UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS 61A
Summer 2008

Evan Chou

CS 61A: Structure and Interpretation of Computer Programs
General Course Information

Introduction

The CS 61 series is an introduction to computer science, with particular emphasis on software and on machines from a programmer's point of view. This first course concentrates mostly on the idea of *abstraction*, allowing the programmer to think in terms appropriate to the problem rather than in low-level operations dictated by the computer hardware. The next course, CS 61B, will deal with the more advanced engineering aspects of software—on constructing and analyzing large programs and on techniques for handling computationally expensive programs. Finally, CS 61C concentrates on machines and how they carry out the programs you write.

In CS 61A, we are interested in teaching you about programming, not about any particular programming language. We consider a series of techniques for controlling program complexity, such as functional programming, data abstraction, object-oriented programming, and query systems. To get past generalities you must have programming practice in some particular language, and in this course we use Scheme, a dialect of Lisp. This language is particularly well-suited to the organizing ideas we want to teach. Our hope, however, is that once you have learned the essence of programming, you will find that picking up a new programming language is but a few days' work.

Instructor

Evan Chou
329 Soda Hall
Email: chou86_e@berkeley.edu
Office Hours:
2:00–5:00 Tuesday,
2:00–5:00 Thursday

Office hours are mainly for questions about the course material or about administrative issues. Please don't wait until the last minute to ask questions; ask as soon as you don't understand something. I'd be happy to help.

Prerequisites

In the past we did not impose any programming-related prerequisites for admission to 61A. However, in recent years we have found that 80% to 90% of 61A students have

had significant prior programming experience, and that students without such experience are at a disadvantage. You certainly have adequate background for this course if you are familiar with the idea of *recursion*: a procedure invoking itself as a subprocedure. If you've taken the CS Advanced Placement AB course in C++ or Java, you are certainly ready for 61A.

If you don't feel ready for 61A, we recommend that you take CS 3, which is a Scheme-based introductory programming course, or CS 3S, the self-paced version. CS 3 and 3S are directed primarily at students who are not Computer Science majors, but are also designed to serve as preparation for 61A. You could then take 61A next semester.

Non-technical students (for instance, ones who prefer Math 16A to Math 1A) should probably avoid CS 61A entirely, and should take CS 3 or 3S to satisfy their need for or interest in a programming course.

Course Materials

The textbook for this course is *Structure and Interpretation of Computer Programs* by Abelson, Sussman, and Sussman, second edition. **You must get the 1996 second edition! Don't buy a used copy of the first edition.** The text is also online; there is a link from the course webpage (see below).

In addition to the textbook, there is a required reader in two volumes. **Volume 1 contains the current semester's assignments, so you must get a new copy this semester.** Volume 2 contains unchanging reference material for the course, so you can use a copy from a previous semester, and at the end of the semester, you can sell a copy to a future student.

Since in the summer we condense two weeks worth of material into one, the course material is labeled by half-weeks (i.e. 1A, 1B, 2A, 2B, ..., 8A). But Volume 2 (since it is unchanging) is still labeled by the week numbers in Fall or Spring (i.e. 1,2,3,4,...,15), and the midterm dates are off.

You will be able to buy the reader at CopyCentral, 2483 Hearst Avenue.

The course reader includes Brian Harvey's lecture notes, which are great. I will be basing my lectures off of these notes.

If you haven't used Unix before, you should also read the *User's Guide to Unix on EECS Instructional Facilities* available on the web at

`http://inst.eecs.berkeley.edu/pub/html/Users.Guide/`

If you have a home computer, you may want to get a Scheme interpreter for it. The Computer Science Division can provide you with free versions of Scheme for Linux, Windows, or MacOS. The distribution also includes the Scheme library programs that we use in this course. You can dowload copies of this distribution at

`http://inst.EECS.Berkeley.EDU/~scheme`

Enrollment—Laboratory and Discussion Sections

In addition to the lectures, the course consists of a discussion/lab section four times per week. Most weeks, the first and third meetings will be in our laboratory room, 271 Soda; the second and fourth meetings will be in the classroom listed in the Schedule of Classes. However, **the first three meetings this week will be in the lab.**

The discussion sections are run by Teaching Assistants; each TA will handle enrollment for his or her sections. We anticipate some rearrangements during the first week in response

to oversubscribed or undersubscribed sections. If you are not pre-enrolled, you should pick a discussion section, but be prepared to shift if your first choice is full. Please try to be in a definite discussion section by the end of the week, though, because much of the coursework will be done in groups of two to four students (the number depends on the activity); these groups will be set up by the TAs within each section.

