CS180 - C++ : References + Pointers

Note Title 8/27/2010

Announcements

- HW1 due Wednesday
- Program 1 due next Friday - checkpoint next Tuesday
- Lab on Friday this week
- Tutoring hours are posted on department webpage

Last time:

- input & output   iostream, fstream
- classes & member data/functions

Point class

C++ forces private data

Simple Point Class

```
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 semi-colon

classes get semi-colons

Comment

/* Comment */

Robust Point Class:

```
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;   ←   _x = _x * factor;
      _y *= factor;
    }

    ⋮
```

mypoint.normalize();

Robust Point class cont:

```
  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);
  }
```

mypoint = point1 + point2;

_x, _y   other

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

mypoint = point1 * 5;

```
  Point operator*(double factor) const {
    return Point(_x * factor, _y * factor);
  }
```

mypoint = point1 * point2;

```
  double operator*(Point other) const {
    return _x * other._x + _y * other._y;
  }
}; // end of Point class (semicolon is required)
```

Things to note:

1) _x + other._x ← allowed if inside the class
(even though _x is private)

2) using operator+, will be x+y

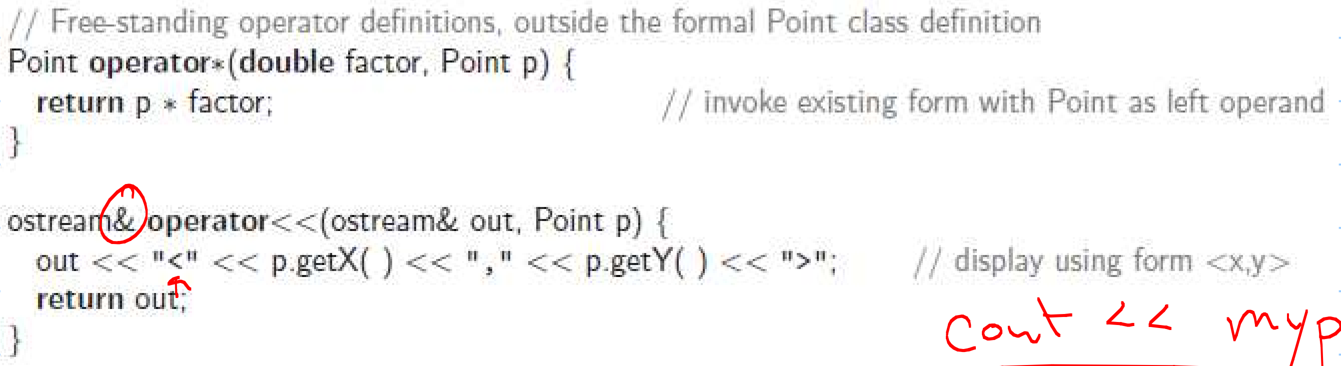
3) two versions of *
one for factors versus one for points

(1,1) * 5 = (5,5)

(1,1) * (3,2) = 5

another issue: 5*(1,1)

Additional functions
(Not in 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 << mypoint;

< 5, 5 >

Why outside of class?

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

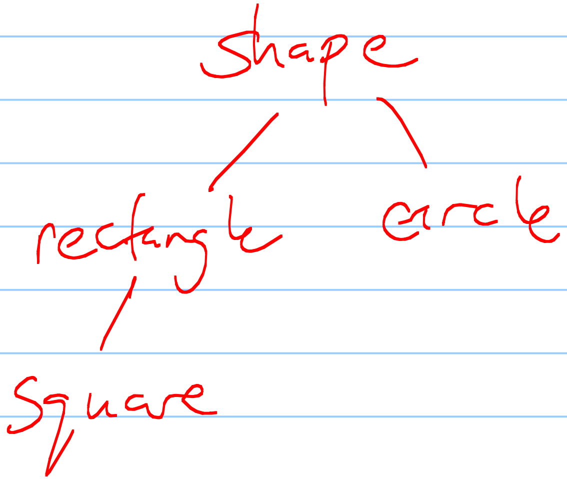
Ch. 2

Inheritance - a good way to be lazy

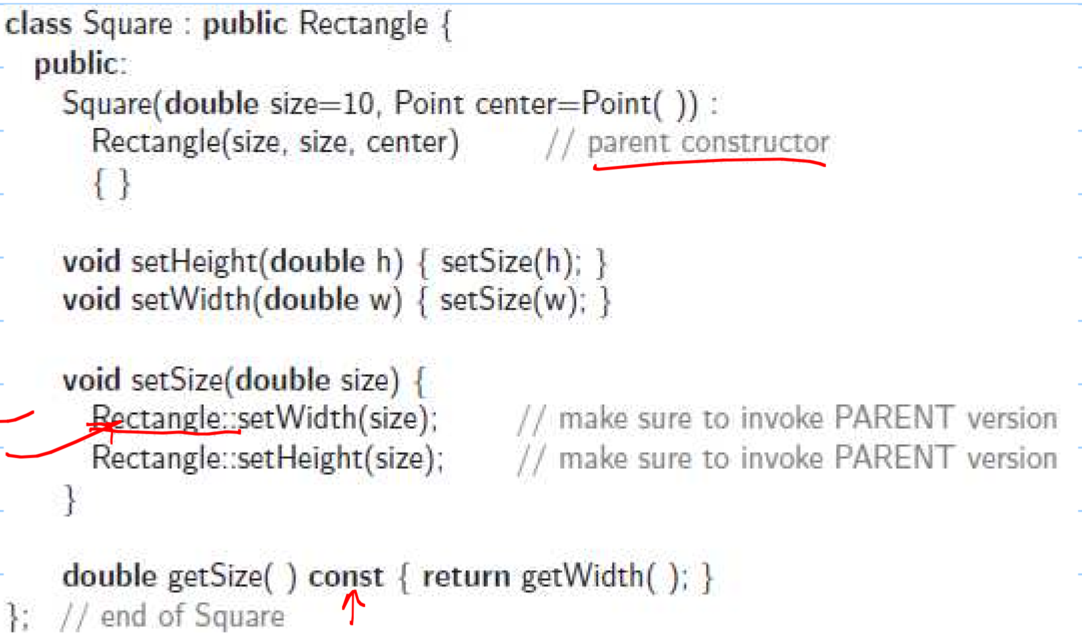
What is it?

Allowing code reuse by declaring sub-class.

Child class "inherits" all data + functions, but additional ones can be added.



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
```

