CURRICULUM '78

Recommendations for the Undergraduate Program in Computer Science

A Report of the ACM Curriculum Committee on Computer Science

Editors: Richard H. Austing, University of Maryland
Bruce H. Barnes, National Science Foundation
Della T. Bonnette, University of Southwestern Louisiana
Gerald L. Engel, Old Dominion University
Gordon Stokes, Brigham Young University

Contained in this report are the recommendations for the undergraduate degree program in Computer Science of the Curriculum Committee on Computer Science (C^3S) of the Association for Computing Machinery (ACM).

The core curriculum common to all computer science undergraduate programs is presented in terms of elementary level topics and courses, and intermediate level courses. Elective courses, used to round out an undergraduate program, are then discussed, and the entire program including the computer science component and other material is presented. Issues related to undergraduate computer science education, such as service courses, supporting areas, continuing education, facilities, staff, and articulation are presented.

Key Words and Phrases: computer sciences courses, computer science curriculum, computer science education, computer science undergraduate degree programs, service courses, continuing education

CR Categories: 1.52

Contents

1. Introduction

Curriculum development work in computer science has been a continuing effort of the Curriculum Committee on Computer Science (C^3S) of the Association for Computing Machinery (ACM). The work leading to the material presented in this report was started under the chairmanship of C^3S of Preston Hammer, and continued when John Hamblen was appointed chairman in 1976.

In the time since the publication of "Curriculum '68" [1] by C^3S, many significant developments have occurred within computer science education, and many educational efforts have been undertaken by C^3S, other groups within ACM, and other professional organizations. As part of the background work in preparation of this report, an extensive survey of the literature of computer science education since "Curriculum '68" was prepared and published [2]. The efforts of C^3S since 1968 are summarized in this document.

The writing group, in its preparation of this set of recommendations, paid considerable attention to the developments as reported in the literature, and to informal comments received regarding "Curriculum '68." In addition to this, a variety of individuals, representing many different types of institutions, and many different interests within computer science, were brought into C^3S meetings and working sessions to present their ideas. A working draft of the report was prepared and published in the June 1977 *SIGCSE Bulletin* in order that the material receive as wide a distribution as possible, and to provide an opportunity for input from interested individuals. Prior to the publication of the working paper, draft reports on specific areas were widely circulated and numerous panel and discussion sessions were held both to inform interested parties of the thinking of the Committee and to allow for comments and suggestions on the work done to that point.

The wide circulation of the various drafts and working papers resulted in numerous suggestions and constructive criticisms, many of which have been incorporated into this final document. In addition to this input, a relationship of mutual benefit has developed by interaction with the parallel, but independent, development of the Model Curricula Subcommittee of the IEEE Computer Society leading to the publication of their curriculum guidelines in Computer Science and Engineering [3].

The writing group is most grateful to all those individuals who contributed to the effort. The Appendix contains the names and affiliations of those people who contributed by serving on C^3S, by supplying course outlines, by supplying comments on the draft report, and in other ways contributing to the final version presented here. The Committee, of course, assumes full responsibility for the substance of this material and the recommendations contained herein.

The report first presents the core curriculum common to all computer science undergraduate programs. This is presented in Section 2 in terms of elementary level material and courses, and intermediate level courses. Section 3 presents computer science electives that may be used to round out an undergraduate program. In Section 4, the full course of study is presented which includes the computer science component, and other material necessary in a program at the bachelor degree level. The important areas of service courses, including general service courses, supporting areas, and continuing education are discussed in Section 5. The report concludes by addressing the areas of facilities, staff, and articulation in Section 6.

In studying this report, it should be recognized that it is a set of guidelines, prepared by a group of individuals working in a committee mode. As such, the recommendations will not satisfy everyone, nor is it intended that they be appropriate to all institutions. It is the hope of the Committee that this report will further stimulate computer science educators to think about their programs and, as appropriate, to share their thinking with others. If this is done, the primary objective of the preparation of these guidelines will have been met.

