A First Course in Computing for Engineers

Kenneth J. Christensen, Dewey Rundus
University of South Florida

Abstract

Without a doubt, the use of computers reaches into all engineering disciplines. Thus, the importance of the first course in computing in the engineering core curriculum should not be underestimated. At the University of South Florida, *Computer Tools for Engineers* is a required course for all engineering students. This course teaches the use of computers for solving engineering problems. The course is innovative in its teaching of not only a high-level programming language, but also a mathematics package, spreadsheet, and formal design methods. A particular emphasis is placed on problem solving through formal methods such as divide-and-conquer and successive refinement. These are methods that form a core "engineering thinking" that is of value for problem solving even outside the realm of computer programming. Two significant challenges in teaching *Computer Tools for Engineers* are the lack of academic discipline of underclass students and the need to accommodate their very wide span of backgrounds.

Introduction

A key component of a traditional engineering curriculum is a first course in computing. With the use of computers pervasive in all engineering disciplines, the successful mastery of computing is important for all engineering students. Today most engineering students have their own Personal Computer (PC) at home and certainly have ready access to PC's in the college. The effective use of the computer as an engineering tool can do much to improve a student's academic performance while in school and workplace performance after graduation. Computers are today used for a far greater number of tasks than they were 10 or 20 years ago. Computers are used for communications, literature research, text processing, presentation graphics, mathematical analysis, data analysis, numerical problem solving, and for specialized applications. The first course in computing must reflect this expanded range of applications by augmenting the teaching of a programming language with a presentation of the computer in its role as an engineering tool.

The first computing course for engineers has traditionally been a "FORTRAN course" intended to teach basic programming concepts and the syntax and usage of a particular high-level language. The choice of "which language to teach" remains an unresolved issue with arguments made for FORTRAN, C, C++, Java, spreadsheet macro languages, and symbolic mathematical languages (see [1], [4], and [12]). Some large engineering colleges allow the student to select the language to be learned by enrolling in a specific section, each section based on a different high-level language (see [7]). We believe that the language debate obscures the real value of a first course in computing. A first course in computing should teach not only programming in a high-level language, but also the use of common computer tools and formal design methods. Many of the key concepts of program design, that are also language independent, are very important as methods of engineering problem solving, or "engineering thinking". For example, methods of divide-and-conquer, successive refinement, and algorithm flowcharting have value for all engineering disciplines.

In this paper we present the design and organization of *Computer Tools for Engineers* as developed and taught in the College of Engineering at the University of South Florida. The second section describes the course design. The third section describes the course pedagogy including the use of lecture and laboratory components. The fourth section contains observations made in teaching this course over a span of several years. The final section is a summary and also describes future directions for the course.

Course Design

The course *Computer Tools for Engineers* at the University of South Florida is a required three credit hour core engineering course and is typically taken in the Freshman or Sophomore years. Since the only course prerequisite is Calculus I, this is the first engineering course taken by many students. The course is organized into six major topics. The topics are:

1. *Motivation* - course motivation, introduction, and basic PC use
2. *Formula crunching* - use of a mathematics package
3. *Number crunching* - use of a spreadsheet
4. *What's under the hood* - computer internals
5. *Think it through* - methods of design
6. *Just program it* - programming with a high-level language

Appendix A contains an example course syllabus and Appendix B contains an example outline of the entire course showing the material covered and the amount of time spent on each topic.

Motivation and introduction

Students begin the course, *Computer Tools for Engineers*, with many different attitudes, personalities, and histories of computer experience. For some, computers are to be feared, for others computers are thought to be non-relevant to their chosen engineering discipline, and for yet others computing is already a common part of their lives. Each of these student personalities poses special challenges to teaching *Computer Tools for Engineers*.

