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;
```

Give me address:

d = &b;

cout << *d; (output is 8) d 287

But d is not an int.

Can't write d = b!

<table>
<thead>
<tr>
<th>Variable</th>
<th>Contents</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>8</td>
<td>281</td>
</tr>
<tr>
<td>d</td>
<td></td>
<td>282</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>283</td>
</tr>
<tr>
<td></td>
<td></td>
<td>284</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>285</td>
</tr>
<tr>
<td>x</td>
<td>5</td>
<td>286</td>
</tr>
<tr>
<td></td>
<td></td>
<td>287</td>
</tr>
</tbody>
</table>
Pointers: getting to the data

Called dereferencing.

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

2 options:

6. get x();
   (*d). get X();

or

return class so now call functions, cout, etc.

Point

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1921</td>
<td>1922</td>
</tr>
<tr>
<td>1239</td>
<td></td>
</tr>
</tbody>
</table>
The new command

```c
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:

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

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 \( \mathbb{R}^2 \): \( <2, 5> \)

A vector in \( \mathbb{R}^4 \): \( <90, 0, 0, 1> \)

Dynamic size!

So how to make a class?

private:

\[
\text{int } \text{size}; \quad \text{vector } \left[ \begin{array}{c} \text{size} \end{array} \right] \quad \text{float* } \text{A} \quad \text{vector } \left[ \begin{array}{c} \text{A} \end{array} \right]
\]
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.

(since no * or ->)

allow [] notation:

```cpp
float8 operator[](int x) {
    return A[x];
}
```

```cpp
my Float Vec pt(3);
pt[0] = 2.0;
pt = 6 * pt;
float x = pt[1];
```
Function to scale by int (in class):

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

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

This holds for any type of variable!
Problem: Pointers

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

```c
int main() {
    int *a = new int(5);
    // a is destroyed
    delete a;
}
```

Rule: If you have a `new`, you must have a `delete`.
If not destroyed, called a memory leak.

Can be large

```cpp
int * c = new int(5);
```

```cpp
delete c;
```

```cpp
delete c;
```
Next time:

- code my FloatVec
  
  - create destructor
to delete that array.