

Syllabus For ACSI-511

Programming Languages and Systems Concepts

Fall 2002

Instructor:	William A. Maniatty Computer Science Dept
Office	LI-95E
Office Hours:	MW 11-12 P.M.
e-mail:	maniatty@cs.albany.edu
URL:	http://www.cs.albany.edu/~maniatty/
Course URL:	http://www.cs.albany.edu/~maniatty/teaching/os
Required Texts:	<i>Programming Language Pragmatics</i> Michael L. Scott Morgan Kaufman, 2000
	<i>ANSI Common Lisp</i> Paul Graham Prentice Hall, 1995
Recommended Texts	<i>ANSI Common Lisp</i> Paul Graham Prentice Hall, 1995
Teaching Asst:	Zhaohuei Yang
Office:	To Be Announced
Office Hours:	To Be Announced

Additional information is on line at: <http://www.cs.albany.edu/~maniatty/teaching/proglang>

1 Goals of the Course

This course is geared to reward good students with some background in applied computing, with an emphasis on systems software, programming languages, interpreters and compile time support systems. In particular we will pay special attention to good engineering practices. This means we will focus on measuring, predicting and controlling:

- Cost — What resources does the software consume? How much memory/disk space does it need? Do networked/distributed programs need to download large amounts of information to run?
- Correctness/completeness — How expressive is the language, that is how general is the class of problems that can be solved using the language in a straight forward manner? If the tool is not general purpose, can it handle many important cases? What conditions cause the solution to become inefficient or give the wrong result?
- Performance — how fast does the software solve the problem? Do programs in the language compile slowly? Is run time slow?

To succeed, good practitioners demonstrate:

- Aptitude — ability to learn the material
- Motivation — desire to learn and apply the material
- Preparation — having good background in the prerequisite material

2 Who should take this Course?

This is a graduate level course in the computer science department, you must have background equivalent to students who have taken the entire undergraduate sequence except for 311 and 402 degree in Computer Science. This means you are responsible for knowing the material covered in CSI 201, 210, 310, 333 and MATH 367. *Unprepared students can be deregistered from the course at the instructor's discretion.* Many courses in the graduate program require a grade of B+ or better in this course (or having the corresponding prerequisites) so please make every effort to do well.

Students must be proficient in programming in some assembler and at least one procedural or object oriented programming language (e.g. C, C++, Java, Pascal, Ada, Eiffel, Smalltalk, Modula 2) and understand data structures and assembler language programming. *If you cannot program, you will fail this course.* Exposure to functional and declarative languages is a plus, but is not strictly needed. Knowledge of basic discrete mathematics and graph theory is assumed.

3 Who should *NOT* take this Course?

If doing the readings prior to lectures, writing your own programs or using applied math is a problem, this class is not for you. If you have too many undergraduate deficiencies the instructor may deem you unprepared and remove you. If you cannot allocate about 20 hours a week to work on this class, you will have difficulty.

4 Some Additional Resources

Some good books for alternative information include:

- Approaches to programming: Jon Bentley's Programming Pearls series is both engaging and insightful [1, 2]. Kernighan and Pike's Practice of Programming gives some tips from the professionals. I've heard good things about (but have not read) McConnell's Code Complete (Microsoft Press), and on the project management side, the XP series (Extreme Programming).
- The Unix Programming Environment — I really like O'Reilly's phenomenal series of books. Linux HowTos are useful. Peek, Todino and Strang have a book "Learning the UNIX Operating System" which has gotten good reviews (but I haven't read it).
- The C programming Language: Kernighan and Ritchie [3] invented C and wrote the canonical reference.

Other resources will be mentioned during the semester. As graduate students, you will read articles from journals and conferences during the course of the semester.

5 Course Overview

This course introduces students to the basic concepts of Operating systems design and their application via systems programming, including:

- scanning (lexical analysis),
- parsing and compiler compiler tools,
- language description notation (BNF),
- Semantic Analysis
- Code Generation Techniques
- Code Optimization Techniques
- Programming language features, and their implementation,
- Memory management and memory protection.
- Program translation (eg. preprocessors, compilers and linkers)

We will investigate correctness/completeness, cost and performance issues of design decisions.. This course assigns equal weight to examinations and projects/homework. Several programming languages will be compared and contrasted, including C, C++, Java, LISP, Prolog, Pascal, Eiffel, TCL/TK, and Python (if time permits). Particular emphasis will be placed on the implementation of these concepts in Unix, but other operating system implementations will also be covered.

5.1 Grading

There will be two tests, and four programming assignments. Each of the tests will count 25% of the course grade, the assignments/projects will be worth 10% each and comprise 50% of the course grade. In particular project work is critical to succeeding in this course, do NOT expect to be able to submit the project work at the end of the semester. Starting the projects early is a good idea, as it gives you time to make your design decisions. Grades are:

Grade	Minimum Score
A	93 %
A-	90 %
B+	87 %
B	83 %
B-	80 %
C+	77 %
C	70 %
E	0 %

5.2 Academic Integrity

Academic integrity is taken seriously here, if you are found in violation, you can expect any and all penalties described in the graduate student bulletin to be applied at the instructor's discretion. In particular, **cheating may result in failure**. Forms of academic dishonesty include:

- Impersonating the instructor.
- Accessing the solution key to a text book assigned for the class.
- Unauthorized sharing of course work with other students.
- Failing to report observed cheating (as this implies complicity).