2. Core Curriculum

2.1 Introduction

Within the present work, C^3S has considered the classification scheme of computer science as defined in "Curriculum '68" with a view to isolating those areas which should be common to all computer science undergraduate degree programs.

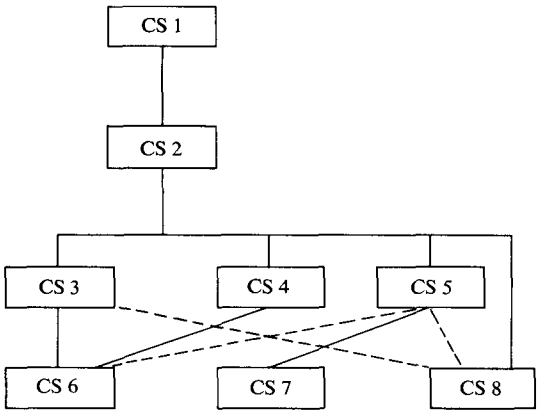
The core curriculum, described in this section, represents this refinement. The material is divided into a section on elementary material, including the specifications of topics at this level and the description of five sample courses, and the intermediate levels, including the description of three sample courses. This collection of eight courses represents one way to include the required core material in the computer science undergraduate major.

While the course material is detailed later on in the section, to gain perspective the eight courses (three semester hours each) are listed here:

CS 1. Computer Programming I
CS 2. Computer Programming II
CS 3. Introduction to Computer Systems
CS 4. Introduction to Computer Organization
CS 5. Introduction to File Processing
CS 6. Operating Systems and Computer Architecture I
CS 7. Data Structures and Algorithm Analysis
CS 8. Organization of Programming Languages

The structuring of the courses as to prerequisites is shown in Figure 1. The solid lines represent required prerequisites, while the dashed lines represent highly recommended prerequisites. This diagram includes courses representing only the computer science material considered to be essential to the program. The entire program, including relevant mathematics requirements, is illustrated in Figure 2 on page 160.

Fig. 1. Computer science core curriculum.



The discussion of the core course material in this section concentrates on the computer science components which are necessary for the undergraduate program. The relationship of this material to two-year programs (especially transfer programs) and the developing high school programs will be considered in Section 6.4.

The elementary core material represents subject matter necessary for all students in computer science in order to be able to achieve the objectives of the undergraduate major. The intermediate level core material follows naturally by providing the students who have been equipped with the basics of the field with the tools to be operational computer scientists.

2.2 Objectives

The core material is required as a prerequisite for advanced courses in the field and thus it is essential that the material be presented early in the program. In learning this material, the computer science student should be provided with the foundation for achieving at least the objectives of an undergraduate degree program that are listed below.

Computer science majors should:

1. be able to write programs in a reasonable amount of time that work correctly, are well documented, and are readable;
2. be able to determine whether or not they have written a reasonably efficient and well organized program;
3. know what general types of problems are amenable to computer solution, and the various tools necessary for solving such problems;
4. be able to assess the implications of work performed either as an individual or as a member of a team;
5. understand basic computer architectures;
6. be prepared to pursue in-depth training in one or more application areas or further education in computer science.

It should be recognized that these alone do not represent the total objectives of an undergraduate program, but only those directly related to the computer science component. Material addressing other requirements and electives is covered in Section 4.4.

2.3 Elementary Material

In order to facilitate the attainment of the objectives above, computer science majors must be given a thorough grounding in the study of the implementation of algorithms in programming languages which operate on data structures in the environment of hardware. Emphasis at the elementary level then should be placed on algorithms, programming, and data structures, but with a good understanding of the hardware capabilities involved in their implementation.

Specifically, the following topics are considered elementary. They should be common to all undergraduate programs in computer science.

