CS180 - Classes + Variables in C++

Note Title 9/12/2011

Announcements

- Program will be up soon
- A note for HW: start early!

Last time:

Classes

```
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;
  }
};
```

Inheritance

What is inheritance?

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

Lets 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
```

child class will call parent's constructor

use parent's version

Other issues

A new type of data. So far, have seen: public and private.

public: main can see

private: only in class

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

protected:

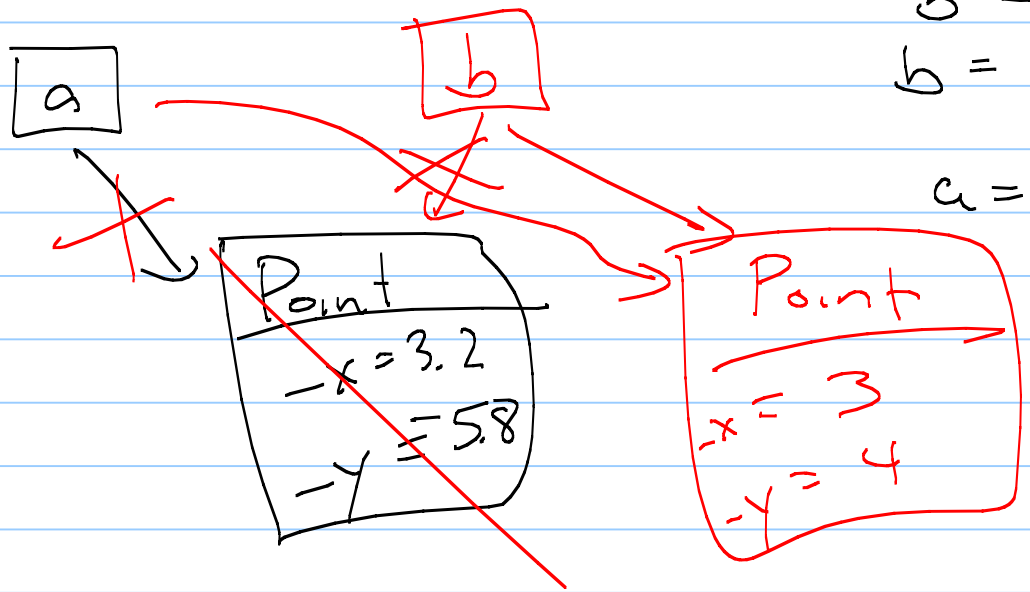
int _height;
int _width;

child classes can use these

Objects

In Python, variables were pointer 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

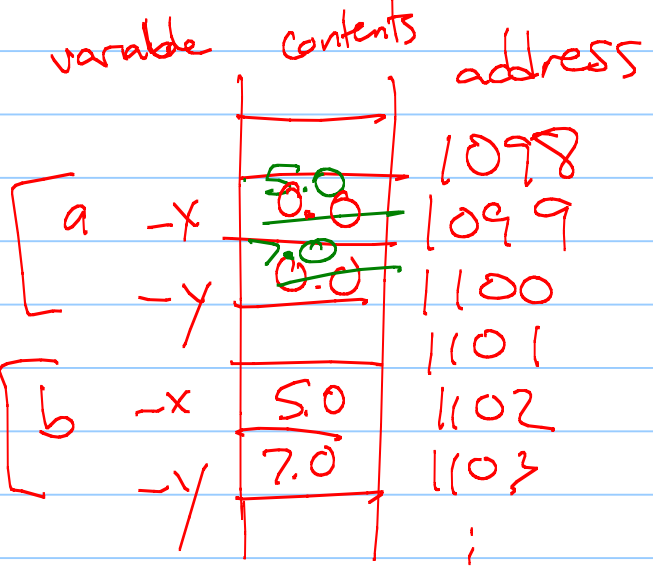
1. Value Variables

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

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

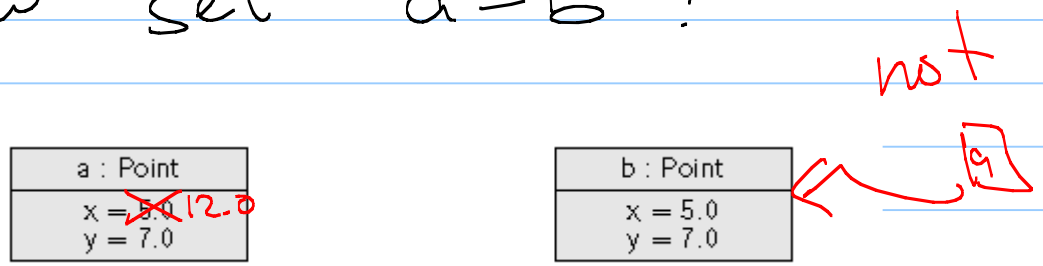
a : Point
x = 0.0 y = 0.0

b : Point
x = 5.0 y = 7.0



More efficient (for both speed & space).

Now set a = b :



They stay separate!

With value variable, get deep copies by default!

