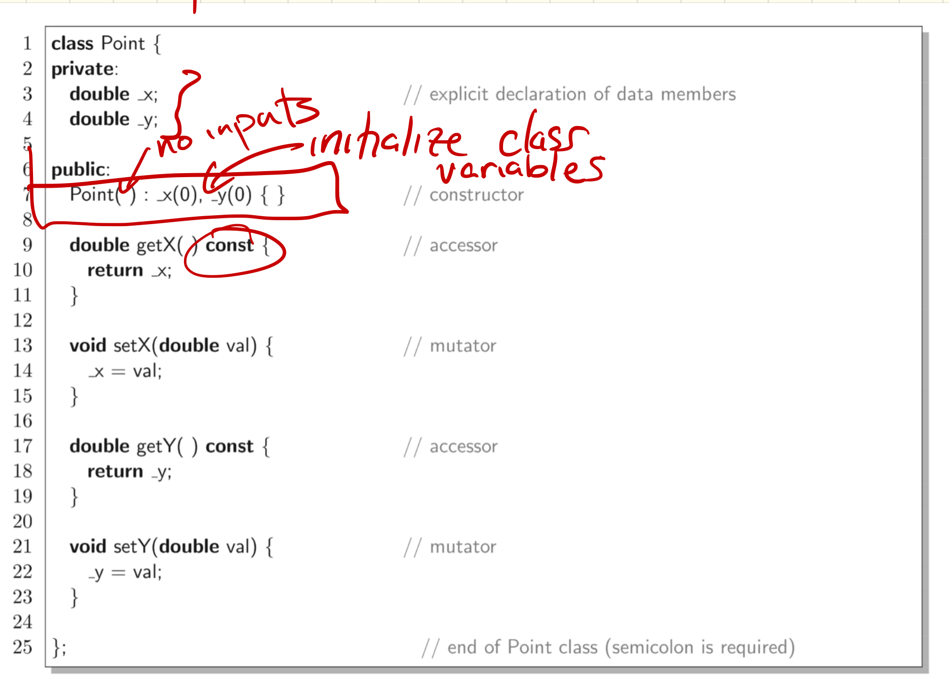
```
> cd ..
> ./a.out
> cp ../file .
```

Can also use IDE (development environment) on own laptop

- eclipse
- code blocks
- Xcode (mac)

Last time:

Simple class file



```
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)
```

no inputs

initialize class variables

Figure 9: Implementation of a simple Point class.

Point.h

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 the class (capital letter fcn)
   - no return type
   - Can initialize in list or in body:

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

More:

③ No self!

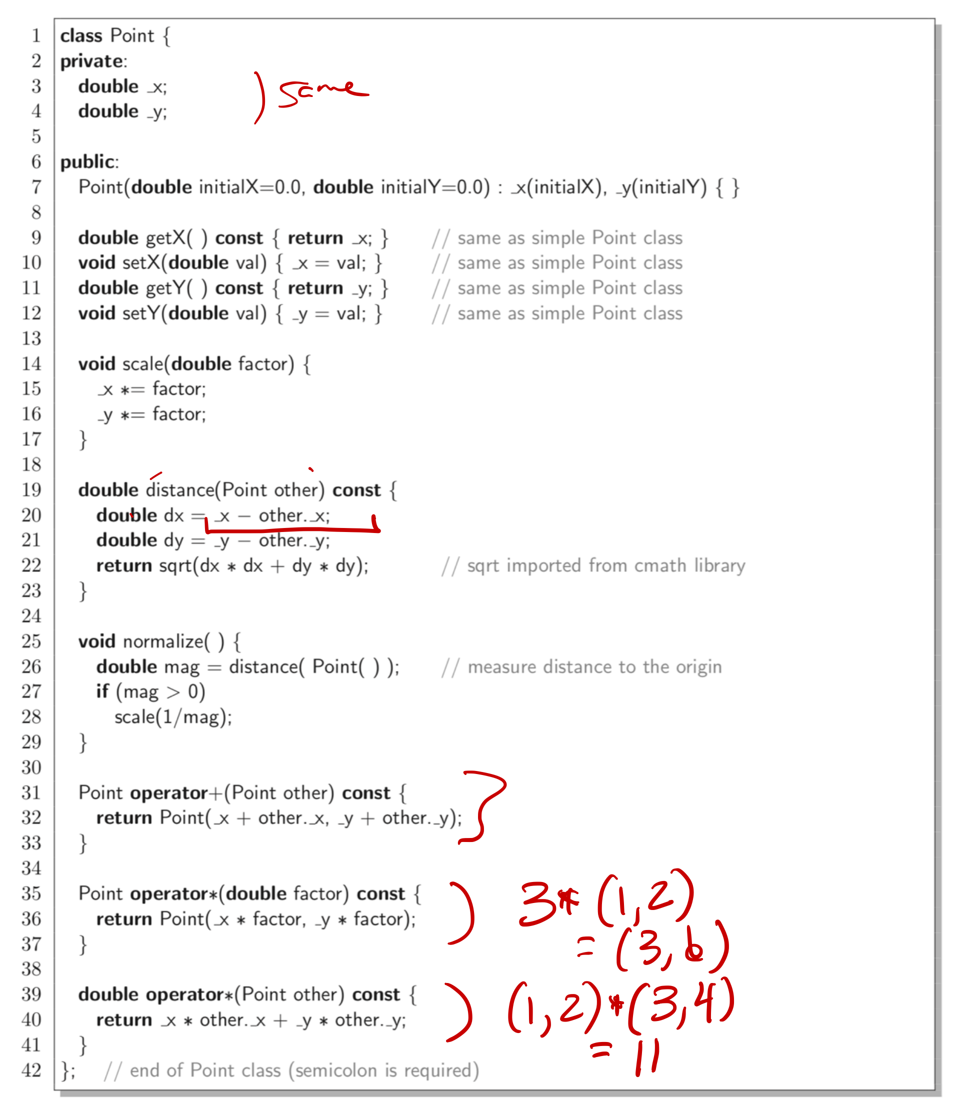
Just say x or y in class functions/ & will use class variables.

