C S 160 - Introduction to Programming with Application to Engineering and Physical Science Problems

Proposed Number: C S 160
New Proposed Title: Introduction to Programming with Application to Engineering and Physical Science Problems
Corequisite(s): ENGR 106
Class Hours: 2
Lab Hours: 2
Credit Hours: 3

Catalog Description: Fundamental principles, concepts, and methods of programming (C & Matlab), with emphasis on applications in the physical sciences and engineering. Basic problem solving and programming techniques; fundamental algorithms and data structures; and use of programming logic in solving engineering and scientific problems. Students are expected to complete assignments in a collaborative learning environment.

Course Objective
Introduce students to structured programming as a tool to solve problems related to Engineering and the Physical Sciences. The language of choice will be C, but will include parallel references to the Matlab environment with which students are already familiar. The focus will be on the implementation of good algorithm design (utilizing the top-down design process, efficient implementations, intelligent debugging approaches, and finished products that abide by good programming and documentation standards) rather than the specific syntax of any given programming language or environment. It is believed that such specific detail of syntax is easily referenced in a text and that students who learn how to program can reference such syntax details as the need for another language arises. The demonstration of such will take near the end of the semester and the students are transferred out of C and Matlab and into another programming or scripting language. The emphasis of learning in the laboratory and on projects will be a constructivist environment as students approach problems collaboratively and provide peer feedback throughout the semester. It is believed that such a collaborative environment encourages informal learning that is beneficial to all students and represents how real-world problems are solved.

Course Topics
Below are the recommended topics for the semester and while syntax will require introduction, the focus will be on the implementation of such approaches to developing a solution to a problem.

<table>
<thead>
<tr>
<th>Week</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Course Introduction. Principles of working effectively and ethically in a team.</td>
</tr>
<tr>
<td>2</td>
<td>Algorithm design and analysis. Translating a written problem statement into a mathematical model that is suitable for algorithm development.</td>
</tr>
<tr>
<td>3</td>
<td>The program design process. Top down design, Stepwise refinement. Using a logical problem solving process for software development.</td>
</tr>
<tr>
<td>5</td>
<td>Compilers. Operating systems. C Language introduction.</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>6</td>
<td>Conditional statements. Repetition (loops).</td>
</tr>
<tr>
<td>7</td>
<td>Real-world engineering problems. (Repetition and Selection)</td>
</tr>
<tr>
<td>8</td>
<td>Modular programming. Functions. Revisit top down design.</td>
</tr>
<tr>
<td>9</td>
<td>Recursion</td>
</tr>
<tr>
<td>10</td>
<td>Arrays. Arrays and functions. Sorting and searching.</td>
</tr>
<tr>
<td>11</td>
<td>Real-world engineering problems. (Modular programming)</td>
</tr>
<tr>
<td>12</td>
<td>Characters, strings, streams, files.</td>
</tr>
<tr>
<td>13</td>
<td>Pointers. Memory management. Dynamic memory allocation and re-allocation.</td>
</tr>
<tr>
<td>14</td>
<td>Real-world engineering problems. (Dynamic memory)</td>
</tr>
<tr>
<td>15</td>
<td>Connections to Fortran, Java, C++, or Perl.</td>
</tr>
</tbody>
</table>

**Laboratory Component**

The two hour weekly laboratory will be utilized as a time in which students will utilize their knowledge of computer programming to solve simple Engineering and Science oriented problems.

The pedagogical approach will encourage learning in a constructivist environment, but will still attempt to hold individuals accountable for comprehension of current course material.

Laboratory assignments throughout the semester will require students to work collaboratively to implement a solution to a brief problem that demonstrates their knowledge and ability to work together. Prior to the next lab meeting students may be required to review the submission of several peers and offer comments and concerns as feedback to assist both the coder and reviewer in reading and writing solutions in the C programming language. It is believed that if beginners are regularly exposed to code, other than that presented in lecture, they will develop faster than if the only feedback received is to come from a course staff member.

At numerous points during the semester, particularly after the introduction of functions, the students will begin an alternative collaborative approach to solve larger problems that require each member of the group to contribute to the design of the solution and its implementation. The process of peer review may continue during such labs as individual students can view the end product created by other groups.

It is believed that learning will continue at a level that is more personal to the individual students that cannot be obtained by large lecture and individual assignments alone. Not only will students use the constructivist environment to learn computer programming but will do so while solving problems relevant to their potential academic and career disciplines.