The first lecture is intended to demonstrate the relevance of computers as a tool for all engineers. We point out that tools are specific for a task and that using the wrong tool may "work" (e.g., using a hammer to put in a screw), but is certainly not efficient. The first laboratory session is intended to level all students with basic PC use including:

- Finding and launching applications in Windows and DOS
- Copying, moving, and renaming files and directories and using a simple text editor (e.g., Edit in DOS or Notepad in Windows)
- Using an email program (e.g., Eudora or Pine)
- Using a browser (e.g., Netscape) to access resources on the World Wide Web (WWW)

Formula crunching - use of a mathematics package

A common tool used by all engineering students is a calculator. It is not assumed that the students can program a calculator (can many faculty really program a calculator?). In many ways, a calculator is the "hammer" that students try to apply to all problems. A calculator works well for non-repetitive numerical calculations, but does not work well for:

- Calculations that need to be repeated many times with different inputs
- Calculations that need to be documented for presentation to others
- Producing graphical output
- Solving for unknowns
- Performing symbolic math

These tasks are summarized as "formula crunching". We cover a mathematics package as a logical extension of a calculator. There are a number of mathematics packages available including Mathematica [10], Maple [11], Matlab [6], and Mathcad [5]. We have chosen to use Mathcad for three reasons:

1. It is easy to learn and includes components typical of all of the other packages.
2. It is affordable, students can buy a copy of Mathcad for use on a home PC and licensing multiple copies for on-campus use is also feasible.
3. It has a very good built-in tutorial.

First we teach the use of Mathcad for solving numerical problems with variables and functions. This introduces students to the important concepts of variables and functions which are encountered in many aspects of computing. The next step is to teach graphing of functions. A break-even analysis example is done in lecture to show how graphing can be used to visually find answers to problems. After graphing we teach the symbolic math capabilities of Mathcad including solving for variables symbolically, solving for N unknowns in N equations, and simple symbolic differentiation and integration. Not all functions in Mathcad are covered. For example, the ability to read data from a file is omitted. For purposes of number crunching existing data, a spreadsheet is a better tool. Also omitted is the programming capabilities of Mathcad. For programming, a high-level language is a better tool.

No textbook is used for this course topic. Instead, in lecture the Mathcad tutorial, quick sheets, and help facilities are introduced. Students are assigned to work through the entire tutorial which is estimated to require about 2 to 3 hours. The use of an online tutorial and help facilities is an achievement in itself, it teaches students that they can "learn from the computer". Two laboratory sessions are devoted to Mathcad, one covering simple calculation and the other covering more advanced graphing and symbolic capabilities.

Number crunching - use of a spreadsheet

A mathematics package is the right tool for formula crunching. However, for the analysis of large data sets, or "number crunching", a spreadsheet is a better tool. Spreadsheets are very good tools for:

- Analysis of large sets of numbers
- Graphical presentation of data analysis
- Development of easy to use "intelligent forms" for application-specific computing

Many spreadsheet packages are available with all having roughly similar capabilities. For this course we use Microsoft Excel due to its ready availability and extensive capabilities. The first lecture teaches the entry of text, numerical values, and formulas into a spreadsheet. Arithmetic operator precedence is covered. Importing data from text files is taught. Addressing modes (relative and absolute) are covered and the use of built-in functions (including the IF() function) is taught. An entire lecture is spent on the graphing of data. With the built-in graphing wizard in Excel, generating high quality graphs is easy. Along with graphing, simple cut-and-paste between applications is taught. At this point the Microsoft Word

Equation Editor is briefly introduced to teach students that mathematical formulas can easily be typeset into their word processing documents. Once students have learned how to manipulate and present data, the use of a spreadsheet to build an "intelligent form" is taught. In an intelligent form a user inputs numerical values in labeled fields and the spreadsheet then computes results as a function of these inputs and their respective cell formulas. An intelligent form allows an engineer to develop a "computer application" for use by someone else who does not need to be familiar with the method of solution for the given problem.

"What's under the hood" - computer internals

A single lecture is devoted to describing how a computer works, or "what's under the hood". The Von Neumann computer architecture model (see Figure 1) is introduced and examples are given for all of the components of the model. The operation of memory and disk storage is described. Having understood spreadsheets, students can make an analogy between memory addressing and spreadsheet addresses. The concept of machine language, and assembly language as a human form of machine language, is introduced. Students see how machine language is capable of only very simple operations such as moving data words to and from memory and simple arithmetic operations. The operation of an assembler follows from the discussion on machine and assembly language.