Note: can't have local x or y in any class function

④ Accessor vs. mutator:

use const (in function)

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)
```

same

3*(1,2)
= (3,6)

(1,2)*(3,4)
= 11

Usage:

```
#include "Point.h"
   ⋮

int main() {
    Point mypoint(1.2, 3.9);
    Point other;
    other.setX(13.2);
    float d = mypoint.distance(other);
        ⋮

}
```

Notes:

1) x + other.x :

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

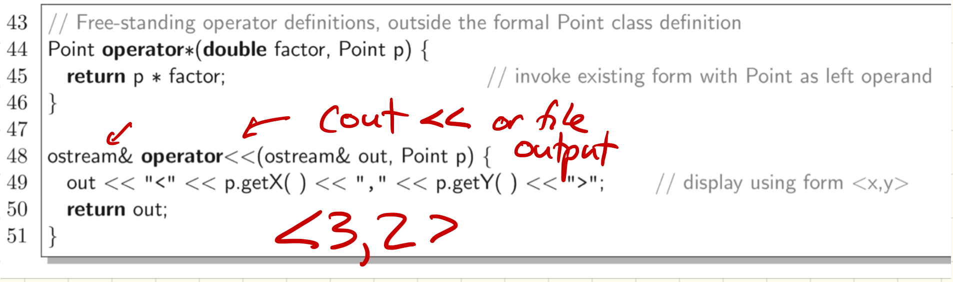
2) operator+ :

```
Point p = mypoint + other;
```

3) two versions of operator*

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;
}
```

cout << or file output

<3,2>

Why? 2*(3,4)

or (3,4)*2

Finally:

.h vs. .cpp files:

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

Idea:

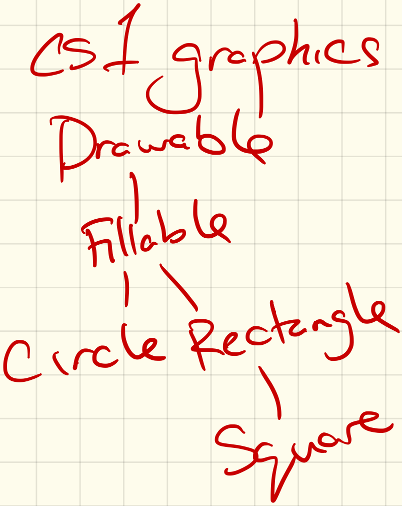
- Seperate classes from main, which might need many of them.
- Then import all needed .h files into one cpp file that has the main

Inheritance

What is it?

