Exam 1: Image Representations and Scheme Basics

Assigned: Friday, 19 September 2008

Due: Beginning of class, Wednesday, 24 September 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 10 problems, each worth 10 points for a total of 100 points. Although each problem is worth the same amount, problems are not necessarily of equal difficulty.

Read the entire exam before you begin.

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's more to life than CS" 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 Weinman (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 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. 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 9 p.m. and after 8 a.m.), feel free to try to call me in the office (269-9812).

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: Drawing with GIMP Tools

1. The following code draws a circle with a slash through it, as on "No Smoking" signs.

```
(context-set-fgcolor! "red")
(context-set-brush! "Circle (05)")
(image-select-ellipse! canvas selection-replace 0 0 100 100)
(image-stroke! canvas)
(image-draw-line! canvas 0 0 100 100)
(image-select-nothing! canvas)
(context-update-displays!)
```

Write a procedure called `image-draw-no-sign!` that generalizes the code as much as possible.

2. Implement the procedure `(image-safe-select-ellipse!` *`image operation left top width height`*`)`. This procedure should select an ellipse based on the given parameters, but should ensure that the entire ellipse falls within the boundaries of the image. That is, it should modify *`left`* and *`top`*, if necessary, so that they fall within the boundaries of the image. The lower right corner of the selection must also fall within the image.

Part Two: Drawings as Values

3. The following code is intended to draw the flag of Japan, a red circle centered on a white field (see Wikipedia: Flag of Japan). This is not actually what it does. Fix it.

```
(define circle (drawing-scale unit-circle 100))
(drawing-recolor circle "red")
(drawing-hshift circle 150)
(drawing-vshift circle 100)
(image-show (drawing->image rectangle 300 200))
```

4. Now that you've drawn the flag of Japan, consider the flag of France: three vertical stripes of equal size, colored blue, white, and red (see Wikipedia: Flag of France). Write a program (that is, a sequence of expressions) that uses drawings as values to draw the flag of France. Make your code as concise as you can.

Part Three: Lists

5. The procedure (`image-select-polygon!` *`image operation points`*) allows the user to select an arbitrary polygon. Each point is a pair consisting of the column and row; the procedure (`point-new` *`col row`*) lets you construct a new point. Try out the following example to see how `image-select-polygon!` can be used to select a parallelogram.

```
(define canvas (image-new 200 200))
(image-show canvas)
(image-select-polygon! canvas selection-replace
                       (list (point-new 10 50) (point-new 60 50)
                             (point-new 80 80) (point-new 30 80)))
```

When you first started programming with GIMP tools, several of you wanted to know how to select a triangle for the roof of a house. You now have the tools to figure this out.

Write a procedure, (`image-select-isosceles-triangle!` *`image operation left top width height`*), that draws an isosceles triangle of the given width and height. Recall that an isosceles triangle has two legs of the same length.

For example, the code below should produce the adjacent image.

```
(define canvas (image-new 200 200))
(image-show canvas)
(image-select-isosceles-triangle! canvas selection-replace
                                  10 30 80 150)
(image-fill! canvas)
(image-select-nothing! canvas)
```

6. Implement a procedure (`list-swap-first-two` *`lst`*) that creates a new list that interchanges the first and second items in the original list, leaving the rest of the list unchanged. For example,

```
> (list-swap-first-two (list 1 2 3))
(2 1 3)
> (list-swap-first-two (list 'a 'b 'c 'd 'e))
(b a c d e)
```

Assuming *`canvas`* is defined, what effect does the following code have?

```
(define parallelogram-points (list (point-new 10 50) (point-new 60 50)
                                   (point-new 80 80) (point-new 30 80)))
(image-select-polygon! canvas selection-replace parallelogram-points)
(image-select-polygon! canvas selection-replace
                       (list-swap-first-two parallelogram-points))
```

7. If we define the list breakfast thus

```
(define breakfast (list 'spam (list 'eggs 'spam) (list 'spam 'spam)))
```

then we might figure out how to extract the symbol `eggs` from the value named `breakfast` as follows:

```
> breakfast
(spam (eggs spam) (spam spam))
> (cdr breakfast)
((eggs spam) (spam spam))
> (car (cdr breakfast))
(eggs spam)
> (car (car (cdr breakfast)))
eggs
```

Note that the first two operations were exploratory; only the final expression is necessary to extract `eggs` from `breakfast`.