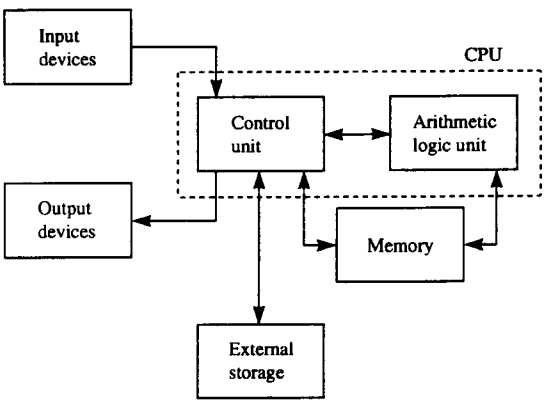

Figure 1 - The Von Neumann architecture model (from [8])

The next step from assembly language is a high-level language. The development of high-level languages as application specific languages to problem solving is introduced. Here again the idea of choosing the right tool for the problem is made. it is shown that FORTRAN is intended for scientific applications, COBOL for business applications, BASIC and Pascal for education, C for systems programming, and Java originally for consumer appliance control applications. The operation of a compiler is described and hands-on laboratory exercises are conducted to familiarize students with the use of a compiler. However, before moving to programming in a high-level language, proper design methods must be covered.

Think it through - methods of design

Possibly the most valuable part of *Computer Tools for Engineers* is teaching the students proper design methods. Engineers are often faced with problems that are:

1. Large, but intellectually simple
2. Small, but intellectually complex

For the first class of problems a means of decomposing the problem into manageable units, or modules, is necessary. The method of divide-and-conquer is taught. The end result of a divide-and-conquer approach to a problem is a structure chart showing at its "leaves" easily manageable tasks. In the context of programming, the leaves of structure diagram become individual program modules or subroutines. Figure 2 shows an example structure diagram for the example of "build a dog house".

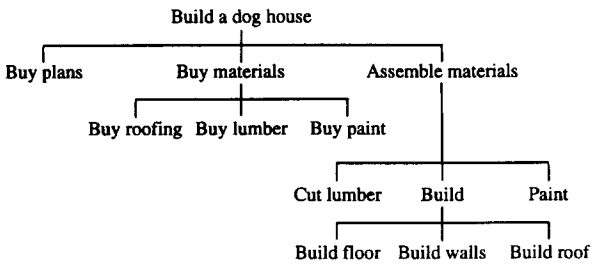

Figure 2 - Example structure diagram

For the second class of problem, very typical of homework and exam problems in the view of the students, a method of successive refinement is taught. In successive refinement a difficult task is first written as a one sentence or single block flowchart solution and then, in successive increments, details are added until finally a complete algorithm is derived. An example used in class is, "Determine if N is prime". Figure 3 shows the successive refinement approach to this problem. Step #4 would be to code the problem in a high-level language. Successive refinement is a very powerful technique for taking an abstract programming problem to a coded solution. We teach flow charting at the same time as successive refinement so that the final design refinement is a flowchart ready to be converted into program code, or *just program it* (with apologies to Nike).

```
Step #1:

Determine if N is prime

Step #2:

Input N
Divide N by all numbers from 2 to (N - 1)
If N divides evenly then output "N is not prime"
If N does not divide evenly then output "N is prime"

Step #3:

J is an integer counter variable
Input N
Loop J = 2 to (N - 1)
  Test if N divides evenly by J
  If yes output "N is not prime" and halt
EndLoop
Output "N is prime"
Halt
```

Figure 3 - Example of the successive refinement method

Just program it - use of a high-level language

If proper design methods have been taught and the concepts of variables and functions (from MathCad and Excel), program compilation, and program execution are understood, then teaching a high-level programming language is less tedious. We introduce a computer high-level language as a "vocabulary" with the analogy that knowing only a vocabulary does not enable one to be a great writer. It is true that the computer language chosen does influence the design, but procedural languages such as FORTRAN, Pascal, and C are really all very similar at a design level. That is, translating a design (for example as in Figure 3) into FORTRAN, Pascal, or C is not very different. This argument does not apply to object-oriented programming (e.g., Smalltalk or C++), which requires a very different thinking process than programming in the more typical procedural languages.