**Project Requirements**

Project assignments outside of class may be reduced as many labs will now require preparation that goes beyond what may have been expected in the current implementation of the course.

The projects that remain will be substantial in length requiring the planning phase to be more of an effort than can be required in the time-constrained lab environment. The amount of code to be implemented should be extensive enough to require collaboration in order to be fully functional. An open-ended nature of projects will encourage the addition or alteration of the assignment such that the end product incorporates all that is expected with what students might
view as potential enhancements. To facilitate such open-endedness, students can consult with their TA when proposing changes not outlined in the project requirement.

It is believed that through close supervision of the collaborative environment in the weekly laboratory that the students will continue their close relationship and work together in the development of a sound and efficient algorithm to solve the problem.

Assignment of Students to Collaborative Teams
During the first lab meeting of the semester students will complete a survey that indicates their level of proficiency with computers and computing environments. Areas such as previous programming experience, operating system exposure, networking experience, and awareness of current topics such as piracy, privacy, viruses, and trends in computing will be evaluated to determine team assignment. The establishment of collaborative teams will initially be based on this assessment and teams will then be composed of students with a variety of computing backgrounds. It is believed that such a team composition will allow more experienced students to share their knowledge informally with those less experienced in their group.

Around the middle of the semester another reassessment of teams will be made, but this time the assignment of teams will be based on progress in the course and not directly on knowledge acquired prior to enrollment in the course. This will allow struggling students to be paired with stronger students and accommodate those teams left short-handed by the final withdrawals from the course.

As the course continues to evolve the number of times new teams are established may vary as best practices and student feedback are evaluated.
1 Course objective

Introduce CS and non-CS majors to the basic concepts of computer programming using Java as the programming language, to (a) allow the students to construct simple sharable applications in the domain of their major and (b) prepare the students for other Computer Science courses.

2 Course description

Introduction to computers and programming: Number representations. Primitive data types and operations. Basic control structures. Programming applets and applications using Graphical User Interfaces. Programming for detecting events and performing actions. Processing multimedia objects such as images and sounds. Throughout the course examples will be drawn from a variety of fields in the natural sciences.

Credit: 4 hours (3 class/2 lab)


Restriction: (a) CS majors who have passed CS 180 with a letter grade of C or better will not be allowed in CS 17x. (b) Non-CS majors who have passed any course on computer programming with a letter grade of C or better will not be allowed in CS 17x.

A weekly lecture plan is provided in Appendix A.

3 Course organization

The course consists of three components: lectures, laboratory assignments, and recitation. During each week there will be two 50 minute lectures, one 2-hour laboratory, and one 50 minute recitation. There will be two midterm examinations and one final examination. The final grade in the course will be determined based on relative performance of the students in the various components listed in the following table.
<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Percent weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-class quizzes</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>Laboratory assignments</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Projects</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>Examinations</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Final exam</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

### 4 Expected outcome

Students who have successfully passed this course with a grade of B or better will be

- prepared to enter CS 180 and learn additional concepts from OO programming.
- able to read and understand more advanced Java tutorials, such as the one [java.sun.com/tutorial](http://java.sun.com/tutorial), and write simple applications using GUIs.
- qualified to participate as undergraduate researchers and support programming activities in their individual disciplines.
- familiar with the applications of computer programming in a variety of domains including molecular biology, neuroscience, plant biology, physics, chemistry, orbital mechanics, statistics, and gaming.

### 5 Challenges and approach

Given the expected outcome of this course and the fact that the students in CS 17x are expected to consist of freshman who have had none or little exposure to computer programming, we face the following questions.

1. What programming language should be used?
2. What minimal set of concepts should the student be familiar with and what skill set should they acquire after having successfully gone through the course?
3. How should the course material be introduced so that the students enjoy the course and acquire a minimal set of skills?

Brief answers to these questions follow.

*Programming language:* (a) Java has become a popular language amongst scientists and engineers. A quick search on the Internet reveals the tremendous use of Java in the design of simple and complex applications in all fields of science and engineering. (b) CS uses
Java in CS 180. Given (a) and (b), we have decided to base CS 17x on Java. The latest version of Java, Java 2 at the time of preparing this proposal, will be used in all examples and exercises.

**Minimal concept/skill set:** (a) Familiarity with classes, objects, primitive data types, assignments, selection, loops, basic GUI objects, multimedia objects. (b) Ability to write simple GUI-based applets and applications in Java using multimedia objects.