For each of the lists defined below, write at least one expression that extracts the symbol `eggs`. You may use any list operations that you see fit.

```
; Problem (a)
(define super-spam-plate (list 'spam 'spam 'spam 'spam 'spam 'eggs 'spam))
; Problem (b)
(define spam-plus (list (list 'spam 'eggs) (list 'toast 'spam)))
; Problem (c)
(define artery-buster (list (list 'spam 'bacon) (list 'eggs) 'spam))
; Problem (d)
(define nested-spam (list (list (list 'eggs 'spam))))
```

Part Four: Fun with Turtles and Triangles

In this part, you will be building the tools needed to draw an isosceles triangle using turtle graphics. You will demonstrate your ability to use numeric procedures and define new procedures. We also hope you will learn something about breaking a complex algorithm down into smaller parts that can be solved independently---what computer scientists call *functional decomposition*.

8. As you may have noticed, turtles understand angles in degrees, while Scheme's trigonometric functions assume angles are measured in radians. Implement a procedure, `degrees->radians`, that takes as its parameter an angle measured in degrees and returns an equivalent angle measured in radians. Recall that 360 degrees is equivalent to 2 pi radians. (Hint: How will you test your procedure to ensure it's correct?)

9. An isosceles triangle has two sides of the same length, and therefore also two angles that are the same. Given the width of the base and the apex angle at which the two equal sides meet, we can determine the length of the two remaining sides and also the two remaining angles.

(a) Implement the procedure (`isosceles-base-angle` *`apex-angle`*), which takes as a parameter the apex angle measured in degrees, and computes the remaining two angles of the triangle as follows: *`base-angle`* = (180 - *`apex-angle`*) / 2

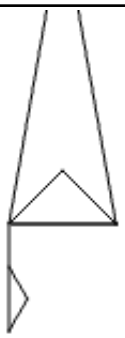
(b) Implement the procedure (`isosceles-leg-length` *`base-width apex-angle`*), which takes as parameters the width of the base and the apex angle, and computes the length of the remaining two legs of the triangle as follows: *`leg-length`* = *`base-width`* / (2 * sine(*`apex-angle`* / 2))

10. Finally, write a procedure, (`turtle-isosceles-triangle!` *`turtle base-width apex-angle`*), that draws an isosceles triangle with the given base width and apex angle. As with most turtle drawing procedures, you should assume that the user has already placed the turtle where they want to start drawing.

Hint: Your procedure will be much simpler if it makes use of the mathematical procedures described in the previous two problems. Even if you were not able to solve the previous two problems, try writing code that would use those procedures if they were correct.

For example, the code below should produce an image similar to the image below.

```
(define canvas (image-new 200 200))
(define terry (turtle-new canvas))
(image-show canvas)
(turtle-teleport! terry 100 100)
(turtle-isosceles-triangle! terry 50 20)
(turtle-isosceles-triangle! terry 50 90)
(turtle-turn! terry 90)
(turtle-forward! terry 20)
(turtle-isosceles-triangle! terry 30 120)
```



Some questions and answers

General

Question: Does “implement the procedure ...” mean the same thing as “write a procedure called ...”?
Answer: Yes, those mean the same thing.

Question 10

Question: The question says to assume the user has placed the turtle where they want it to start drawing. Can you also assume it is facing right?
Answer: Assume the turtle is also facing the direction that the user wants it to face. In this example, the turtle was initially facing counterclockwise along the base of the triangle; that is, to the right.

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.

- We forgot to make the example images available before class. [JLND, 1 point]
- The original image accompanying question 5 was wrong. According to the parameters, it should be 80 pixels wide and 150 pixels tall. Since it is on a 200*200 pixel canvas the height of the triangle should be 2/3 of the canvas, which was clearly not the case. [RS, 1 point]
- We meant to write `image-select-polygon!`, etc., and not `gimp-select-polygon!` [EH, 1 point]

Jerod Weinman