3 things are almost always needed when writing a class that manages dynamic memory:

① copy constructor
② assignment operator
③ destructor

Lets start w/ ②.

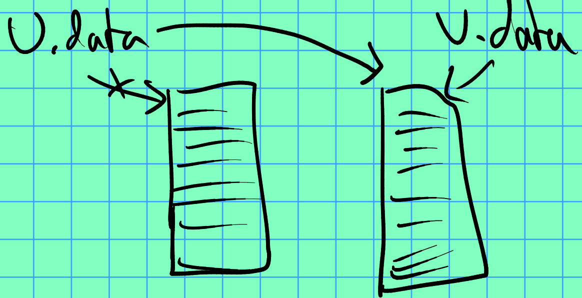
Q: what does this do?

```
vector2 V;
   :
vector U;
   :
U = V;
```

A: member-wise assignment:

```
U.size     = V.size;
U.capacity = V.capacity;
U.data     = V.data;
```

← this is the problem...



Issues: old U.data is lost & can't be deleted; V.data will be deleted twice (but usually the second attempt will crash the program with "double free" error from libc).

Similar issue arises with the copy constructor.

Btw, copy constructors look like this:

```
vector2::vector2(const vector2& v);
```

It makes copies of other vector2's. ← must be by ref!

Example usages:

```
vector2 v;
    ; //set up v...

vector2 u(v); // calls copy const.
```

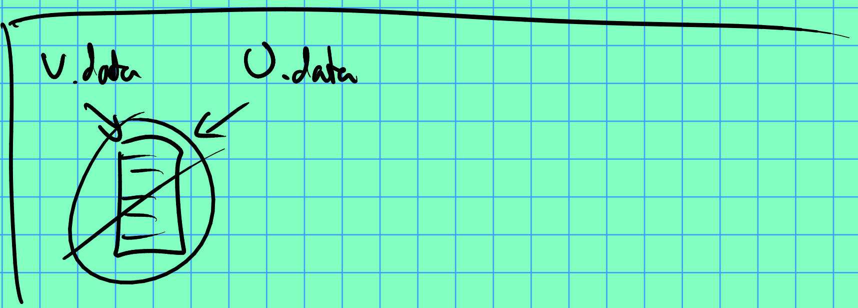
Also called to initialize parameters in a by-value function call. (This is why the parameter for the copy const. must be by reference.)

One more example of why you need a copy const.:
say we had a function like this:

```
void f(vector2 u) {
      // ... stuff ...

} // destructor for u called here.
```

```
int main() {
    vector2 v;
     // set up v...
    f(v);
    v.push_back(10);   ←
      :
```



x_x v's array was deleted by u's destructor.

Solution: make "deep copies"...