Programming Topics

P1. Algorithms: includes the concept and properties of algorithms; the role of algorithms in the problem solving process; constructs and languages to facilitate the expression of algorithms.

P2. Programming Languages: includes basic syntax and semantics of a higher level (problem oriented) language; subprograms; I/O; recursion.

P3. Programming Style: includes the preparation of readable, understandable, modifiable, and more easily verifiable programs through the application of concepts and techniques of structured programming; program documentation; some practical aspects of proving programs correct. (Note: Programming style should pervade the entire curriculum rather than be considered as a separate topic.)

P4. Debugging and Verification: includes the use of debugging software, selection of test data; techniques for error detection; relation of good programming style to the use of error detection; and program verification.

P5. Applications: includes an introduction to uses of selected topics in areas such as information retrieval, file management, lexical analysis, string processing and numeric computation; need for and examples of different types of programming languages; social, philosophical, and ethical considerations.

Software Organization

S1. Computer Structure and Machine Language: includes organization of computers in terms of I/O, storage, control and processing units; register and storage structures, instruction format and execution; principal instruction types; machine arithmetic; program control; I/O operations; interrupts.

S2. Data Representation: includes bits, bytes, words and other information structures; number representation; representation of elementary data structures; data transmission, error detection and correction; fixed versus variable word lengths.

S3. Symbolic Coding and Assembly Systems: includes mnemonic operation codes; labels; symbolic addresses and address expressions; literals; extended machine operations and pseudo operations; error flags and messages; scanning of symbolic instruc-

tions and symbol table construction; overall design and operation of assemblers, compilers, and interpreters.

S4. Addressing Techniques: includes absolute, relative, base associative, indirect, and immediate addressing; indexing; memory mapping functions; storage allocation, paging and machine organization to facilitate modes of addressing.

S5. Macros: includes definition, call, expansion of macros; parameter handling; conditional assembly and assembly time computation.

S6. Program Segmentation and Linkage: includes subroutines, coroutines and functions; subprogram loading and linkage; common data linkage transfer vectors; parameter passing and binding; overlays; re-entrant subprograms; stacking techniques; linkage using page and segment tables.

S7. Linkers and Loaders: separate compilation of subroutines; incoming and outgoing symbols; relocation; resolving intersegment references by direct and indirect linking.

S8. Systems and Utility Programs: includes basic concepts of loaders, I/O systems, human interface with operating systems; program libraries.

Hardware Organization

H1. Computer Systems Organization: includes characteristics of, and relationships between I/O devices, processors, control units, main and auxiliary storage devices; organization of modules into a system; multiple processor configurations and computer networks; relationship between computer organization and software.

H2. Logic Design: includes basic digital circuits; AND, OR, and NOT elements; half-adder, adder, storage and delay elements; encoding-decoding logic; basic concepts of microprogramming; logical equivalence between hardware and software; elements of switching algebra; combinatorial and sequential networks.

H3. Data Representation and Transfer: includes codes, number representation; flipflops, registers, gates.

H4. Digital Arithmetic: includes serial versus parallel adders; subtraction and signed magnitude versus complemented arithmetic; multiply/divide algorithms; elementary speed-up techniques for arithmetic.

H5. Digital Storage and Accessing: includes memory control; data and address buses; addressing and accessing methods; memory segmentation; data flow in multimemory and hierarchical systems.

H6. Control and I/O: includes synchronous and asynchronous control; interrupts; modes of communication with processors.

H7. Reliability: includes error detection and correction, diagnostics.

Data Structures and File Processing

D1. Data Structures: includes arrays, strings, stacks, queues, linked lists; representation in memory; algorithms for manipulating data within these structures.

D2. Sorting and searching: includes algorithms for in-core sorting and searching methods; comparative efficiency of methods; table lookup techniques; hash coding.

D3. Trees: includes basic terminology and types; representation as binary trees; traversal schemes; representation in memory; breadth-first and depth-first search techniques; threading.

