Fundamentals of Computer Science I: Media Computing (CS151.01 2008S)

Exam 2: Control: Conditionals, Iteration, and Recursion

Assigned: Wednesday, 5 March 2008

Due: Beginning of class, Wednesday, 12 March 2008

Preliminaries

Exam format

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.

There are ten problems, each worth 10 points, giving a total of 100 points. Although each problem is worth the same amount, problems are not necessarily of equal difficulty. Nonetheless, you should plan to spend about twenty minutes on each problem. If you spend more than ten minutes on a problem without making progress, move on to another problem and come back to it later. Once you hit more than twenty minutes on a problem, you should consider asking for help from Professor Rebelsky (or Professor Davis).

Read the exam as soon as possible so that you can bring questions to class. Read the entire exam before you begin working.

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 two to three hours, depending on how well you've learned the topics and how fast you work. You should not work more than four hours on this exam. Stop at four hours and write "There is more to life than CS" (on both the electronic and printed versions) and you will earn at least 75 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 three hours on the exam or completed the exam. (At that point, I may then change the exam.)

Academic honesty

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. 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.

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 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.

Presenting Your Work

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 tell you to document your procedures, you need not document them. When you are asked to document procedures, you should make sure to document them using the six-P style.

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. I am best able to give such partial credit if you include a clear set of work that shows how you derived your answer. You ensure the best possible grade for yourself by clearly indicating what part of your answer is work and what part is your final answer.

Getting Help

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

Part A: Transforming Color Names

Topics: Color representations (color names and RGB colors), Color transformations, List iteration, Composition.

In our explorations of colors, we found that we could start with a procedure such as `rgb-redder` that takes a color as a parameter and produces a new color, and use that procedure to transform a whole image. Let's try a similar thing with lists of color names. (Yes, it's clear that lists of color names cannot be directly converted to images, but it is certainly possible to use them to create images.) That is, let's consider procedures that transform each color name in a list. Note that this process will be slightly more difficult, because the color transformations expect RGB colors rather than color names.

1. Making lots of color names much darker

Write (but do not yet document) a procedure, (`cnames-much-darker` *`color-names`*), that, given a list of color names, returns a new list of color names, with each color name in the result list an approximation of the color that results from applying `rgb-darker` twice to the corresponding color in the original list. For example,

```
> (cnames-much-darker (list "red" "blue" "green" "black" "white" "grey"
                            "brown" "spicy pink" "neon avocado" "neon blue"))
("blood orange" "blue" "green" "black" "quartz" "medium grey"
 "dark forest green" "fuchsia" "neon avocado" "medium blue")
```

You should make the body of this procedure as concise as you can.

2. Documenting Transformations

Using the six-P style, document `cnames-much-darker`.

A Tip. `cnames-much-darker` converts each color name to an RGB color, applies a color transformation to that RGB color, and then converts it back to a color name. (Well, you should have done that. If you're hitting your head now, that should be a lesson to read through the exam first.) In the past, we've observed that the transformations don't always work as we might expect for colors with extreme values. Even if we ignore such colors, you may still find that the transformations you've just written don't work uniformly for all colors. Why? You may want to provide some examples (under Practica or

Problems) in your documentation.

Part B: Computing Pixels

As you've seen, `image-compute-pixels!` can be used to draw interesting color ranges and to draw shapes. Let's consider once again a problem in which you might do both.

3. Drawing Multiple Circles

Suppose we want to draw three circles on canvas.

- The first has radius 50, is centered at (30,40) and is colored with a function that computes colors with (3*col/2*row/row+col).
- The second has radius 30, is centered at (50,50), and is colored with shades of purple based on their distance from (60,60).
- The third has radius 40, is centered at (20,80), and is filled with vertical stripes.

