Computer Programming: Skills and Concepts
Tutorial 3 (Tue 17 Oct – Fri 20 Oct)

Descartes

In the current Labsheet we are working with the `descartes` library of graphics functions, and applying them to solve some simple drawing tasks. In this question we will consider a simple *segment-drawing* task, and think about the different ways we can solve it.

The task is to draw a line segment between positions `(40, 300)` and `(400, 100)`. You are given the starting template below which includes some variable declarations. You should:

(i) write code which uses the `descartes` functions to set up values for `q, r` and `seg`, and then draws the line segment (between `(40, 300)` and `(400, 100)`) on the graphics window.

(ii) re-write your code so that you draw the line segment between `(40, 300)` and `(400, 100)` on the graphics window without using `q, r, seg` or any other variables.

```
#include <stdio.h>
#include <stdlib.h>
#include "descartes.h"

int main (void) {
  lineSeg_t seg;
  point_t q, r;
  OpenGraphics();

  ...

  CloseGraphics();
  return EXIT_SUCCESS;
}
```

Descartes 2

In the header file `descartes.h` we are given a type declaration for the structured type `rectangle_t`. We are also given declarations for the functions `Rectangle`, `BottomLeft` and `TopRight`. All of these functions *require* that the first `point_t` making the rectangle is the "bottom left" corner and that the second `point_t` making the rectangle is the "top right" corner. However, if the pair of input points passed to `Rectangle` does *not* satisfy this requirement, `Rectangle` will print an error message and move the 'top right' point until it is above and right of the 'bottom left'.

In this question you are asked to design a function `IsRectangleOK` which will check whether the implied rectangle does satisfy this assumption (returning 1 if it does, and 0 otherwise). The order in which the points are passed to `IsRectangleOK` should influence the result. The function prototype for `IsRectangleOK` is:

```
int IsRectangleOK(point_t, point_t);
```

You are also asked to implement a second function `Rectangle2` which will take two points of type `point_t`, and return an object of type `rectangle_t` which defines the *exactly* same rectangle as the input points, but which satisfies the "bottom left"/"top right" assumptions. The function prototype for this function is:

```
rectangle_t Rectangle2(point_t, point_t);
```

Note that the bottom-left position of graphics window is `(0,0)`. You will need to use the `descartes` library to implement the functions.

In testing your code, it will be helpful to consider some examples:

- If we consider the two points `(40, 300), (400, 100)` in *either* order, the call to `IsRectangleOK` should return 0.

  After calling `Rectangle2` with these two points (in either order), the rectangle returned should have corner `(40, 100)` as "bottom left" and corner `(400, 300)` as "top right".

  If we *now* call `IsRectangleOK` on this "bottom left", "top right", it should return 1.

- If we consider the two points `(400, 100), (50, 50)` in *this* order, the call to `IsRectangleOK` should return 0.

  After calling `Rectangle2` with these two points (in either order), the rectangle returned should have corner `(50, 50)` as "bottom left" and corner `(400, 100)` as "top right".

  If we *now* call `IsRectangleOK` on this "bottom left", "top right", it should return 1.

Pointers

Consider the following code:

```
int a = 10;
int b = 5;
int *p = &a;
int *q = &b;
int c = *p;
c = c + 1;
*p = *p + 2;
int d = *p;
*p = *q;
int e = *p;
b = b + 5;
int f = b;
int g = *p;
int h = *q;
```

What are the values of `b`, `c`, `d`, `e`, `f`, `g`, and `h`? (use lecture 8 on pointers to help with this).