D4. File Terminology: includes record, file, blocking, database; overall idea of database management systems.

D5. Sequential Access: includes physical characteristics of appropriate storage media; sort/merge algorithms; file manipulation techniques for updating, deleting, and inserting records.

D6. Random Access: includes physical characteristics of appropriate storage media; physical representation of data structures on storage devices; algorithms and techniques for implementing inverted lists, multilists, indexed sequential, hierarchical structures.

D7. File I/O: includes file control systems (directory, allocation, file control table, file security); I/O specification statements for allocating space and cataloging files; file utility routines; data handling (format definition, block buffering, buffer pools, compaction).

2.4 Implementation Considerations

Throughout the presentation of the elementary level material, programming projects should be assigned; these projects should be designed to aid in the comprehension and use of language details, to exemplify the problem solving process, and/or to introduce more advanced areas of computer science.

Good programming style should be stressed in the teaching of all of this material. The discipline required to achieve style will promote the development of effective algorithms and should result in students writing correct, understandable programs. Thus emphasis in the programming exercises should be placed on efficient algorithms, structured programming techniques, and good documentation.

A specific course on structured programming, or on programming style, is not intended at the elementary level. The topics are of such importance that they should be considered a common thread throughout the entire curriculum and, as such, should be totally integrated into the curriculum. They provide a philosophy of discipline which pervades all of the course work.

Throughout the presentation of this elementary material, meaningful actual computer applications should be cited and reviewed. In the process of so doing, reference must be made to the social, philosophical, and ethical considerations involved in the applications. Like structured programming, these issues are of such importance to the development of the computer scientist that they must permeate the instruction at this level.

It would be desirable, though not necessary, for the

computer science major to be familiar with all of the elementary level topics before taking intermediate level courses. This, however, may not always be possible. Factors influencing how and when courses are offered which include the material are: the purpose and circumstances of a particular department within the context of its educational institution, the availability of computer resources, and whether an institution is on the quarter or semester system.

Most courses at this level should include laboratory sessions. These laboratories provide the student with the opportunity to gain practical experience by actually solving problems on the computer. Laboratory sessions should be implemented in such a way that the student can develop good programming techniques under close supervision. The instructor may or may not be the same as for the lecture portion of the course. The absence of a specific laboratory in a course description does not imply that programming should not be required.

2.5 Sample Elementary Level Courses

The following set of courses is provided merely as a sample to illustrate one of the ways in which core material at the elementary level might be presented. Other implementations are possible. No matter what implementation is attempted, however, all of the elementary material specified in Section 2.3 should be included so that students are equipped with adequate background for intermediate and advanced level material.

Each course described in the sample set is assumed to be offered on a semester basis. Suggested numbers of hours of credit are given in parentheses immediately after the course titles. For example, (2-2-3) indicates two hours of lectures and two hours of laboratory per week for a total of three semester hours of credit.

CS 1. Computer Programming I (2-2-3)

The objectives of this course are:

(a) to introduce problem solving methods and algorithm development;

(b) to teach a high level programming language that is widely used; and

(c) to teach how to design, code, debug, and document programs using techniques of good programming style.

COURSE OUTLINE

The material on a high level programming language and on algorithm development can be taught best as an integrated whole. Thus the topics should not be covered sequentially. The emphasis of the course is on the techniques of algorithm development and programming with style. Neither esoteric features of a programming language nor other aspects of computers should be allowed to interfere with that purpose.

