Lab 8
Helpful Tips
point_array.h -> 2 structures

typedef struct point
{
    double x, y, z;
} point_t;

typedef struct
{
    // number of points in the array
    size_t len;

    // pointer to an array of point_t structs
    // There is space for 'allocated' structs, but
    // 'len' structs have been used so far.
    point_t* points;

    // to be discussed in class - see Demo
    size_t reserved;
} point_array_t;

point_array_t
len
points
reserved

array of "len" point_t

point_t
x
y
z
point_array.h - 4 functions

/* ALL THESE FUNCTIONS REQUIRE A VALID POINT_ARRAY_T POINTER AS THEIR FIRST PARAMETER. THEY SHOULD FAIL ON ASSERTION IF THIS POINTER IS NULL */

/* TASK 1 */

// Safely initialize an empty array structure.
void point_array_init( point_array_t* pa );

/* TASK 2 */

// Resets the array to be empty, freeing any memory allocated if necessary.
void point_array_reset( point_array_t* pa );

/* TASK 3 */

// Append a point to the end of an array. If successful, return 0, else return 1.
int point_array_append( point_array_t* pa, point_t* p );

/* TASK 4 */

// Remove the point at index i from the array, reducing the number of elements stored in the array by one. The order of points in the array may change. If successful, return 0, else return 1.
int point_array_remove( point_array_t* pa, unsigned int i );
len versus reserved?

- The idea of Lab 8 is for you to implement these 4 functions using 2 different Memory Allocation Strategies.
  - These Memory Allocation Strategies are described in details in Lab 8 Demo.
- In Task 1, you are to use ...
  - `len` -> to represent the number of points in the array as well as the size of allocated (or reallocated) memory for the array of points.
- In Task 2, you are to use ...
  - `len` -> to represent the number of points in the array
  - `reserved` -> to represent the size of allocated (or reallocated) memory for the array of points.
Compiling and testing our `t1.c` and `t2.c`

- Use `demo.c` as a testing program (test driver)
  - You may have to tweak it first
- Compile and test your code before submitting it to your Git repo
assert( ) and free( )

- You may want to investigate the function assert( )
  - How it works
  - What it returns
  - Use it to check the validity of functions’ parameters

- When we call free( aPtr ), let’s make sure we set aPtr to NULL:

```c
free( aPtr );
aPtr = NULL;
```