variable scoping

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:~~

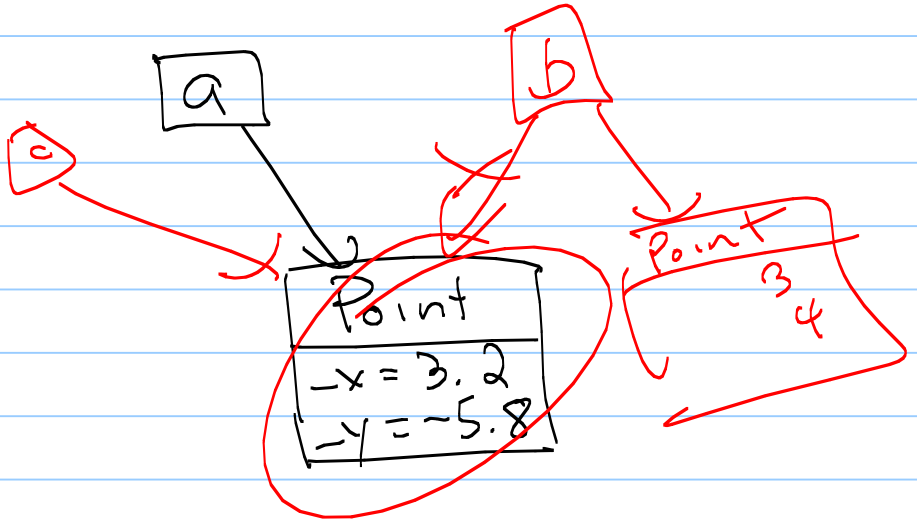
~~int height;~~
~~int width;~~

protected:

int height;
int width;

Objects & Memory Management

In Python, variables were pointers to data:



```
-> b = a;
   b = Point(3,4);

   c = a;
   c = c+b;
```

C++: A more versatile setup

C++ allows 3 different models for storing & passing information.