TOPICS

A. *Computer Organization.* An overview identifying components and their functions, machine and assembly languages. (5%)

B. *Programming Language and Programming.* Representation of integers, reals, characters, instructions. Data types, constants, variables. Arithmetic expression. Assignment statement. Logical expression. Sequencing, alternation, and iteration. Arrays. Subprograms and parameters. Simple I/O. Programming projects utilizing concepts and emphasizing good programming style .(45%)

C. *Algorithm Development.* Techniques of problem solving. Flowcharting. Stepwise refinement. Simple numerical examples. Algorithms for searching (e.g. linear, binary), sorting (e.g. exchange, insertion), merging of ordered lists. Examples taken from such areas as business applications involving data manipulation, and simulations involving games. (45%)

D. *Examinations.* (5%)

CS 2. Computer Programming II (2-2-3)

Prerequisite: CS 1

The objectives of this course are:

(a) to continue the development of discipline in program design, in style and expression, in debugging and testing, especially for larger programs;

(b) to introduce algorithmic analysis; and

(c) to introduce basic aspects of string processing, recursion, internal search/sort methods and simple data structures.

COURSE OUTLINE

The topics in this outline should be introduced as needed in the context of one or more projects involving larger programs. The instructor may choose to begin with the statement of a sizeable project, then utilize structured programming techniques to develop a number of small projects each of which involves string processing, recursion, searching and sorting, or data structures. The emphasis on good programming style, expression, and documentation, begun in CS 1, should be continued. In order to do this effectively, it may be necessary to introduce a second language (especially if a language like Fortran is used in CS 1). In that case, details of the language should be included in the outline. Analysis of algorithms should be introduced, but at this level such analysis should be given by the instructor to the student.

Consideration should be given to the implementation of programming projects by organizing students into programming teams. This technique is essential in advanced level courses and should be attempted as early as possible in the curriculum. If large class size makes such an approach impractical, every effort should be made to have each student's programs read and critiqued by another student.

TOPICS

A. *Review.* Principles of good programming style, expression, and documentation. Details of a second language if appropriate. (15%)

B. *Structured Programming Concepts.* Control flow. Invariant relation of a loop. Stepwise refinement of

both statements and data structures, or top-down programming. (40%)

C. *Debugging and Testing.* (10%)

D. *String Processing.* Concatenation. Substrings. Matching. (5%)

E. *Internal Searching and Sorting.* Methods such as binary, radix, Shell, quicksort, merge sort. Hash coding. (10%)

F. *Data Structures.* Linear allocation (e.g. stacks, queues, deques) and linked allocation (e.g. simple linked lists). (10%)

G. *Recursion.* (5%)

H. *Examinations.* (5%)

CS 3. Introduction to Computer Systems (2-2-3)

Prerequisite: CS 2

The objectives of this course are:

(a) to provide basic concepts of computer systems;
(b) to introduce computer architecture; and
(c) to teach an assembly language.

COURSE OUTLINE

The extent to which each topic is discussed and the ordering of topics depends on the facilities available and the nature and orientation of CS 4 described below. Enough assembly language details should be covered and projects assigned so that the student gains experience in programming a specific computer. However, concepts and techniques that apply to a broad range of computers should be emphasized. Programming methods that are developed in CS 1 and CS 2 should also be utilized in this course.

TOPICS

A. *Computer Structure and Machine Language.* Memory, control, processing and I/O units. Registers, principal machine instruction types and their formats. Character representation. Program control. Fetch-execute cycle. Timing. I/O operations. (15%)

B. *Assembly Language.* Mnemonic operations. Symbolic addresses. Assembler concepts and instruction format. Data-word definition. Literals. Location counter. Error flags and messages. Implementation of high level language constructs. (30%)

C. *Addressing Techniques.* Indexing. Indirect Addressing. Absolute and relative addressing. (5%)

D. *Macros.* Definition. Call. Parameters. Expansion. Nesting. Conditional assembly. (10%)

E. *File I/O.* Basic physical characteristics of I/O and auxiliary storage devices. File control system. I/O specification statements and device handlers. Data handling, including buffering and blocking. (5%)

F. *Program Segmentation and Linkage.* Subroutines. Coroutines. Recursive and re-entrant routines. (20%)

G. *Assembler Construction.* One-pass and two-pass assemblers. Relocation. Relocatable loaders. (5%)

H. *Interpretive Routines.* Simulators. Trace. (5%)

I. *Examinations.* (5%)

CS 4. Introduction to Computer Organization (3-0-3) or (2-2-3)

Prerequisite: CS 2

The objectives of this course are:

(a) to introduce the organization and structuring of the major hardware components of computers;
(b) to understand the mechanics of information transfer and control within a digital computer system; and
(c) to provide the fundamentals of logic design.