**Approach to introducing the material:** (a) *Learning through experimentation.* Prior to discussing the fundamental concepts and linguistic elements used in constructing the code, students will be shown portions of the actual Java code and its trace in the class. During laboratory sessions, students will be asked to experiment by changing small portions of the code to understand how the program works. Laboratory exercises, projects, and other components of the evaluation will reinforce this approach. (b) *Interesting examples.* In class illustrative examples, laboratory exercises, and projects, will make use of a variety of application domains. For example, the application to display a gallery of images could use annotated images of the brain.

### 6 Similar approaches

A large number of institutions offer programming courses for non-CS majors. The number of such offerings using Java has increased dramatically over the years. Recently, notable attempts have been made at Georgia Tech by Mark Gudziale and at Colorado State by Elizabeth Boese. Though both instructors use multimedia objects in examples, Gudziale uses Jython and Boese uses Java.

Jython is a Java implementation of Python and is easier to learn and use than Java. However, a Jython programmer must learn from the beginning the syntax and OO related concepts in Java. Considering that one of the goals of CS 17x is to prepare a subset of the CS freshman class for CS 180, we decided against using Jython. Furthermore, though Jython is good for “quick” programming, it is seldom used in the development of multimedia tools that will be used and maintained over a long period and will be shared by researchers across many platforms. Java is finding an increasingly large number of such uses as in BrainImageJ at Stanford, NeoBio at King’s College, and CrazyQuant at the University of Washington. The popularity of Java in all the sciences is another reason for choosing Java for CS 17x.

### 7 Impact on CS 180

In the following we compare the coverage of CS 180 with that of the proposed CS 17x. Note that both courses use Java as the programming language. A “No” entry in the following table means that either the topic is not covered or that it is covered “lightly” only where necessary. Students who complete CS 17x with a grade of B or better will likely find few challenges in CS 180 unless the latter is revised to include more advanced topics or more
in-depth coverage (e.g., threads).

<table>
<thead>
<tr>
<th>Topic</th>
<th>Covered?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CS 180</td>
<td>CS 17x</td>
</tr>
<tr>
<td>Number representation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Primitive types</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Control structures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if-then-else</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>while-do</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>switch</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>for</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Exception handling</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Classes and objects</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Inheritance, polymorphism</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Wrapper classes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Parameters</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Applets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GUI</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multimedia objects</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Recursion</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Arrays (1-D)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Arrays (&gt;1-D)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Events and listeners</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic data structures</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
## A Lecture Plan

A 15-week course plan follows.

<table>
<thead>
<tr>
<th>Week</th>
<th>Topic</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1    | Basics            | 1       | - Programming, languages, editing, compiling.  
        |                   |         | - Number representation.  
        |                   |         | - Classes, objects, events: images, sounds, movies, and GUI objects.  
        |                   |         | - Sample applets and applications from biology, statistics, graphics, etc., downloaded over the Internet, will be shown to the students. |
| 2    | Applets           | 2.1, 2.2, 2.3 | - Constructing simple applets.  
        |                   |         | - Buttons and labels.  
        |                   |         | - Variables, assignment, and simple operations.  
        |                   |         | - Handling programming errors. |
| 3    | GUI design-1      | 3       | - Components, textual widgets.  
        |                   |         | - GUI construction examples. |
| 4    | Data types, opera- | 5.1, 5.2, 5.3, 10.1 | - Integers, floats, characters, and strings  
        | tions, and I/O     |         | - String to number conversion.  
        |                   |         | - Constructing simple applications. |
| 5    | Selection         | 6.1     | - Type boolean, conditions and the if-then-else construct. Construct applications that require if-then-else. |
| 6    | Loops             | 8.1     | - The while-do construct and applications that require its use. |
| 7    | Arrays            | 8.2, 8.3, 8.4 | - Single dimensional arrays.  
        |                   |         | - the for construct  
        |                   |         | - Sorting and searching. |
| 8    | GUI design-2      | 4       | - Containers, layouts, menus.  
        |                   |         | - GUI construction examples. |