You must have a computer account on the 61A course facility. You must set up your account *this week* because that is how we know who is really in the class. If you are pre-enrolled but do not set up your account this week, you may be dropped from the course. Account forms will be distributed at the first lecture, along with this handout. If you missed the first lecture, see Jenny Jones in 339 Soda Hall to get your account. The first time you log in, you will be asked to type in your name and student ID number, if you have one. Please follow the instructions carefully. You must get your account *and log into it* no later than **5pm Thursday** so that we have an accurate class count.

If you have any problems with the computer account (i.e. forgetting the password), email the Instructional sys admin staff at `inst@inst.eecs.berkeley.edu` or visit them in 333 Soda, 378 Cory or 386 Cory. Please do not activate two accounts, especially if you already have some work graded under the old account!

If you are not pre-enrolled and want to take 61A, you have to add the course using Tele-BEARS. If you are something other than a regular Berkeley undergraduate, then you probably need a signature on a form admitting you to the course.

Information Resources

Your first and most important resource for help in learning the material in this course is your fellow students. Your discussion section TA will assign you to a group of four students, and you will do all course activities with this group. You are responsible for helping each other learn.

The class will have a staff of undergraduate Lab Assistants (LAs). Each LA will have scheduled hours to be in the lab. Whenever an LA is in the lab you may request that s/he answer questions about the homework or programs (but *not* do them for you).

The instructor and the Teaching Assistants who teach the discussion sections are also available to answer questions. You may drop in during office hours, make appointments for other times, or communicate with us by e-mail.

For technical questions about the homework or about the computer facility, post your questions to the class newsgroup (see below), but please *don't* post your homework solutions before the assignment is due. You can also send e-mail to your TA or to me about intellectual questions. If you have an administrative question about a *future* assignment (e.g., "I have to be out of town..."), send e-mail to your TA or your reader, as appropriate. If you have a complaint about a missing or incorrect grade on a *past* assignment, there is an online complaint form accessible from the course web page.

Solutions to homeworks and labs are posted online the day after each assignment is due in the directory `~cs61a/solutions` and on the course web page. You should definitely read these! They discuss most problems in some depth, with alternate solutions and suggestions for thinking about similar problems. Project and exam solutions are also posted, two days after the due date of each project, and whenever we finish grading each

exam (because the posted solutions include a discussion of common wrong answers, which we don't know until we grade them).

There is an electronic bulletin board system that you can use to communicate with other 61A students. To do this, subscribe to the `ucb.class.cs61a` newsgroup using any news reader, such as trn, Mozilla, or Thunderbird.

Your TA will help you set this up. You can also find some information here:

`http://inst.eecs.berkeley.edu/connecting.html#news`

There is a class web page, with online versions of some of the documents we hand out:

`http://www-inst.eecs.berkeley.edu/~cs61a`

The web page for the textbook, with additional study resources, is

`http://www-mitpress.mit.edu/sicp/sicp.html`

Tutoring services are provided by Eta Kappa Nu (HKN), the EECS honors society (345 Soda, `hkn@hkn`), and Upsilon Pi Epsilon (UPE), the Computer Science honors society (346 Soda, `upe@cory`).

Additional information to help you in studying, including hints from the course staff and copies of programs demonstrated in lectures, is available on the course web page.

Computer Resources

The computing laboratory in 271 Soda consists of about 30 SunRay terminals connected to a Sun Solaris compute server. The lab is normally available for use at all times, but you need a card key for access to the lab. To gain card key access to Soda Hall you must go to 387 Soda Hall to complete the appropriate form to get your CAL ID proximity card activated, or to obtain a white card key (if you don't have a CAL ID).

During scheduled lab sessions, only students enrolled in that particular section may be in the lab. At other hours, any 61A student may use the lab on a drop-in basis. (Until you get your card key, you may find it easier to use the labs on the second floor of Soda Hall for drop-in use, since they are unlocked during daytime hours.)

These machines use the Unix operating system. The course readers include introductory documentation about Unix and about Emacs, the text editing program we are recommending for your use. Although the use of Unix is not extensively taught in 61A lectures, it will be extremely worthwhile for you to spend some time getting to know how the system works.

Each weekly homework assignment includes a suggested "feature of the week" for you to explore. These are entirely optional, and there is nothing to hand in about them.

If you have a home computer you may wish to use your class account remotely. The TAs will help you set this up.

