EECE.2160: ECE Application Programming

Spring 2017

Programming Assignment #8: Nested Structures

Due **Wednesday, 4/19/17**, 11:59:59 PM

1. Introduction

This assignment focuses on the use of structures. You will model a group of rectangles as a collection of their vertices, using structures to represent each of those points as well as the rectangles as a whole. In doing so, you will gain familiarity with the typical file organization used when defining structures, as well as the syntax required to work with structures nested inside one another.

2. Deliverables

This assignment uses multiple files, versions of which are on the course website:

- ***prog8_main.c:*** Source file containing your main function. ***THIS FILE SHOULD NOT BE MODIFIED.***
- ***Point.h:*** Header file containing definition of Point structure and prototypes of relevant functions.
- ***Point.c:*** Source file containing definitions of Point functions. *Replace the code in the starter file with your own code.*
- ***Rectangle.h:*** Header file containing definition of Rectangle structure and prototypes of relevant functions.
- ***Rectangle.c:*** Source file containing definitions of Rectangle functions. *Replace the code in the starter file with your own code.*

You are required to complete the function definitions in Point.c and Rectangle.c. You do not need to modify any other files—and absolutely should not modify the main() function—but you may add other functions in Point.h/Point.c or Rectangle.h/Rectangle.c if you deem them necessary.

Submit all five files (even the unchanged ones) by uploading them to your Dropbox folder. Ensure your file names match the names specified above. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

The main program (***prog8_main.c***) recognizes five different single-letter commands, most of which call functions described in *Point.h* or *Rectangle.h* and defined in the corresponding source file:

- **'A', 'a'**: Add a Rectangle to the array of Rectangles, which can contain up to 10 elements, using the **`readPoint()`** function to read all four vertices in clockwise fashion, starting with the lower left corner.
- **'P', 'p'**: Print the entire array of Rectangles using the **`printList()`** function.
- **'D', 'd'**: Prompts the user to enter an integer representing an index into the Rectangle array, then print the dimensions (area and perimeter) of that Rectangle, using the **`area()`** and **`perimeter()`** functions.
- **'O', 'o'**: Prompts the user to enter two integers representing indices into the Rectangle array, then checks to see if the two Rectangles overlap one another, using the **`overlap()`** function.
- **'Q', 'q'**: Exit the program.

Point.h contains the definition of the Point structure, which consists of two coordinates, x and y. It also contains prototypes for the following functions, which you must properly define in Point.c:

```
void printPoint(Point *p);
```

Print the x and y coordinates of the Point referenced by the pointer p using the following format: `(x.xx, y.yy)` (for example, `(3.14, -5.22)`)

```
void readPoint(Point *p);
```

Read x and y coordinates from the user input into the Point referenced by the pointer p. Note that this function does not have to prompt the user for coordinates.

```
double dist(Point *p1, Point *p2);
```

Return the distance between the Points referenced by pointers p1 and p2. The "distance formula" is fairly well known and can be found with a quick web search.

Rectangle.h contains the definition of the Rectangle structure, which consists of an array (**`vert[]`**) of four Point structures, defined as follows:

- `vert[0]` is the lower left corner of the Rectangle.
- `vert[1]` is the upper left corner of the Rectangle.
- `vert[2]` is the upper right corner of the Rectangle.
- `vert[3]` is the lower right corner of the Rectangle.

3. Specifications (continued)

Rectangle.h also contains prototypes for the following functions, which you must properly define in Rectangle.c:

```
void printRectangle(Rectangle *r);
```

Print the four vertices of the Rectangle in the appropriate relative positions. For example, given a 4 x 2 rectangle with lower left corner at (1, 3), this function would generate the following output:

```
(1.00, 5.00)    (5.00, 5.00)
(1.00, 3.00)    (5.00, 3.00)
```

```
void printList(Rectangle list[], int n);
```

Print an array of Rectangles containing n elements. This function should leave an extra blank line between Rectangles to easily distinguish one from the next.

```
double area(Rectangle *r);
```

Return the area of the Rectangle referenced by the pointer r.

```
double perimeter(Rectangle *r);
```

Return the perimeter of the Rectangle referenced by the pointer r.

```
int overlap(Rectangle *r1, Rectangle *r2);
```

Test the two Rectangles pointed to by r1 and r2 to see if they overlap with one another. Return 1 if the Rectangles overlap and 0 if they do not. (Hint: There are essentially four cases you have to account for, and it may help to draw a picture of each case before writing this function.)