*Continued on the next page.*
<table>
<thead>
<tr>
<th>Week</th>
<th>Topic</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Events and listeners-1</td>
<td>6.2, 6.3</td>
<td>• Event-driven programming. Event hierarchy.</td>
</tr>
<tr>
<td>10</td>
<td>Events and listeners-2</td>
<td>6.4</td>
<td>• Listener interfaces.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Loading, playing, and stopping sounds.</td>
</tr>
<tr>
<td>11</td>
<td>Program design</td>
<td>7</td>
<td>• Problem solving: decomposition into subproblems, classes and methods.</td>
</tr>
<tr>
<td>12</td>
<td>Images and drawing-1</td>
<td>12.1, 12.2</td>
<td>• Image processing and filters.</td>
</tr>
<tr>
<td>13</td>
<td>Images and drawing-2</td>
<td>12.3</td>
<td>• Drawing on canvas.</td>
</tr>
<tr>
<td>14</td>
<td>Animation</td>
<td>11.1, 11.2</td>
<td>• Threads, synchronization, time.</td>
</tr>
<tr>
<td>15</td>
<td>Course summary and miscellaneous topics</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B  Reference material for the instructor.


End of the proposal for CS 17x.
CS 240: Programming Laboratory (C)

List of Topics (By Week):

1. The C compilation model:
   Preprocessor, C compiler, some useful compiler options, using libraries, UNIX library functions, lint \em C program verifier

2. C program structure:
   Variables, input, output, constants, arithmetic operations, comparison operators, logical operators, order of precedence

3. Conditionals, loops:
   if, ? operator, switch, for, while, do-while, break, continue

4. Arrays and strings

5. Functions:
   Void functions, functions and arrays, function prototyping

6. Further data types:
   Structures, unions, coercion or type-casting, enumerated types, static variables

7. Pointers:
   Pointers and functions, pointers and arrays, arrays of pointers, multidimensional arrays and pointers, static initialization of pointer arrays, pointers and structures, common pointer pitfalls,
pointers to pointers, command line input, pointers to a function

8. Dynamic memory allocation and dynamic structures: Malloc, sizeof, free, linked lists

9. Low level operators and bit fields: Bitwise operators, bit fields

10. C preprocessor: #define, #undef, #include, #if, preprocessor compiler control, other preprocessor commands

11. C, UNIX, and standard libraries: Integer functions, random numbers, string conversion, searching and sorting, mathematics

12. Streams: Predefined streams, redirection, basic I/O, formatted I/O

13. String handling: Basic string handling functions, character conversions and testing

14. Writing larger programs: Header files, external variables and functions, advantages of using several files, how to divide a program between several files, organization of data in each file, "make" utility

15. File access and directory system calls: Directory handling functions, file manipulation routines, creating temporary files

16. Exception handling: Try block, catch handler, function throw list, exception handling and classes
17. Processes and threads:
Running UNIX commands from C, exec functions, fork(), wait(), exit(), interprocess communication (IPC), pipes, interrupts and signals, message queues, semaphores, shared memory, sockets, thread libraries, processes and threads, benefits of threads vs. processes, multithreading vs. single threading, synchronization

18. Features of C++:
Function prototypes, call by value, call by reference, default parameter values, function overloading, classes and objects, data members, member functions, constructor functions, default constructor functions, destructor function, member function prototypes, member function default arguments, inheritance, friend functions and classes, overloaded member functions, virtual functions, class scope, "this" pointer, object instantiation, access specifiers private and public, encapsulation, information hiding, private data members, public member functions, private member functions, array of class objects

2001.03
Course: CS 240
Title: Programming in C
Prerequisite(s): CS 180
Corequisite(s):
Credit: Class 2, Lab 2, Credit 3
Computer science majors have priority.

Catalog Description:
This course assumes that students already know Java. It teaches them to program in C. Includes Preprocessor, Pointers, Dynamic Memory Allocation, Streams, String Handling, Header files, Process Control, Interprocess Communication, Threads, Interrupts, Software Engineering Principles, and features of C++.

Changes made by Dennis Brylow. Changes in this revision:

(1) Revised to list 15 weeks, rather than the 18 currently in the canonical syllabus.

(2) Some topics re-ordered to better fit current Kernighan and Ritchie (The C Programming Language) textbook.

(3) One week of software engineering principles, design, and testing added to syllabus, per earlier suggestions.

