CS180 - Pointers & garbage collection

Note Title 9/6/2013

Announcements

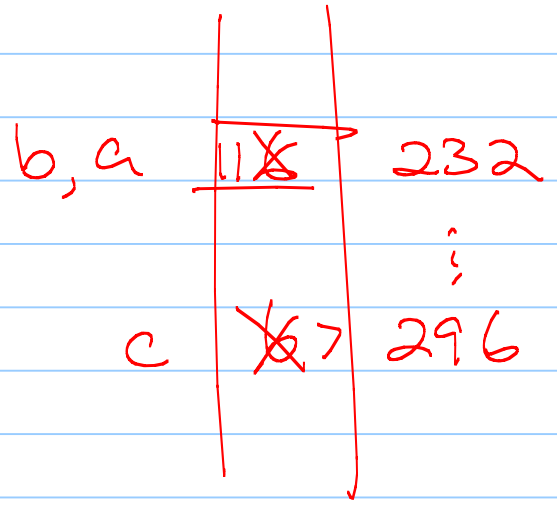
- HW2 due in 1 week
- No office hours today
  - move to Wed. at 2pm
- Thursday - no office hours at 9am
  - stay after 11 for office hours

Last Time

Variables

→ references &
- alias

```
→ int a = 6;
→ int& b(a);
→ int  c(a);
  c = 7;
  b = 11;
```



(3) Pointer variables

Syntax: `int * d;`

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

Ex:

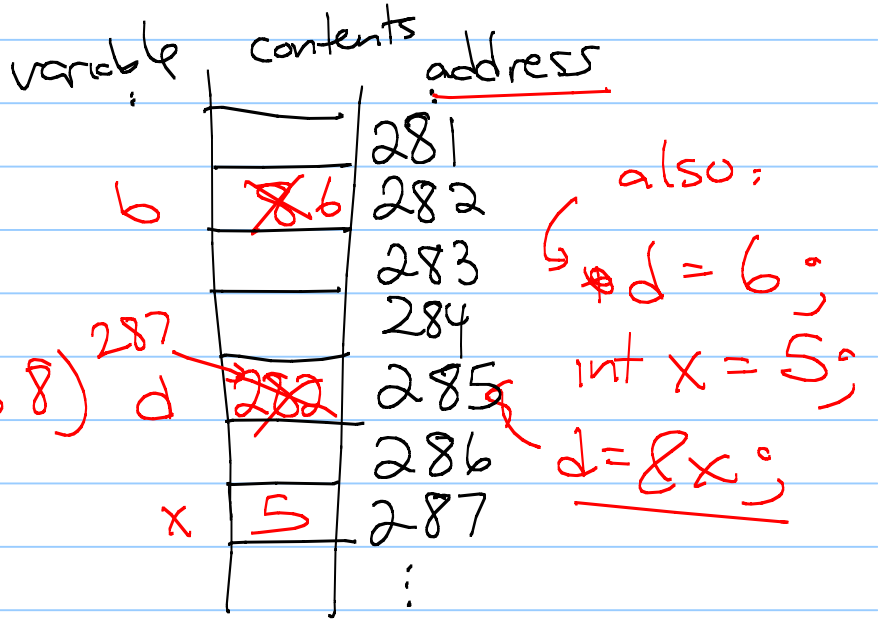
```
int b(8);
int* d;
d = &b;
```

(give me address)

`cout << *d;` (output is 8)

But d is not an int.

Can't write d=b!



Pointers: getting to the data

Called dereferencing.