Programming, and FORTRAN as the high-level language in which to implement a program, is taught in five learning modules covering approximately eight weeks. The argument that programs should be readable by both computers and humans is strongly made. Thus, the habit of careful documentation is developed from the beginning. A standard header and documentation style is introduced, see Figure 4, and students are required to use this header format for all programs and subprograms. The five learning modules are:

1. **Programs with straight-line execution** -- Data types, arithmetic, and program input/output are covered. Special care is given to discussing mixed-mode arithmetic problems. Program documentation, organization (header, specification, and execution), and naming of variables is covered.
2. **Programs with selection** -- Logical expressions and the various IF constructs are taught as the means of selecting portions of code to execute based on logical conditions. Having been introduced to IF and logical expressions in spreadsheets, the application of the various flavors of the FORTRAN IF is not very difficult for students.
3. **Programs with repetition** -- The next step from selection is repetition. DO loops, WHILE-DO, and DO-UNTIL are taught. Debugging with trace statements is taught.
4. **Programs with subprograms** -- Subprograms such as FUNCTION and SUBROUTINE are seen as ways to break-up large monolithic programs into modular programs conforming to a structure diagram. A FUNCTION is seen to be very similar to a Mathcad function.
5. **Complex data structures** -- As a final topic, one and two dimensional arrays are covered. Having seen spreadsheets, the concept of an array is not difficult. Manipulating two-dimensional arrays, for example in a matrix multiplication subroutine, reinforces the learning of loop constructs.

We intentionally do not cover all of the FORTRAN language. Archaic constructs, such as computed GOTO's, are omitted. Also minimized, are FORTRAN-specific features that do no extend well into other languages. For example, FORTRAN's FORMAT statement, DATA statement, and COMMON statement are presented so that the student can recognize their use when encountered in existing code, but these features are only minimally practiced.

```
C=============================================================
C= Program name (assignment #)            filename = _______ =
C= Your name and SSN                                         =
C=============================================================
C= Description of program purpose and function               =
C=============================================================
C= Variables used are:                                       =
C=                                                           =
C=============================================================
C= Program inputs are:                                       =
C= Program outputs are:                                      =
C=============================================================
```

Figure 4 - Standard program and subprogram header

Course Pedagogy

Computer Tools for Engineers is offered Fall, Spring, and Summer semesters and has an enrollment of up to 240 students per semester. A two-hour lecture meets once per week in a large auditorium. Each student attends one smaller laboratory session with each laboratory session meeting in a "PC lab" with one PC for each student. The lecture is given by a full-time faculty member and the laboratory sections are handled by graduate teaching assistants. Important considerations for the class text include clarity of expression, extensive use of engineering-oriented examples, and inclusion of study problems of varying degrees of difficulty.

Lecture pedagogy

The lecture room in which the course is taught contains the standard blackboard, but also contains a computer projection screen. The computer projection screen is particularly useful for this course. It not only supports the display of presentation-graphics slides, but also allows display of real-time creation, execution, and debugging of programs. This promotes a feeling of involvement for the students. The digitally based format of the lecture materials allows them to be made available to the students on the course WWW homepage. The course WWW homepage is shown in Figure 5 (and also described in detail in [3]). When covering FORTRAN programming, some example programs are created and executed in class while other small sample programs are presented and then the students are asked to work-through the operation of the program. In this way the concepts of, for example, a loop and its implementation in FORTRAN can be learned. Students have the option of downloading and printing the WWW "handouts" before class for study and note-taking in class. Thus, the lecture is driven not only by required textbook readings, but also by materials presented on the class homepage as handouts. Other materials used to support lecture activities include the old quizzes and exams that are available from the class homepage. Within the lecture sessions, readings and laboratory assignments are made. These assignments are completed and submitted within individual laboratory sessions.

