CS180 - Variables

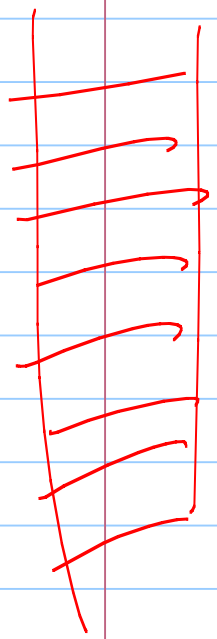
Note Title 1/27/2011

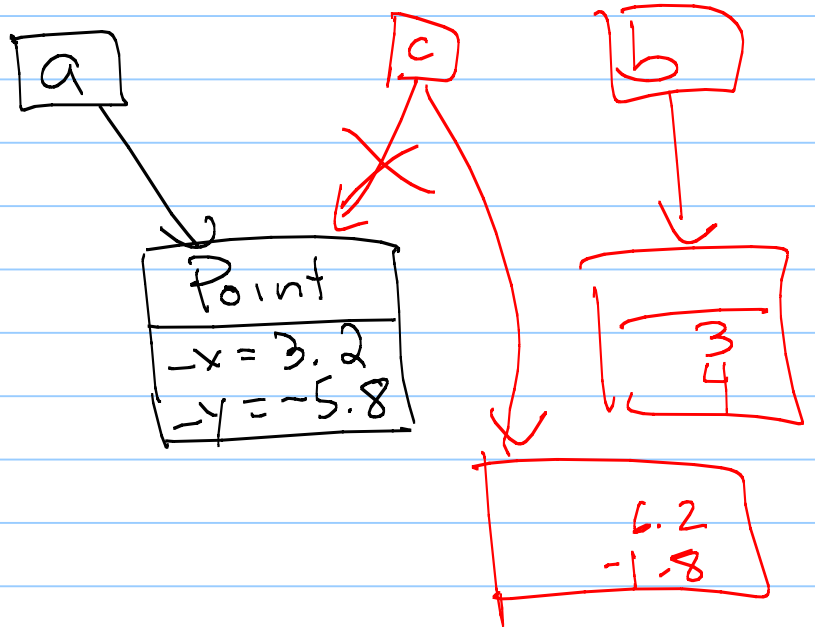
Announcements

- Schedule page is up on website
- Next hw up today

Objects & Memory Management

In Python, variables were pointers to data!






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

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

creates new point

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 (standard)

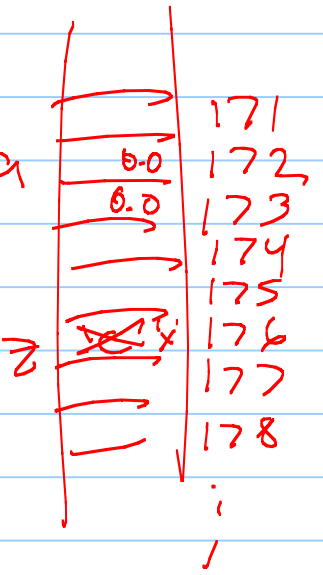
When a variable is created, a precise amount of memory is set aside:

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

a : Point
x = 0.0 y = 0.0

b : Point
x = 5.0 y = 7.0

```
char z='c';
z='x';
```



This is more efficient, both for space and speed.

Now suppose we set a = b:

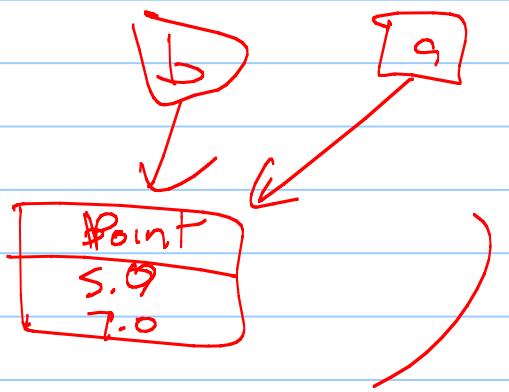
a : Point
x = 5.0 y = 7.0

b : Point
x = 5.0 y = 7.0

They stay separate!