List of Topics (By Week):

1. The C compilation model: Preprocessor, C compiler, some useful compiler options, using libraries, UNIX library functions

2. C program structure: Types, precision, constants, variables, simple I/O, arithmetic operations, comparison operators, logical operators, order of precedence

3. Representation and Control Flow: Underlying binary representation, bitwise operations, sizeof(), if-then-else, ? operator, switch, for, while, do-while, break, continue, nesting, lexical blocks

4. Functions: Function definitions, scope, static modifier, register modifier, Void functions, functions and arrays, function prototyping, Header files, preprocessor macros #define, #undef, #include, #if, preprocessor compiler control, other preprocessor commands

5. Arrays and strings

6. Software Engineering: Design, testing

7. Pointers: Pointers and functions, pointers and arrays, arrays of pointers, multidimensional arrays and pointers, static initialization of pointer arrays, pointers and structures, common pointer pitfalls, pointers to pointers, command line input, pointers to a function

8. Dynamic memory allocation: Malloc(), free(), word-alignment, structures
9. Dynamic structures and further data types: Unions, bit fields, coercion or type-casting, enumerated types, Lists, trees

10. Streams: Predefined streams, redirection, formatted I/O

11. File access and directory system calls: Directory handling functions, file manipulation routines, creating temporary files

12. C, UNIX, and standard libraries: Integer functions, random numbers, string conversion, searching and sorting, mathematics, Makefiles, header files, advantages of using several files, how to divide a program between several files, organization of data in each file

13. Processes and threads: Running UNIX commands from C, exec functions, fork(), wait(), exit(), interprocess communication (IPC), pipes, message queues, semaphores, shared memory

14. Interrupts and signals: interrupts and signals, sockets, synchronization

15. Features of C++: Function overloading, classes and objects, data members, member functions, constructor functions, default constructor functions, destructor functions, member function prototypes, member function default arguments, inheritance, friend functions and classes, overloaded member functions, virtual functions, class scope, "this" pointer, object instantiation, access specifiers private and public, encapsulation, information hiding, private data members, public member functions, private member functions, array of class objects
CS 483 --- Theory of Computation --- Fall 2004

Professor:

Greg N. Frederickson  
Office: CS 224  
Office hours: TBA  
email: gnf@cs.purdue.edu  
Time and location: MWF 10:30, in REC 227

Text:

M. Sipser,  

Motivation:

Why you should take this course (excerpted from the preface to Sipser's book)

CS 483 has changed. Here's how, and why.

Prerequisites:

CS 381 and CS 352, or the consent of the instructor.

Topics will include:

Turing machines and the Church-Turing thesis
Decidability, Halting problem
Reducibility, Undecidable problems
Decidability of logical theories, Kolmogorov complexity
Time classes: P, NP, NP-complete
Space classes: Savitch's theorem, PSPACE-completeness, NL-completeness
Hierarchy theorems, Approximation algorithms, Probabilistic algorithms
Applications of complexity to parallel computation and cryptography

**Course Work:**

| Written assignments (8-10) | 30% |
| Midterm exam (evening)    | 35% |
| Final Exam                | 35% |
Why CS 483 is important (and even exciting!)

Michael Sipser, in the preface to his textbook, *Introduction to the Theory of Computation*, explains why this course is important, and exciting:

Welcome!

You are about to embark on the study of a fascinating and important subject: the theory of computation. It comprises the fundamental mathematical properties of computer hardware, software, and certain applications thereof. In studying this subject we seek to determine what can and cannot be computed, how quickly, with how much memory, and on which type of computational model. The subject has obvious connections with engineering practice, and, as in many sciences, it also has purely philosophical aspects.

... 

Theory is relevant to practice. It provides conceptual tools that practitioners use in computer engineering. ... Confronted with a problem that seems to require more computer time than you can afford? Think back to what you learned about *NP-completeness*. Various application areas, such as modern cryptographic protocols, rely on theoretical principles that you will learn here.

Theory is also relevant to you because it shows you a new, simpler, and more elegant side of computers, which we normally consider to be complicated machines. The best
computer designs and applications are conceived with elegance in mind. A theoretical course can heighten your aesthetic sense and help you build more beautiful systems.

Finally, theory is good for you because studying it expands your mind. Computer technology changes quickly. Specific technical knowledge, though useful today, becomes outdated in just a few years. Consider instead the abilities to think, to express yourself clearly and precisely, to solve problems, and to know when you haven't solved a problem. These abilities have lasting value. Studying theory trains you in these areas.