Ex: Point * d;

→ Point b(3,5);

d = &b;

cout << d;

2 options:

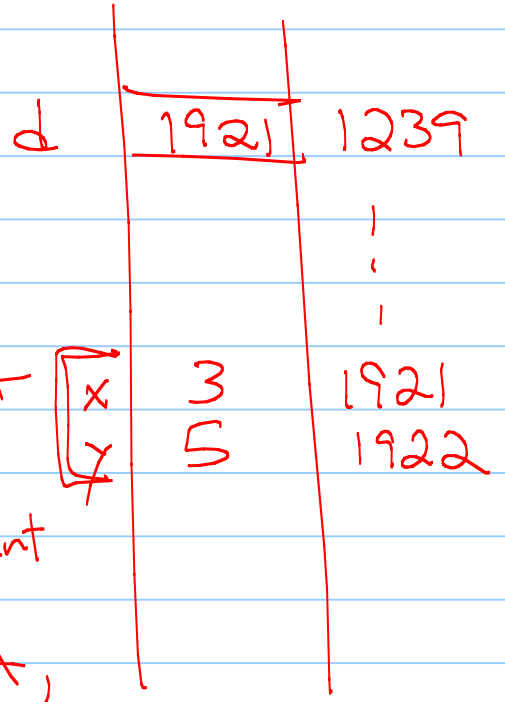
b.getX();

(*d).getX();

or

d -> getY();

return class, instance of Point so now all functions, cout, etc.



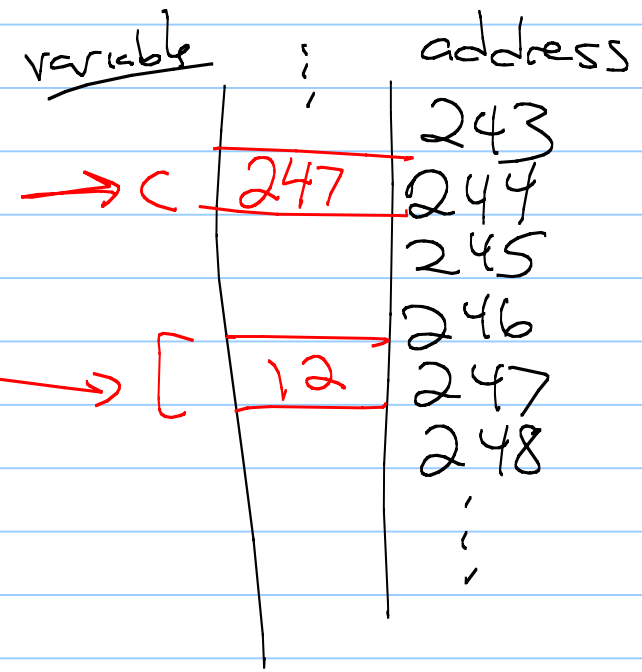
The new command

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

} // c is destroyed
```

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

So can create or modify inside multiple functions.



Passing pointers

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

= NULL

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

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!

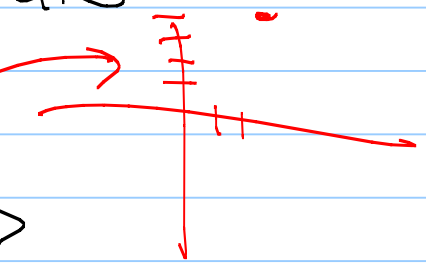
What do we need when creating an array?

Size, type: int myArray[60];

Example class: vector of floats

A vector in $\mathbb{R}^2$: $\langle 2, 5 \rangle$

A vector in $\mathbb{R}^4$: $\langle 0, 0, 0, 1 \rangle$

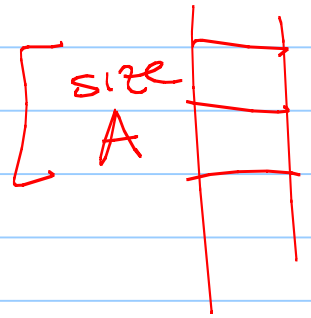


Dynamic size!

So how to make a class?