Homework and Programming Assignments

Each week there will be problems assigned for you to work on, most of which will involve writing and debugging computer programs. You'll work on some of these problems individually, and some in groups. These assignments come in three categories:

• **Laboratory exercises** are short, relatively simple exercises designed to introduce a new topic. Most weeks you'll do these during the scheduled lab meetings the first half of the week, in groups of two to four students.

• **Homework assignments** consist mostly of exercises from the textbook; you'll do these whenever you can schedule time, either in the lab or at home. You may be accustomed to textbooks with huge numbers of boring, repetitive exercises. You won't find that in our text! Each assigned exercise teaches an important point. These assignments are included in the course reader. (The first half-week's assignment is also attached to this handout.)

You are encouraged to *discuss* the homework with other students, but each of you must prepare and turn in your own solutions. (More on this later.)

• **Projects** are larger assignments intended both to teach you the skill of developing a large program and to assess your understanding of the course material. There are four projects during the semester. The first two projects will be done individually; the last two in groups of two students.

Most of the weekly assignments include problems labelled as "Extra for Experts." These problems are entirely optional; do them only if you have finished the regular assignment and want to do something more challenging. There is no extra *credit* for these problems; people who need more credit shouldn't even be trying them, and people who are doing well in the course should be motivated by the desire to learn.

The purpose of the **homework** is for you to learn the course, not to prove that you already know it. Therefore, the weekly homeworks are not graded on the correctness of your solutions, but on effort.

You will get full credit for an entirely wrong answer that shows reasonable effort! (But you should test your work. If your solution is incorrect, the grader will want to see some evidence that you know it's incorrect.)

Each homework is worth two points for a reasonable effort, zero points for a missing homework or one that seems to show no effort, or **negative four (−4) points** for a solution copied from someone else.

The four programming **projects** *are* graded on correctness, as well as on your understanding of your solution. The first two projects will be done individually; the last two in pairs, but the work is split up so that each problem is done by one student. You must work together to ensure that both group members understand the complete results.

Projects must be turned in both online and on paper (see below).

Copying someone else's work does not count as "reasonable effort"! This includes copying from your friend who took the course last semester as well as copying from other current students. You will get negative credit for copied solutions, and repeated offenses will lead to more severe penalties. If you don't know how to do something, it's better to leave it out than to copy someone else's work.

You should try to complete the *reading* assignment for each week **before** the lecture. (Read section 1.1 as soon as possible this week!) The schedule is posted below.

Each week, two homeworks are assigned, the first will be due Thursday and the second will be due Monday, both at 11am. There are two ways to turn in assignments: online and on paper. The first homework must be turned in *both* online and on paper. (Part of the purpose of this assignment is to make sure you know how to print things.) The remaining homework assignments *must* be turned in online, and *may also* be turned in on paper if you would like detailed comments on your work from your reader.

Paper turnin: There are boxes with slots labelled by course in room 283 Soda Hall.

(Don't put them in my mailbox or on my office door!) What you turn in should include transcripts showing that you have tested your solution as appropriate. **Keep your graded papers** until the semester is over. You may need them in case a grade is entered incorrectly.

Online turnin: You must create a directory (you'll learn how to do that in the first lab) with the official assignment name, which will be something like `hw5` or `proj2`. Put in that directory all files you want to turn in. Then, still in that directory, give the shell command `submit hw5` (or whatever the assignment name is). We'll give more details in the lab.

The programming projects must also be turned in online as well as on paper in the homework box; the deadline is also at 11am the day the project is due. For the group projects, only one member of the group will submit the entire project for the group.

Keep your graded papers until the semester is over. You may need them in case a grade is entered incorrectly.

Everything you turn in on paper must show your name(s), your computer account(s), and your section number. Please cooperate about this; make sure they're visible on the *outside* of the paper you turn in, not buried in a comment in a listing. For online submissions, your name(s), computer account(s), and section number must be included in a comment at the beginning of the file.

Webcast of Lectures

For those of you who wish to review the material covered in lectures, the webcast for previous semesters of CS61A are available online:
`http://webcast.berkeley.edu/courses`

Testing and Grading

Your course grade is computed using a point system with a total of 300 points:

2 midterms	2 * 50	100	15 homeworks	15 * 2	30
final		90	4 projects	15,15,25,25	80

There will be two midterms (at the beginning of the fourth and seventh weeks of the term) and a final. The exams will be open book, open notes. (You may not use a computer or PDA during the exam.) In the past, some students have complained about time pressure, so we'll hold the midterm tests Monday evenings (at 7pm, room 145 Dwinelle) instead of during the lecture hour.

