A FOUNDATIONS COURSE FOR A DEVELOPING COMPUTER SCIENCE PROGRAM

By Mark Benard
Computer Science Department
Tulane University
New Orleans, LA 70118

I. Introduction

The demand of industry for personnel educated in computer science has created a two-edged problem for colleges and universities. Student demand for computer science courses has increased as the use of computers has increased. In addition, industry has attracted computer science faculty and potential faculty away from academics so that there is a very real shortage of faculty members who have strong backgrounds in computer science. Thus the increased demands for courses coupled with the shortage of staffing for such courses have caught institutions of higher education in a squeeze and have caused them to examine the extent that they participate in computer science education.

Since colleges and universities are competing with one another for students more now than in the past, they cannot ignore strong student interests in any area of study. Courses in computer science are not just being demanded by students who are aiming for computer-related careers. Students with majors in such diverse fields as business, psychology, and engineering are being told that some computer science coursework is essential for careers in these fields. In addition, more students majoring in the humanities are choosing to take computer science courses as electives for "insurance" in case that they are unable to pursue careers related to their majors.

Most colleges and universities without full-fledged computer science programs meet the general demand by offering courses which are staffed by faculty from other fields who have some experience with computer programming. Because of the limited expertise of the staff, such course offerings may be restricted to courses such as Introduction to Data Processing and courses in which programming languages such as FORTRAN and COBOL are studied. This solution makes good use of faculty resources to meet some basic needs for service courses in computer science, particularly when the faculty member involved is supported and encouraged by the college in refining the skills needed to teach such courses.

Schools which go a little further and try to prepare students for computer-related careers may also have an assembly language course and a numerical analysis course in their curriculum. If the assembly language course focuses on assembly language programming and not on computer organization, then the student is exposed to very little computer science in this curriculum, no matter how many programming languages he or she may learn to use. Such a combination of course offerings does not provide an education which will enable the graduate to contribute to solving computer-related problems as they change over the next forty years. In order to adapt to the changing demands which are already apparent in today's jobs, the computer professional needs a broad background in the fundamentals of computer science, which clearly goes beyond programming. Furthermore, training in computer programming and nothing beyond is not consistent with the principles of a college education. Such training is more suitable for a vocational-technical institute both from the standpoint of breadth of coverage and that of the depth of understanding.

II. The Role of the Foundations Course

This paper discusses a course, referred to as Foundations, which has been used to partially satisfy the need for a broad program in computer science in a situation where staffing is limited. This course was introduced at Tulane University in 1974 and was taught until recently when a full-fledged major program was established. At the time that it was introduced, the only other computer science courses that were being taught involved programming in several languages (including assembler), some

 0-89791-036-2/81/0200/0188 $00.75

applications, and numerical analysis. My experience in teaching the course for five years has led me to believe that it is an ideal course for a school which wishes to offer something beyond programming but which is restricted by the size and expertise of its faculty.

The Foundations course covers several topics, each of which is related to discrete mathematics. These topics are data structures, formal languages, graph theory and boolean algebras. While the course is organized around the theoretical subjects, it includes a great variety of applications to computer science. The course also includes a major project which ties in several of the topics studied and provides a design and programming problem that is more complex than those seen by students in preceding courses. The project involves the construction of an interpreter for a simple algebraic language. More details are given on the exact contents of the course and on the interpreter project in Sections IV thru VI.

Most students came into the course with a view that computer systems are black boxes. They had very little idea of what happened to their programs during compilation and execution. An introduction to the translation process for programming languages, reinforced by the interpreter project, changed that black-box view very quickly. The study of the use and implementation of some simple data structures and the study of applications of boolean algebras such as the logical design of adders gave them some insight into the organization of computer systems. Students who took the course during its first two or three years of existence were mostly participants in a combined mathematics/computer science program. Feedback from these students indicate that the course did accomplish its goals of giving them a broad view of computer science (which resulted in more opportunities for them in the job market) and of providing them with the background needed for further study in the field.