① Value

② Reference

③ Pointer

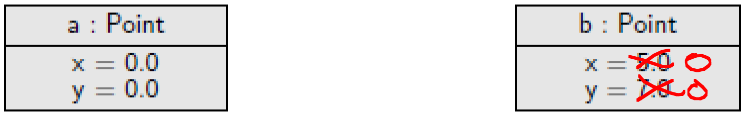
(Remember that strange & a few slides ago?)

Value Variables

When a variable is created a precise amount of memory is set aside:

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

```
b = a;
```



a : Point

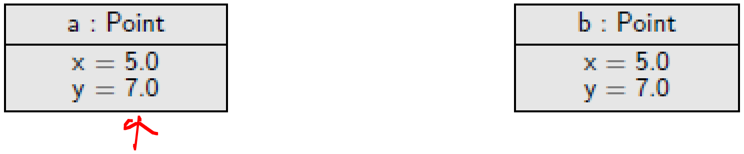
x = 0.0
y = 0.0

b : Point

x = ~~5.0~~ 0
y = ~~7.0~~ 0

This is more efficient, both for space and speed.

Now suppose we set a = b:



a : Point

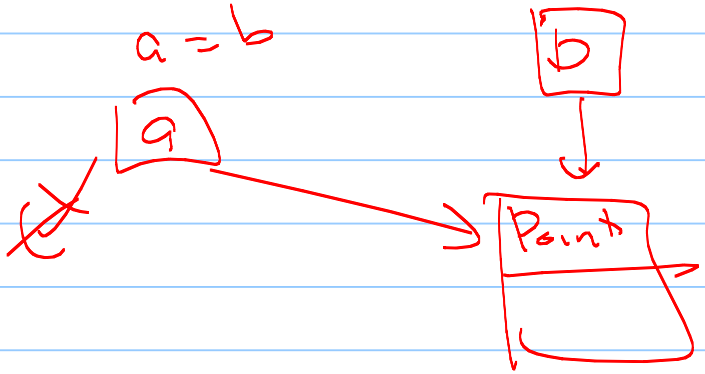
x = 5.0
y = 7.0

b : Point

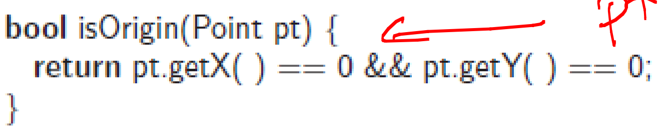
x = 5.0
y = 7.0

They stay separate!

Different than Python:



Functions: Passing by Value



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

pt.setX = 5;

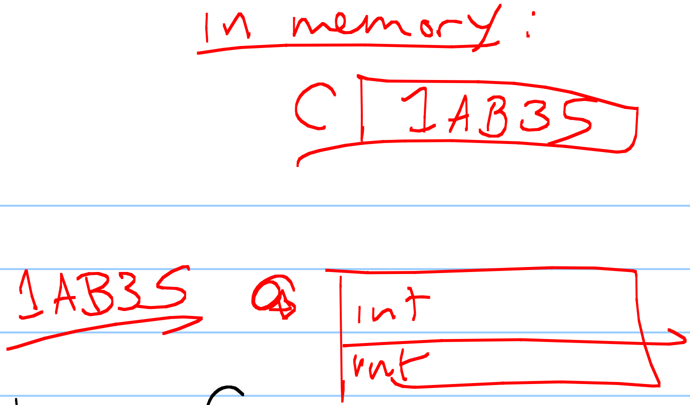
wouldn't change myPoint

When someone calls isOrigin(myPoint) later, the value pt in the function is initialized as though a new variable was created:

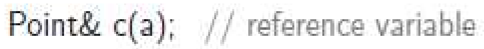
Point pt(myPoint);

So changes in function to pt don't affect myPoint!

