

Exam 1: Scheme Fundamentals

Distributed: Friday, 21 September 2007

Due: 10:00 a.m., Friday, 28 September 2007

No extensions.

This page may be found online at

<http://www.cs.grinnell.edu/~rebelsky/Courses/CS151/2007F/Exams/exam.01.html>.

This exam is also available in PDF format.

Contents

- Preliminaries
- Problems
 - Problem 1: Some Filters
 - Problem 2: Approximating Colors
 - Problem 3: Drawing Blends
 - Problem 4: Drawing Circles
- Some Questions and Answers
- Errors

Preliminaries

There are four problems on the exam. Some problems have subproblems. Each problem is worth twenty-five (25) points. The point value associated with a problem does not necessarily correspond to the complexity of the problem or the time required to solve the problem.

This examination is open book, open notes, open mind, open computer, open Web. However, it is closed person. That means you should not talk to other people about the exam. Other than as restricted by that limitation, you should feel free to use all reasonable resources available to you. As always, you are expected to turn in your own work. If you find ideas in a book or on the Web, be sure to cite them appropriately.

Although you may use the Web for this exam, you may not post your answers to this examination on the Web (at least not until after I return exams to you). And, in case it's not clear, you may not ask others (in person, via email, via IM, by posting a "please help" message, or in any other way) to put answers on the Web.

This is a take-home examination. You may use any time or times you deem appropriate to complete the exam, provided you return it to me by the due date.

I expect that someone who has mastered the material and works at a moderate rate should have little trouble completing the exam in a reasonable amount of time. In particular, this exam is likely to take you about four to six hours, depending on how well you've learned topics and how fast you work. *You should not work more than eight hours on this exam. Stop at eight hours and write "There's more to life than CS" and you will earn at least 80 points on this exam.*

I would also appreciate it if you would write down the amount of time each problem takes. Each person who does so will earn two points of extra credit. Since I worry about the amount of time my exams take, I will give two points of extra credit to the first two people who honestly report that they've spent at least five hours on the exam or completed the exam. (At that point, I may then change the exam.)

You must include both of the following statements on the cover sheet of the examination.

1. I have neither received nor given inappropriate assistance on this examination.
2. I am not aware of any other students who have given or received inappropriate assistance on this examination.

Please sign and date each statement. Note that the statements must be true; if you are unable to sign either statement, please talk to me at your earliest convenience. You need not reveal the particulars of the dishonesty, simply that it happened. Note also that "inappropriate assistance" is assistance from (or to) anyone other than Professor Rebelsky (that's me) or Professor Davis.

Because different students may be taking the exam at different times, you are not permitted to discuss the exam with anyone until after I have returned it. If you must say something about the exam, you are allowed to say "This is among the hardest exams I have ever taken. If you don't start it early, you will have no chance of finishing the exam." You may also summarize these policies. You may not tell other students which problems you've finished. You may not tell other students how long you've spent on the exam.

You must present your exam to me in two forms: both physically and electronically. That is, you must write all of your answers using the computer, print them out, number the pages, put your name on the top of every page, and hand me the printed copy. You must also email me a copy of your exam. You should create the emailed version by copying the various parts of your exam and pasting them into an email message. In both cases, you should put your answers in the same order as the problems. Failure to name and number the printed pages will lead to a penalty of two points. Failure to turn in both versions may lead to a much worse penalty.

In many problems, I ask you to write code. Unless I specify otherwise in a problem, you should write working code and include examples that show that you've tested the code. Do not include images; I should be able to regenerate those.

Unless I explicitly ask you to document your procedures, you need not write introductory comments.

Just as you should be careful and precise when you write code and documentation, so should you be careful and precise when you write prose. Please check your spelling and grammar. Since I should be equally careful, the whole class will receive one point of extra credit for each error in spelling or grammar you identify on this exam. I will limit that form of extra credit to five points.

I will give partial credit for partially correct answers. You ensure the best possible grade for yourself by emphasizing your answer and including a *clear* set of work that you used to derive the answer.

I may not be available at the time you take the exam. If you feel that a question is badly worded or impossible to answer, note the problem you have observed and attempt to reword the question in such a way that it is answerable. If it's a reasonable hour (before 10 p.m. and after 8 a.m.), feel free to try to call me in the office (269-4410) or at home (236-7445).

I will also reserve time at the start of classes next week to discuss any general questions you have on the exam.

Problems

Problem 1: Some Filters

Topics: Simple computation, Colors, Images, Mapping, Filters

a. Write a procedure, `(image.flatten image base)` that flattens the given image by ensuring that every component is the nearest exact multiple of *base* which is less than or equal to 255.

Your definition should look something like the following:

```
(define flatten
  (lambda (image base)
    (image.map (lambda (color) ____
                image)))
```

b. Write a procedure, `(color.greyer color)`, that makes the given color a bit greyer by halving the difference of each component from the average component. For example, consider the color 100/200/30. The average of 100, 200, and 30 is 110. The difference of 110 and 100 is 10, so we add 5 (half of 10) to 100, giving 105. The difference of 110 and 200 is -90, so we subtract 45 (half of 90) from 200, giving us 155. The difference of 110 and 30 is 80, so we add 40 to 30, giving us 70.

As you write `color.greyer`, you may find the following procedure useful.

```
(define average-component
  (lambda (color)
    (/ (+ (rgb.red color) (rgb.green color) (rgb.blue color)) 3)))
```

c. Write a procedure, `(image.greyer image)`, that applies `color.greyer` to each pixel in *image*.

d. Write a procedure, `(color.enhance color)`, that does the opposite of `color.greyer`. That is, your procedure should double the difference of each component from the average component. For example, if we start with 105/155/70 (the result in the example above), we note that the average of the three values is 110. The difference of 110 and 105 is 5. Hence, to double that difference, we subtract 5 from 105, giving us 100. The difference of 110 and 155 is -45. Hence, to double that difference, we add 45 to 155, giving us 200. The difference of 110 and 70 is 40. To double that difference, we subtract 40 from 70, giving us 30. The result is therefore 100/200/30.

e. Write a procedure, (`image.enhance image`), that applies `color.enhance` to each pixel in `image`.

