CS180 - Pointers

Note Title 9/12/2011

Announcements

- Program 1 is up — due next Monday (pair project)

- Lab tomorrow

3 Pointer variables

`int * a,b;`

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!

variable	contents	address
		281
b	8	282
		283
d	282	284
		285
		286
		287
		⋮

The new command

```
int* c;
c = new int(12);
```

creates a separate piece of memory

Main use: the data persists even after the pointer is gone!

So can create or modify inside multiple functions.

```
cout << *c << endl;
```

→ print 12

variable	⋮	address
		243
		244
c	248	245
		246
		247
	12	248
		⋮

Pointers: getting to the data

Called dereferencing.

Ex:

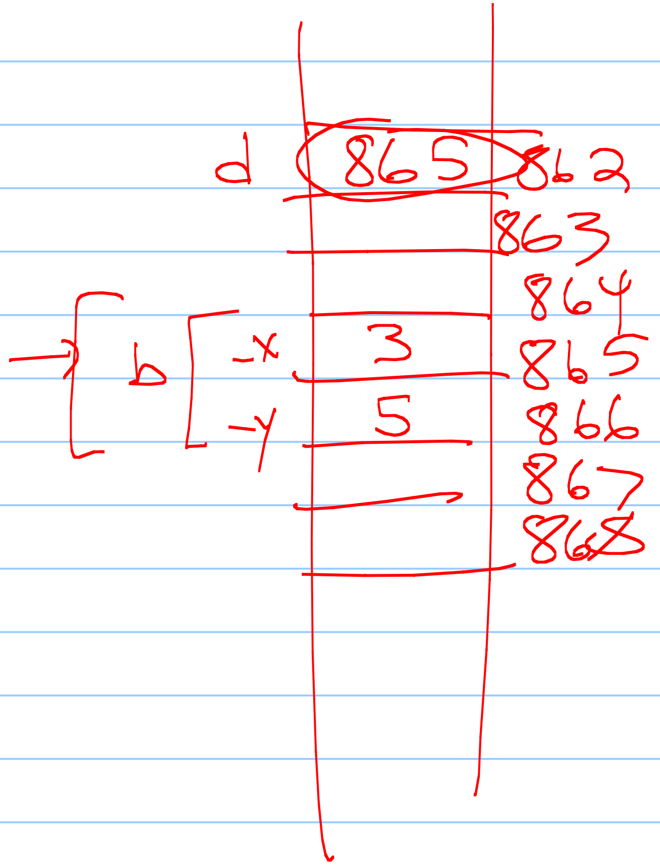
```
Point * d;
Point b(3,5);
d = &b;
```

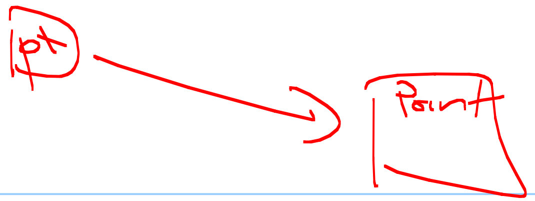
2 options:

```
(*d).getX()
```

or

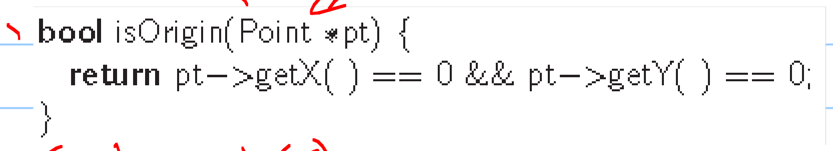
```
d -> getY()
```





Passing pointers

address stored in input parameter

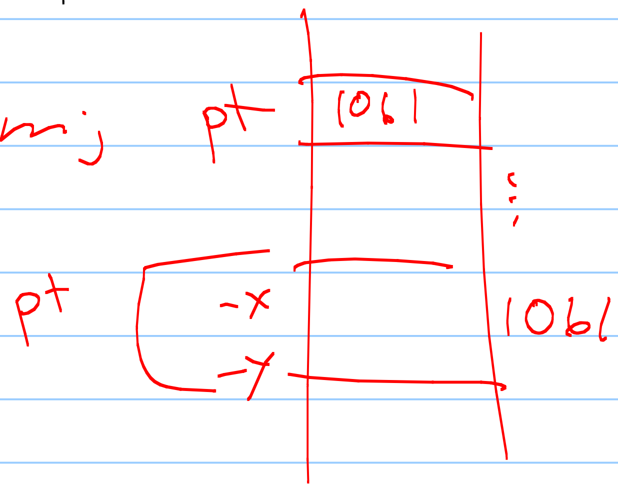


```
bool isOrigin(Point *pt) {
    return pt->getX( ) == 0 && pt->getY( ) == 0;
}
```

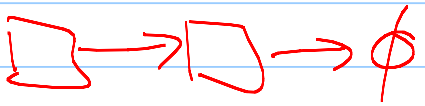
(*pt).setY(5);

Similar to passing by reference, but allows passing a NULL pointer also.

