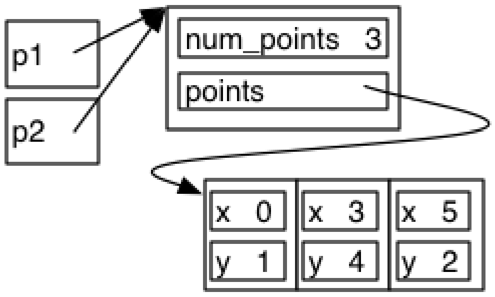
Shallow vs. Deep Copying

coursera.org/learn/interacting-system-managing-memory/supplement/beo60/shallow-vs-deep-copying

Suppose we had a **polygon_t * p1**pointing at a polygon we created (*e.g.*, by calling **makeRectangle**), and we wanted to make a copy of the polygon it points to. If we just wrote the following, we would only copy the pointer—we would not actually copy the object it points to:

```
polygon_t * p2 = p1; //just copy pointer
```

The figure below illustrates the (hopefully now familiar) effects of this statement. After assigning the pointer **p1** to the pointer **p2**, both point at the exact same memory location. The only new box that was created was the box for the pointer **p2**, and if we change anything about the polygon through one pointer, we will see the change if we examine the values through the other pointer—since they point at the same values.

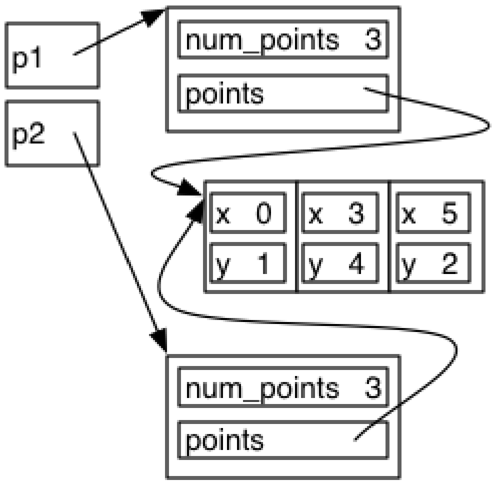


If we were to use malloc, we could create a copy.
For example, we might write:

```
*p2 = *p;
polygon_t * p2 = malloc(sizeof(*p2));
```

The figure below illustrates the effects of this piece of code.

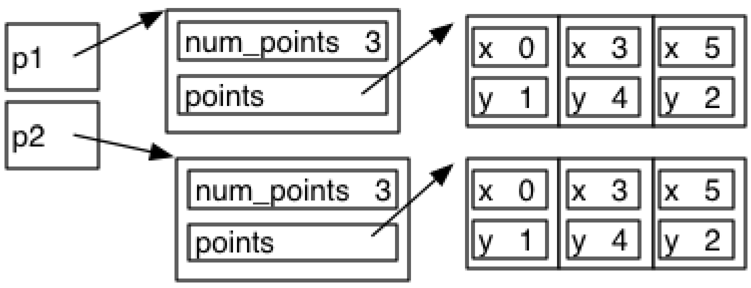
Now, we have two polygons, but only *one* array of points. We have created a *shallow copy* of the polygon—we have made a copy of the polygon, by exactly copying the fields inside of it. Each pointer in the shallow copy points at exactly the same location in memory as the corresponding pointer in the original. In some cases, a shallow copy may be what we want. However, if we did **p1->points[0].x = 42**; we would change thexof **p2**'s 0 point, since **p1->points** points at the same memory as **p2->points**. Notice that we must also be careful when freeing an object which has had a shallow copy made of it—we need to take care to only free the array of points when we are done with both shallow copies. As they share the memory, if we free the points array when we are done with one, then try to use the other copy, it will have a dangling pointer.



If we want two completely distinct polygon objects, we want to make a *deep copy*—in which we do not just copy pointers, but instead, allocate new memory for and make deep copies of whatever the pointers point to. To make a deep copy of our *polygon_t*, we would write:

```c
polygon_t * p2 = malloc(sizeof(*p2));
p2->num_points = p1->num_points;
p2->points = malloc(p2->num_points * sizeof(*p2->points));
for (size_t i = 0; i < p2->num_points; i++) {
  p2->points[i] = p1->points[i];
}
```

The figure below illustrates the effects of the deep copy from this fragment of code (which again, should not be surprising—as always, it follows the same rules you have learned).



Here we have created two completely distinct polygons which have the same values for their points, but their own memory. If we change the x or y of one polygon's points, the other remains unaffected, as they are two completely distinct data structures. Similarly, we can now completely free one polygon when we are done with it without worrying about anything else sharing its internal structures.

✓

Completed