This course should be appealing to a college with limited faculty expertise in computer science. With a moderate amount of reading and preparation, a faculty member who is familiar with discrete mathematics and who is experienced in programming can teach such a course. In addition, teaching such a course can also provide an incentive to delve more deeply into computer science. Hence, the course might provide the school with a springboard for developing the faculty expertise needed for an expanded computer science program.

III. Further Course and Faculty Development

The Foundations course can also be used as a transition course in the development of a minor or major in computer science. Data Structures would probably be the most desirable next course to be added to the curriculum. A faculty member with experience with the Foundations course should be able to develop Data Structures, which is a natural spin-off from Foundations. In the case of Tulane, we inserted a sophomore-level Data Structures course and altered the junior-level Foundations course to cover other material in more depth. Another option would be to build on Foundations and schedule Data Structures to follow it.

A minor program might easily be built from this point. Such a minor might include two introductory programming courses, asssembly language, Data Structures, Foundations, and some coverage of the organization of computer systems and system software. A major program would require considerably more courses, but several can be regarded as spin-offs from Foundations. Courses in programming languages and their translation, analysis of algorithms, and discrete mathematics could be inserted in place of the Foundations course.

I have referred to the role of the Foundations course in faculty development. The current situation in which many computer science courses are being taught by those acquainted only with computer programming is hopefully a temporary one. Computer science should be taught by computer scientists for the same reason that history should be taught by historians. College professors should have a thorough understanding of what they are teaching and should be knowledgeable of new developments in the field; it is also important that they have an overview of the field which goes beyond mere knowledge of the course being taught. However, many of the non-experts who are currently teaching computer science may provide a key source of the computer scientists that are badly needed in academics. Often these people are already dedicated to teaching but may not find much incentive to learn about computer science beyond the subject of "programming" if they do not expect to be teaching anything other than the introductory courses which are primarily service courses. However, if they are given a chance to teach a course like Foundations, they may be stimulated to become more deeply involved in computer science. With the support of their institutions in the form of travel to conferences and short courses, and paid leave to study with computer scientists elsewhere, many of these people may be transformed into contributing computer scientists in a fairly short period of time.

IV. Contents of the Course

The Foundations course is divided into two main parts; data structures and formal languages are covered in the first part while graph theory and boolean algebras comprise the second part. The theme that dominates the first part is the specification and translation of languages. In an introductory section on mathematical induction, an algorithm for determining whether a string of parentheses is properly paired is presented. The algorithm is a simple counting algorithm,

assigning the value of +1 to left parentheses and -1 to right ones, but the proof that it works involves looking at how strings of properly paired parentheses can be generated. That is, a grammer which generates the language of "properly paired parentheses" is informally presented. Much later, after formal languages have been introduced, that grammar is discussed again. The language of reverse Polish expressions is also studied prior to the section on formal languages. Translation algorithms for infix to reverse Polish are used to illustrate applications of stacks and binary trees. Thus the student deals with grammars and languages in several situations before they are presented more rigorously in the section of formal languages.

The section on graph theory does not include any indepth discussion of prominent applications in computer science. Because there are so many definitions and concepts to understand before one can use graph theory, this section concentrates on those basics. The applications presented are to problems for which a computer solution is being sought, such as problems that involve resource allocations (e.g., PERT and CPM). Hence, graph theory is viewed as a mathematical tool which can be used to reduce a problem to a form that is appropriate for computer solution. Direct applications to computer science, such as flow analysis of programs and analysis of computer networks, are somewhat too sophisticated for this course and are only briefly mentioned.

One application of boolean algebras appears very prominently in the second part of the course, the application to the design of digital logic circuits. This gives the student some exposure to the hardware-level organization of computers. There are other reasons for including a thorough discussion of boolean algebras. Mathematical logic and the propositional calculus are useful tools for a computer scientist, and the study of boolean algebras provides an introduction to this material.