My goal will be to write one-hour tests, but you'll have two hours to work on them. The relatively large number of tests should reduce your anxiety about ruining your grade by misunderstanding any one question. In general, tests concentrate on the material that has been covered up to and including the week before the test. In this course, the later topics depend on the early ones, so you mustn't forget things after each test is over!

(A note about open book tests: The reason for making the tests open book is to free you of the need to memorize details, such as the order of arguments to a procedure. It is *not* a good idea to try to teach yourself the big ideas of the course during the exam!)

Each letter grade corresponds to a range of point scores: 280 points and up is an A+, 270–279 is A, and so on by steps of ten points to 170–179 points for a D–.

A+	280-300	A	270-279	A-	260-269
B+	250-259	B	240-249	B-	230-239
C+	220-229	C	210-219	C-	200-209
D+	190-199	D	180-189	D-	170-179

If you make the effort to do the assigned work, you will do well on the weekly homework, since those points are awarded for effort and general understanding rather than for specific correct results. The projects do require correct solutions for full credit. Finally, the tests are meant to be easy for anyone who understands the material; they do not require great creative leaps.

This grading formula implies that **there is no curve**; your grade will depend only on how well you (and, to a small extent, your group partners) do, and not on how well everyone else does. (If everyone does exceptionally badly on some exam, I may decide the exam was at fault rather than the students, in which case I'll adjust the grade cutoffs as I deem appropriate. But I won't adjust in the other direction; if everyone gets an A, that's great.)

If you believe we have misgraded an exam, first be sure that you understand the solutions and grading standards that we'll post online soon after the exam. If your paper was misgraded *according to those standards*, return it to your TA and file an online complaint (from the course web page) explaining your complaint. Only if you are unable to reach an agreement with the TA should you bring the test to me. The TA will carefully regrade *the entire test*, so be sure that your score will really improve through this regrading!

By University policy, final exams may *not* be regraded. They may be viewed at times and places to be announced.

Incomplete grades will be granted only for dire medical or personal emergencies that cause you to miss the final, and only if your work up to that point has been satisfactory.

Cooperative Learning Policy

With the obvious exception of exams, we *encourage* you to discuss *all* of the course activities with your friends as you are working on them.

Each student must solve the weekly homework assignments individually; **it is by struggling with the homework problems that you learn the course material!** However, before you develop your own solution to each problem you are encouraged to discuss it with other students, in groups as large or small as you like. **When you turn in your solution, you must give credit to any other student(s) who contributed to your work.** Working on the homework in groups is both a good way to learn and a lot more fun! Although the homework is graded on effort rather than on correctness, if you take the opportunity to discuss the homework with other students then you'll probably solve every problem correctly.

Similarly, each student should solve each problem in the scheduled lab activities, but you are welcome to discuss your efforts with your neighbors in the lab.

The first two programming projects will be done individually. The last two projects are larger, and will be done in groups of two, but with each individual student responsible for specific problems within the project. The groups will be chosen by the TA of your discussion section, although we'll consider requests for specific partners or unusual constraints. Your group will turn in *one copy* of each project, with both of your names and

logins listed on it.

Parts of some midterm exams will be done in groups of four; this will be explained further when the time comes.

If some medical or personal emergency takes you away from the course for an extended period, or if you decide to drop the course for any reason, please don't just disappear silently! You should inform your project partner and your TA, so that nobody is depending on you to do something you can't finish.

Since the textbook exercises are largely the same from one semester to the next in this course, you may be tempted to turn the official published solutions collected by a friend who's already taken the course. Don't do it, for three reasons: First, it's dishonest. Second, the readers will recognize those solutions and you'll get caught. Third, *doing the homework is the main way you learn in this course.* **Read the published solutions** *after* **you struggle with each problem yourself. Reading the homework solutions each week is an excellent way to prepare for the exams.**

Unlike the homework and projects, the tests in this course (except for the parts specifically designated as group parts) must be your own, individual work. I hope that you will work cooperatively with your friends *before* the test to help each other prepare by learning the ideas and skills in the course. But during the test you're on your own. The EECS Department Policy on Academic Dishonesty says, "Copying all or part of another person's work, or using reference materials not specifically allowed, are forms of cheating and will not be tolerated." (61A tests are open-book, so reference materials are okay.) The policy statement goes on to explain the penalties for cheating, which range from a zero grade for the test up to dismissal from the University, for a second offense.