Different than Python:

(in Python:



Functions: Passing by Value

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

(in main or another function)

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

```
Point pt(myPoint);
```

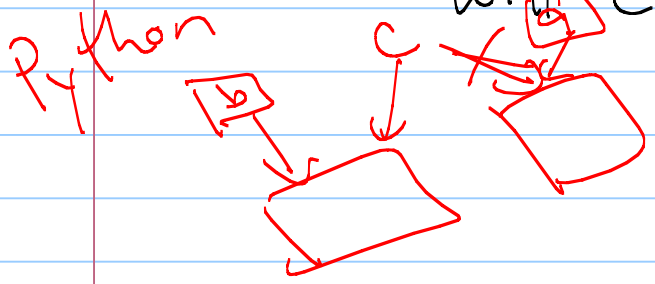
So changes in function to pt don't affect myPoint!

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).

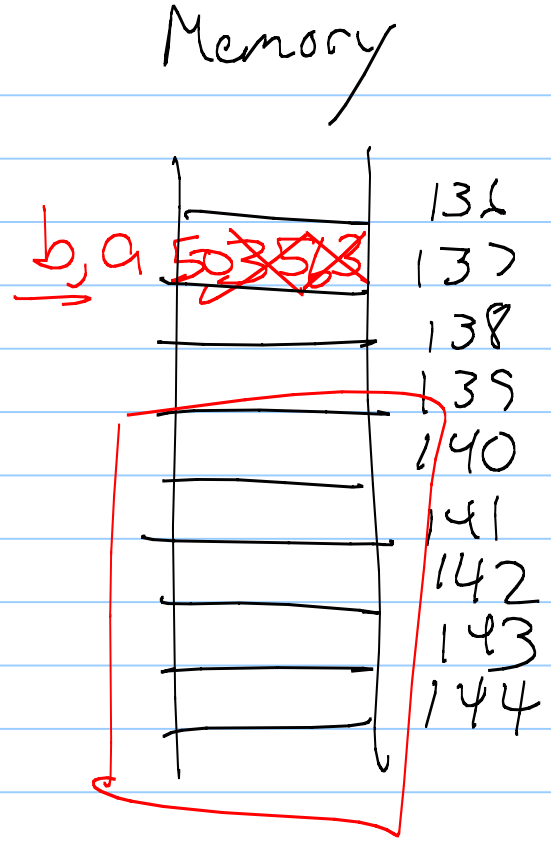


(NOT in C++)

Ex:

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

binds to a



Seems useless...

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;
}
```

this runs Point & pt(myPoint);

Passing by reference (cont.)

Why?

- Sometimes, we want changes to the variable to persist outside the function

- Value variables copy all the data which uses both time & space.

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;
}
```

makes it a constant reference, so function cannot change that variable.

Compiler will ensure that pt isn't modified.

pt.x = 5; ← compiler error

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.

3) Pointer variables

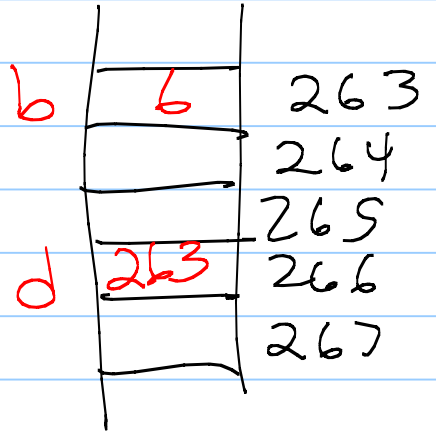
Syntax: ~~Point~~ int *d; // d is a pointer variable

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

So: int b(6);

d = &b;

↑ gives memory address of b



But d is not an int ! can't say d=6

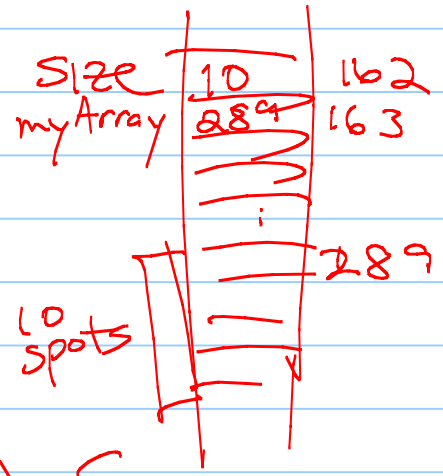
Using pointers in a class:

Suppose we need an array as private data, but don't know size.