Practical considerations aside, nearly everyone working with computers is curious about these amazing creations, their capabilities, and their limitations. A whole new branch of mathematics has grown up in the past 30 years to answer certain basic questions. Here's a big one that remains unsolved: If I give you a large number, say, with 500 digits, can you find its factors (the numbers that divide it evenly), in a reasonable amount of time? Even using a supercomputer, no one presently knows how to do that in all cases within the lifetime of the universe! The factoring problem is connected to certain codes in modern cryptosystems. Find a fast way to factor and fame is yours!
Why CS 483 has changed

When CS 483 was taught in Spring 2000 and Spring 2001, it had the following list of topics (taken from chapters 0, 1, 2, 3, 4, 5, 7, 8 of the text by Sipser):

- Proof techniques
- Finite automata, nondeterminism, regular expressions
- Nonregular languages
- Context-free grammars, pushdown automata
- Noncontext-free languages
- Turing machines
- Decidability, Halting problem
- Reducibility, undecidable problems
- Time classes: P, NP, NP-complete
- Space classes

The course was very successful, except that some students (rightly) pointed out that it duplicated material on regular languages and machines and context-free languages and pushdown automata that they had studied in CS 352. Furthermore, in the 2001-2002 academic year, the department introduced CS 182, which included material on proof techniques and an introduction to regular languages and machines and context-free languages and pushdown automata. At the same time, many students expressed an (unanticipated) enthusiasm for topics covered late in the course, especially space complexity and its relationship to games. This reflected the very accessible manner in which the Sipser text treated these topics. It now seems that good undergraduates are able to handle this material.

When CS 483 was taught in Fall 2002, the department was dealing with increasing enrollment pressures and the desire to teach courses in
emerging and expanding areas. Thus CS 483 and CS 584 were offered together. However, the mix of undergraduate and graduate students pushed the level at which the course was taught a bit too high for the undergraduates. This time CS 483 is offered by itself, so that the course will be taught at a level appropriate for advanced juniors and seniors.

**How CS 483 has changed**

CS 483 course will cover the following topics (taken from chapters 3, 4, 5, 6, 7, 8, 9, 10 of the text):

- Turing machines and the Church-Turing thesis
- Decidability, Halting problem
- Reducibility, Undecidable problems
- Decidability of logical theories, Kolmogorov complexity
- Time classes: P, NP, NP-complete
- Space classes: Savitch's theorem, PSPACE-completeness, NL-completeness
- Hierarchy theorems, Approximation algorithms, Probabilistic algorithms
- Parallel Computation, Cryptography

Thus the course will focus on computability theory (chapters 3-6) and complexity theory (chapters 7-10). The instructor reserves the right to trim topics as necessary.
CRITERIA FOR ACCREDITING COMPUTING PROGRAMS

Effective for Evaluations During the 2004-2005 Accreditation Cycle

Incorporates all changes approved by the ABET Board of Directors as of November 1, 2003

ABET
Computing Accreditation Commission
ABET, Inc.
111 Market Place, Suite 1050
Baltimore, MD 21202

Telephone: 410-347-7700
Fax: 410-625-2238
E-mail: accreditation@abet.org
Website: www.abet.org
### TABLE OF CONTENTS

INTRODUCTION ........................................................................................................ iv

CRITERIA FOR COMPUTER SCIENCE PROGRAMS ................................................. 1
  - Objectives and Assessments ................................................................. 1
  - Student Support ................................................................................. 1
  - Faculty ............................................................................................... 2
  - Curriculum .......................................................................................... 2
  - Laboratories and Computing Facilities .............................................. 4
  - Institutional Support and Financial Resources ................................. 4
  - Institutional Facilities ......................................................................... 5

CRITERIA FOR INFORMATION SYSTEMS PROGRAMS ...................................... 7
  - Objectives and Assessments ................................................................. 7
  - Students ............................................................................................. 7
  - Faculty ............................................................................................... 8
  - Curriculum .......................................................................................... 8
  - Technology Infrastructure ................................................................. 10
  - Institutional Support and Financial Resources ................................. 10
  - Program Delivery ............................................................................... 11
  - Institutional Facilities ......................................................................... 11

PROPOSED CHANGES TO THE CRITERIA .......................................................... 13

Requests for further information about ABET, its accreditation process, or other activities may be addressed to the Accreditation Director, ABET, Inc., 111 Market Place, Suite 1050, Baltimore, MD 21202 or to accreditation@abet.org.
Criteria for Accrediting Computer Science Programs
Effective for Evaluations during the 2004-2005 Accreditation Cycle

I. Objectives and Assessments

Intent
The program has documented, measurable objectives, including expected outcomes for graduates. The program regularly assesses its progress against its objectives and uses the results of the assessments to identify program improvements and to modify the program’s objectives.

