12-1-1992

ACM model high school computer science curriculum.

Susan M. Merritt

SCHOOL OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

TECHNICAL REPORT

Number 57, December 1992

Draft Report

ACM Model High School Computer Science Curriculum

by the

Curriculum Task Force of the ACM
Pre-College Education Committee

Susan M. Merritt
Committee Chairperson

School of CS and IS
Pace University
1 Martine Avenue
White Plains, NY 10606

Carol E. Wolf
Draft Report Co-Editor

Dept. of Computer Science
Pace University
New York, NY 10038

Charles Rice
Draft Report Co-Editor

Computer Science Laboratory
The Dalton School
108 East 89th Street
New York, NY 10128

PACE UNIVERSITY

Susan M. Merritt is Professor of Computer Science and Dean of the School of Computer Science and Information Systems at Pace University.

Dr. Merritt holds the Ph.D. in computer science from NYU's Courant Institute and has authored many scholarly papers, including "an Inverted Taxonomy of Sorting Algorithms" which appeared in the Communications of the ACM and, with Dewar and Sharir, "Some Modified Algorithms for Dijkstra's Longest Upsequence Problem" which appeared in Acta Informatics. She continues to do research in algorithm development as well as in the cultural and educational impact of information technology.

Carol E. Wolf received a BA from Swarthmore College and an MA and Ph.D. from Cornell University, all in mathematics. She taught at the State University College at Brockport and Iowa State University before joining Pace University in 1986 as an Associate Professor of Computer Science.

While at Swarthmore College, she worked as a summer intern on the UNIVAC I, one of the first commercial digital computers. Her work at Cornell was in partial recursive functions and recursively enumerable sets. At Iowa State she became interested in cellular automata and graph grammars. There she cultivated a general expertise in computer science, a field not yet developed when she was in graduate school, through several years of intensive study.

At Pace she serves as Chairperson of the New York Computer Science Department as well as Chairperson of the joint New York and Westchester Computer Science Curriculum Committee.

≈ ≈ ≈

Dean Merritt has been Chairperson of the ACM's Pre-College Education Committee for the past four years. The present report is a late (but not final) product of the task force she formed in 1989 to prepare recommendations on the presentation of computer science in secondary schools. In addition to Dean Merritt, Dr. Wolf, and Mr. Rice; those who worked on it include *Charles J. Bruen* from the Bergenfield High School in New Jersey, *J. Philip East* from the University of Northern Iowa, *Darlene Grantham* from the Montogomery County School System in Maryland, *Viera K. Proulx* from Northeastern University in Boston, and *Gerry Segal* from the Bank Street College of Education in Manhattan.

Draft Report

ACM Model High School Computer Science Curriculum

Task Force of the Pre-College Committee of the Education Board of the ACM

Susan Merritt, Charles J. Bruen, J. Philip East, Darlene Grantham
Charles Rice, Viera K. Proulx, Gerry Segal, Carol Wolf

Introduction

Computer technology has had a profound effect on our society and world. Every citizen should have some familiarity with this technology and its consequences in the home, school, work place and society. Yet since the details of the technology change from day to day, the study of the subject has been difficult and so should concentrate upon those fundamental scientific principles and concepts which underlie the field.

The Association for Computing Machinery (ACM) made recommendations for college level computer science curricula in 1968 [1] and 1978 [2]. The latter was followed in 1981 [9] and 1985 [8] by recommendations for high school computer science curricula. A joint task force of the ACM and the Institute of Electrical and Electronics Engineers - Computer Society (IEEE-CS) chaired by Allen Tucker and Bruce Barnes has now published new proposals for college curricula [11] and [12]. It therefore seems appropriate to take another look at the high school curriculum.

In 1989 the Pre-college Committee of the Education Board of the ACM formed a task force to consider the college proposals and prepare recommendations for a new high school computer science course. The task force is composed of high school, college and university faculty members and computer administrators. This report is the third draft. The previous two [5, 6] have been extensively reviewed and many recommendations have been incorporated.

Background

Scientific studies are an important part of everyones education. Computer science is unique because it is not restricted to the experimental limitations of the real world. Most computing activities must be carried out in an artificial environment which is designed by people and not by nature. The ability to use abstraction to build these models is an essential skill for all to master. These artificial environments are diverse, ranging from organizing data in a database which supports the operations of an institution to defining a virtual world.

Most people require mental models to feel comfortable in these new worlds. The difficulties which many experience when programming their VCRs illustrate this. Background knowledge of fundamental concepts is as important as specific operational techniques. Pre-college education is a natural place to acquire this background.

The original charge to the task force was to develop a computer science curriculum for secondary schools. It quickly became apparent that to accomplish this the task force had to first focus on a clearer definition of computer science and its value to society. In the course of developing this definition, the task force realized that there were many ways to approach and present computer science.

The task force next identified specific model courses which seem to introduce high school students to this field successfully. University curricula and proposals were also examined to build this definition. The goal of these investigations was to arrive at an agreed core definition for computer science which all approaches would use.

The intention is that these courses would be similar in their scope, depth, breadth and methodology to typical high school science courses. They should serve all students in the same way that introductory biology, chemistry and physics courses do. These courses present the background of the field, discuss important issues, study and solve problems in the field, and apply mathematics to problem solution.

The intended level for a computer science course is approximately tenth grade. It could serve as a minimal

requirement in itself or as a prerequisite for advanced computing courses. Student preparation would usually include first year algebra and some computing experience.

The focus of the course is on fundamental concepts of computer science. Several model course curricula show how these concepts can be presented in different settings that illustrate their applications. As much as possible, students should conduct experiments and write programs that demonstrate the abstract concepts, confirm the theory and expose the power of computers.

Computer literacy courses which have evolved could be included in this definition of computer science by adjusting their contents to contain essential core topics. An example of such a model will be included in an appendix.

Different options in the method of delivery permit the course to be adopted in a wide variety of schools, those with extensive computing facilities as well as those with minimal computer availability.

The introduction of computers into our society has brought with it fundamental changes in thinking, and in the organization and use of information in the workplace and in daily life. These changes are based both on theoretical advances in computer science and on technological advances of computer architectures. Courses based on these models study both the fundamentals of theory and technology that made the computer revolution possible, as well as the impact of this revolution on today's society - from both a technical and a social point of view.

Motivation

The current situation in American education makes this an appropriate time to propose this course.

- In 1983, a national commission issued a report on A Nation at Risk [7], which recommended a required computer science course and achievement test.
- The ACM and IEEE-CS [11] and [12] have again considered the college curricula.
- The emphasis on more rigor and basics in education dictates a rethinking of the high school curriculum.
- There appears to be a declining interest in mathematics and science in high schools at a time of greater national emphasis on technology issues.
- A declining interest toward computer science by all students and especially by women and minorities has also been observed at the university level.
- There has been a lack of clarity as to what constitutes computer science.

In 1988, an ACM task force chaired by Peter Denning published a report [3] that offered a clear definition of the computing disciplines. This task force was followed by a joint ACM/IEEE-CS task force chaired by Allen Tucker and Bruce Barnes [11] and [12]. Its report, just published, details the content both required and elective of the computer science curriculum.

The 1988 and 1991 reports identify the following subject areas for the courses in a curriculum in computing and the recommended course hours.

Algorithms and data structures	47 hours
Architecture	59 hours
Artificial intelligence and robotics	9 hours
Database and information retrieval	9 hours
Human-computer communication	8 hours
Numerical and symbolic computing	7 hours
Operating systems	51 hours
Programming languages	46 hours
Software methodology and engineering	44 hours
Social, ethical, and professional issues	11 hours
Intro. to a Programming Language	12 hours

Course Topics and Models

In designing a high school curriculum to introduce computer science, it was desired to be consistent with ACM/IEEE-CS topics which define computer science and their relative importance. However the curriculum also must reflect the abilities, interests, and school situation of high school students. The above topics were used as a guideline, but were combined in a different grouping to develop this curriculum.

The task force identified seven areas which provide a broad introduction to computer science for high school students. Algorithms, programming languages, operating systems, computer architecture, and social, ethical and professional context are included with modified content

from the ACM/IEEE-CS report. Advanced topics include most of the other topics from the report. A new area, Computer applications, was added to reflect how some computer science topics are introduced at the high school level.

Recommended course hours for these seven areas are not included since the times allotted depend on the implementation model.

The seven areas are:

Algorithms
Programming Languages
Operating Systems and User Support
Computer Architecture
Social, Ethical, and Professional Context
Computer Applications
Advanced Applications

Recognizing the diversity of school systems, teachers and students, the task force has identified several different approaches for presenting these computer science areas. These approaches require the seven areas be introduced in different ways. The first five areas have been divided into essential, recommended, and optional topics. Each approach should cover the essential topics, with students working on examples, exercises, projects, and reports. The teacher should introduce as many of the recommended topics as possible, some possibly only on a survey level. Other recommended and optional topics should be covered as appropriate.

The last two areas include topics which can be introduced to broaden students' understanding of computer science by using computer applications and investigating more advanced topics. It is recommended that each approach include at least one topic from these last two areas to illustrate the current state of the art in some area of computer science. Additional topics from these areas may be covered as time permits.