Problem 2: Approximating Colors

Topics: Predicates, Conditionals, Colors

In creating images, it is often useful to convert colors into approximations that use “nearby” colors. Why? Sometimes we want to describe the colors, and not every color has a name. Sometimes we want to use a more limited color spectrum. Sometimes we just want to experiment.

a. Write a procedure, (`rgb.distance color1 color2`) that computes a number that represents the difference between rgb colors `color1` and `color2`. A reasonable metric for distance between two colors is the sum of the absolute values of the differences of each component.

b. Write a procedure, (`rgb.closer2 color estimate1 estimate2`), that returns whichever of `estimate1` or `estimate2` is closer to `color`.

c. Write a procedure, (`rgb.closer4 color est1 est2 est3 est4`), that returns whichever of `est1`, `est2`, `est3`, or `est4` is closer to `color`. In this case, the body of the procedure should consist only of calls to `rgb.closer2`.

Problem 3: Drawing Blends

Topics: Colors, Blends, Mathematical Operations, Positions, Region Maps

In a variety of assignments and lab exercises, we have worked with a number of approaches to computing color blends. Let’s combine some of those ideas to write a procedure that creates and draws a color blend.

Write a procedure, (`image.fill-with-blend! image start-color end-color`) that, given an image, builds a horizontal blend from `start-color` to `end-color` that fills the image. That is, all the pixels in the first column of the image should be `start-color`, all the pixels in the last column should be `end-color`, and each pixel in between should be somewhere in between.

Note that you should use `region.compute-pixels!` to draw this blend. You should plan to test your new procedure on relatively small images (e.g., 17x3 or 10x10). Once the tests succeed, you might try a much larger image.

Problem 4: Drawing Circles

Topics: Smiley Faces, Drawing Shapes, Generalizing Code

As you may recall, in our initial experiments with drawing faces, we found it convenient to be able to say “draw a circle of *this radius* centered at *this point* in *this color*”. We now have enough tools to write a procedure that does just that. In particular, we can use `region.compute-pixels!` and a procedure that colors a pixel with the specified color if it falls within the circle, and with the special value `transparent` if the pixel does not fall within the circle.

a. Write a procedure, (`image.draw-circle! image radius column row color`), that draws a filled circle of radius *radius* and color *color* in *image*, centered at (*column,row*).

Hint: The lab on conditionals includes a few problems in which circles are drawn. You should find Exercise 5 and Extra 1 particularly helpful.

In case you've forgotten, the formula for a circle of radius *r* centered at (*x₀,y₀*) is $(x-x_0)^2+(y-y_0)^2=r^2$.

b. Write a sequence of calls to `image.draw-circle!` that will draw a smiley face in a 200x200 image.

Some Questions and Answers

These are some of the questions students have asked about the exam and my answers to those questions.

Problem 1

Can you please explain what you mean in the flatten problem?

Colors have three components: red, green, and blue. Each has a value between 0 and 255.

Problem 2

Problem 3

Problem 4

Could you remind me of how one computes the points in a circle?

I have added that formula to the test. (I didn't really get the question, but I thought I might.)

My code runs really slowly. Is it supposed to?

See if you can avoid using `sqrt` in your computations.

How do I draw "nothing" in the parts of the diagram that don't include the circle?

Use the color `transparent`.

Miscellaneous

Will you take off if we do not indent in the same way that you do?

I have not yet discussed proper indentation style, so I don't consider it appropriate to take off points if you don't follow an unstated style. But don't worry, you'll get guidelines and criticisms soon.

What's the best way to create a color?

Using `rgb.new` is the fastest way to create a color. Using `cname->rgb` is probably the slowest.

Can we get extra credit for noticing errors of grammar and spelling in the Q&A or Errors sections?

Not anymore.

I understand that we turn in working code, but what exactly do you mean by examples? Does this mean we should include examples in our code?

It depends on the problem. For things like `rgb.greyer`, you can write an example like the following that shows that it works correctly.

```
> (define sample (rgb.new 83 120 95))
> (rgb->string (rgb.greyer sample))
"...."
```

For problems in which you create images, you can simply turn in a sequence of instructions that would create an image. (Do *not* attach the images you create.)

Also, the preliminaries state that "You ensure the best possible grade for yourself by emphasizing your answer and including a clear set of work that you used to derive the answer." What does this entail? Should I be justifying every step of every problem?

The "Show Your Work" is primarily useful for people who fail to obtain a working solution. In this case, putting in preliminary steps and indicating the intent of what they tried will help. However, even those who write correct solutions might explain the parts of a solution. The *emphasize your answer* was added in response to people who send me twenty solutions, and don't make it clear which one they consider the solution appropriate for grading.

Errors

Here you will find errors of spelling, grammar, and design that students have noted. Remember, each error found corresponds to a point of extra credit for everyone. I usually limit such extra credit to five points. However, if I make an astoundingly large number of errors, then I will provide more extra credit.

- What in *** is "Conditionaa"? [SR, 1 point]
- Sam once again got the order of parameters wrong in `image.map`. [DB, 1 point]
- Sam spelled "criticism" wrong in the Q&A section. [AZ, 1 point]

Copyright © 2007 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (Selected materials copyright by John David Stone and Henry Walker and used by permission.) This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.