We might write

```
(define colors (map cname->rgb (list "orange" "yellow" "orange" "red")))
(define canvas (image-new 100 100))
(image-show canvas)
(image-compute-pixels!
 canvas 0 0 (- (image-width canvas) 1) (- (image-height canvas) 1)
 (lambda (pos)
   (if (>= 2500 (+ (square (- (position-col pos) 30)) (square (- (position-row pos) 40))))
       (rgb-new (* 3 (position-row pos)) (* 2 (position-col pos)) (+ (position-row pos) (position-col pos)))
       color-transparent)))
(image-compute-pixels!
 canvas 0 0 (- (image-width canvas) 1) (- (image-height canvas) 1)
 (lambda (pos)
   (if (>= 900 (+ (square (- (position-col pos) 50)) (square (- (position-row pos) 50))))
       (let ((c (* 2 (sqrt (+ (square (- (position-col pos) 60))
                              (square (- (position-row pos) 60)))))))
         (rgb-new c 0 c))
       color-transparent)))
(image-compute-pixels!
 canvas 0 0 (- (image-width canvas) 1) (- (image-height canvas) 1)
 (lambda (pos)
   (if (>= 1600 (+ (square (- (position-col pos) 20)) (square (- (position-row pos) 80))))
       (list-ref colors (modulo (quotient (position-col pos) 5) 4))
       color-transparent)))
```

Rewrite this code so that there is only one call to `image-compute-pixels!`

Hint: Use a `cond` to decide which, if any, of the circles each position belongs in.

Hint: Which circle appears in front, and which appears behind? Your code should produce exactly the same image as the code above.

Part C: Quadrachromic Pictures

Topics: Transforming images, RGB Colors, Distance computations.

Consider the following procedure, `rgb-distance`, that provides a metric for the distance between two colors.

```
;;; Procedure:
;;;   rgb-distance
;;; Parameters:
;;;   c1, an RGB color
;;;   c2, an RGB color
;;; Purpose:
;;;   Compute a "distance" between c1 and c2 that tells us how
;;;   well c1 estimates c2 (or vice versa).
;;; Produces:
;;;   distance, a real number
;;; Preconditions:
;;;   [No additional]
;;; Postconditions:
;;;   For any color c,
;;;     (rgb-distance c c) is 0
;;;   If d is likely to be perceived as closer to c than e is to c, then
;;;     (rgb-distance c d) < (rgb-distance c e)
(define rgb-distance
  (lambda (c1 c2)
    (+ (square (- (rgb-red c1) (rgb-red c2)))
       (square (- (rgb-green c1) (rgb-green c2)))
       (square (- (rgb-blue c1) (rgb-blue c2))))))
```

4. The closer of two colors

Write a procedure, (`rgb-closest-estimate-of-2` *`color estimate1 estimate2`*), that, given three RGB colors, returns whichever of *`estimate1`* or *`estimate2`* is closer to *`color`*.

If *`estimate1`* and *`estimate2`* are equidistant from *`color`*, you may return either one.

For example,

```
> (rgb->cname (rgb-closest-estimate-of-2 (cname->rgb "wheat")
                                         (cname->rgb "white")
                                         (cname->rgb "black")))
"white"
> (rgb->cname (rgb-closest-estimate-of-2 (cname->rgb "bakers chocolate")
                                         (cname->rgb "white")
                                         (cname->rgb "black")))
"black"
> (rgb->cname (rgb-closest-estimate-of-2 (cname->rgb "spicy pink")
                                         (cname->rgb "magenta")
                                         (cname->rgb "blue")))
"magenta"
```

```
> (rgb->cname (rgb-closest-estimate-of-2 (cname->rgb "neon blue")
                                         (cname->rgb "magenta")
                                         (cname->rgb "blue")))
"blue"
```

5. The closer of two colors, revisited

Using the six-P style, document `rgb-closest-estimate-of-2`.

6. The closer of four colors

Consider the following procedure,

```
;;; Procedure:
;;;   rgb-closest-estimate-of-4
;;; Parameters:
;;;   color, an RGB color
;;;   est1, an RGB color
;;;   est2, an RGB color
;;;   est3, an RGB color
;;;   est4, an RGB color
;;; Purpose:
;;;   Find the closest estimate to color from among the four
;;;   estimates.
;;; Produces:
;;;   est, an RGB color
;;; Preconditions:
;;;   [No additional]
;;; Postconditions:
;;;   est is either est1, est2, est3, or est4
;;;   (rgb-distance color est) <= (rgb-distance color est1)
;;;   (rgb-distance color est) <= (rgb-distance color est2)
;;;   (rgb-distance color est) <= (rgb-distance color est3)
;;;   (rgb-distance color est) <= (rgb-distance color est4)
(define rgb-closest-estimate-of-4
  (lambda (color est1 est2 est3 est4)
    (rgb-closest-estimate-of-2 color
                               (rgb-closest-estimate-of-2 color est1 est2)
                               (rgb-closest-estimate-of-2 color est3 est4))))
```

Explain, in your own words, how this procedure works.