```
:
a.setX(12.0);
```

Functions: passing by value

```
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);

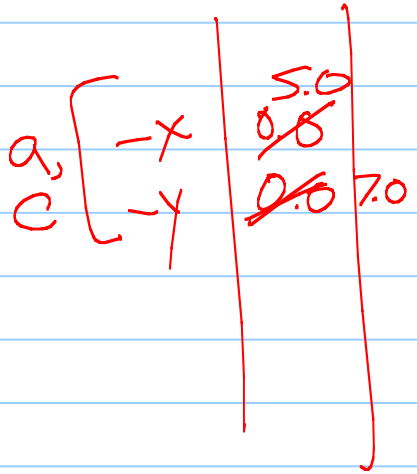
is run at the beginning of the function!

```
Point pt;
pt = myPoint;
```

② 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;

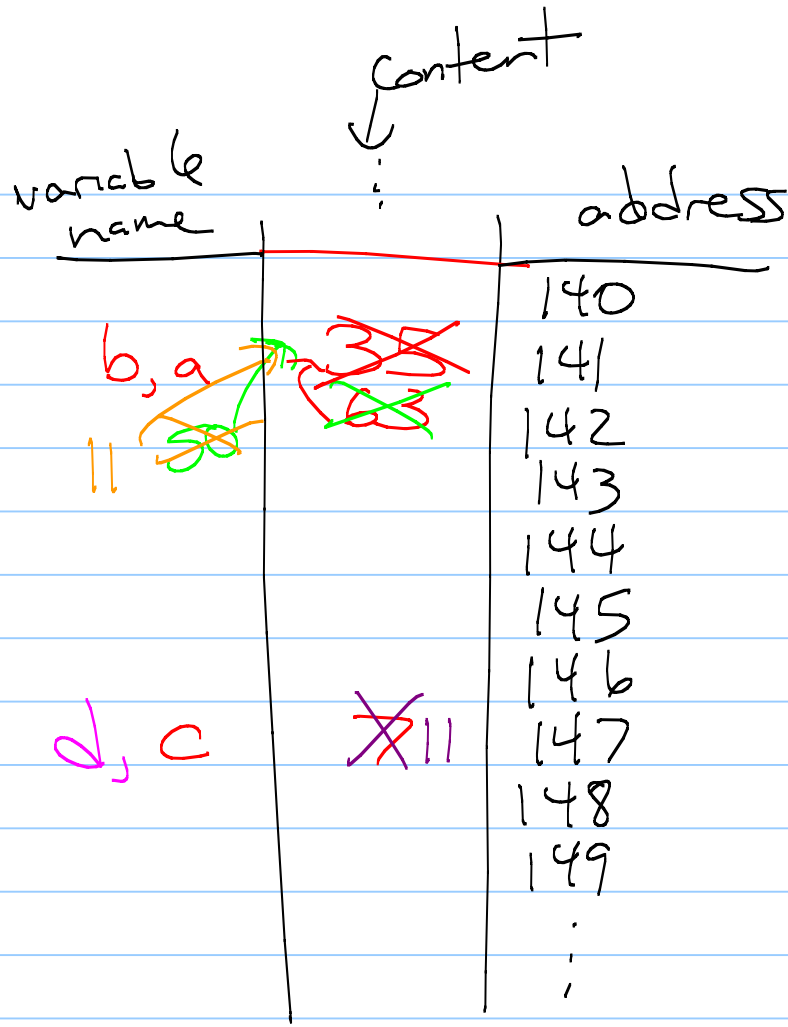
Will not make c point to b, but will actually change value of a.

Ex:

```
int a;
a = 35;
int & b(a);
int c(7);
b = 63;
c = 11;
a = 50;
b = c;
```

int & d(c);

Why useful??



Passing by reference

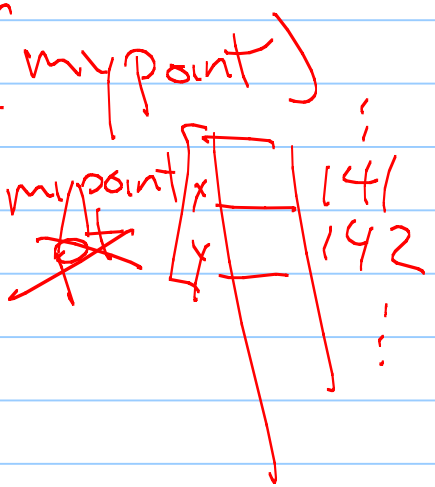
Reference variables aren't generally use in main.

Instead, primary purpose is in functions:

Ex:

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

Point& pt (mypoint)



aliases the outside variable during the function.

Why pass by reference?

2 main reasons

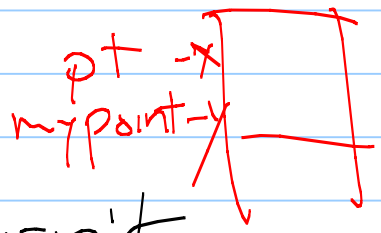
1. Changes made in the function will persist.
2. Faster (no making a new copy)

   Less space (no new copy)

If we want the speed of passing by reference, but we don't want changes to variable, use const:

const goes before variable!

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



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

Recall: Point output

```
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

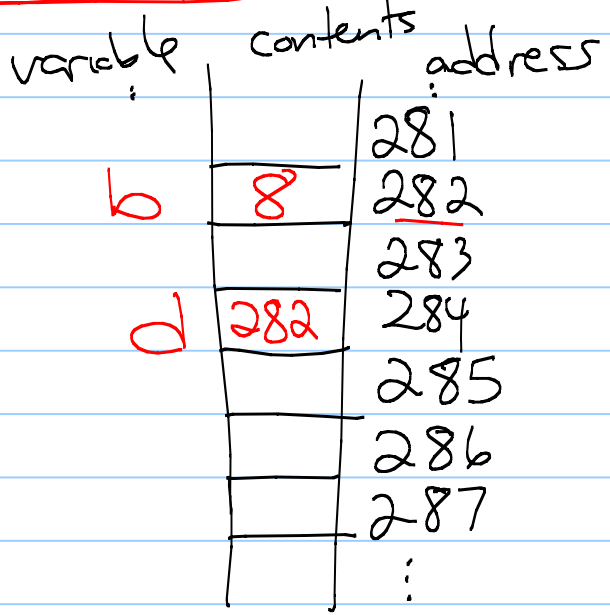
Syntax: `int * d;`

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

Ex:

```
int b(8);
int * d;
d = &b;
```

memory address of b



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