Essential topics	**Recommended topics**	**Optional topics**
1. Algorithms		
Algorithms in daily life	Methods to test algorithms for flaws	Characteristics of an algorithm
Techniques used to design and represent algorithms	Basic data structures	Complexity of algorithms
		Limits of computability
Examples of important algorithms		
Fundamental problem solving concepts		
2. Programming Languages		
Introduction to a specific computer language	Components of a structured language: assignments, conditionals, loops, external communication, subprograms	Types of languages: procedural, structured, functional, object-oriented, parallel
Concept of conditionals and iteration		Theoretical machines and formal languages
Levels of language: machine, assembly, high level, very high level, natural	Compilers, interpreters and assemblers	

3. Operating Systems and User Support

Command language and its use

Files and disk management

Human-computer communications: user interfaces, graphics, hypertext, CD-ROM technology

Communication networks: graphs, protocols

Memory management and virtual memory

Task scheduling: interrupts, buffered I/O

Single and multi-user machines

Accuracy of numerical computation

4. Computer Architecture

Basic computer model: CPU, memory, I/O

Basic data representation: numbers vs. characters, ascii vs non-ascii

Boolean algebra and logic: circuits, gates

Data representation: bits and bytes, binary numbers, real numbers

von Neumann stored program model: opcodes, registers, clock, fetch-execute cycle

Physical disk organization

Sequential and parallel processing

Special data representation: graphics, sound

Data compression

5. Social, Ethical and Professional Context

Privacy, reliability and system security

Uses, misuses, and limits of computer technology

Impact of technology on today's society: workplace, daily life

Software: public domain and private

Electronic crime: stealing and spying

Future of computer technology

Risks and liability in computing

Legal issues

Intellectual property, infringements and penalties

6. Computer Applications

Spreadsheets and data analysis

Data base systems

Word processing and desktop publishing

Presentation and statistical graphics

Electronic mail

Audio: speech and music synthesis

Scientific analysis: Mathematica, Mathlab

Simulation: statistics, modeling

7. Advanced Topics

Graphics: image generation, two and three dimensional, artistic

Artificial intelligence: games, expert systems, robotics, knowledge representation

Software engineering and system development: software development cycle, modelling and diagramming

Diverse Models and Course Presentation Methods

Computer Science topics can be presented in a spectrum of approaches ranging from a concentration on applications to a concentration on intensive programming. A breadth first approach can also be used to balance programming and applications. The task force has identified a number of models which can be used successfully to introduce computer science.

These models include:

Applications based
Breadth approach using applications and programming modules
Breadth approach interweaving applications, computer science topics, and programming
Project development approach using programming language
Apprenticeship model
Advanced placement (AP) computer science

Applications based models illustrate computing concepts with exercises involving commercial software such as spreadsheets and data bases. As an example, a discussion of data representation should accompany the creation of a spreadsheet or data base.

In another model, students learn some concepts while programming and others while using application software. The applications could be covered first so that students see what computers can be programmed to do. Then when they learn to program, they will have some knowledge of useful software.

Computing topics and applications can be interwoven into a course that concentrates on programming. The third model adds breadth to the traditional programming course. When the teacher feels it appropriate, topics can be introduced and discussed.

A course concentrating on student developed programming projects which illustrate computing concepts is another model.

Another approach suggests an apprenticeship model. In schools where this is feasible, interested students can learn much about computing and computers by working with a master programmer on a real world programming project. [4]

Finally, many schools offer advanced placement (AP) courses. These are comprehensive introductions to traditional university computer science following the CS1 and CS2 programming curricula. AP courses often have prerequisites but can be offered to some students as a first level course.

Descriptions of these models including sample syllabi are contained in the appendices.

These courses are designed to be taught in a full year. The material that can be covered in one semester does not give the student full appreciation of the subject and necessarily omits some of the key concepts of computer science. However, for practical purposes the task force felt it necessary to support the option of offering a limited version of the course in one semester.

Laboratories and Exercises

Laboratories and exercises should give students an opportunity to carry out experiments that illustrate topics in a realistic setting and at the same time teach the specifics of the software used. Students may also be assigned to work on projects too large to be completed during a single class period.

Both open-ended, independent, or homework type assignments and the more directed laboratory exercises such as in physics and chemistry are recommended.

Students would be given an objective, guidance on how to achieve it, and supervision. They then report their findings, not just as programs, but in the form of charts, illustrations, and written reports. In addition there should be experimentation with software which demonstrates algorithms and applications.

Barriers to Implementation

High schools offer several types of computing courses [8, page 11], often as part of the mathematics, science or technology departments. Students can take courses to study computer applications software in general literacy courses; learn an introduction to programming in a language such as LOGO, BASIC, Pascal, Scheme, or Turing; or study AP (advanced placement) computer science courses.

Many college bound students take only a general literacy course because they feel that word processing applications are important, but programming is not. Computer science is seen by students and counselors as either a programming course or an advanced honors type course. They often find programming irrelevant and dull and thus fail to elect any additional computing courses.

In part this may be due to the fact that the current computer science curriculum (with its emphasis on small programming projects) leads students to perceive computer science as uninteresting. Students do not learn what a computer scientist does or why computer science is an exciting field. Nor do they see the overall impact of computing on their lives.

In addition their schedules are usually filled with other courses required for college admission. There is no College Board Achievement Test in computer science. Thus there is little motivation for them to study computing. (Most also have a full program and limited time outside class to experiment with computer projects.)

Many high school computer teachers were educated in other fields and have minimal formal preparation in computer science. Some are entirely self taught. While many teachers effectively teach the use of applications and/or programming languages; without guidance, most would need some formal preparation to deal with concepts from modern computer science.

Moreover, the computing field is changing very rapidly making it difficult to keep up with developments, and high school teachers have little time to study all the aspects of these changes. Few states require certification to teach computer science. A teacher could be certified in any subject discipline and then allowed to teach computer science.

Computer teachers are expected to perform additional duties outside of the classroom such as monitoring computer labs, servicing computing equipment and providing computer expertise for others in the school.

Guidance counselors, school administrators and the public at large do not really know what computer science is. In most schools, computer science courses cannot be used to fulfill mathematics and science requirements. Thus guidance counselors often do not recommend that students take computer science courses. They also have a tendency to steer students away from computer science and towards easier applications so that the students can maintain the high averages needed for college admission.

Many high school teachers are uncertain about college and university expectations and requirements. It is difficult to keep abreast of what universities are teaching and what preparation is expected from high schools. College admissions officers only recognize the significance of the AP computer science courses in a student's schedule.

And finally, the poorest schools with the least computer equipment often serve large minority populations. To attract minority students into computing, courses must be adaptable to technology limited settings.

Task Force Activities

The task force was formed in 1989, had a number of meetings, and them presented preliminary results at conferences in 1991 (CSC '91 in San Antonio and NECC '91 in Phoenix) and in 1992 (CSC '92 in Kansas City and NECC '92 in Dallas). Also several workshops for teachers were held in 1992.

In addition, the second draft report (November 1991) together with a short questionnaire was sent out to over 1000 reviewers in fall 1991 and winter 1992. Over 200 questionnaires were returned. In their responses most high school teachers welcomed a definition of computer

science and a discussion of its important topics. They indicated that it would be very useful to have specific curriculum models.

Most university faculty indicated a willingness to support efforts to improve high school offerings. They recognized the importance of a proper introduction to computer science at the high school level.

Summary

The task force is charged with developing a high school course in computer science. The proposal contains topics from seven important computing areas including a number of applications and advanced topics, that are consistent with the new ACM/IEEE-CS university recommendations. In addition, six models incorporating these topics have been listed together with syllabi for some of the models.

The proposal not only describes the list of concepts which these courses should cover, but also addresses the delivery of the material in a wide variety of circumstances. The task force recognizes the barriers that may hinder the implementation of these courses and is working to provide solutions to some of these problems.

References

1. ACM Curriculum Committee on Computer Science, Curriculum '68. Communications of the ACM, 1968.

2. ACM Curriculum Committee on Computer Science, Curriculum '78. Communications of the ACM. 1979.

3. Denning, P., Comer, D., Gries, D., Mulder, M., Tucker, A., Turner, A., and Young, P., Report of the ACM Task Force on the Core of Computer Science. ACM Press, NY 1988.

4. Harvey, B., Symbolic Programming vs. the A.P. Curriculum, The Computing Teacher, pages 27-29, February 1991.

5. Merritt, S., Bruen, C., East, P., Grantham, D., Rice, C., Proulx, V., Segal, G., Wolf, C., ACM Model High School Computer Science Curriculum, Task Force of the Pre-College Committee of the Education Board of the ACM, June Draft Report, June 1991.

6. Merritt, S., Bruen, C., East, P., Grantham, D., Rice, C., Proulx, V., Segal, G., Wolf, C., ACM Model High School Computer Science Curriculum, Task Force of the Pre-College Committee of the Education Board of the ACM, November Draft Report, November 1991.

7. National Commission on Excellence in Education, A Nation at Risk: The Imperative for Educational Reform, Washington, D.C.: US Government Printing Office, 1983.

8. Rogers, J., Achberger, F., Aiken, R., Arch, J., Haney, M., Lawson, J., Lemke, C., Swanson, T., Tumolo, S., Computer Science in Secondary Schools: Course Content. Communications of the ACM, March 1985, pages 270-274.

9. Rogers, J. and Austing, D., Computer Science in Secondary Schools: Recommendations for a one-year course. In Topics: Computer Education for Elementary and Secondary Schools. Joint Issue Education Board ACM/SIGCES and ACM/SIGCUE, January 1981, pages 48-54.