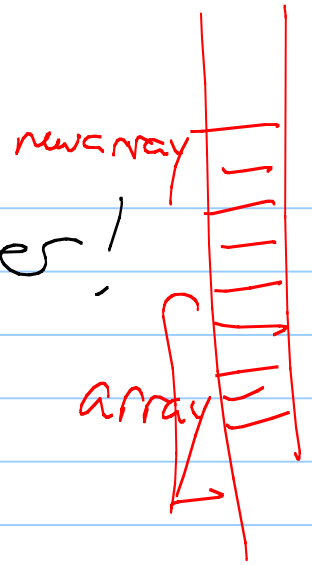
```
class MyObject {
  private:
    int size;
    // array?
    int * myArray;

  public
  MyObject (int s=10) : size(s) {
    myArray = new int[size];
  }
```

initializes some other spot



This lets you create arrays later!



Then, if need to resize:

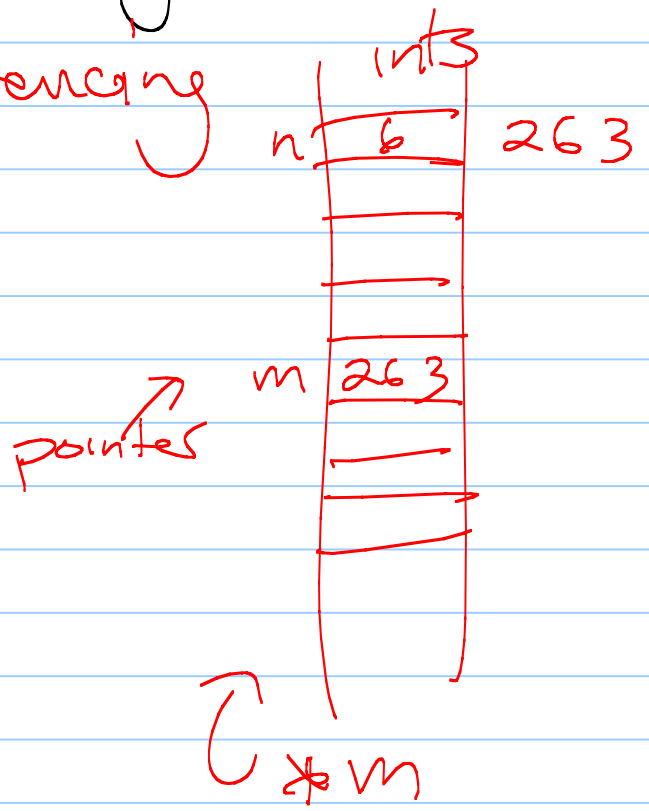
```
void resizearray(int newsize) {
    int * newarray = new int[newsize];
    for(int i=0; i<size; i++)
        newarray[i] = myArray[i];
    delete[] myArray;
    myArray = newarray;
}
```

Using pointer variables to get to data

2 options: called dereferencing

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

```
d -> getY();
```



Using pointer variables to get to data

With an array, even better:

d[i] works!
(when d is a pointer to an array & i is an array)

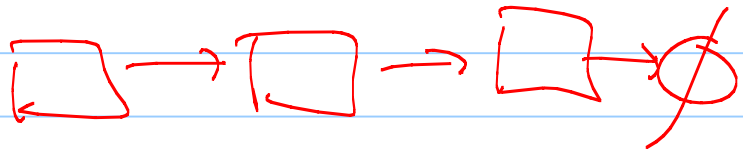
(no need to dereference)

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.



Garbage Collection

- In Python, variables that are no longer used will be automatically destroyed

Pros: lets programmer ignore deletes

Cons: time — writing it yourself is much faster

C++:

In C++, things are sometimes handled for you:

```
int n = 5;
for (int i = 1; i < n; i++) {
    int value;
    value = i;
}
cout << value << endl;
```

value is deleted at end of loop!

value is gone

Output? Error "value is not declared in this scope"

In a C++ program, all value variables are destroyed for you.

The problem is pointers.

Rule: If you use new, you need to use delete!

More on Classes: (next time)

Destructors:

If your class opens files or allocates memory, then can't just use delete.

Must create a destructor.

~ClassName() - no arguements, no return type

~Point()