Point * pt = inputparam;



Linked Lists:



Pointers in a class

Pointers are especially useful in classes.

Often, we don't know all the details of private variables to put in the private declaration.

Example: arrays!

What do we need when creating an array?

size & type

int array class
add, sort, average,
max

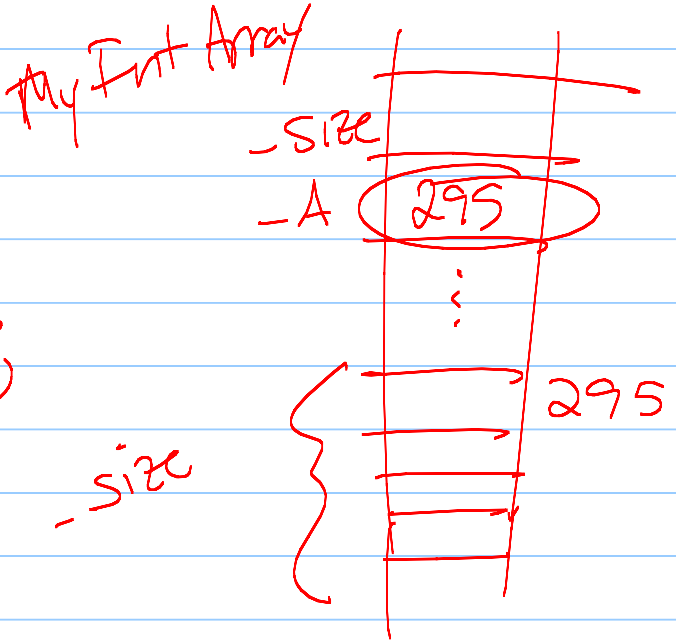
```
class   MyIntArray {
   private:
      int   _size;     // size of this array
      int*  _A;        // pointer to our array


   public:
      MyIntArray( int   s = 10) : _size(s) {
         // need to create array
         _A = new int[_size];
   }
```

Accessing the array: (*d)

With an array, can just pretend the variable isn't a pointer.
(so no * or ->)

Ex:

```
_A[0] = 12;
_A[_size-1] = 1;
```

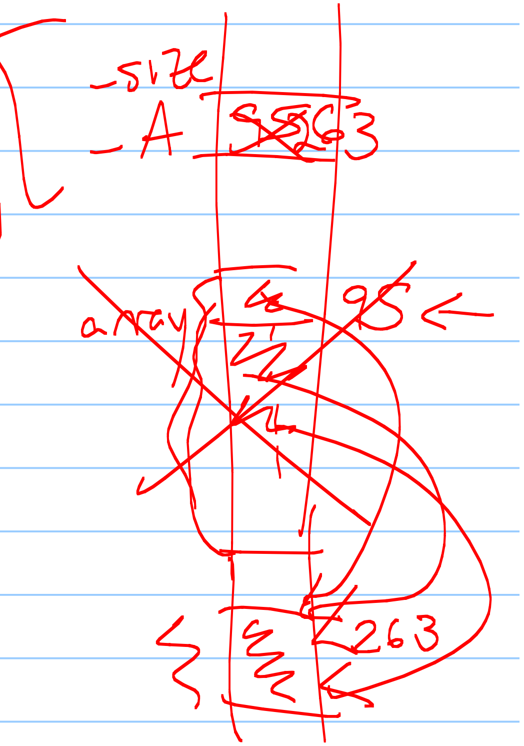


myArray.resize(50);

This lets you delay creating the array!

Also, if you need to change size:

```
void resize(int newsize) {
  int* newarray = new int[newsize];
  //assume newsize > _size
  for (int i=0; i< _size; i++)
    newarray[i] = _A[i];
  delete _A;
  _A = newarray;
  _size = newsize;
  // need to delete old array
}
```



Variables (recap)

1. Value – standard

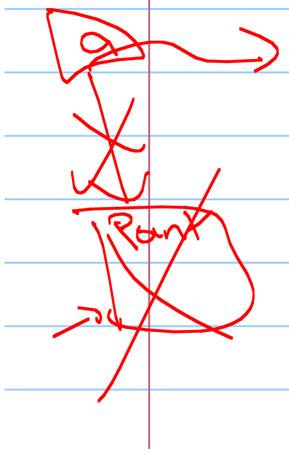
2. Reference – alias (usually used in function passing)

3. Pointer - just a memory address

Garbage Collection

In Python, variables that are no longer in use are automatically destroyed.