10. Taylor, H. and Aiken, R., Informatics in Secondary Schools, Draft report, July 1991.

11. Tucker, A., Barnes, B., Aiken, R., Barker, K., Bruce, K., Cain, J.T., Conry, S., Engel, G., Epstein, R., Lidtke, D., Mulder, M., Rogers, J., Spafford, E., and Turner, A.J., Computing Curricula 1991, Report of the ACM/IEEE-CS Joint Curriculum Task Force, ACM Press, NY 1991.

12. Tucker, A., Barnes, B., Aiken, R., Barker, K., Bruce, K., Cain, J.T., Conry, S., Engel, G., Epstein, R., Lidtke, D., Mulder, M., Rogers, J., Spafford, E., and Turner, A.J., A Summary of the ACM/IEEE-CS Joint Curriculum Task Force Report Computing Curricula 1991. Communications of the ACM, June 1991, pages 68-84.

13. Tucker, A. and Garnick, D., A Breadth-First Approach to the Introductory Curriculum in Computing, preprint.

Appendix A: The ACM/IEEE-CS Report

According to the ACM/IEEE-CS report [11, page 9], 'computing is simultaneously a mathematical, scientific, and engineering discipline,' so 'different practitioners in each of the nine subject areas employ different working methodologies, or processes, during the course of their research, development, and applications work.' The three processes are theory, abstraction and design. Theory is mathematical, abstraction involves constructing hypotheses from data and design is 'rooted in engineering'.

The task force described in some detail the topics in each area which computer science students should know. These were called 'knowledge units' (KU's). They also listed elective topics. These topics were arranged in a number of sample curricula. The report also divided laboratory work into that done in open and closed labs. The former consist of the usual homework assignments which students do on their own time. The latter are performed during scheduled times with an instructor present. A specific task is to be accomplished during the time allotted. For high schools this translates into projects that can be done in one or two class periods (closed labs) versus ones that require time outside of class for preparation (open labs).

One recommendation that grew out of the 1988 report [3] was that computer science be taught on the physics and chemistry models. These curricula begin with broad but non-trivial surveys of the fields. Subsequently those majoring in these disciplines study specialized areas in depth. This broad-based approach is described in more detail by Allen Tucker and David Garnick in [13]. Tucker and Garnick also added social context to the three processes of theory, abstraction and design.

Appendix B: The Second Draft Report and Its Evaluation.

Viera K. Proulx, Northeastern University, MA

In November 1991 the first Draft Report was sent to over 1000 individuals from high schools and colleges for comments and critique. The evaluators were asked to reply to nine specific questions, and to add any other comments they thought would be helpful. The task force received 188 responses, 30 of them from college instructors, 152 from high school teachers or administrators, and 6 which fit into other categories. The overall reception of this new curriculum was positive, but there were several areas of concern that the task force has addressed in this draft.

Questions 1 and 2.

The first two questions focused on the place of computer science as a subject in a high school curriculum. The questions were:

1. Do you feel that it is a good idea to develop a high school computer science curriculum as described in the Draft Report? If no, why not? What alternatives would you suggest?

2. Is the proposed course in the Draft Report a good way to make computer science a more valuable subject to extend a student's computing knowledge?

The answers to the first question were overwhelmingly positive, but a higher percentage of the college respondents had doubts. While 82% of high school reviewers responded positively to the second question, only 50% of the college respondents did - a full 40% were not sure that this was a good way to make computer science a more valuable subject.

The majority of comments on these two questions were concerned with clarification of some of the goals of the course. The first comment was "where does the course fit into the curriculum" - compared to literacy courses at the low end and the Advanced Placement Pascal course at the high end. Others asked for a statement about the benefits of teaching and/or taking this course, the expected outcomes, and what new knowledge will the students gain from the course.

The task force expects that this course will serve the same purpose as a typical science course - similar to learning about cells, and classifying living things which one does in biology, or similar to learning the periodic table, the composition of molecules, and the basic laws of chemistry, etc. It is not intended to be a computer literacy course, nor is it a programming course. Some programming is needed to explain the tools a computer scientist works with, but emphasis should be on concepts and algorithmic thinking. At the end, students should understand how a computer works and what computer scientists need to do to make the computers as powerful and useful as they are today; what are the limits of what a computer can do, and what are the effects of computer technology on today's

society. In order to understand this, students will have to learn a new way of reasoning and problem solving - namely working out the solution in the form of an algorithm (expressed in natural language, as well as in program form).

Some respondents commented that what makes this course valuable is that it shows computer science is not just programming, that it differs from 'hacking' often present in programming courses, and it teaches new valuable ideas, skills, new types of reasoning, and problem solving; goes beyond the syntax of the language.

Others voiced concern about the course being too ambitious, or asked the task force to clarify the depth and approach to various topics. The list of topics by itself could certainly fill two years of a college level curriculum. To illustrate the intended depth of coverage and possible methodologies for introducing the material the task force has included in this draft several sample course syllabi to show typical implementations of this curriculum.

Questions 3 and 4.

The next two questions, concerning the selection of topics, skills, and priorities suggested for the course, were:

3. Does the Draft Report identify all of the key areas of computer science as you would define it? If no, which topics should be included/deleted?

4. Does the Draft Report place the proper priorities on suggested topics and skills desired for a high school computer science curriculum? If no, suggest changes.

About 71% of all respondents felt all key areas have been listed. While 66% of the high school respondents felt that the draft curriculum contained the proper priorities on topics and skills, only 37% of the college respondents agreed - another 37% were not sure, and 27% thought the priorities were wrong. To find out what were the main concerns we made several wish lists from comments received.

Wish Lists

College respondents would like to add testing, correctness, and reliability, ethics and privacy (already there), the role of mathematical and logical reasoning, problem solving, algorithmic thinking (there already, but needs to be emphasized), and computer support for group work. Among the items suggested for deletion were Turing machines, computational complexity, existence of non-computable functions, shortest path and graph algorithms, normal distributions, interrupts and buffered I/O, and advanced graphics. The task force feels that all of these topics can be explained to high school students on a conceptual level, together with a few basic exercises to illustrate the point.

Others commented that the breadth first approach is good, students will learn what computer science is about, and if they wish to continue, they will have the material again anyway (in a manner similar to the first physics course - in high school and in college).

The wish list of the high school teachers can be characterized as "take away the topics that we, the teachers, have never learned - they are too hard" and "add all the exciting stuff that is just happening - we want to know, and our students want to know". So the "Add" wish list consists of hardware, transistors, chip building, conversion of data formats, files, databases, graphics, fractals, animation, sound, virtual reality, hypermedia, supercomputing, telecommunications, communications links, project management, problem solving, processes of learning, structured design, number theory (Fibonacci numbers), sorting - searching - arrays, more simple math. The "Delete" list contains computer architecture, circuitry, electronics, assembly language, networking, statistics, theoretical computer science, expert systems, and all topics in the AP course (one respondent).

Question 5.

The fifth question focused on justification for such a course:

5. Does the Draft Report identify a clear justification for a high school computer science curriculum? What suggestions can you offer to make the need for computer science studies stronger?

Over 75% percent of respondents thought the justification was clear, others suggested improvements. College respondents asked to emphasize the preparation for a computerized society, and that students are at a disadvantage without it. They asked to emphasize the new kind of reasoning, and also broad applications, and integration with other subjects. The high school

respondents agreed: emphasize that job qualifications are changing and so the preparation needs to change too. Mention the Education 2000 goals, mention that it fits in with the new "applied education" wave, emphasize the place of this course between applications and programming, talk about the computer as a problem solving tool.

Question 6.

The sixth question looked at the role of this course in a student's preparation for university studies:

6. If a student completes a computer science course such as the proposed course in the Draft Report, will this improve the student's preparation for general or scientific studies at the university level? If no, what improvements should be made to computer studies?

Sixty seven per cent of college respondents thought this would be a good preparation, some commenting that students will benefit from better problem solving skills. Most of the others were unsure (27%) - they had doubts that the course would be done right, whether it is too engineering oriented, or whether a math course, general broad education, or a course on relevant applications, would not be better.

The high school respondents were more enthusiastic - 82% agreed. They felt that colleges need to require such courses for admission. It needs to count for credit towards high school graduation. The depth and the level of presentation needs to be good for the course to be beneficial. It will help if colleges require the use of computers. Some wondered whether the course is too technical, whether other subjects are not more valuable, whether it is suitable for pre-med and pre-law students.

Questions 7 and 9.

Questions seven and nine asked for comments on the clarity of the report, suggestions for changes, and for the problems that need to be addressed:

7. Are any parts of the Draft Report unclear or difficult to understand? Please make suggestions to improve any part(s) of the Draft Report which are unclear to you.

9. If the Draft Report curriculum receives wide acceptance, what suggestions would you make to implement it in your school, school district, state education department, or throughout the country?

Most felt the report was clear, some asked the task force to comment on its impact on women and minorities, others pointed out that a report to ACM should be different from a report to administrators.

Question 8.

8. Would you like to be a reviewer of the next Draft Report?

Many respondents wished to see the next report. Altogether, 78% of the high school evaluators and 87% of the college marked yes.

Appendix C: Breadth Approach Interweaving Applications, Computer Science Topics, and Programming.

Charles W. Rice, Dalton School, NY, NY.

Description

This course outline is a summary of the introductory computer science course which is taught at the Dalton School. The course is designed for students with no computer background. It is assumed that students have studied some algebra. In most cases students have used a computer in some context. Students range from 9th to 12th graders.

