CSE 150
Programming Languages for Artificial Intelligence
UCSD, Winter 1997

Instructor: Gary Cottrell
Office: 5220 AP&M
Office Hours: Wed 1-2, Thursday 3-4 or by appointment
email: cs150w@ieng9 or gary@cs

T.A.: Glenn Reinman
Office: 3337A AP&M
Office Hours: M/W 11:00-12:00
email: cs150w@ieng9 or greinman@cs

Class Lectures:
Tuesday/Thursday: 10-11:20, PETERSON 103

Programming Languages covered in this class: LISP, Prolog

Computer Accounts: See the TA as soon as possible to obtain your computer account if you do not have an OCE account.

Note that you may access the class machine via the work stations in the UAPE Laboratory in the basement of AP&M (or from any terminal room on campus or by telephone).

Prerequisites

You must have all the prerequisites listed below. This is the BARE MINIMUM. If that is all you have, you will have to work hard to keep up. You cannot make up for any missing prerequisite with extra work or a friend who can help you or even the grace of God. In determining the bare minimum, we already figured in all those friends and extra help. IF YOU DO NOT HAVE ALL THE PREREQUISITES DO NOT TAKE THIS CLASS.

(1) An introductory programming class such as CSE 9a/B, CSE 10, CSE 11, (or the old CSE 62A-B or 65) AND CSE 12 AND CSE 100. CSE 100 may be taken concurrently. If you took a course someplace else it matters little what programming language you used but you need to have covered at least the following topics: procedures, local variables, arrays, recursion and pointers. If you are not very comfortable with recursion in some language, then you do not have the prerequisites and you should not take this class.

(2) Enough knowledge of UNIX to move files around and to use the vi editor. (You will receive absolutely no UNIX instruction in this class, because we will assume that you are already comfortable using UNIX.)

(3) Knowledge of very basic data structures: If you understand both depth first and breadth first searches of binary trees and can implement them in some programming language, then you probably know enough data structures to get by. If you do not feel comfortable with the notions of depth first and breadth first searches, then you should not take this class.

Required Texts

Winston and Horn, *LISP, 3rd Edition*

Clocksin and Mellish, *Programming in Prolog*.

Note: We will not use the PROLOG text until about halfway through the course, but the bookstore has an unfortunate policy of returning books about midterm. You should buy the Prolog book as soon as you are certain you will remain in the class. It may not be available yet in the bookstore anyway.

Grading

You will be given a series of programming assignments. Programming assignments will *not* be accepted late. There will also be two in-class midterms. *There will not be a final exam*. The various components are weighted as follows:

Homework	40%
Midterm 1	30%
Midterm 2	30%

Midterm Dates

The midterms are closed book and will be at the regular class times. The first midterm will be on or about Thursday February 13, 1997. This date is subject to change so watch for any change of date notice. The second midterm will be very near the end of the quarter, usually the last day. You must bring a picture ID to the midterms.

Cheating Policies

Unfortunately, cheating has become a serious problem on this campus. A number of students have complained that they are being hurt by others cheating in classes. This is correct. This cheating lowers the value of honestly earned grades. Unfortunately, we have no alternative but to enforce the following very strict rules against cheating.

Exams: Anybody talking during an exam will receive a zero for that exam. There will be no exceptions to this rule. If, as you leave an exam, you turn to a friend and say "meet me at Porter's when you finish," then you will receive a zero for that exam. "I forgot" or "I only said ..." will not change the zero. This may sound unreasonable. However, we have no good alternative. We can easily tell if you are talking, but we cannot listen to and evaluate everything that anybody in the room might say. Similarly, books or notes may not be with you at your seat during an exam. If you bring books into the exam room, then you must leave them in the front of the room and not pick them up until after the exam. If you are found to have any books or notes at your seat, you will receive a zero for the exam, even if you did not look at them. Anybody found clearly cheating will receive more severe penalties as described below.

Programming Assignments

NOTE: Many messages regarding programming assignments will appear as the "Message of the day" when you log in to your account. YOU ARE EXPECTED TO KNOW THAT INFORMATION!

When you have a question about the programming assignment, use e-mail to clarify points and ask brief questions. Do not mail your code and ask what is wrong with it. If you do, that mail will be ignored. For big problems or particularly stubborn bugs see the TA or instructor in person. If it is about your program, bring a listing with you.

Naming of Functions

Part of the grading will be done automatically by grading programs. Be sure to spell function names exactly as the exercise states, otherwise the grading program will not find your function and will mark it incorrect. This applies to the choice of uppercase versus lowercase letters as well as the words and their spellings.

Style rules for LISP programs:

All programs must be easy to read and well documented. You may not use **setf**, **setq**, **set**, or any thing that has similar effects (unless the assignment explicitly says you may). You may not use **prog** or any loop construct (e.g. **do**) unless the assignment explicitly says that you may. More details guidelines will be given in class and you must follow them. If a program is not easy to read or if it contains a **set** or some other forbidden construct, then the grader will take off half of the style points for each one.