Standards
I-1. The program must have documented, measurable objectives.
I-2. The program’s objectives must include expected outcomes for graduating students.
I-3. Data relative to the objectives must be routinely collected and documented, and used in program assessments.
I-4. The extent to which each program objective is being met must be periodically assessed.
I-5. The results of the program’s periodic assessments must be used to help identify opportunities for program improvement.
I-6. The results of the program’s assessments and the actions taken based on the results must be documented.

II. Student Support

Intent
Students can complete the program in a reasonable amount of time. Students have ample opportunity to interact with their instructors. Students are offered timely guidance and advice about the program’s requirements and their career alternatives. Students who graduate the program meet all program requirements.

Standards
II-1. Courses must be offered with sufficient frequency for students to complete the program in a timely manner.
II-2. Computer science courses must be structured to ensure effective interaction between faculty/teaching assistants and students in lower division courses and between faculty and students in upper division courses.
II-3. Guidance on how to complete the program must be available to all students.
II-4. Students must have access to qualified advising when they need to make course decisions and career choices.
II-5. There must be established standards and procedures to ensure that graduates meet the requirements of the program.

III. Faculty

Intent

Faculty members are current and active in the discipline and have the necessary technical breadth and depth to support a modern computer science program. There are enough faculty members to provide continuity and stability, to cover the curriculum reasonably, and to allow an appropriate mix of teaching and scholarly activity.

Standards

III-1. There must be enough full-time faculty members with primary commitment to the program to provide continuity and stability.

III-2. Full-time faculty members must oversee all course work.

III-3. Full-time faculty members must cover most of the total classroom instruction.

III-4. The interests and qualifications of the faculty members must be sufficient to teach the courses and to plan and modify the courses and curriculum.

III-5. All faculty members must remain current in the discipline.

III-6. All faculty members must have a level of competence that would normally be obtained through graduate work in computer science.

III-7. Some full-time faculty members must have a Ph.D. in computer science.

III-8. All full-time faculty members must have sufficient time for scholarly activities and professional development.

III-9. Advising duties must be a recognized part of faculty members' workloads.

IV. Curriculum

Intent

The curriculum is consistent with the program's documented objectives. It combines technical requirements with general education requirements and electives to prepare students for a professional career in the computer field, for further study in computer science, and for functioning in modern society. The technical requirements include up-to-date coverage of basic and advanced topics in computer science as well as an emphasis on science and mathematics.
Standards

Curriculum standards are specified in terms of semester hours of study. Thirty semester hours generally constitutes one year of full-time study and is equivalent to 45 quarter hours. A course or a specific part of a course can only be applied toward one standard.

General

IV-1. The curriculum must include at least 40 semester hours of up-to-date study in computer science topics.

IV-2. The curriculum must contain at least 30 semester hours of study in mathematics and science as specified below under Mathematics and Science.

IV-3. The curriculum must include at least 30 semester hours of study in humanities, social sciences, arts and other disciplines that serve to broaden the background of the student.

IV-4. The curriculum must be consistent with the documented objectives of the program.

Computer Science

IV-5. All students must take a broad-based core of fundamental computer science material consisting of at least 16 semester hours.

IV-6. The core materials must provide basic coverage of algorithms, data structures, software design, concepts of programming languages, and computer organization and architecture.

IV-7. Theoretical foundations, problem analysis, and solution design must be stressed within the program’s core materials.

IV-8. Students must be exposed to a variety of programming languages and systems and must become proficient in at least one higher-level language.

IV-9. All students must take at least 16 semester hours of advanced course work in computer science that provides breadth and builds on the core to provide depth.

Mathematics and Science

IV-10. The curriculum must include at least 15 semester hours of mathematics.

IV-11. Course work in mathematics must include discrete mathematics, differential and integral calculus, and probability and statistics.

IV-12. The curriculum must include at least 12 semester hours of science.

IV-13. Course work in science must include the equivalent of a two-semester sequence in a laboratory science for science or engineering majors.

IV-14. Science course work additional to that specified in Standard IV-13 must be in science courses or courses that enhance the student’s ability to apply the scientific method.

Additional Areas of Study
IV-15. The oral communications skills of the student must be developed and applied in the program.