There are several aims for the course. First a student will learn how to solve a variety of problems using a computer by programming methods and other application techniques. Second the course will give a student an understanding of what computer science is, how it differs from other fields and how it can contribute to other fields of study.

The course primarily uses classroom discussions and student lab work techniques to introduce and master material. Some topics are covered by class lecture, but it is always better to keep the students actively participating in some topic. The course is broken up into approximately 3 week long assignment periods. Minimal tests are given because the students constantly demonstrate what they know and how well they are

learning. The course is designed so that the students will perform most of the work during scheduled class periods. Computers are available for all class periods but not always used.

Pascal, Turing and Fortran programming languages have been used at different times to conduct this course, but the course does not depend on any specific language. The language is used only to give a student a method to solve different types of problems using a computer when problem solutions must be designed, implemented and analyzed.

A list of computer science discussion and lecture topics are referred to in the course outline. These topics are an important part of the course. The topics are presented to students at various times throughout the course. It is important to note that not all of the topics are presented every year or to every student. Selection of specific topics is made depending on the type of student and amount of time available.

These topics are usually presented by discussion and demonstration. The order that the topics are presented is suggested in this outline but is not rigid. A topic is sometimes omitted or moved to a different assignment period.

Problem and lab work topics included in the course outline are designed for students to work on during lab class periods. The purpose of these assignments is to give a student a chance to learn by investigation. Students will learn how to solve problems using the computer with applications and by writing programs. In addition some assignments are designed to explore important underlying computer science principles. A detailed assignment sheet with required and optional exercises is provided for each of these assignments.

By the end of the semester or year the students have learned to solve problems using programming and other methods but have also been introduced to the wider spectrum of the computer science field.

Course Outline: First Semester

Assignment 1: Introduction to Computers, Computer Science and Computer Applications

Discussion and Lecture Topics: 1, 2, 3, 4, 20 and 21.
Problem and Lab Topics:

Program Familiarity: enter it, operate it, analyze it

Computer Applications: spreadsheets - solve simple problem or project using this technique

Assignment 2: Algorithms and Computer Science Background

Discussion and Lecture Topics: 5, 6, 7, 8, 9, 10, 11, 12 13, 14, 16, 17, 26, 29, 30, 31, 32, 34, 39 and 41.

Problem and Lab Topics:

Algorithm Solutions(2): prepare and analyze an algorithm, informally analyze other students work

Assignment 3: How Computers Work and Computer Science Background

Discussion and Lecture Topics: 15, 19, 22, 23, 24, 25, 35, 36, 37, 40 and 42.

Problem and Lab Topics:

Electronic Circuits: conduct experiments, prepare lab report

Mid Semester Test

Assignment 4: Introduction to Programming

Discussion and Lecture Topics: 38, 43, 49, 50, 51 and 52.

Problem and Lab Topics:

Programming Assignment 1: assignment, I/O, documentation, testing

Assignment 5: Using Feedback and Decisions to Solve Problems

Discussion and Lecture Topics: 45

Problem and Lab Topics:

Programming Assignment 2: decisions, loops, procedures, testing

Assignment 6: Using Alphabetic Data

Discussion and Lecture Topics: 44

Problem and Lab Topics:

Programming Assignment 3: array and strings

Course Outline: Second Semester

Assignment 7: Simplifying Complex Problems and More on how Computer Hardware Components Work

Discussion and Lecture Topics: 25, 27, 28 and 30

Problem and Lab Topics:

Programming Assignment 4: abstraction and decomposition using subroutines and functions
Computer Combinatoric Circuits: flip flop, binary counter, adders, 7-segment display; conduct experiments and write lab report

Assignment 8: Data Storage, Data Design and Knowledge Structures

Discussion and Lecture Topics: 17, 18, 46 and 53

Problem and Lab Topics:

Programming Assignment 5: data storage, records, files
Artificial Intelligence: Knowledge Systems Expert system experiment and lab report

Mid Semester Test

Assignment 9: Computer Graphics and Sound Generation

Discussion and Lecture Topics: 47

Problem and Lab Topics:

Programming Assignment 6: using simple graphics and sound

Assignments 10 and 11: Solving Real World Problems and Projects

Discussion and Lecture Topics: 48

Problem and Lab Topics:

Guided Independent Project Assignment: proposal, feasibility study, task definitions, peer design review and walkthrough, PERT chart, progress report, final documentation, peer final testing review walkthrough

Note For the course offered in 1991-92 the following topics were covered at some time during the year: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 29, 30, 31, 32, 34, 36, 37, 38, 41, 42, 47, 48, 49 and 53.

The following topics were omitted: 1, 18, 27, 28, 33, 35, 39, 40, 43, 44, 45, 46, 50, 51 and 52.

Also Assignment 9 was omitted.

List of Discussion and Lecture Topics

General Topics

1. Evolution of Computer Science: math, physics, electrical engineering, management, science, linguistics, psychology

2. Four interest areas of Computer Science & applications: algorithms, programming languages, operating systems, architecture
Computer Science topics: general applications, professional and ethical issues

14. Types of computers: microprocessor, micro, mini, mainframe, supercomputers

10. Contrasting types of computers: analog and digital

16. General uses of computers: control, simulation, design, communication, education, AI

41. Social and societal issues with computing, examples and cases: crime, privacy, risks, reliability, viruses, legal issues, misuses

General Application Topics

3. Computer tools introduction and use:text editor, word processors, spreadsheets, data base, presentation software, hypercard, multimedia

20. Computer tools: spread sheets and problem assignment

21. Computer tools: electronic mail and problem assignment

Computer Science Application Topics

17. Artificial Intelligence: computer uses and examples expert systems, games, theorem proving, robotics, image recognition, natural language

18. Neural Networks: what look like, how they work, differences (use handout sheet)

45. Computer simulations: examples, nondeterministic, random numbers

46. Computer database systems: purposes, simple and complex types, examples

47. Computational problem examples: fractals, etc.

48. Software Engineering Methodology: systems analysis, project management, PERT, life cycle

Data Representation Topics

4. Hierarchy of data inside and outside a computer: bit, nibble, byte, word; element, field, record, file, library

5. Why 8 bits in a byte?

6. Information representation in binary, how data is stored and why: numbers, characters and symbols, operations, pictures, sound

7. Codes: ASCII and variable length codes

44. Data compression: huffman coding, other techniques

Computer Language Topics

13. Computer languages and levels from machine: machine, macro, assembly, high level, very high level, 4GL, natural

49. How procedural computer languages are organized and designed: data and data structures, statements, assignment, control, I/O, subprogram, declarations, special

50. Types programming languages and examples: structured, procedural, functional, logic, object oriented

52. Linguistic design techniques for computer languages: BNF

Computer Architecture and Design Topics

11. Computer processing: sequential and parallel examples

12. Types of parallel processing problem solution possibilities: none, minor improv, binary improv, one step, pipelining

15. Computer architecture: chips define the computer and its functions (8080, 80286, 6500, 68000 etc.), types of chips: CPU, memory, communication

8. Computer memory: characteristics K (size), Herz (speed), flipflops

9. Computer memory: types (RAM, ROM, PROM, EPROM, DRAM, etc.)

35. Conceptual and functional organization of computers

36. Computer architecture: simple hand computer example; terms: register, clock, address, machine instructions, etc.; simple problem flow through

37. Parallel processing architectures: ring, grid, network, specific machine examples

19. Semiconductors: switch, composition and how it works, different types

22. Boolean logic, gates and truth tables: AND/OR/NOT, schematics, symbols, other gates possibilities

23. Binary numbers: how represented, how arithmetic and logic work, base 10 differences

24. Combinatoric circuits and boolean problem solutions: purposes, construct with gates, truth tables, data flow through; 3 types in computer: arithmetic and logic, storage, communication and bus

25. Electronic lab assignment

40. Micro computer system schematic diagram

42. Switching device hardware-generations: relay, vacuum tube, transistor, IC, optical?

53. Computer Disk Storage: conceptual - directory, tree structure, special sectors; physical - byte, block, sector, track, cylinder, R/W mechanism

Algorithms and Problem Solving Topics

26. Algorithm characteristics: finite, definite, generalized, produce results, effectiveness

27. Types of algorithm: enumeration, iteration, trial and error, divide and conquer

28. Computer algorithms: require logic and data structures

29. Algorithm problem complexity classifications and examples: trivial, tractable (simple and complex), intractable, impossible (paradox)

30. Algorithm constructs and regular structures needed: requirements - sequence, branch, repetition, subroutines; structures - counting, totaling, swapping, comparing, save a value

31. Algorithm representation techniques: list, flowchart diagram, structured narrative, program

32. Flowcharts including assignment: symbols, structure, examples, advantage and disadvantage

33. Warnier-Orr diagrams including assignment

34. Structured narrative including assignment: top down, outline form, symbols, decomposition, advantages and disadvantages

38. Steps to take an idea through a computer system: idea, result?, solve by hand, algorithm, program, enter, compile, test and debug, accept?

Operating Systems and User Services

43. Operating systems: purposes, how they work, DOS, GUI and examples

39. Types of computer system processing techniques: batch, timesharing, multiprocessing, distributed processing networks

51. Compilers and Interpreters: difference; how they work - lexical analysis, syntactic analysis, code, optimization, source code, object code, machine operational code transformations

Appendix D: High School Computer Science Using Programming Projects

J. Philip East, University of Northern Iowa, IA.

The basis for this work is a contention that project-based computer programming and associated study is a better way to introduce students to computer science and programming than a language oriented course. Included in the discussion below are the major guidelines or beliefs that influenced the design of the course, some general management information, and an example showing one possible instantiation of such a course.

