1. Apply for a department Unix account in the usual way. Do that even if you already have a Unix account, so that you can tell the system that you are enrolled in this course. Do not ever change the protections on this account (or on any files or subdirectories in it).

When I send email to the class, I will send it to the cs450-1 email list. This list consists of everyone who has signed up for the class in this way, and the email will go to your email address at cs.umb.edu. So be sure to read your email there, or forward it to some place where you will read it (there is a FAQ section at the cs webpage with instructions on how to open an account and how to forward).

2. Read pages 1-21 of the text (Abelson and Sussman’s Structure and Interpretation of Computer Programs) once over lightly, and try not to worry about what doesn’t seem clear. Answer the following 4 exercises (and note that the third one has a number of parts to it):

(a) 1.1 (page 20) Work this out with pencil and paper, thinking it through by yourself before you run Scheme to see what it really does. Then evaluate these expressions in Scheme to make sure you got them right.

(b) 1.6 (page 25). Note that this exercise calls for an explanation, not a program.

(c) (Call this Problem A on your paper, so I’ll know which one it is.) Suppose we have this code:

```scheme
(define (f x) (* x x))
```

i. What is this turned into internally when the Scheme interpreter reads this definition?

ii. Given this definition, and given the following expression

```scheme
(f (+ 3 5))
```

A. Show how this expression is evaluated using applicative-order evaluation.

B. Show how this expression is evaluated using normal-order evaluation.

(d) For extra credit: 1.5 (page 21). And notice that the problem asks for an explanation. If all you put is the answer but not a clear and convincing explanation for it, I’m not even going to read it. Please keep this in mind – it will apply to virtually everything we do.

To test your answers and your understanding you will want to be able to play with Scheme while you read the text. To run the UMB Scheme interpreter simply type

```
% scheme
```

at the Unix prompt. Then enjoy.

To work with a file of Scheme code rather than simply typing at the Scheme prompt you can redirect input (and output) with

```
% scheme < somefile.scm > somefile.out
```
or (load "somefile.scm") while in Scheme.

**Note:** Your answers to the exercises should come along with (or be incorporated in) a short essay in which you discuss what you discovered about Scheme. How does it compare to other languages you know? Did any of its responses surprise you? What is the largest integer Scheme will handle? What message appears when you type `'+` at the Scheme prompt (and what does the message mean)? What other experiments did you perform? What do you like or dislike (so far) about Scheme?

This essay and the others I will regularly ask for are an important part of how you learn, and an important part of your grade. I will read them before I read your code. So take them seriously.