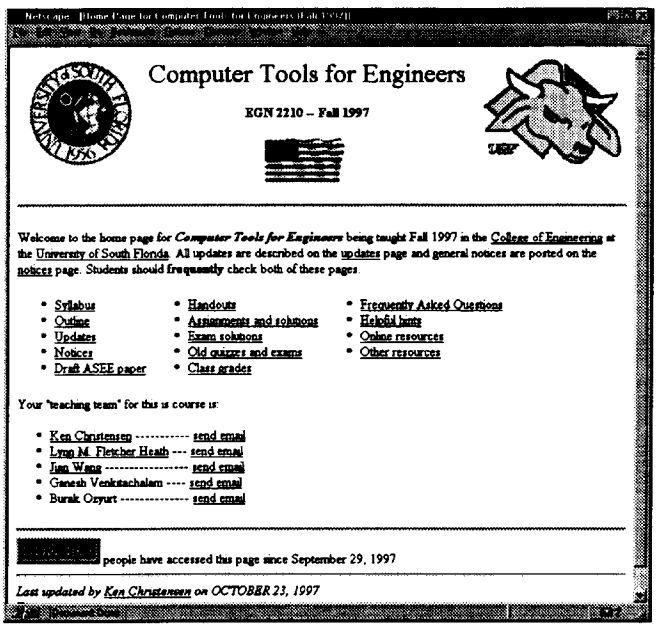

Figure 5 - Homepage for *Computer Tools for Engineers*

Laboratory pedagogy

The hands-on learning occurs in the laboratory sessions. Laboratory sessions meet every week. The weeks alternate with hands-on assignments and quizzes. Each laboratory session serves approximately 20 to 40 students and is conducted by two graduate teaching assistants. Within the laboratory session a short lecture is given on the particulars of the biweekly assignment and then the students are allowed to work the assignment on their own initiative. All laboratory assignments must be completed and checked-off during the laboratory session to earn credit. Quizzes are given on alternate weeks and are also completed on the PC's. For example, the Mathcad quiz requires solving several problems in Mathcad and then submitting a diskette containing the Mathcad sheets for grading.

Observations and Challenges

Computer Tools for Engineers is not an easy course for the students or the instructor. For many students this is their first engineering course and they do not understand the time commitment and discipline needed for such a course. On the first day of class, students are asked to sign two identical forms as shown in Figure 6. One copy is to be kept by the student, the other to be returned to the instructor. Time is spent in class to discuss studying strategies for exams and quizzes. This time is well spent and also helps improve student attitudes.

I _____ understand that to be successful in *Computer Tools for Engineers* I must:

1) Attend and participate in all classes and labs.

2) Dedicate at least 6 hours per week outside of the classroom for reading, studying, and working exercises and assignments.

3) Write down a question and have it answered by a) myself, b) the instructor, c) a teaching assistant, or d) another student whenever I encounter something that I do not understand.

4) Have a positive mental attitude. This can be a difficult course, but with a positive attitude the material can be learned and applied.

Figure 6 - First day commitment form signed by the students

The problem of commitment is exacerbated by the feeling of many students that since their chosen engineering specialty is not computers and programming, this course is merely something to suffer through. The first day motivational lecture is intended to demonstrate that computers are relevant to everyone. It is made clear that the computer is as much a part of the professional engineer's desktop as is a telephone. Competent use of both of these tools is essential for professional success! The algorithmic

approach to problem solving with its emphasis on unambiguous detail does not come easily to many students. This is why we believe that additional time must be spent on problem solving techniques before any actual programming is undertaken.

Most of the students taking this course will not go on to write computer programs. They are more likely to need to be able to understand the logic of an existing program so that they can effectively use or modify that program. This course should therefore focus on providing a reading level of programming literacy without expecting much in the way of programming production. Any programming assignments or exam questions which seem "interesting" to the instructor should therefore probably be avoided.

There is no easy way to overcome the wide range of student backgrounds. The extremes of the computer experience dimension, the total computer novices (a surprisingly sizable group even within engineering majors) and the computer hobbyists, pose different problems. The former can usually be helped by the patient provision of individual assistance during laboratory sections, office hours, and help sessions. Attempts to deal with the lack of challenge and consequent, often disruptive, boredom seen in the latter group have been much less satisfying. We would welcome any really practical ideas for enhancing the class experience for those students.

Summary and Future Directions

In summary, *Computer Tools for Engineers* at the University of South Florida teaches incoming engineering students how to use computers to solve problems. Not only is a high-level programming language taught, but so are the use of a mathematics package for "formula crunching" and a spreadsheet for "number crunching". The basics of computer operation are reviewed. Design methods, including divide-and-conquer and successive refinement, are stressed as ways of "engineering thinking" that can be applied to solving problems in many engineering domains. Future directions for the course include finding ways to address the wide range of backgrounds of incoming students. An Instructional Development Grant proposal to be submitted to the University of South Florida is currently under preparation (see [2]). This proposal intends to develop self-paced tutorial and active-learning modules for improved presentation of the course. It is hoped that the developed learning tools can be widely distributed to other engineering colleges.

