York University
Electrical Engineering and Computer Science

EECS2031: Software Tools
SU2016
Assignment #9

Chapter 16: Exercises

3. (a) Show how to declare a tag named `complex` for a structure with two members, `real` and `imaginary`, of type `double`.

   (b) Use the `complex` tag to declare variables named `c1`, `c2`, and `c3`.

   (c) Write a function named `make_complex` that stores its two arguments (both of type `double`) in a `complex` structure, then returns the structure.

   (d) Write a function named `add_complex` that adds the corresponding members of its arguments (both `complex` structures), then returns the result (another `complex` structure).

5. Write the following functions, assuming that the `date` structure contains three members: `month`, `day`, and `year` (all of type `int`).

   (a) `int day_of_year(struct date d);`

   Returns the day of the year (an integer between 1 and 366) that corresponds to the date `d`.

   (b) `int compare_dates(struct date d1, struct date d2);`

   Returns –1 if `d1` is an earlier date than `d2`, +1 if `d1` is a later date than `d2`, and 0 if `d1` and `d2` are the same.

10. The following structures are designed to store information about objects on a graphics screen:

```
struct point { int x, y; };
struct rectangle { struct point upper_left, lower_right; };
```

A `point` structure stores the *x* and *y* coordinates of a point on the screen. A `rectangle` structure stores the coordinates of the upper left and lower right corners of a rectangle. Write functions that perform the following operations on a `rectangle` structure `r` passed as an argument:

(a) Compute the area of `r`.
(b) Compute the center of `r`, returning it as a `point` value. If either the *x* or *y* coordinate of the center isn't an integer, store its truncated value in the `point` structure.
(c) Move `r` by `x` units in the *x* direction and `y` units in the *y* direction, returning the modified version of `r`. (`x` and `y` are additional arguments to the function.)
(d) Determine whether a point `p` lies within `r`, returning `true` or `false`. (`p` is an additional argument of type `struct point`.)

Chapter 16: Programming Projects

2. Modify the `inventory.c` program of Section 16.3 so that the `p` (print) operation displays the parts sorted by part number.

Chapter 17: Exercises

1. Having to check the return value of `malloc` (or any other memory allocation function) each time we call it can be an annoyance. Write a function named `my_malloc` that serves as a "wrapper" for `malloc`. When we call `my_malloc` and ask it to allocate `n` bytes, it in turn calls `malloc`, tests to make sure that `malloc` doesn't return a null pointer, and then returns the pointer from `malloc`. Have `my_malloc` print an error message and terminate the program if `malloc` returns a null pointer.

3. Write the following function:

```
int *create_array(int n, int initial_value);
```

The function should return a pointer to a dynamically allocated `int` array with `n` members, each of which is initialized to `initial_value`. The return value should be `NULL` if the array can't be allocated.

Chapter 17: Programming Projects

1. Modify the `inventory.c` program of Section 16.3 so that the `inventory` array is allocated dynamically and later reallocated when it fills up. Use `malloc` initially to allocate enough space for an array of 10 `part` structures. When the array has no more room for new parts, use `realloc` to double its size. Repeat the doubling step each time the array becomes full.