More Complex Structures

coursera.org/learn/interacting-system-managing-memory/supplement/2Vcg6/more-complex-structures

Even though our examples so far have shown mallocing arrays of ints, we can, of course, malloc any type that we want. We can malloc one of a thing if we need (rather than an array) or any number of things (as memory permits). We can form complex data structure in the heap by mallocing structs which have pointers, and then setting those to point at other locations in the heap (which themselves point at blocks of memory allocated by malloc). We state this explicitly because it is important; however, you could also realize that all of this is possible due to the principle of composability —we can put all these different pieces together to make larger things.

For example, we might write

```c
struct point_tag {
  int x;
  int y;
};
typedef struct point_tag point_t;

struct polygon_tag {
  size_t num_points;
  point_t * points;
};
typedef struct polygon_tag polygon_t;

polygon_t * makeRectangle(point_t c1, point_t c2) {
  polygon_t * answer = malloc(sizeof(*answer));
  answer->num_points = 4;
  answer->points = malloc(answer->num_points * sizeof(*answer->points));
  answer->points[0]   = c1;
  answer->points[1].x = c1.x;
  answer->points[1].y = c2.y;
  answer->points[2]   = c2;
  answer->points[3].x = c2.x;
  answer->points[3].y = c1.y;
  return answer;

}
```

Here, we have a function that mallocs space for a polygon (one struct), which itself has a pointer to an array of points. This particular function makes a rectangle, so it mallocs space for 4 points and fills them in before returning its answer to the caller.

✓

Completed