Pros: Easy (not our problem)



Cons: Less control
Slow

C++

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

Basically, any standard variable is automatically destroyed at the end of its scope.

This holds for <u>any</u> type of variable!

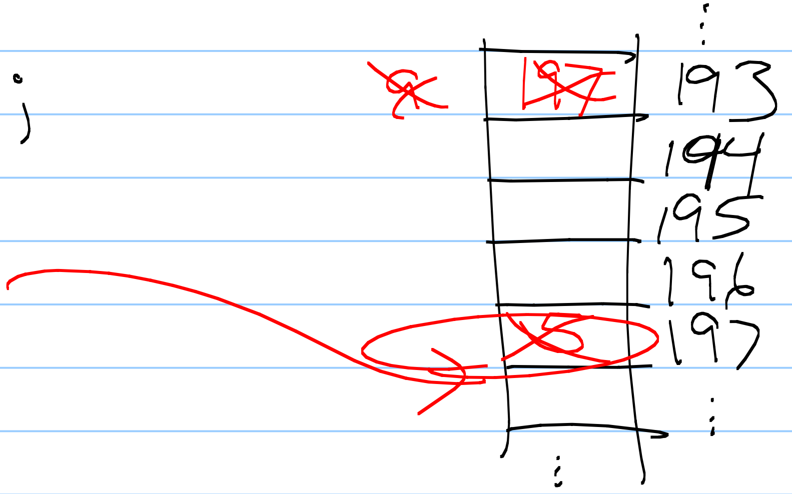
```
int main {
    int a;


} //a is destroyed
```

Problem: Pointers

While the pointer variable is deleted, the spot you created with a "new" is not.

```
main {
  int * a = new int(5);

  delete a;
  cout << *a
       ↑ seg fault
} // a is destroyed
```

memory leak



Rule: If you have a new, must have a delete!

Destructors

If your class opens files or allocates memory, then you must have a destructor.

~ClassName() ← no inputs
no return type

Ex:
```
~MyIntArray() {
    delete _A;
}
```

```
int main {
    MyIntArray Q;
    ⋮
} // Q is destroyed
```

Copy Constructor

Consider that MyIntArray class.

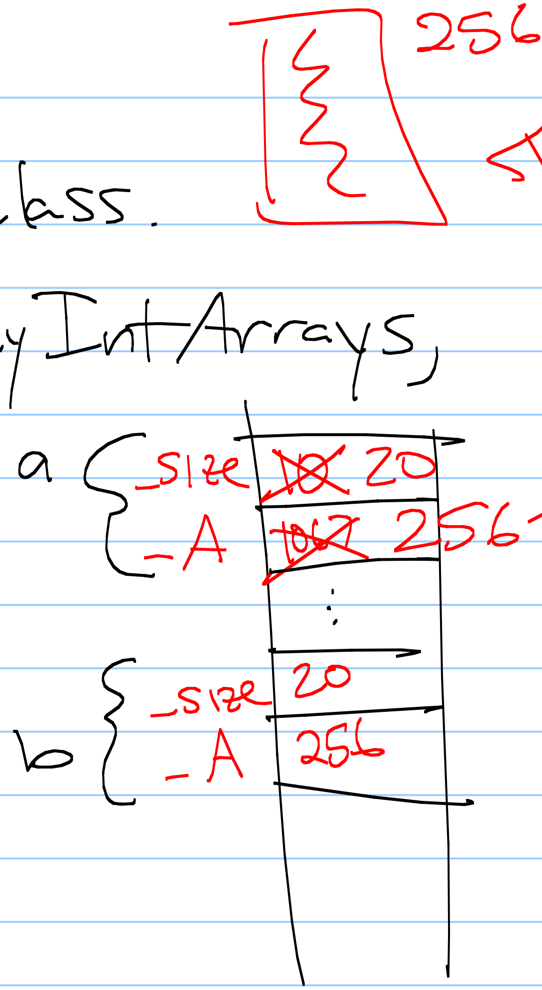
What if we have 2 MyIntArrays, & set a=b?

By default, compiler sets each private variable equal to other.

a.size = b.size

a._A = b._A

Shallow copy



To avoid shallow copies, we need to make a copy constructor function.

```
MyIntArray(const MyIntArray& other) {

}
```