References

[1] G. Bjedov and P. Andersen, "Should Freshman Engineering Students be Taught a Programming Language," *Proceedings of the 26th Annual Frontiers in Education Conference*, pp. 90 - 92, November 1996.

[2] K. Christensen and D. Rundus, "A Technology-Supported 'Boot Camp' for Computer Tools for Engineers (EGN 2210)," Proposal for USF Instructional Development Grants Program, January 30, 1998.

[3] K. Christensen and A. Barrett, "Using the Internet to Enhance Off-Campus Engineering Education," *Proceedings of the 1997 ASEE Southeastern Section Meeting*, pp. 35 - 42, April 1997.

[4] F. Hosch, "Java as a First Language: An Evaluation," *SIGCSE Bulletin*, Vol. 28, No. 3, pp. 45 - 50, September 1996.

[5] Mathsoft, Mathcad 7.0, 1997. URL: *http://www.mathsoft.com/*.

[6] MATLAB 5.1, The MathWorks, Inc., 1997. URL: *http://www.mathworks.com/*.

[7] North Carolina State University College of Engineering, Civil Engineering Semester-by-Semester Curriculum Display, 1997. URL: *http://www.engr.ncsu.edu/academic/curricula/ce.html*.

[8] L. Nyhoff and S. Leestma, FORTRAN77 for Engineers and Scientists, fourth edition, Prentice Hall, Upper Saddle River, NJ, 1996.

[9] K. Pierce, L. Deneen, and G. Shute, "Teaching Software Design in the Freshman Year," *Proceedings of Software Engineering Education*, pp. 219 - 231, October 1991.

[10] Wolfram Research, Mathematica 3.0, 1997. URL: *http://store.wolfram.com/catalog/mathematica/*.

[11] Waterloo Maple, Maple V, Release 4, 1997. URL: *http://www.maplesoft.com/*.

[12] J. Zachery, et al., "An Entry-Level Course in Computational Engineering and Science," *SIGCSE Bulletin*, Vol. 27, No. 1, pp. 209 - 213, March 1995.

Appendix A - Course Syllabus for *Computer Tools for Engineers*

This course introduces all engineering majors to the use of the computer as a problem solving tool. Three representative tools are introduced. The tools are: a high-level mathematics package (Mathcad), a spreadsheet (Excel), and a high-level engineering programming language (FORTRAN). The material learned in this class will be of value to all engineering majors for the remainder of their undergraduate engineering curriculum and in their professional careers.

Instructor: Ken Christensen (*Assistant Professor*)
Office: ENB 319
Office Phone: 974-4761
Office FAX: 974-5456
Home Phone: 991-4401 (not after 10pm, please)
Email: *christen@csee.usf.edu*
WWW: *http://www.csee.usf.edu/~christen*

Office hours: To be announced.

Teaching assistants: To be announced.

Textbook: The textbook for this course is *FORTRAN77 for Engineers and Scientists*, fourth edition by Larry Nyhoff and Sanford Leestma.

Prerequisites: The only prerequisites are 1) the desire to learn, and 2) the discipline to work very hard. This is not a course for slackers.

Grades: Your grade will be computed as 6 quizzes at 5% each for 30%, 5 assignments (drop lowest of 6) at 4% each for 20%, midterm exam for 20%, and final exam for 30%. Final grade will be: A = 90% and above, B = 80% to 89%, C = 70% to 79%, D = 60% to 69%, F = less than 60%.

Late and missed work policy: Missed lab session cannot be made-up unless discussed prior with the instructor prior to the missed lab session. No missed quizzes or exams will be excused. Exceptional circumstances should be discussed with the instructor.

Attendance policy: The decision to attend a given class or lab is up to you. However, 100% attendance is expected if you are to pass this course. Look at it this way... when you take your first professional job, how many times in three months do you expect to be late and/or absent and still keep your job?

Academic honesty: If you are dishonest, you will be asked to leave the course and take an "F". On quizzes and exams, you do not give or receive help. For the laboratory exercises, you must submit independent work.