Lateness

A programming project that is not ready by the deadline may be turned in the day after the due date at 11am. These late projects will count for 2/3 of the earned score. No credit will be given for late homeworks, or for projects turned in after the late deadline. Please do not beg and plead for exceptions. If some personal crisis disrupts your schedule one week, don't waste your time and ours by trying to fake it; just be sure you do the next week's work on time.

By the way, if you wait until the last day to do the homework, you will probably experience both a shortage of available workstations and unusually slow computer response.

Lecture Outline and Reading Assignments

In the following chart, the readings refer to section numbers in the Abelson and Sussman text. Remember, the reading should be done *before* the week indicated.

week	Monday/Wednesday		Tuesday/Thursday	reading
1A	6/23	functional programming	6/24	1.1
1B	6/25	higher-order procedures	6/26	1.3
2A	6/30	recursion and iteration	7/1	1.2.1–4
2B	7/2	data abstraction, sequences calculator	7/3	2.1, 2.2.1
3A	7/7	hierarchical data	7/8	2.2.2–3, 2.3.1,3
3B	7/9	interpreter generic operators	7/10	2.4–2.5.2
		Midterm Monday 7/14, 7–9pm		
4A	7/14	object-oriented programming	7/15	OOP (reader)
4B	7/16	assignment, state, environments	7/17	3.1, 3.2
5A	7/21	mutable data vectors	7/22	3.3.1–3
5B	7/23	client/server concurrency	7/24	3.4
6A	7/28	streams	7/29	3.5.1–3, 3.5.5
6B	7/30	metacircular eval. mapreduce	7/31	4.1.1–6
		Midterm Monday 8/4, 7–9pm		
7A	8/4	mapreduce analyzing eval.	8/5	4.1.7
7B	8/6	lazy eval. nondeterministic eval.	8/7	4.2, 4.3
8A	8/11	logic programming	8/12	4.4.1–3
		Final Thursday, 8/14, 9:30–12:30pm		

CS 61A Summer 2008 Week 1A

Topic: Functional programming

Lectures: Monday 6/23, Tuesday 6/24

Reading: Abelson & Sussman, Section 1.1 (pages 1–31)

Note: Follow the reading schedule posted on the first day handout or at the front of the Volume 1 reader. The reading should be done *before* the Monday or Wednesday lecture.

Homework due 11am Thursday, 6/26:

People who've taken CS 3: Don't use the CS 3 higher-order procedures such as `every` in these problems; use recursion.

1. Do exercise 1.6, page 25. This is an essay question; you needn't hand in any computer printout, unless you think the grader can't read your handwriting. If you had trouble understanding the square root program in the book, explain instead what will happen if you use `new-if` instead of `if` in the `pigl` Pig Latin procedure.

2. Write a procedure `squares` that takes a sentence of numbers as its argument and returns a sentence of the squares of the numbers:

```
> (squares '(2 3 4 5))
(4 9 16 25)
```

3. Write a procedure `switch` that takes a sentence as its argument and returns a sentence in which every instance of the words `I` or `me` is replaced by `you`, while every instance of `you` is replaced by `me` except at the beginning of the sentence, where it's replaced by `I`. (Don't worry about capitalization of letters.) Example:

```
> (switch '(You told me that I should wake you up))
(i told you that you should wake me up)
```

4. Write a predicate `ordered?` that takes a sentence of numbers as its argument and returns a true value if the numbers are in ascending order, or a false value otherwise.

5. Write a procedure `ends-e` that takes a sentence as its argument and returns a sentence containing only those words of the argument whose last letter is E:

```
> (ends-e '(please put the salami above the blue elephant))
(please the above the blue)
```

Continued on next page.

Week 1A continued...

6. Most versions of Lisp provide `and` and `or` procedures like the ones on page 19. In principle there is no reason why these can't be ordinary procedures, but some versions of Lisp make them special forms. Suppose, for example, we evaluate

```
(or (= x 0) (= y 0) (= z 0))
```

If `or` is an ordinary procedure, all three argument expressions will be evaluated before `or` is invoked. But if the variable `x` has the value `0`, we know that the entire expression has to be true regardless of the values of `y` and `z`. A Lisp interpreter in which `or` is a special form can evaluate the arguments one by one until either a true one is found or it runs out of arguments.

Your mission is to devise a test that will tell you whether Scheme's `and` and `or` are special forms or ordinary functions. This is a somewhat tricky problem, but it'll get you thinking about the evaluation process more deeply than you otherwise might.