Lets "child" class use data + methods of parent class

Ex:

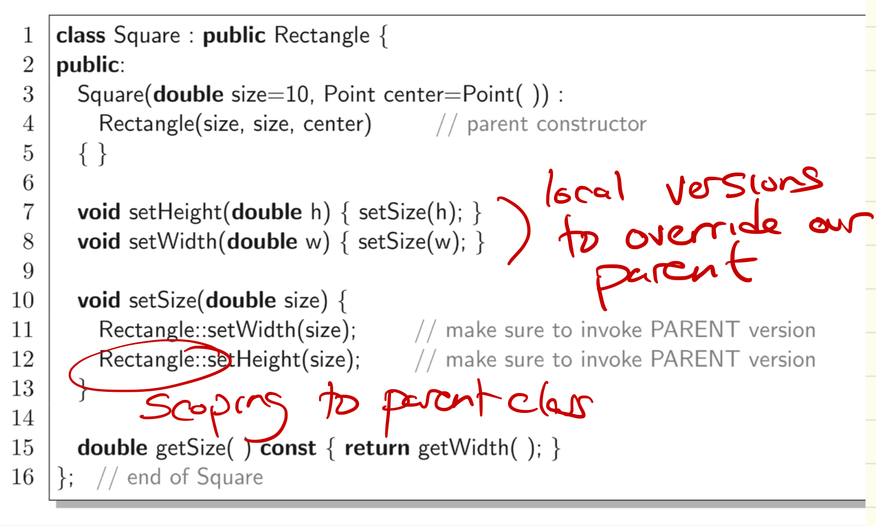


Code example:

Suppose we make a Rectangle class:

- two private variables (height & width)
- functions to reset each

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
```

local versions to override our parent

Scoping to parent class

std::cin

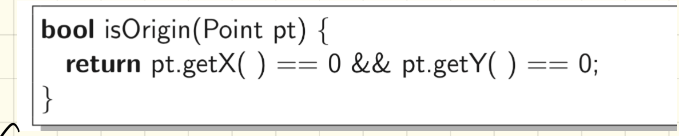
And protected data:

- Public : open to all
- Private: no one!
- Protected :
  - child classes can see
  - friend classes can see
  - main can not

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>
```

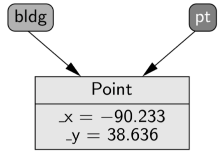


Figure 14: An example of parameter passing in Python.

Shallow copy

C++: Much more versatile:

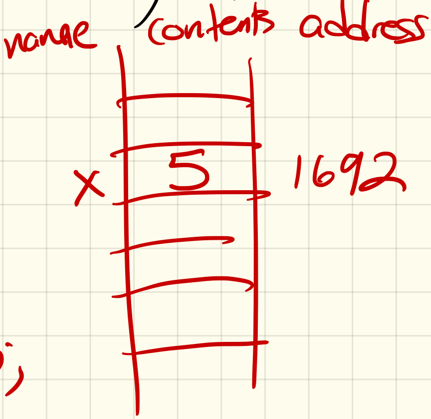
3 parameter types

1. Value
2. Reference
3. Pointer

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

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

abstract picture of memory:



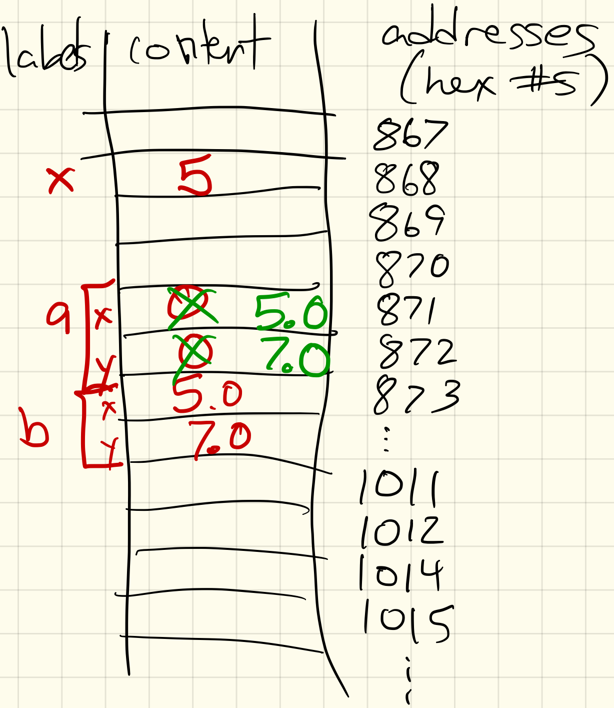
int x=5;

1 Value Variables

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

```
int x = 5;
Point a;
Point b(5,7);
```

Memory:



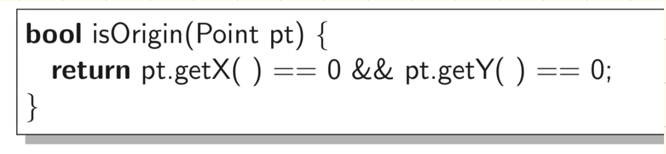
Now:

a=b ;

What happens?

deep copy

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: