8. Let `color` be the following structure:

```
struct color {
  int red;
  int green;
  int blue;
};
```

(a) Write a declaration for a `const` variable named `MAGENTA` of type `struct color` whose members have the values 255, 0, and 255, respectively.

(b) (C99) Repeat part (a), but use a designated initializer that doesn't specify the value of `green`, allowing it to default to 0.

9. Write the following functions. (The `color` structure is defined in Exercise 8.)

(a) `struct color make_color(int red, int green, int blue);`

Returns a `color` structure containing the specified red, green, and blue values. If any argument is less than zero, the corresponding member of the structure will contain zero instead. If any argument is greater than 255, the corresponding member of the structure will contain 255.

(b) `int getRed(struct color c);`

Returns the value of `c`'s `red` member.

(c) `bool equal_color(struct color color1, struct color color2);`

Returns `true` if the corresponding members of `color1` and `color2` are equal.

(d) `struct color brighter(struct color c);`

Returns a `color` structure that represents a brighter version of the color `c`. The structure is identical to `c`, except that each member has been divided by 0.7 (with the result truncated to an integer). However, there are three special cases: (1) If all members of `c` are zero, the function returns a color whose members all have the value 3. (2) If any member of `c` is greater than 0 but less than 3, it is replaced by 3 before the division by 0.7. (3) If dividing by 0.7 causes a member to exceed 255, it is reduced to 255.

(e) `struct color darker(struct color c);`

Returns a `color` structure that represents a darker version of the color `c`. The structure is identical to `c`, except that each member has been multiplied by 0.7 (with the result truncated to an integer).

Section 16.3

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`.)