Another topic which is included actually appears spread throughout the entire course. This topic is the analysis of algorithms. Numerous algorithms are presented in each section. These algorithms are carefully presented so that their validity can be discussed. In cases where a simple proof (using tools such as mathematical induction) of the correctness of the algorithm exists, the proof is formally presented. Recurring techniques, such as the use of depth-first and breadth-first searches in graphs, are pointed out to illustrate that algorithm design often involves the adaption of well-known methods in new situations.

Several different books ([1], [3], [5]) have been used as textbooks for the course, but none of these was followed very closely. Selected readings from these books and several other references ([2], [4]) can be used to supplement classroom lectures.

V. Course Outline

A. Mathematical preliminaries
   1. Mathematical induction
   2. Set theory

B. Linear data structures
   1. Arrays
   2. Linked lists
   3. Stacks
   4. Applications of stacks: subroutines; conversion from infix to reverse Polish notation; execution of reverse Polish expressions

C. Binary trees
   1. Traversal algorithms
   2. Implementation
   3. Applications: relationship between infix and reverse Polish notation; symbol tables

D. Formal languages
   1. Formal grammars and their languages
   2. BNF notation
   3. Syntax trees and parsing
   4. Simple precedence grammars

E. Graph theory
   1. Undirected graphs - basic definitions
   2. Trees and spanning trees; spanning tree algorithms using depth first search
   3. Cycle basis; algorithm based on the spanning tree
   4. Degrees of connectivity
   5. Directed graphs - basic definitions
   6. Applications of networks using the max flow - min cut algorithm and shortest path algorithms (PERT, CPM)

F. Boolean algbras
   1. Characterization of (finite) boolean algebras
   2. Boolean functions
   3. Canonical forms of boolean polynomials
   4. Minimization using Karnaugh maps
   5. Applications to digital logic design (example: a full adder-subtractor)

VI. The Interpreter Project

While additional programming assignments have been given to implement some of the algorithms studied, the interpreter project has been used as the primary programming work. It ties together the material in the first part (sections B,C and D) of the course and gives the student a better understanding of language translators.

The language to be interpreted consists of algebraic assignment statements, with integer values and single character variable names. Optionally, the language can include an IF-THEN-ELSE construct or multicharacter variable names.

The project assignment is handed out at the first class to give the students a goal to focus on. Of course, the students cannot fully understand the project at that time and it is actually broken down into a series of assignments to be done when appropriate. Stack algorithms are implemented after stacks are introduced, symbol table routines are implemented after binary trees are studied, and the parsing module is implemented after the section on formal languages is completed. The most difficult part of that project for most students is the identification of the various syntax errors that can occur. Certain types of errors (e.g., improperly paired parentheses) can be detected with the usual precedence-based algorithm which can be used to convert an assignment statement to reverse Polish notation, but transition tables of some type are needed in either the lexicographic scanning module or the parsing module to detect most errors. When time has permitted, this problem has been used to motivate a brief study of finite state machines and automata.

VII. Conclusions

The Foundations course, as outlined here, can be a valuable asset to a small or developing computer science program. It exposes the students to many apects of computer science which are not apparent to those who have studied only "programming" previously; yet it provides a natural transition between "programming" and a more indepth study of computer science. The course also presents a number of tools which are needed for the study of more advanced topics.

In addition, this course can be taught by a faculty member with minimal expertise in computer science. It can also provide the incentive for such a person to pursue a deeper understanding of computer science. With the help of the college or university, this person could develop the expertise to teach a wide variety of computer science courses and hence contribute to a more comprehensive computer science program at that institution.

REFERENCES

[1] A.T. Berztiss, "Data Structures," Second Edition, Academic Press, 1975.

[2] D.E. Knuth, "The Art of Computer Programming," 3 volumes, Addison-Wesley.

[3] R.R. Korfhage, "Discrete Computational Structures," Academic Press, 1974.

[4] C.W. Marshall, "Applied Graph Theory," Wiley-Interscience, 1971.

[5] J.P. Tremblay and R. Manohar, "Discrete Mathematical Structures with Applications to Computer Science," McGraw-Hill, 1975.