All programs submitted must be your own work, and you are expected to develop your programs independently. You may receive as much help as you wish on the use of the operating system, text editors, debuggers, and so on. You may consult with other members of the class about interpreting the assignment, and you may get help in finding bugs, but not fixing bugs, but you are not allowed to look at or copy another person's code or discuss design decisions with others, and you cannot show your code to others. It is also expected that your examination work will be your own, and soliciting or giving help in solving the examination questions will be considered a violation of the course guidelines. **Reputation is our most important asset, guard it carefully! Students found to be in violation of academic integrity will fail the course and could be subject to more severe disciplinary actions.**

6 Programming assignments

This is a programming intensive course, and students are expected to be strong programmers. Assignments will be made in a variety of languages. to be specified later. While you may use whatever platforms or compilers you wish to develop your code, all code submitted must run on the specified platform (most likely eve.albany.edu) and must build (if compiled) and run without errors or significant warnings. Assignments will be submitted by using the `turnin` script. You must submit an automated build procedure (often a make file) with each assignment, and a README file is strongly suggested. Late submissions may (at the instructors and TA's discretion) not be accepted, and will almost certainly have a large penalty.

7 General Project Guidelines

All projects should follow guidelines for good programming practices. Here is my list.

- Your code must adhere to the specified naming conventions and functionality.
- Be sure to have your submitted code disable debugging output as the default (otherwise our grading programs might choke on it and your grade would suffer).
- Your program should have a comment at the top with your name, login id, a brief description of what the program is, and any special compiling instructions.
- `main()` (and most functions/routines in general) should delegate work to other functions as appropriate.
- Most functions should be short, and each should do only one thing.

- Each function should have a brief comment at the top of the function describing what it does (and if necessary describing the algorithm or citing where it was published). This comment should describe the functions:
 - inputs
 - results
 - side effect
 - any other notes
- It is not necessary to comment every line, and if your code is well written, it should not require a lot of in-line commenting. However, you should use comments to describe any unusual code or hard to follow code or complicated code, and describe any non-obvious variables. Additionally it is usually good to avoid using non-obvious numeric constants (e.g. in C and C++ you should instead use the `#define` or `const` constructs).
- variables and functions should have meaningful names, but you can use single letters like `i` for loop counters and such.
- Your software must check the return code for any system call that can fail and have an appropriate error handler.
- Your code should have enough error checking so that no matter what the user does, your program will not seg fault or loop forever, or do other strange things. It is very hard to make your program idiot proof; idiots are very clever.
- Systems software can bring down machines (and sometimes networks). Be careful that your software does not crash the system!
- All code must be written by you and you alone, except that you can use and modify any sample code from the text or from class as long as you provide credit to the source.
- Minimize the number of global variables.
- **Be prepared to debug your own code.** While the instructor and TA may sometimes assist in diagnosing some particularly troublesome errors, we cannot give any one student (or project team) an inordinate amount of help at the expense of others.

8 Schedule

This schedule is tentative and is subject to change. The course schedule is described in Table 1. Project assignments and deadlines are to be announced during the semester.

References

- [1] J. L. Bentley. Programming pearls: Selection. *Communications of the ACM*, 28(11):1121–1127, November 1985.
- [2] J. L. Bentley. *More Programming Pearls: Confessions of a Coder*. Addison-Wesley, Reading, Mass., 1988.
- [3] B. W. Kernighan and D. M. Ritchie. *The C Programming Language: ANSI C Version*. Prentice-Hall, Englewood Cliffs, NJ USA, 1988.

Week	Topics	Readings
1.	Introduction, Programming Languages, Hardware/Software.	PLP 1, ACL 1
2.	Syntax Analysis	PL 2
3.	Finish Parsing, Scope Rules	PL3
4.	Semantic Analysis	PL 4
5.	Target Architecture Features	PL 5
6.	Midterm and intro to Control Flow	PL 6, ACL 9
7.	Flow of Control	PL 6, ACL 3
8.	Data Types	PL 7, ACL 3
9.	Subroutines and Control	PL 8, ACL 3
10.	Intermediate Code Generation, Optimization	PL 9, PL 13 ACL 3
11.	Data Abstraction and Object Oriented Programming	PL 10, ACL 3
12.	Functional and Logic Languages	PL 11, ACL 3
13.	Special Topics/Presentations	
	Final Exam, Vacation!	

Table 1: (Tentative) Reading Schedule for Programming Languages and Systems Concepts (CSI 511) — ACL refers to *ANSI Common Lisp* by Graham and PL refers to Scott's *Programming Language Pragmatics* by Stevens