Course Foundation

A course in programming seldom has as its primary goal the development of student knowledge of the language. Rather, we typically want the students to develop general skill in program design and implementation. Additionally, we would prefer to have students learn something of "computer science" while they are learning programming. Project-based programming can be used to meet both these goals, so long as one is careful about the teaching of "programming" and in the selection of projects. The programming instruction needs to focus on concepts rather than language syntax and semantics. The projects need to be selected or designed to illustrate various areas of computing and to provide motivation for discussion about them--to go beyond just the programming necessitated by the project. These notions are the basis for the discussion below.

A major conclusion about this course is that learning programming requires learning both programming concepts and a programming language. It is not, however, necessary to learn both at the same time. Indeed, it would seem difficult for students to focus on the concepts while simultaneously being required to produce products (programs) that absolutely require knowledge of the language and which will determine their grade in the course. Thus, it is proposed that the concepts of programming--the knowledge, skills, and processes often called problem solving--be taught prior to the introduction of a programming language.

In a procedural programming environment, the concepts of programming are, primarily, threefold. First, there are a few limited fundamental actions that computers are capable of (principally input, output, and assignment). Second, there are a few limited ways to organize these actions, i.e., sequence, selection, iteration, and modularity and recursion. Third, data is necessary and consists of various types of low level data elements (numbers and characters primarily), various organizations of such data elements (arrays, etc.), and using the elements and organizations to represent more complex information such as sound, graphics, a business enterprise, etc. All of these concepts are, in themselves, relatively easy to communicate and comprehend. When they get mixed in with programming problems requiring actual working, coded programs they become quite complex.

Arguably, developing expertise with a particular language is relatively simple provided one already understands the concepts of programming. Thus, for this course the standard programming concepts noted above will be introduced and learned prior to any use of a language.

Programming can be thought of as skill in organizing instructions for the computer. As with any skill, some knowledge is required before one can develop the skill. We will not be able to begin directly with programming projects. The first "project" will be devoted primarily to basic programming knowledge acquisition. Later projects can be selected to expose students to a cross-section of computing and to appeal to expected student interests.

Most of the modules will share a number of management and methodological characteristics. While student interest will be considered in the selection of specific projects, a (the) major motivational factor will be intellectual challenge. Generally, class discussion and other activity including algorithm and program development will likely be group-based. Each module will end with reflection on what has been learned and on the social and professional context of computing that relates to the module's topic. It is likely that grading in the course will be based primarily on performance on the programming projects and other assigned activity rather than examinations.

Course Synopsis

Background including how computers work, logic, pseudocode, simple programming statements, and data representation is presented first. Later work will use and/or build on this groundwork. The approach taken

here is that the computer is nearly magic in what it does but yet quite simple and easily understood. Next students get a chance to examine the area of artificial intelligence. The programming language will be formally introduced and students will see the importance of data organization in computing. The third module/project is less programming intense. It provides an opportunity for students to examine human-computer communication and to write modules based on their experience, teacher lecture, and research into the area. Next, students will tackle a database and information retrieval project. Beginning with a broad problem statement, students (in groups and with the guidance of the teacher) will explicate requirements and specifications, design the algorithm and data organization, and implement and test the program. The fifth project involves simulation of a component of a standard architecture or operating system. By simulating the system, students will better understand how it works as well as have encountered the important idea of system simulation. The final preplanned module will be the programming of a component of a language translator. Students will incorporate program components written in earlier projects. For the duration of the year, students will be able to design and implement a system of their choice. The only restriction will be that the project must introduce the student to something new.

Course Topics and Process

1. Groundwork

The purpose of this module is to provide students with the knowledge and skills basic to an understanding of programming. The content includes 1) the basic (machine level) capabilities of computers and the context for those actions--the instruction cycle; 2) the representation of data and ways of collecting or organizing data (both physical and abstract); 3) ways for organizing actions on data--in a procedural language: sequence, selection, iteration, and modularity (with recursion)--and the representation of algorithms using pseudo code (and/or flowcharts); 4) the logic necessary to facilely interpret and produce the conditionals used in selection and iteration; and 5) problems associated with finite representation of infinite values.

Weeks 1-2: The Magic of Computers. Through lecture, discussion, class activity, and video students will be introduced to the computer as a very limited device which through speed and human ingenuity in representation of problems and data can be used to accomplish myriad complex tasks. (The model of activity presented here--the instruction cycle--should be used to explain later concepts.) An introduction to pseudocode and data representation/structures (e.g., pictures, sound, etc. as well as arrays, sets, files, trees, etc.) will provide students with insight into how such a simple device can be used so extensively. Student activity is primarily to listen and think. At the end of the unit, students will write a paper describing the capabilities of computers.

Weeks 3-4: Logic/Thinking. Primarily through examples, practice, and recitation and criticism students will gain facility with conditional expressions and pseudocode algorithms for the solution to problems expressed in English. Student activity has three major components, each of which will involve teacher provided examples and feedback on student performance. The activities are 1) exercises for developing an understanding of logic and interpreting conditional expressions and pseudocode algorithms, 2) exercises for transforming English expressions of problems into pseudocode involving conditionals, and 3) practice developing pseudocode algorithms for multiple-step problems expressed in English.

Weeks 5-6: Machine Representation of Numbers. Through an introductory lecture and a series of closed laboratory experiences students will be exposed to problems associated with the computer representation of numbers. Individual reasoning and class discussion will be used to identify potential problems related to numeric computation on computers. Students will have homework exercises that relate to interpreting and understanding the outcomes of laboratory exercises. These activities will be followed by a group discussion of conclusions and principles with respect to this area and by a programming activity that tests students understanding and utilization of the principles. The programming project will be coded with significant help from the instructor. Students will gain experience in the mechanics of programming--editing, compiling, debugging, and eventually running a program.

Week 7: Reflection. Class discussion and individual student reflection will address machine-level and general capabilities and limits in computers, reliability and numeric accuracy of computer programs, and review of pervasive use of computers in modern society.

2. Twenty Questions

This project introduces students to the exciting or intriguing area of artificial intelligence. The twenty questions game provides a basis for discussion of the area and various themes within AI such as knowledge representation and search of or access to knowledge. Students will also receive more programming background in the form of syntax diagrams for better understanding of the syntax and semantics of a particular language. The program itself will require the use of sophisticated data structures and can be used to introduce the importance of data structures in computing.

Week 8: AI and Twenty Questions. This overview of artificial intelligence and the introduction to the game of twenty questions will be provided primarily through lectures and student readings. Various elements of AI will be discussed and illustrated, when possible, with the twenty questions game. Students will respond to homework questions from the overview of AI.

Week 9: Twenty Questions Algorithm. Class activity will be devoted to discussing the problem and developing an algorithm and data structure for the problem.

Week 10: Our Programming Language. The teacher will provide general instruction in interpreting the syntax and semantics of elements of a programming language. Students will work exercises to enhance their understanding of the language syntax. They will also get additional practice in converting pseudocode to code.

Weeks 11-12: The Program. Students work to implement the program. Class time is spent discussing problems and otherwise working on implementing and testing the program. Difficulties will be noted and recorded for later discussion.

Week 13: Reflection. Class discussion and individual student reflection will address software development process, determining program correctness, and applications and limits of AI in our society.

3. Human-Computer Communication

This module will introduce students to the area and methodology relating to designing good program interfaces. The associated need for validity-checking of data will also be included.

Week 14: Issues and Problems. Through lecture, discussion, class and individual activity, and video students will be introduced to 1) the various media/modes of communication, 2) the need for valid data, and 3) ways to enhance efficient and effective communication. Students will hear and read about these concepts and experience them through the use of various programs. Initial student activity will be to listen to, read about, and discuss various issues. Next students will begin to formalize the issues by noting good and bad characteristics of program interfaces they experience and by responding to teacher questioning and probing after those experiences. Ultimately, groups or individuals will develop explicit guidelines for developing user interfaces.

Weeks 15-16: Components of Human-Computer Communication. Students will discuss alternatives for various types of human computer interaction including data entry (e.g., simple input such as number of trials, record input such as sales data, extended or varied input such auto service requirements) and continuous interaction (e.g., tutorials, application programs such as word processing). After arriving at general conclusions, students will design and implement at least one of the interaction components discussed. Additionally, they will design a different interaction component for the previous project. (The next project will also included extensive attention to the human-computer interface.)

Week 17: Reflection. Class discussion and individual student reflection will address both general (command-driven, menu-driven and graphical) user interfaces and generalizations about program-specific user interfaces.

4. Database and Information Retrieval

This module will introduce students to the area and methodology relating to database or the storage, retrieval, and analysis of extensive amounts of information. Students will also extend their previous exposure to human-computer interaction to this area of computing.

Weeks 18-19: Introduction. Through lecture, readings, and reflection students will be exposed to the needs of information storage, retrieval, and processing. General

conclusions about or requirements of large systems (databases) will be formed. Students will be introduced to the need for disciplined development when tackling large or involved projects. The various stages of and methods relating to the software life cycle will be presented. Students will analyze their experience with database systems to develop preliminary ideas as to the aspects of database that require significant attention (e.g., ease of access, redundancy problems, storage limitations). They will also provide written responses to questions after lectures and readings on both topics (database and software engineering). Explicit, written student statements and discussion of the principal aspects of database systems and the major software engineering methods/steps will be the final outcome of this segment of the module.