```
private:
    int _size;
    float* A;
```

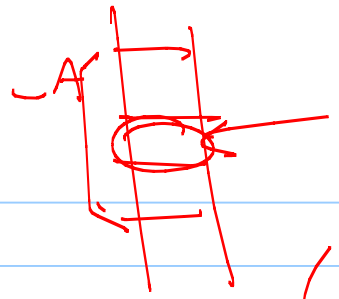
vector



```
class MyFloatVec {
    private:
        int _size;      // size of this array
        float* _A;      // pointer to my array

    public:
        MyFloatVec ( int s=10 ) :_size(s) {
            _A = new float [_size];
        }
        :
```

Accessing the array:



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

allow [] notation:

```
float& operator[](int x) {
    return _A[x];
}
```

```
myFloatVec pt(3);
pt[0] = 2;
pt = 6*pt;
float x = pt[1];
```

pt * 6;

Function to scale by int (in class):

```
void operator*(int x) {
    for (int i=0; i<_size; i++)
        _A[i] = x * _A[i];

}
```

Garbage Collection

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

Pros: space, easy

Cons: speed

C++

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

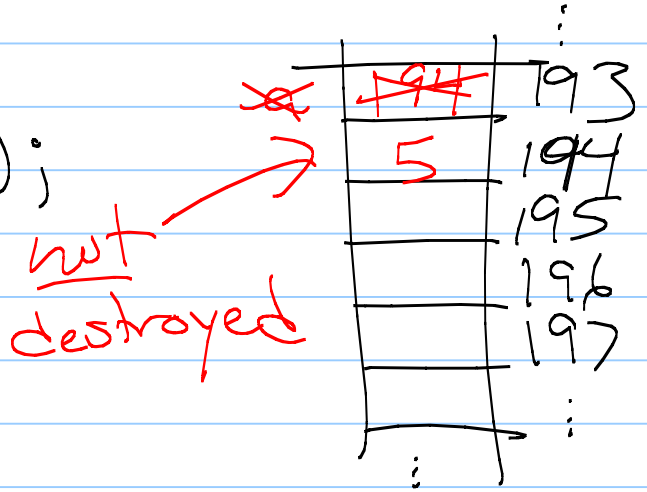
Basically, any standard variable is automatically destroyed at the end of its scope.

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

Problem: Pointers

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

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

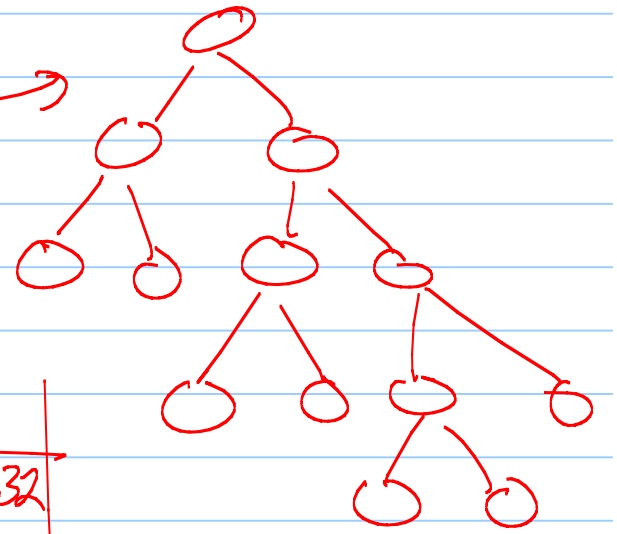



```
} // a is destroyed
```

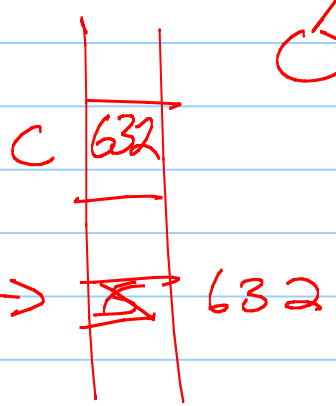
Rule: If you have a new, must have a delete!

If not destroyed, called a memory leak.

can be large



```
int * c = new int(5);
   :
   ;
delete c;
```



Next time:

Code myFloatVec

- create destructor to delete that array.