Why might it be advantageous for an interpreter to treat `or` as a special form and evaluate its arguments one at a time? Can you think of reasons why it might be advantageous to treat `or` as an ordinary function?

Unix feature of the week: `man`

Emacs feature of the week: `C-g`, `M-x apropos`

There will be a "feature of the week" each week. These first features come first because they are the ones that you use to find out about the other ones: Each provides documentation of a Unix or Emacs feature. This week, type `man man` as a shell command to see the Unix manual page on the `man` program. Then, in Emacs, type `M-x` (that's meta-`X`, or `ESC X` if you prefer) `describe-function` followed by the Return or Enter key, then `apropos` to see how the `apropos` command works. If you want to know about a command by its keystroke form (such as `C-g`) because you don't know its long name (such as `keyboard-quit`), you can say `M-x describe-key` then `C-g`.

You aren't going to be tested on these system features, but it'll make the rest of your life a *lot* easier if you learn about them.

CS 61A Summer 2008 Week 1A First Lab

Monday 6/23 afternoon

Try to get as much done as possible, but don't panic if you don't finish everything.

1. Start the Emacs editor, either by typing `emacs` in your main window or by selecting it from the alt-middle mouse menu. (Your TA will show you how to do this.) From the `Help` menu, select the Emacs tutorial. You need not complete the entire tutorial at the first session, but you should do so eventually.

2. Start Scheme, either by typing `stk` in your main window or by typing meta-S in your Emacs window. Type each of the following expressions into Scheme, ending the line with the Enter (carriage return) key. **Think about the results!** Try to understand how Scheme interprets what you type.

```
3
(+ 2 3)
(+ 5 6 7 8)
(+)
(sqrt 16)
(+ (* 3 4) 5)
+
'+
'hello
'(+ 2 3)
'(good morning)
(first 274)
(butfirst 274)
```

```
(first 'hello)
(first hello)
(first (bf 'hello))
(+ (first 23) (last 45))
(define pi 3.14159)
pi
'pi
(+ pi 7)
(* pi pi)
(define (square x) (* x x))
(square 5)
(square (+ 2 3))
```

3. Use Emacs to create a file called `pigl.scm` in your directory containing the Pig Latin program shown below:

```
(define (pigl wd)
  (if (pl-done? wd)
      (word wd 'ay)
      (pigl (word (bf wd) (first wd)))))

(define (pl-done? wd)
  (vowel? (first wd)))

(define (vowel? letter)
  (member? letter '(a e i o u)))
```

Make sure you are editing a file whose name ends in `.scm`, so that Emacs will know to indent your code correctly!

4. Now run Scheme. You are going to create a transcript of a session using the file you just created:

```
(transcript-on "lab1")       ; This starts the transcript file.
(load "pigl.scm")            ; This reads in the file you created earlier.
(pigl 'scheme)               ; Try out your program.
                             ; Feel free to try more test cases here!
(trace pigl)                 ; This is a debugging aid. Watch what happens
(pigl 'scheme)               ; when you run a traced procedure.
(transcript-off)
(exit)
```

5. Use `lpr` to print your transcript file.

CS 61A Summer 2008 Week 1A Second Lab
Tuesday 6/24 afternoon

1. Predict what Scheme will print in response to each of these expressions. *Then* try it and make sure your answer was correct, or if not, that you understand why!

```
(define a 3)
```

```
(define b (+ a 1))
```

```
(+ a b (* a b))
```

```
(= a b)
```

```
(if (and (> b a) (< b (* a b)))
    b
    a)
```

```
(cond ((= a 4) 6)
      ((= b 4) (+ 6 7 a))
      (else 25))
```

```
(+ 2 (if (> b a) b a))
```

```
(* (cond ((> a b) a)
         ((< a b) b)
         (else -1))
   (+ a 1))
```

```
((if (< a b) + -) a b)
```

2. In the shell, type the command

```
cp ~cs61a/lib/plural.scm .
```

(Note the period at the end of the line!) This will copy a file from the class library to your own directory. Then, using emacs to edit the file, modify the procedure so that it correctly handles cases like `(plural 'boy)`.

3. Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

4. Write a procedure `dupls-removed` that, given a sentence as input, returns the result of removing duplicate words from the sentence. It should work this way:

```
> (dupls-removed '(a b c a e d e b))
(c a d e b)
> (dupls-removed '(a b c))
(a b c)
> (dupls-removed '(a a a a b a a))
(b a)
```