Weeks 20-21: Project Selection and Definition. Through small group and class discussion students will gain experience in the requirements analysis and project specification phases of development. They will also gain further insight into data and file organizations through readings, lecture, and discussion. In order to reinforce recently gained knowledge about user interaction, those aspects of the project will receive particular attention. (The actual project topic will be subject to student selection and perhaps multiple group projects will be allowed.) Students will produce requirements documentation and a rough set of specifications for whatever project is selected.

Weeks 22-24: Algorithm Development and Implementation. Individual and group work and discussion with the teacher will be the mode of activity during this time. Occasional class discussion will be used to make salient points from other groups' experiences and to help students generalize from the experiences. Issues/questions about correctness will be raised toward the end of the experience to set the stage for more explicit treatment later--this large project should be very useful in pointing out the need for earlier and more formal attention to correctness concerns. Students will develop appropriate algorithms and implement them.

Week 25: Reflection. Class discussion and individual student reflection will focus on security, privacy, and other issues revolving around the collection and manipulation of large databases.

5. System Simulation

This module returns to the topic of how computer systems work. Students will continue to enhance their programming through the development of a simulation of some component of a computer system (e.g., opcode interpretation, adders, memory management, process scheduling). The component could relate to either hardware or to operating systems (including networking or parallelism). Recent project experience should have set the stage for a solid treatment of correctness which will also be included with this module.

Week 26: Computer System Review/Overview. Students will read and hear about the organization and operation of computer "systems". Reference will be made to experience in earlier projects. The goal will be to have students form a general model of the operation of a computer system. There will also be lectures and readings regarding program testing and developing correct programs. The primary student related outcome of this week will be student discussion and oral question answering.

Week 27: Project Selection and Background. One or more aspects of a computer system will be focussed upon. Students will again read and hear about (and perhaps see video's or simulations of) the system component(s) that will be the basis for the programming project. Class discussion of critical points in the component will occur. Students will perform requirements analysis and develop specifications for the component to be simulated.

Weeks 28-29: Algorithm Development and Implementation. Individual and group work and discussion with the teacher will be the mode of activity during this time. Students will develop appropriate algorithms and implement them.

Week 30: Reflection. Class discussion and individual student reflection will work to help students generalize their experiences about systems operation and program correctness.

6. Programming Languages

This module will provide an opportunity to expose students to concepts related to programming languages--what they accomplish (and how) and how they are developed. Students will continue to enhance their programming through the development of an appropriate

program (e.g., expression input and evaluation, simulation of control and/or data structure implementation).

Weeks 31-32: Introduction. Through lecture and reading students will be introduced to the utility of various types of higher level programming languages. If time and resources allow, students will also be exposed to some special purpose or fourth generation languages. The particular topic for the programming project will also be introduced. Students may be asked to respond to various general questions regarding programming languages. A requirements analysis for the programming project will be the object of class discussion.

Weeks 33-34: Algorithm Development and Implementation. Individual and group work and discussion with the teacher will be the mode of activity during this time. Students will develop appropriate algorithms and implement them.

Week 35: Reflection. Class discussion and individual student reflection will address levels and types of programming languages and the translation necessity and process.

7. Individual Projects

A whole course reflection will occur during the last week. Students will consider societal issues relating to computing, computing in vocations, professional responsibilities, and trends in computing hardware, software, and applications. If additional time is available, students will have the opportunity to work on writing or programming projects of their own design.

Coverage of Topics in ACM Recommendation

The ACM Precollege Committee recommendations for topics to be included in a high school course covered the areas of algorithms; programming languages; operating systems and user support; computer architecture; and social, ethical, and professional context. In these areas, the course described above:

> Includes all the "essential topics" except that no explicit attention is given to "command language and its use".
>
> Includes all the "recommended topics" except (from "Operating Systems and User Support") the topics of graphics, hypertext, and CD-ROM technology from "human-computer communication" and "communication networks".
>
> Includes only a few of the optional topics explicitly, "types of languages" (under "Programming Languages" can easily be addressed and most of the optional "Social, Ethical, and Professional Context" topics are addressed either during programming-related discussion or in the discussion at the end of the course.

Very few of the topics under the areas of "Computer Applications" are taught directly. Databases will be addressed from the computer science or programming perspective but not from the point of view of using a database system. Some aspects of simulation might be included in some of the projects. Under advanced topics, "artificial intelligence" and "software engineering" concepts are encountered and the provision of a solid introduction to software engineering principles is attempted.

Appendix E: Breadth Approach Using Applications and Programming Modules.

Viera K. Proulx, Northeastern University, MA
Carol E. Wolf, Pace University, NY

The topics recommended in the report can be organized into courses in a number of different ways. One method is to divide the information into modules, each covering related material. The purpose of each module is to introduce concepts in a concrete setting, so that from the given examples and exercises students can formulate abstractions and gain understanding of the theory. Collectively the modules expose students to a number of different uses of computers, with all their excitement, as well as their possible misuse. They enable students to gain an appreciation of computer science as a whole.

One possible division into modules follows. Schools unable to offer the entire semester course may choose to cover only modules 1 to 5 and module 7.

1. Overview of Computer Science

a) Problem solving and algorithms
Algorithms in daily life: recipes, directions, appliance instructions
Simple programs

b) Computer architecture
Components of a computer: CPU, memory, disk drives, monitor, printer
Internal and external storage
Basic computer operation

c) Operating systems
Command language
Files and storage

d) Societal impact of computer technology
History from the abacus to EDVAC
Changes in daily life since 1960
Everyday computer usage today

Exercises

a) With a computer
Use the operating system in the lab.
Create, save, copy and delete files.
Edit and compile a simple program.
Use a word processor.

b) Without a computer
Write an essay on the changes brought about by computers in our lives and in society.
Devise algorithms to solve everyday problems.

2. Data Representation, Storage, and Analysis

a) Data representation
Number systems: binary, octal and decimal
Binary and octal arithmetic
Codes: negative numbers, characters (ascii)
Real numbers and the problems with accuracy

b) Arithmetic expressions
Order of evaluation, precedence rules
Definition of a function as a rule
Writing expressions for computers to read

c) Analysis using a computer spreadsheet
The grid, cells, rows, columns and ranges
Data types: labels or values, formatting, significant digits stored and displayed
Commands and menus
Formulas: built-in and user created
Potential for errors when using formulas

Exercises

a) With a computer
Write a program using arithmetic expressions.
Find the value of the largest and smallest integers which the system can represent.
Create a spreadsheet for balancing checkbooks, managing inventory, budget preparation or loan analysis.
Use a spreadsheet to graph some data.

b) Without a computer
Experiment with an octal odometer.
Check the accuracy of a calculator when computing (1/9)*9 or performing multiple additions.
Play the binary card number guessing game.
Explore simple codes such as Morse code or those found in mystery stories.

3. **Managing Data**

a) Database management systems
Fields, records, data types, index files, updating

Searching using a logical condition
Examples found in the school such as library catalogs and course information systems
Privacy and integrity of databases

b) Boolean algebra and logic
Truth tables
Logical data types
Boolean expressions and conditional statements

c) Files, input and output
External files: paths, directories, deletion and recovery
File security and encryption of sensitive data

Exercises

a) With a computer
Create a data base using a commercial package.
Search through the data base using logical conditions.
Sort the data using different keys.
Write programs which contain conditional statements.

b) Without a computer
Use index cards to design a data base. Make additions and deletions, search using a logical condition, sort on different keys, and create an index file.
Research the problems of privacy and the laws regulating databases.

4. **Computer Architecture**

a) The computer at the chip level
Transistors, circuits, gates
Combining gates to form adders, I/O tables
Components of the CPU

b) Machine language level
von Neumann stored program model
Machine and assembly language
Opcodes, registers, memory
Fetch-execute cycle

c) History of computers from EDVAC to the present
Computer generations
Sizes of computers from desk-top to supercomputers
Networks and distributed computer systems
Problems of open systems and standards

Exercises

a) With a computer
Use a simulator for a simple machine.
Write a program in a real assembly language.
Use electronic mail on a real network.

b) Without a computer
Simulate a simple machine on paper.
Design a simple circuit such as an LCD digit display.
Write a paper on computer viruses or the problem of misuse of computer networks.

5. **Algorithms, Problem Solving, and Programming Languages**

a) Programming languages
The development of programming languages
Components of a structured language: assignment statements, conditionals, loops, input/output, subprograms
Examples of non-procedural languages
Compilers, interpreters and assemblers

b) Algorithms
Definition and properties: clear starting point, finite number of steps, next step always determined
Analysis of code complexity, time and storage requirements
Examples from computer applications and daily life

c) Problem solving techniques
Top-down design and stepwise refinement
Subprograms: procedures and functions

Divide and conquer
Flow diagrams and charts
Importance of structured design in creating and maintaining correct software

Exercises

a) With a computer
Write a computer program using loops and subprograms.
Investigate several different programming languages.
Experiment with animated sorting algorithms.

b) Without a computer
Carefully write out some algorithm used in daily life.
Play a game of strategy and describe the algorithms used.
Provide three kinds of directions from one place to another
i) the simplest to explain: shortest program
ii) the fastest to travel: least time
iii) the shortest route: least space.
Use index cards to simulate different sorting methods.
Draw flow charts to explain problem solutions.