2 Reference Variables



Syntax:



```
Point& c(a);   // reference variable
```

- c is created as an alias for a
- More like Python model, but can't be changed later

Ex: c=b;

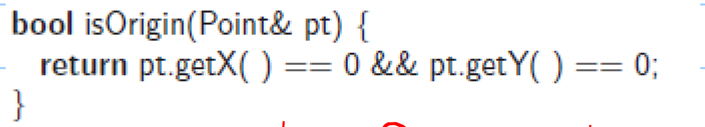
Will not rebind c to point to b, but will change the value of c (and a).

Passing by reference:

Reference variables aren't usually needed in main program.

Instead, they are primarily used for passing to functions!

Ex:



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

instead of making a local copy of input, makes a reference

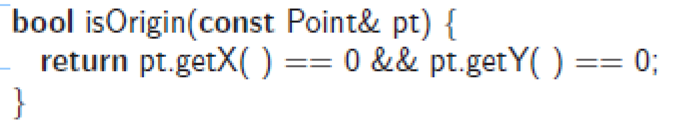
here, changes to pt persist outside fcn

Passing by reference (cont.)

Why?

- changes persist
- saves memory
- increase speed

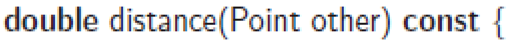
If we want the speed of passing by reference but don't want our object mutated, use const.



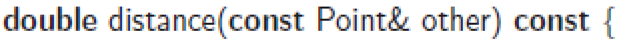
```
bool isOrigin(const Point& pt) {
  return pt.getX( ) == 0 && pt.getY( ) == 0;
}
```

Compiler will ensure that pt isn't modified.

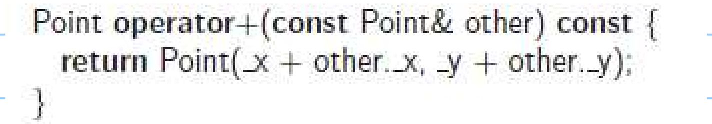
Speeding up the Point class:

original: 

```
double distance(Point other) const {
```

faster: 

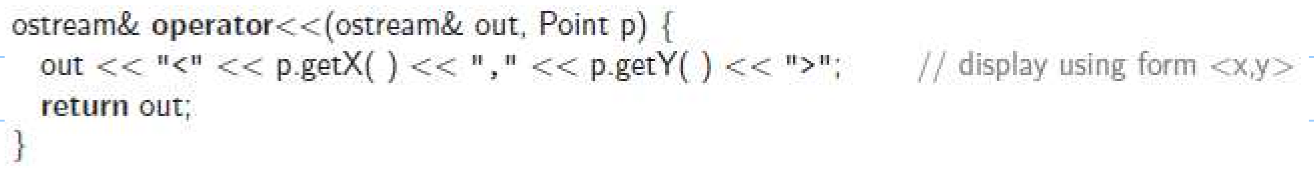
```
double distance(const Point& other) const {
```

Another: 

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

Note: Return type is still value. Why?

Recall: Point output



```
ostream& operator<<(ostream& out, Point p) {
   out << "<" << p.getX( ) << "," << p.getY( ) << ">";        // display using form <x,y>
   return out;
}
```

Here, & is required because streams cannot be copied.

Note that we don't use const, since we are changing the stream by adding data.

Pointer variables

Syntax :

```
Point *d;     // d is a pointer variable
```

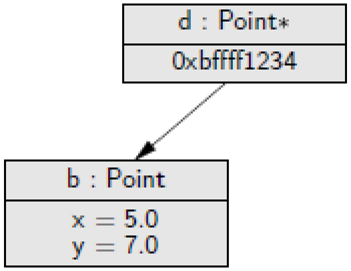
d is created as a variable that stores a memory address.

So:

```
d = &b;
```

gives

↑ memory address of b



But d is not a Point! can't say d = b

Using pointer variables

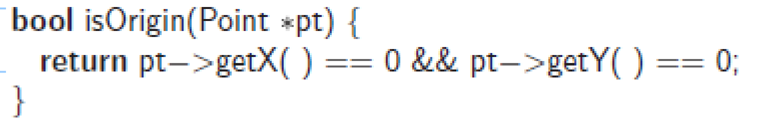
2 options:

```
(*d).getY();

d -> getY();
```

Passing by Pointer

Point *pt = NULL



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

This is similar to passing by reference, but allows you to also pass a null pointer.