CIS 15 Spring 2010, Assignment III

Instructions

- This is the assignment for Unit III.
- It is worth 10 points.
- **It is due on Sunday March 21st** and must be submitted by email (as below).
- **Follow these emailing instructions:**
  1. Create a mail message addressed to **parsons@sci.brooklyn.cuny.edu** with the subject line **CIS15 HW3**.
  2. Write your name, that is the name under which you registered for the course, in the email (when I get an email from `deathmetal@aol.com` or `pinkprincess@yahoo.com`, I can usually guess whose program it is, but that is not as good as *knowing* whose program it is).
  3. Attach ONLY the **.cpp** source code file created below.
  4. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions . . . (which can make it a lot harder for me to grade your work)

Program description

For this assignment, you will develop a program that does some elementary graphics.

The main object that the program will manipulate will be a `display`. You can think of this as being the window that your program will display, the canvas on which your program will draw. The `display` also contains some objects — for simplicity they will all be rectangles — and the user can specify how many of these they want to enter, and then give their position within the display.

The intermediary test programs are for your benefit as a developer and should not be submitted. These are called *unit tests* and are created to let you test and debug the individual units of a complex program. By the end of the semester, you should have gained the skills and experience to devise and construct your own unit tests, without me giving you step-by-step instructions.

A. Create and test a `point` class

1. Create a `point` class.

   The class should have two `int` data members: x and y. The class should have four function members:

   - `void print() const;`
     This function prints out the values of the data members x and y.
   - `void set(int u, int v);`
     This function sets the values of the data members x and y.
   - `int getX()`
     This function returns the value of the x data member.
   - `int getY()`
     This function returns the value of the y data member.

2. Create a `main()` to test this class. This should read in two values from the user, store them in an instance of the `point` class, and then print them out.

B. Create a `rectangle` class

1. Create a class called `rectangle` that has four data member and eight function members (two of which are constructors):
   - the four data members are `point` objects, called `tl`, `tr`, `bl` and `br` which hold the top left, top right, bottom left and bottom right corners of the rectangle.
   - The first constructor sets all four corners to (0, 0).
   - The second constructor takes two arguments that are `point` objects and uses them to initialise the top left and bottom right corners.
     The function then initialises the other corners using the `x` and `y` values of the two arguments.
   - The next function member is `void print() const;`
     This function prints out the location of the four corners of the rectangle.
   - Then we have a function set `void set(point u, point v);` which, just like the constructor, initialises all four corners from the top left and bottom right corners.
   - The remaining functions `point getTL()`, `point getTR()`, `point getBL()`, `point getBR()` return the respective corner of the rectangle.
2. Modify the `main()` that you created in part A to instantiate a `rectangle` object and then to take values from the user for the coordinates of the top-left and bottom-right corners, use these values to set the `x` and `y` values of the `members` and then print the four corners of the rectangle out.

C. Create and test a `display` class

1. Create a class called `display` that has three data members and five function members (the usual access functions):
   - The first data member is a `rectangle` object called `frame`.
   - The second data member is an integer called `number`. This holds the number of rectangles that are part of the display.
   - The third data member is a pointer to objects of type `rectangle`. Let's call this member `rects`.
   - The first function member is the function `void setFrame(rectangle)` which takes a `rectangle` as an argument and sets `frame` to be that rectangle.
   - The second function member is `rectangle getFrame(void)`. This returns the value of `frame`.
   - The third function member is `void setNumber(int)`. This sets the value of `number`.
   - The fourth function member is `void addRectangles(rectangle*)`. It sets the value of the pointer `rects`.
   - The final function is `void print(void)`. This prints out the coordinates of all four corners of the frame and each rectangle in the display.
2. Modify the `main()` that you created in part C so that it creates a display, sets the frame of the display, has the user enter the number of rectangles they want in the display, has the user enter the coordinates of the top-left and bottom-right corners of each rectangle, passes these to the display object, and the prints the display object.

D. The function `inFrame()`

1. Write a function `bool inFrame(rectangle r1, rectangle r2)` which returns `true` if `r2` is inside `r1`.
   "Inside" here means that all four corner points of `r2` fall within the four corners of `r1`.
2. Modify the `main()` you wrote in the previous part so that each time the user enters a rectangle, `inFrame` is used to check that it is inside the frame of the display.
   If any part of the entered rectangle sticks out of the frame, print an error message and get the user to re-enter the rectangle.

E. The function `render()`

1. Write a member function of the `display` class called `render` that prints out the display on the screen.

   (a) The frame should be rendered so that each point in the frame is represented by a character. The frame should thus be represented by 15 lines each of 40 characters.

   (b) Each rectangle should be rendered by printing a `1` for every point in the rectangle. A four by four square should thus be rendered:

   ```
   1111
   1111
   1111
   1111
   ```

   (c) Print a `0` for every point in the frame that is not covered by a rectangle. Thus the result of a `render` should look like:

   ```
   0000000000000000000000000000000000000000
   0000000000000000000000000000000000000000
   0000000000111111111111111100000000000000
   0000000000111111111111111100000000000000
   0000000000111111111111111100000000000000
   0000000000111111111111111100000000000000
   0000000000111111111111111100000000000000
   0000000000111111111111111100000000000000
   0000000000000000000000000000000000000000
   0000000000000000000000000000000000000000
   0000000000000000000000000000000000000000
   0111111100000000000000000000000000000000
   0111111100000000000000000000000000000000
   0111111100000000000000000000000000000000
   0111111100000000000000000000000000000000
   0000000000000000000000000000000000000000
   ```

2. Modify the `main()` you wrote in the previous part so that after all the rectangles are entered, the display is rendered.

F. `main()`

The final version of `main()` should contain the following:

1. A display object is declared, and the frame of the display is set to have a top left corner at (0, 15) and a bottom right corner at (40, 0).
2. The user is asked to enter the number of rectangles that will be in the display.
3. An array is declared that will hold the rectangles.
4. The coordinates of the rectangles are entered by the user, and they are checked to make sure they fall within the frame of the diaply.

   If any rectangle doesn't fall within the frame of the display, the user has to re-enter it.
5. The data member `rects` of the display is set to point to the array of rectangles.
6. The display is rendered, showing the rectangles and the display.

G. For additional credit

You will get additional credit for any of the following that you complete:

1. Modify `main()` so that the user has the option of having the program reads in the coordinates of the rectangles from a file.

   If you modify your program in this way, include a file of coordinates in the right format to work with your program.

2. Add an attribute `color` to `rectangle`, and render different colored rectangles in different ways.

   If you add this functionality to your program, you should modify `main()` so that is demonstrates what you have done.

3. Add a function `move` to `rectangle` which moves the rectangle relative to the frame of the display.

   If you add this functionality to your program, you should modify `main()` so that is demonstrates what you have done.

H. Marking rubric

This assignment is worth 10 points. The breakdown is as follows:

- `point` class with two data members (x and y) and four function members (print(), set(), getX() and getY())

  (1 points)

- `rectangle` class with four data members and eight function members.

  (2 points)

- `display` class with three data members, and the function members `print()` `setFrame`, `getFrame`, `addRectangles` and `setNumber`.

  (2 points)

- The function `inFrame` that tests if a given rectangle is completely inside another rectangle.

  (1 point)

- The member function `render` for the display class which prints out a representation of the frame and the rectangles in the frame..

  (2 points)

- A `main` that sets up a display object, sets the frame of the object, dynamically sets up an array of rectangles and passes this to the display, then renders the display.

  (2 points)

Note that I only want you to hand in the final `main()`. The intermediate ones are for your testing only.