Note from the Provost: "Students who anticipate the necessity of being absent from class due to the observation of a major religious observance must provide notice of the date(s) to the instructor, in writing, by the second class meeting."

Appendix B - Course Outline for *Computer Tools for Engineers*

Week #1: Motivation and introduction
- Lecture - The computer as a tool for solving engineering problems. Motivation problem to compare business and engineering salaries solved with Mathcad.
- Lab - "PC 101" including Windows and DOS basics, using DOS Edit, Netscape, and Email.

Week #2: Mathcad (reading - Mathcad tutorial)
- Lecture - Basic calculation and graphing with Mathcad including a break-even analysis example
- Lab - ***Lab assignment #1***

Week #3: Mathcad and Excel (reading - Mathcad tutorial)
- Lecture - Built-in functions, vectors, matrices, equation solving, least squares fit, and symbolic math capabilities in Mathcad. Formula entry, addressing modes, and functions including IF function in Excel.
- Lab - ***Quiz #1***

Week #4: Excel (reading - supplementary material)
- Lecture - Introduction to equation editor. Spreadsheet advanced functions including trendlines, solver, and data analysis tools.
- Lab - ***Lab assignment #2***

Week #5: Computer internals (reading - chapter 1)
- Lecture - Von Neumann architecture, machine, assembly, and high-level languages. Introduction to the FORTRAN compiler.
- Lab - ***Quiz #2***

Week #6: Design methods (reading - chapter 1)
- Lecture - Flow charting, divide and conquer, and successive refinement as design methods.
- Lab - ***Lab assignment #3***

Week #7: FORTRAN (reading - chapter 2)
- Lecture - Basic structure of a program and documentation rules. Data types, variables and constants, and arithmetic.
- Lab - ***Quiz #3***

Week #8: ***Midterm exam*** (covers all topics from week #1 to #7)
- Lecture - No lecture this week
- Lab - ***Lab assignment #4***

Week #9: FORTRAN (reading - chapter 2 and 5.1)
- Lecture - Console and file input/output, FORMAT, and simple debugging.
- Lab - ***Quiz #4***

Week #10: FORTRAN (reading - chapter 3)
- Lecture - Logic expressions and program selection using IF
- Lab - ***Lab assignment #5***

Week #11: FORTRAN (reading - chapter 4)
- Lecture - Repetition using single and nested loops (DO, WHILE-DO, DO-UNTIL)
- Lab - ***Quiz #5***

Week #12: FORTRAN (reading - chapter 6 and 7)
- Lecture - Subprograms using FUNCTIONS and SUBROUTINES
- Lab - ***Lab assignment #6***

Week #13: FORTRAN (reading - chapter 8 and 9)
- Lecture - One and two dimensional arrays. Application examples of sorting and matrix multiplication.
- Lab - ***Quiz #6***

Week #14: Course review (reading - none required)
- Lecture - Overview of FORTRAN features not covered, course summary, and preparation for final exam
- Lab - No lab this week

Finals Week: ***Final exam*** (comprehensive)

DEWEY RUNDUS

Dewey Rundus received his Ph.D. in Experimental Psychology from Stanford University in 1971 and his M.S. in Computer Engineering from the University of South Florida in 1985. He is currently an Associate Professor and Associate Chair in the Department of Computer Science and Engineering at the University of South Florida. His research interests focus on Human / Computer Interaction and System Usability. His concern for undergraduate education has earned him a University of South Florida Outstanding Teaching Award and a State of Florida Teaching Incentive Program Award. His homepage is at, *http://www.csee.usf.edu/~rundus*.

KENNETH J. CHRISTENSEN

Kenneth J. Christensen received his Ph.D. in Electrical and Computer Engineering from North Carolina State University in 1991. He is currently an Assistant Professor at the University of South Florida. His research and teaching interests are in the areas of computer network systems, architectures, and performance modeling with an emphasis on integrating voice, video, and data in existing and future networks. He has over fifteen conference and journal publications, eight U.S. patents, and several patents pending all in the areas of computer networks and performance modeling. He was awarded a USF 1996/1997 Outstanding Undergraduate Teaching Award. He is a member of ASEE and ACM and a senior member of IEEE. His homepage is at, *http://www.csee.usf.edu/~christen*.