6. **Computer Graphics**

a) Method used to store and display pictures
Difference between way text and graphics are stored
Grid, coordinates, scaling, pixels, colors
Display of slanted lines on a rectangular grid

b) Applications of computer graphics
User interfaces for computer programs
TV and films
Transmittal of pictures electronically: FAX, newspapers, from satellites and space probes
Scientific visualization of data: molecules, architectural and engineering structures, geological data

c) Arrays
Giving a single name to a number of storage locations
Storing tables of data using rows and columns

Exercises

a) With a computer
Use Logo or a graphics package to draw pictures.
Write a program using some of the graphics features of a language.
Write a program using arrays.
Use a program to create animations.

b) Without a computer
Use graph paper to simulate a computer screen.
Experiment with drawing lines and figures and see what should be stored in memory.
Make a simple animated cartoon.
Write a paper on the use of computer graphics in medicine and science.

7. **Operating Systems and User Support**

a) Tasks performed by the operating system
Managing files and external devices
Interrupts and buffered I/O

b) Human-computer communications
User interfaces: menus and graphics

c) Management of memory
Virtual memory and paging
Multi-user machines and networks
Scheduling and process coordination
Sequential and parallel processing

d) Hypertext
How hypertext organizes information
Uses of hypertext in encyclopedias, expert systems
Stacks and queues

Exercises

a) With a computer
Use diagnostics to check out a computer.
Use several graphical user interfaces on PC's.
Use an operating system on a larger machine.
Write a program using a stack or a queue.

Use a hypertext application. Create an application using a hypertext product.

b) Without a computer
Play a scheduling and resource allocation game.
Use index cards to simulate an interrupt stack.
Find out how computer viruses work.
Write a paper on ways to protect systems and files.
Study the problems with large systems and their vulnerability to errors.

8. Simulation, Statistics, and Probability

a) Using computers to make predictions
Modeling processes in the real world: weather, the stock market, election results, airflow around planes, pollution, population, traffic
Priority and wait queues
Probability of system failure and methods to safeguard against it

b) Probability
Set theory and Venn diagrams
Outcomes when tossing coins or throwing dice
Counting arguments, combinations and permutations
Random number generators

c) Statistics
Binomial and normal distributions
Distribution functions, the mean and standard deviation

Exercises

a) With a computer
Write and test a random number generator.
Use a random number generator to simulate tossing coins or rolling dice.
Play computer simulation games such as SimEarth.
Use a commercial simulation package to create simulations.
Change settings and parameters, then compare outcomes.

b) Without a computer
Play a traffic light simulation game.
Use real dice, coins, roulette wheels, etc. and compare the results with those obtained theoretically.
Design a 'house' strategy or 'best' strategy for blackjack.
Write a paper on the economic and environmental need for predictions and forecasts and the problems caused by incorrect or incomplete models.

9. Artificial Intelligence

a) Computers that play games
Games of strategy
Games with known strategy
Games programed to learn to win
Game trees as an example of binary trees

b) Expert systems and knowledge based reasoning
Examples of expert systems: physician's assistant, legal assistant, travel agent, 'mock psychoanalyst'

Deductive logic
Rules of propositional logic
Simple predicate logic

c) Robotics and prosthetic devices
Robots used in industry and the military
Computer controlled devices to aid the disabled

Exercises

a) With a computer
Write a program to play Nim or tic-tac-toe.
Implement a binary tree in an array.
Play Nim against the computer.
Devise several learning strategies. Try to play the game in a way that will compromise these strategies.
Use a commercial expert system and create an application.

b) Without a computer
Play Master Mind and try to write down the strategy used.
Then try to play following these rules blindly.
Do the same with some other games.
Build a simple robot from a kit.

Write a paper on an expert system in actual use.
Write a paper on the use of computer technology for the disabled.

10. **Theoretical Foundations of Computer Science**

a) Theoretical machines and formal languages
Finite state automata and Turing machines
Grammars and parsing
Compilers and interpreters

b) Complexity of algorithms
Rates of growth
Best, average and worst case analysis
Counting lines of code or comparisons
Simple sequences and series

c) Limits to computability
Logical paradoxes and puzzles
Existence of non-computable functions

Exercises

a) With a computer
Run a simulator of a finite automaton or a Turing machine.
Time different sorting algorithms to investigate their complexity.

b) Without a computer
Evaluate the complexity of a game such as 'Instant Insanity'.
Check whether a given word is accepted by a finite automaton.
Write a simple program for a Turing machine.
Check whether a given string can be derived from a given grammar.
Use logical deduction to solve puzzles and mysteries.
Write a paper on the problems caused by the complexity of large computerized systems.

11. **Graphs and Networks**

a) Computer networks
Topology, routing, redundancy
Network protocols, acknowledgments, handshakes
Problems of privacy, security, and breakdowns
Networked and distributed systems
Use in telephones, airline reservation systems, ATM networks

b) Graphs
Edges and vertices, adjacency, paths, weights, loops
Basic graph algorithms: traversals, mazes, backtracking, shortest path
Traveling salesman problem
Expression trees and their evaluation

c) Matrices
Adjacency matrices for graphs
Matrix operations

Exercises

a) With a computer
Use electronic mail where available.
Use a LAN (local area network) if available.
Run and if possible modify graph algorithm programs.

b) Without a computer
Work with graphs and mazes.
Multiply matrices and compute inverses.
Find the shortest path from one city to another.
Create an expression tree and evaluate it.
Write a paper on the breakdown problems of the telephone networks.
Write a paper on computer network crime.

12. **Advanced Computer Graphics**

a) Recursive drawings
Concept of recursion
Snowflakes, trees drawn from L-systems
Towers of Hanoi
Game of life and cellular automata
Modelling life with fractal drawings
Number of moves in Towers of Hanoi and length of fractal curves

b) Two and three dimensional pictures
Animation

Rotation of three dimensional pictures
CAD/CAM, computer aided manufacture and design
CAT scanners and ultrasound monitors

Exercises

a) With a computer
Examine programs that create recursive drawings, modify them, and investigate the effects of random variations.
Write programs to draw snowflakes or trees.
Run programs that create fractal drawings.
Run programs that draw three dimensional pictures.

b) Without a computer
Write a simplified CAT scan program using an article in Scientific American, September 1990.

Hand draw some recursive drawings.
Investigate some recursive traversals of graphs and trees.

Location of Ten Areas of Computer Science

The ten areas listed in the 1988 (3) and 1991 (11) and (12) reports are all covered in these modules. A table of areas and modules follows:

Areas		Modules
1.	Algorithms and Data Structures	
	Algorithms	1,5,9,10,11
	Data Structures	2,3,6,7,8,9,11
2.	Architecture	1, 4
3.	Artificial Intelligence & Robotics	9
4.	Database & Information Retrieval	3
5.	Human-Computer Communication	6, 7
6.	Numerical & Symbolic Computation	2
7.	Operating Systems	1, 7
8.	Programming Languages	4, 5, 10
9.	Software Methodology & Engineering	5
10.	Social, Ethical and Professional Issues	All

Location of Discrete Mathematics Topics

These modules also cover all but one of the discrete mathematics topics recommended in the 1991 report [11, page 27]. The one not included, proof techniques, was felt to be inappropriate for high school students. The mapping of areas to modules follows:

Areas	Modules
1. Sets	8
2. Functions	2
3. Elementary propositional and predicate logic	3, 9
4. Boolean algebra	3
5. Elementary graph theory	11
6. Matrices	11
7. Proof techniques	Not included
8. Combinatorics	8
9. Probability	8
10.Random numbers	8

Appendix F: A One Semester Introductory Computer Science Course

Carol E. Wolf, Pace University, NY

The following is an outline of a one semester computer science course for students in Arts and Science. It assumes little or no prior computer experience, but students should have familiarity with algebra.

If the course is offered to first year college students, it should consist of two one hour lectures a week followed by a two hour lab. In a high school setting, lectures and labs would probably be combined. However, even here, it might be wise to devote some days exclusively to class discussions and others to working on computers.

My own experience is that students become very involved with their computer work and do not wish to stop to listen to explanations. Separating class lectures and discussion from machine exploration seems to work best. In my classes, I require two students to work together on each machine. This works very well, since they teach each other.

For a first semester course with inexperienced students, it is necessary to have a laboratory work-book which explains what is to be done and lays out in detail how to do it.

Part way through the semester, term projects can be assigned which are open-ended. By this time, most students should be able to figure out how to do something without step by step instructions.

Logo and Pascal are the programming languages used here. However, these are only suggestions. Several schools have had great success with Scheme, and BASIC has a long useful history. Since the languages are only introduced in an elementary way, the particular choice is not too significant.

Week 1

Lectures Introduction to computers and the course:
Describe what computers can do and what the course requirements are.

Lab School computer system, files, Logo:
Give students enough information so that they can get started using the computer system available in the school. This includes formatting disks and loading programs. The rest of the time is taken up with discovering features of Logo, drawing pictures and saving them on disks.

Week 2

Lectures Computer components, algorithms:
Explain the basic parts of a computer including its CPU (central processing unit), memory, and I/O (input/output) devices. Introduce the concept of an algorithm as a recipe.

Lab Word processing:
Teach students enough about word processing to create a document, alter it, check its spelling, and print it out.

Week 3

Lectures Algorithms, Logo procedures:
Explain what an algorithm is both with everyday examples and with an introduction to Logo procedures.