7. Quadrachromic Images

Write (but do not document) a procedure, (`image-quadrachromic` *`image c1 c2 c3 c4`*), that, given an image and four RGB colors, builds a new image in which each pixel is whichever of *`c1`*, *`c2`*, *`c3`*, or *`c4`* is closest to the color of the corresponding pixel in *`image`*.

Warning! This procedure may be slow. Try it with a small image first.

For example, here is the result of

```
(image-quadrachromic portrait
                     (cname->rgb "bakers chocolate")
                     (cname->rgb "Seattle salmon")
                     (cname->rgb "wheat")
                     (cname->rgb "light avocado green"))
```

Part D: More Fun with Spots

Topics: Recursion, Spots, Filtering, Maximum/Minimum.

In our initial explorations with recursion, we worked primarily with lists of RGB colors or lists of numbers. Let's consider some recursive procedures that one might write with lists of spots. (We know that recursion, rather than `map` or `foreach!`, is appropriate for these problems because they either return a type other than a list or return a list that may not be the same length as the original.)

You should find the following procedures helpful for this problem.

```
;;; Procedure:
;;;   spot-new
;;; Parameters:
;;;   col, an integer
;;;   row, an integer
;;;   color, a color (name, RGB, etc.)
;;; Purpose:
;;;   Create a new spot.
;;; Produces:
;;;   spot, a spot
;;; Preconditions:
;;;   [No additional]
;;; Postconditions:
;;;   (spot-col spot) = col
;;;   (spot-row spot) = row
```

```
;;;   (spot-color spot = color
(define spot-new
  (lambda (col row color)
    (list col row color)))

;;; Procedure:
;;;   spot-col
;;; Parameters:
;;;   spot, a spot
;;; Purpose:
;;;   Extract the col from a spot.
;;; Produces:
;;;   col, an integer
(define spot-col
  (lambda (spot)
    (car spot)))

;;; Procedure:
;;;   spot-row
;;; Parameters:
;;;   spot, a spot
;;; Purpose:
;;;   Extract the row from a spot.
;;; Produces:
;;;   row, an integer
(define spot-row
  (lambda (spot)
    (cadr spot)))

;;; Procedure:
;;;   spot-color
;;; Parameters:
;;;   spot, a spot
;;; Purpose:
;;;   Extract the color from a spot.
;;; Produces:
;;;   color, an integer
(define spot-color
  (lambda (spot)
    (caddr spot)))
```

8. Selecting Spots

We might use an approximation of the diagonal (most simply, places where the row and column are equal) as a way of selecting spots. In particular, we might say that spots whose row and column differ by no more than two are near the diagonal.

Write (but do not document) a procedure, `(spots-select-near-diagonal` *`spots`*`)` that, given a list of spots, returns a list of the spots in *`spots`* that meet that criterion for being near the diagonal. For example,

```
> (spots-select-near-diagonal (list (spot-new 5 2 "red")
                                    (spot-new 5 3 "red")
                                    (spot-new 5 4 "red")
                                    (spot-new 6 3 "red")
                                    (spot-new 6 4 "red")
                                    (spot-new 7 7 "red")
                                    (spot-new 7 8 "red")
                                    (spot-new 7 9 "red")
                                    (spot-new 7 10 "red")))
((5 3 "red") (5 4 "red") (6 4 "red") (7 7 "red") (7 8 "red") (7 9 "red"))
```

9. The Highest Spot

Write a procedure, `(spots-highest spots)`, that finds the highest spot in *`spots`*. (The highest spot is the one with the smallest row number.) If more than one spot is in the topmost row, you may return any of those spots. For example,

```
> (spots-highest (list (spot-new 5 2 "red")
                       (spot-new 5 3 "red")
                       (spot-new 5 4 "red")
                       (spot-new 6 3 "red")
                       (spot-new 6 4 "red")))
(5 2 "red")
```

10. The Highest Spot, Revisited

Using the six-P style, document `spots-highest`.

Some Questions and Answers

Errata

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.

- The example using `spots-select-near-diagonal` in Problem 8 was missing a close parenthesis. [AC, 1 point]
- Misspelled “Practica” as “Pratica”. [AS, 1 point]
- Referred to `rgb-distance` as `color-distance`. [DM, 1 point]
- Typo in documentation for `rgb-distance`: “perceived” misspelled as “preceived”. [JB, 1 point]

Copyright (c) 2007-8 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.