Generally, LISP programming encourages lots of small functions that do simple tasks. These are then composed to form the solution. We encourage you to follow this idea. "Small" is around 5 or 6 lines, but obviously, things get longer sometimes. We like you to have function headers on your functions, and a page at the top of the file explaining your approach - data structures, your algorithm (if not specified in the assignment), explanation of your heruistics (if heuristics are asked for in the assignment) etc. The page at the top of your file is the "documentation". The style you use for your programming, and comments, are "style".

NOTE: HALF OF YOUR GRADE FOR PROGRAMS IS BASED ON DOCUMENTATION AND "STYLE". IT IS BETTER TO TURN IN A NON-WORKING PROGRAM ON TIME THAT IS WELL_DOCUMENTED THAN A WORKING PROGRAM LATE, WHICH WILL RECEIVE A ZERO!

Programming assignment schedule:

Programs are due at midnight on the due date. The programs should all be done well before the deadlines. The deadlines are late to allow for any unforseen problems.

Turning in Progams:

Use the "turnin" facility to turn in your programs. To use it, first "prep cs150" if you have an OCE account. (Do nothing otherwise!). Then, simply type "turnin <filename>".

You MUST execute this command BEFORE MIDNIGHT ON THE DUE DATE. We may sometimes collect them a little after midnight, but will check the time stamp on the file; if it has been submitted after the midnight deadline, it will be considered late and not collected. If you have a medical excuse for a missed assignment, see the instructor about special arrangements for making up the missed assignments.

Assignment Updates:

NOTE: Many messages regarding programming assignments will appear as the "Message of the day" when you log in to your account. YOU ARE EXPECTED TO KNOW THAT INFORMATION!

Rules for collaboration on Programming Assignments:

You may not work in groups unless the assignment explicitly says that you may. You must write your own programs. We ask that you follow the "Gilligan’s Island Rule": The rule is: You may discuss assignments with classmates but you may not take away any written notes, and you must watch an hour’s worth of Gilligan’s Island or equivalent before writing anything down. If a question regarding cheating arises, we reserve the right to give you an oral exam on the homework to see if you understand your answer. If you do not understand the assignment, you will receive a zero for that assignment, even if the program you have is perfect in every way. Some randomly chosen students will receive oral exams on some of the homework assignments. If you are caught blatantly copying we will impose more severe penalties as described below.

Penalty for Cheating

Anybody caught clearly cheating will receive an F in the class and will be reported to their Provost’s office. For example, if you are caught talking at all in an exam then you receive a zero for that exam, no matter how innocent the talk. If we are not sure you are cheating, that is the end of that. However, if we hear you giving or receiving answers, then you receive an F for the class. Scary, huh? We mean it to be.

Getting Started with LISP

To get the feel of LISP simply log on and then type

cl

This stands for "Common Lisp". You will then get a prompt that looks like the following:

USER(1):

You are now in the LISP interpreter. You can evaluate any LISP function by giving it along with its arguments in the correct format, e.g.,

USER(1): (+ 2 3)

produces the response:

5

Number are easy to work with at first because LISP does not care whether they have a quote in front or not. For other types of arguments and most functions, the arguments must normally be precede by a single quote when using the interpreter. The details are in the text and will be discussed in class, but the following simple example may help illustrate enough to let you start playing with the interpreter. The function **car** normally takes a list as its single argument. It returns the first thing on that list. For example,

USER(2): (car ’(a b c))

produces the response:

A

Common Lisp is case insensitive. **a** and **A** are the same to it. It will write in all caps. You can use either.

If you make a mistake, the prompt changes to what looks like an error count. For example, if you forget the quote and type

USER(3): (car (a b c))

the response will be:

Error: bla bla bla

[1]USER(4):

If you make a second error, the "1" will change to "2"s and so on. This can be treated just like any other prompt, but if you want to get back the old "clean" prompt, type **:reset**, that is, a colon followed by the word **reset**.. In other words to get rid of the error count type:

[1]USER(4)::reset

(The error count is there for reasons other than to remind you of how many mistakes you have made, but for now that need not concern us.)

You are not allowed to use **setf** in your homework assignments, but it is perfectly acceptable to use it in the interpreter to save values for testing. **setf** can be used much like an assignment operator. It binds values to atoms and then the atoms may be used as if they were the value. This works because of the way LISP evaluates functions and arguments. There will be more about that in class and there is more in the text book. To see a simple example before reading the details, you might type in something like the following to the LISP interpreter:

```
 USER(5):(setf x ’(a b c))
   (A B C)
 USER(6):(car x)
   A
 USER(7):x
   (A B C)
```

You type what is after the prompt **USER(1):**. The following line is the response of the interpreter. Note that **setf**, like all actions in LISP, is a function to be evaluated. It returns its second argument and has the side effect of binding a value to its first argument. Note that the second argument must be quoted. Its first argument is not quoted. That is a kludge to make your life easy. The first argument is quoted "automatically". Also, note that the **x** in the above example is not quoted. That is because we do not want **’x** we want the value of **x**. Also note that **x** by itself can be evaluated by LISP. That produces the value bound to **x**. If you make an error and the prompt changes, do not be concerned, just keep going or use **reset** as we described above.

To leave LISP, type

USER(8): :exit

Enjoy!