Lab Logo:
Have students type in and run some Logo procedures, and then require them to compose some on their own.

Week 4

Lectures History, societal impact:
Discuss some of the history of how computers were developed and have entered everyone's lives.

Lab Introduction to spreadsheets:
Have students use a work-book to set up a spreadsheet, calculate some formulas, make changes, save it on a disk, and print it out.

Week 5

Lectures Number systems, ASCII code:
Introduce binary numbers, two's complement integers, scientific notation, and the ascii code for characters.

Lab Formulas in spreadsheets:
Have students create a more complicated spreadsheet, such as an interest table. Have them copy and move formulas.

Week 6

Lectures Expressions in a spreadsheet and in Logo:
Discuss arithmetic operators and built in functions in both a spreadsheet and Logo. Explain operator precedence.

Lab Graphing with spreadsheets:
Create graphs of some of the data stored in an earlier spreadsheet. If available, also introduce a presentation graphics package.

Week 7

Lectures Problem solving, Logo variables and the conditional statement:
Explain what a variable is and how it is used to store information. Use variables to create conditions in Logo procedures.

Lab Logo:
Give students examples of Logo procedures with parameters and global variables, and then have them write and run some of their own.

Week 8

Lectures Boolean variables and truth tables, database concepts:

Introduce truth tables for and, or, and not, and then explain logical data types. Describe files, records, fields and data types in a database.

Lab Database creation and use:
Have students set up a database and then use it. They should search using different conditions and also make reports of selected portions of the data.

Week 9

Lectures Database queries, searching and sorting, indexing, privacy:
Introduce several searching and sorting algorithms, and explain how indexing works. Discuss the widespread use of databases and their uses and abuses.

Lab Database queries:
Create another database, sort it and index it. Then use more complicated conditions to search and create reports.

Week 10

Lectures Chips, gates, circuits, and adders:
Describe the AND, OR, NOT, NAND, and NOR gates. Show how to combine them to produce different outputs including a simple adder.

Lab Database queries:
Practice with more complicated databases having date and logical fields.

Week 11

Lectures Problem solving and algorithms:
Discuss the algorithms used in searching and sorting, and introduce the concept of recursion.

Lab Recursion in Logo:
Have students type in and run a few simple recursive Logo procedures, and then let them write a few of their own. Show them more complicated recursive procedures such as the fractal snowflake curve.

Week 12

Lectures Computer architecture, simple machine example:
Describe a simple example of a von Neumann machine having a few opcodes and registers. Trace through the execution of a short program.

Lab Machine simulation:
Have students run a program which simulates the simple machine described in lectures. Give them a few programs to test out and then have them write several themselves.

Week 13

Lectures Programming languages, introduction to Pascal:
Describe the different types of programming languages, their uses and special features. Explain the structure of a Pascal program as contrasted with a Logo procedure.

Lab Pascal programming:
Give students a few simple programs to type in and run, and then have them modify these and create their own programs.

Week 14

Lectures More on Pascal:
Show how some of the features of Logo have their counterparts in Pascal. Explain the differences between a compiler, an interpreter and an assembler.

Lab Pascal programming:
Give examples of more complicated Pascal programs and then have students write similar ones themselves.

Week 15

Lectures Societal impact of computers:
Discuss the ways that computers have made our lives easier on the one hand, but have created their own problems for society on the other.

Lab Demonstration of term projects:
Have the student teams give oral reports and demonstrate their projects.

We Weren't Exactly Laughing...

People have started reporting their best reads -- books that were memorably pleasant. What's emerging is that lightness and playfulness in itself isn't what makes for special appeal; rather it is the presentation's power to enrich.

Based upon this tiny and non-random collection of volunteered titles, it seems that there are five not necessarily mutually exclusive realms of "pleasurable enrichment": i) extensions upon professional knowledge of a many concepts-per-page parade of new ideas or a put-it-all-together synthesis; ii) supplements of professional knowledge consisting of a novel look at things well known or offering insights into things we just never thought to inquire about before; iii) nonfiction "stories" -- books about the way in which the industry does business and biographies of leading figures; iv) fiction and novelties; and v) books that are fun because we don't have to read them but which were probably not construed by the author as being for pleasure (and which we would probably consider a chore if they were required.)

Area one includes A.K. Dewdney's anthologies of "computer recreations from Scientific American": The Armchair Universe and The Magic Machine. Speaking personally, my first real understanding of the Mandelbrot set, Julia sets, chaos, and strange attractors came from these articles. The new book by Steven Levy on Artificial Life is in this category as is a wonderful though largely unnoticed treatment of this same area by William Poundstone, The Recursive Universe. Douglas Hofstadter's Metamagical Themas, another collection of pieces from Scientific American, is here as is his Godel, Escher, Bach. The Cuckoo's Egg, by Cliff Stoll, belongs here too because of its information bearing on security.

Area two includes A.K. Dewdney's The Turing Omnibus: 61 Excursions into Computer Science with its extraordinary treatments of otherwise ordinary fare such as random numbers, text compression, NP-completeness, the fast Fourier transform, and public key cryptography (all of which are in Sedgewick's Algorithms text). Also, Martin Gardner's anthologies tend to fall into this area. Mathematical Puzzles & Diversions has a chapter with lots of interesting sidelights on the Towers of Hanoi, including an algorithm for its non-recursive solution. His Sixth Book of Mathematical Games has a nice chapter on parity checks with puzzles and magic tricks never found in serious textbooks (but which can add plenty of zest to lectures!). Mathematical Circus has a chapter on Boolean algebra giving biographical information about George and his very remarkable "six ladies." [These happen to be his wife and five daughters, each a woman of tremendous intellectual accomplishment.]

I've had an idea for book which would be fun to write and valuable: a comprehensive index to all of Gardner's math books (Donald Knuth, incidently, is the person to whom Mathematical Circus is dedicated). The index would enable us to put our hands right on general topics, algorithms, games, puzzles, tricks, riddles, and ridiculous questions that relate to lectures we want to spice-up.

Other area two books are Jon Bentley's two books of programming pearls and William Poundstone's Labyrinths of Reason. Area two and area one overlap because many predominately area two books survey recent developments.

Samples of books in area three are The Soul of a New Machine by Tracy Kidder, Hackers ("the story of the whiz kinds whose irreverence, idealism, and sheer genius changed the world") by Steven Levy, and Accidental Empires by Robert X. Cringely.

Area four samples are the short stories "The Last Question" by Isaac Asimov and "The Riddle of the Universe and Its Solution" by Christopher Cherniak. The latter is in The Mind's I by Douglas Hofstadter and Daniel Dennett along with other such items. Books include Donald Knuth's 3:16 Bible texts Illuminated, Alexander Tzonis's Hermes and the Golden Thinking Machine, Dennis Shasha's The Puzzling Adventures of Dr. Ecco, and Michael Wiesenberg's Puzzled Programmers.

Area five examples are Consciousness Explained by Daniel Dennett, The Emperor's New Mind by Roger Penrose, the Little Black Book of Computer Viruses by Mark Ludwig and, really, any could-be textbook you pick-up and read out of interest but not necessity.

Space is definitely posing a problem to me! Send in the names of computer pertinent books that you've especially enjoyed. We'll build and share a recommended reading list. Anyone interested in handling the compilation, possibly supplementing titles with annotations?

☺☺☺☺☺

Introduce truth tables for and, or, and not, and then explain logical data types. Describe files, records, fields and data types in a database.

Lab Database creation and use:
Have students set up a database and then use it. They should search using different conditions and also make reports of selected portions of the data.

Week 9

Lectures Database queries, searching and sorting, indexing, privacy:
Introduce several searching and sorting algorithms, and explain how indexing works. Discuss the widespread use of databases and their uses and abuses.

Lab Database queries:
Create another database, sort it and index it. Then use more complicated conditions to search and create reports.

Week 10

Lectures Chips, gates, circuits, and adders:
Describe the AND, OR, NOT, NAND, and NOR gates. Show how to combine them to produce different outputs including a simple adder.

Lab Database queries:
Practice with more complicated databases having date and logical fields.

Week 11

Lectures Problem solving and algorithms:
Discuss the algorithms used in searching and sorting, and introduce the concept of recursion.

Lab Recursion in Logo:
Have students type in and run a few simple recursive Logo procedures, and then let them write a few of their own. Show them more complicated recursive procedures such as the fractal snowflake curve.

Week 12

Lectures Computer architecture, simple machine example:
Describe a simple example of a von Neumann machine having a few opcodes and registers. Trace through the execution of a short program.

Lab Machine simulation:
Have students run a program which simulates the simple machine described in lectures. Give them a few programs to test out and then have them write several themselves.

Week 13

Lectures Programming languages, introduction to Pascal:
Describe the different types of programming languages, their uses and special features. Explain the structure of a Pascal program as contrasted with a Logo procedure.

Lab Pascal programming:
Give students a few simple programs to type in and run, and then have them modify these and create their own programs.

Week 14

Lectures More on Pascal:
Show how some of the features of Logo have their counterparts in Pascal. Explain the differences between a compiler, an interpreter and an assembler.

Lab Pascal programming:
Give examples of more complicated Pascal programs and then have students write similar ones themselves.

Week 15

Lectures Societal impact of computers:
Discuss the ways that computers have made our lives easier on the one hand, but have created their own problems for society on the other.

Lab Demonstration of term projects:
Have the student teams give oral reports and demonstrate their projects.