IV-16. The written communications skills of the student must be developed and applied in the program.

IV-17. There must be sufficient coverage of social and ethical implications of computing to give students an understanding of a broad range of issues in this area.

V. Laboratories and Computing Facilities

Intent

Laboratories and computing facilities are available, accessible, and adequately supported to enable students to complete their course work and to support faculty teaching needs and scholarly activities.

Standards

V-1. Each student must have adequate and reasonable access to the systems needed for each course.

V-2. Documentation for hardware and software must be readily accessible to faculty and students.

V-3. All faculty members must have access to adequate computing facilities for class preparation and for scholarly activities.

V-4. There must be adequate support personnel to install and maintain the laboratories and computing facilities.

V-5. Instructional assistance must be provided for the laboratories and computing facilities.

VI. Institutional Support and Financial Resources

Intent

The institution's support for the program and the financial resources available to the program are sufficient to provide an environment in which the program can achieve its objectives. Support and resources are sufficient to provide assurance that the program will retain its strength throughout the period of accreditation.

Standards

VI-1. Support for faculty must be sufficient to enable the program to attract and retain high-quality faculty capable of supporting the program's objectives.

VI-2. There must be sufficient support and financial resources to allow all faculty members to attend national technical meetings with sufficient frequency to maintain competence as teachers and scholars.
VI-3. There must be support and recognition of scholarly activities.

VI-4. There must be office support consistent with the type of program, level of scholarly activity, and needs of the faculty members.

VI-5. Adequate time must be assigned for the administration of the program.

VI-6. Upper levels of administration must provide the program with the resources and atmosphere to function effectively with the rest of the institution.

VI-7. Resources must be provided to acquire and maintain laboratory facilities that meet the needs of the program.

VI-8. Resources must be provided to support library and related information retrieval facilities that meet the needs of the program.

VI-9. There must be evidence that the institutional support and financial resources will remain in place throughout the period of accreditation.

VII. Institutional Facilities

Intent

Institutional facilities including the library, other electronic information retrieval systems, computer networks, classrooms, and offices are adequate to support the objectives of the program.

Standards

VII-1. The library that serves the computer science program must be adequately staffed with professional librarians and support personnel.

VII-2. The library’s technical collection must include up-to-date textbooks, reference works, and publications of professional and research organizations such as the ACM and the IEEE Computer Society.

VII-3. Systems for locating and obtaining electronic information must be available.

VII-4. Classrooms must be adequately equipped for the courses taught.

VII-5. Faculty offices must be adequate to enable faculty members to meet their responsibilities to students and for their professional needs.
STAT 511 Statistical Methods

Applied statistics for students with a calculus background. Some probability theory is presented but applicable statistics is emphasized. May lead to STAT 512 or STAT 513. Taken by both undergraduate and graduate students from many subject areas, especially engineering and physical sciences.

Text

** Tentative - check with Purdue bookstores for verification **

Spring 2004 text is:


Outline

1. Data (sample vs. Population, histograms, sample mean, median, variance, standard deviation)
2. Elementary probability (axioms, basic rules; little attention to counting and conditional probability)

3. Random variables (discrete vs. Continuous, moments)

4. Binomial, hypergeometric, Poisson distributions, Poisson approximation to binomial

5. Normal distribution, normal approximation to binomial

6. Sampling distributions, distribution of mean, central limit theorem

7. Confidence intervals for means

8. Confidence intervals for proportions

9. Tests of hypotheses for means and proportions

10. Regression and correlation

11. One way analysis of variance

12. Contingency tables

13. Exams
- 82 Total Recommended
  - 23 Fellowships offers
  - 51 TA offer
  - 1 RA offer
  - 7 Admission only

- 8 Accepted
  - 3 Fellows
  - 4 TAs
  - 1 Admission only

- 9 Declined
  - 7 Fellows
  - 2 TAs

- 65 Open

- 23 Fellowship Offers
  - 8 females, 4 minorities
    - 3 accepted, 7 declined, 13 open
  - 10 GAANN to US students
  - 4 Other to US students
  - 9 Andrew/Rosa to Internationals
    (4 China, 2 India, 2 Europe, 1 Korea)

- 51 TA Offers
  - 7 females, 1 minority
    - 4 accepted, 2 declined, 45 open
  - 22 US
  - 10 Indians
  - 9 Chinese
  - 10 Other