4. Hints

Design process: I would suggest handling the program in the following order:

1. Write the Point functions first, in the order listed above. As with the nested structures we used in PE4 (Lecture 11), you'll call at least some of these functions inside the Rectangle functions. You may want to write a short, separate program that tests only the Point functions before moving on.
2. Next, write the Rectangle print functions (**`printRectangle()`** and **`printList()`**).
   a. Once these functions are done, you can use the main program to test them, using only 'A' and 'P' commands.
3. Next, write the **`area()`** and **`perimeter()`** functions.
   a. Test these functions by using the 'D' command in the main program.
4. Finally, write the **`overlap()`** function.
   a. Test this function using the 'O' command in the main program. Note that you'll need at least two Rectangles in your list to test it.
   b. Try testing each of the four overlap cases referenced on the previous page.

If you encounter errors, running your program in the debugger is the most effective way to find them. Recall that the debugger offers the ability to "step into" a function (F11 in Visual Studio) so that you can see each step within the function you have written, or simply "step over" (F10) the function and treat a function call as a single statement.

5. Test Cases

Your output should match these test cases exactly for the given input values. I will use these test cases in grading of your lab, but will also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.

The test cases were generated in Xcode, so I've copied and pasted the output below, rather than showing a screenshot of the output window. User input is underlined, although it won't be when you run the program.

```
Enter command <A | P | D | O | Q>: A
Enter coordinates as x y, starting with lower left hand corner:
0 0
0 3
4 3
4 0

Enter command <A | P | D | O | Q>: A
Enter coordinates as x y, starting with lower left hand corner:
1 1
1 5
5 5
5 1

Enter command <A | P | D | O | Q>: P
(0.00, 3.00)  (4.00, 3.00)
(0.00, 0.00)  (4.00, 0.00)

(1.00, 5.00)  (5.00, 5.00)
(1.00, 1.00)  (5.00, 1.00)


Enter command <A | P | D | O | Q>: D
Enter index into array: 1
Area of rectangle 1: 16.00
Perimeter of rectangle 1: 16.00

Enter command <A | P | D | O | Q>: D
Enter index into array: 0
Area of rectangle 0: 12.00
Perimeter of rectangle 0: 14.00

Enter command <A | P | D | O | Q>: O
Enter indices to test: 0 1
Rectangles 0 and 1 overlap
```

5. Test Cases (continued)

```
Enter command <A | P | D | O | Q>: A
Enter coordinates as x y, starting with lower left hand corner:
-1 -1
-1 0.5
0.5 0.5
0.5 -1

Enter command <A | P | D | O | Q>: P
(0.00, 3.00)  (4.00, 3.00)
(0.00, 0.00)  (4.00, 0.00)

(1.00, 5.00)  (5.00, 5.00)
(1.00, 1.00)  (5.00, 1.00)

(-1.00, 0.50)  (0.50, 0.50)
(-1.00, -1.00)  (0.50, -1.00)


Enter command <A | P | D | O | Q>: D
Enter index into array: 2
Area of rectangle 2: 2.25
Perimeter of rectangle 2: 6.00

Enter command <A | P | D | O | Q>: O
Enter indices to test: 0 2
Rectangles 0 and 2 overlap

Enter command <A | P | D | O | Q>: O
Enter indices to test: 1 2
Rectangles 1 and 2 do not overlap

Enter command <A | P | D | O | Q>: Q
Program ended with exit code: 0
```

```
Enter command <A | P | D | O | Q>: A
Enter coordinates as x y, starting with lower left hand corner:
-1 -1
-1 0.5
0.5 0.5
0.5 -1

Enter command <A | P | D | O | Q>: P
(0.00, 3.00)  (3.00, 4.00)
(0.00, 0.00)  (0.00, 4.00)

(1.00, 5.00)  (5.00, 5.00)
(1.00, 1.00)  (5.00, 1.00)

(-1.00, 0.50)  (0.50, 0.50)
(-1.00, -1.00)  (0.50, -1.00)


Enter command <A | P | D | O | Q>: O
Enter indices to test: 0 2
Rectangles 0 and 2 overlap

Enter command <A | P | D | O | Q>: O
Enter indices to test: 1 2
Rectangles 1 and 2 do not overlap

Enter command <A | P | D | O | Q>: Q
Program ended with exit code: 0
```