COURSE OUTLINE

The three main categories in the outline, namely computer architecture, arithmetic, and basic logic design, should be interwoven throughout the course rather than taught sequentially. The first two of these areas may be covered, at least in part, in CS 3 and the amount of material included in this course will depend on how the topics are divided between the two courses. The logic design part of the outline is specific and essential to this course. The functional, logic design level is emphasized rather than circuit details which are more appropriate in engineering curricula. The functional level provides the student with an understanding of the mechanics of information transfer and control within the computer system. Although much of the course material can and should be presented in a form that is independent of any particular technology, it is recommended that an actual, simple minicomputer or microcomputer system be studied. A supplemental laboratory is appropriate for that purpose.

TOPICS

A. *Basic Logic Design.* Representation of both data and control information by digital (binary) signals. Logic properties of elemental devices for processing (gates) and storing (flipflops) information. Description by truth tables, Boolean functions and timing diagrams. Analysis and synthesis of combinatorial networks of commonly used gate types. Parallel and serial registers. Analysis and synthesis of simple synchronous control mechanisms; data and address buses; addressing and accessing methods; memory segmentation. Practical methods of timing pulse generation. (25%)

B. *Coding.* Commonly used codes (e.g. BCD, ASCII). Parity generation and detection. Encoders, decoders, code converters. (5%)

C. *Number Representation and Arithmetic.* Binary number representation, unsigned addition and subtraction. One's and two's complement, signed magnitude and excess radix number representations and their pros and cons for implementing elementary arithmetic for BCD and excess-3 representations. (10%)

D. *Computer Architecture.* Functions of, and communication between, large-scale components of a computer system. Hardware implementation and sequencing of instruction fetch, address construction, and instruction execution. Data flow and control

block diagrams of a simple processor. Concept of microprogram and analogy with software. Properties of simple I/O devices and their controllers, synchronous control, interrupts. Modes of communications with processors. (35%)

E. *Example.* Study of an actual, simple minicomputer or microcomputer system. (20%)

F. *Examinations.* (5%)

CS 5. Introduction to File Processing (3-0-3)

Prerequisite: CS 2

The objectives of this course are:

(a) to introduce concepts and techniques of structuring data on bulk storage devices;

(b) to provide experience in the use of bulk storage devices; and

(c) to provide the foundation for applications of data structures and file processing techniques.

COURSE OUTLINE

The emphasis given to topics in this outline will vary depending on the computer facilities available to students. Programming projects should be assigned to give students experience in file processing. Characteristics and utilization of a variety of storage devices should be covered even though some of the devices are not part of the computer system that is used. Algorithmic analysis and programming techniques developed in CS 2 should be utilized.

TOPICS

A. *File Processing Environment.* Definitions of record, file, blocking, compaction, database. Overview of database management system. (5%)

B. *Sequential Access.* Physical characteristics of sequential media (tape, cards, etc.). External sort/merge algorithms. File manipulation techniques for updating, deleting and inserting records in sequential files. (30%)

C. *Data Structures.* Algorithms for manipulating linked lists. Binary, B-trees, B*-trees, and AVL trees. Algorithms for traversing and balancing trees. Basic concepts of networks (plex structures). (20%)

D. *Random Access.* Physical characteristics of disk/drum and other bulk storage devices. Physical representation of data structure on storage devices. Algorithms and techniques for implementing inverted lists, multilist, indexed sequential, and hierarchical structures. (35%)

E. *File I/O.* File control systems and utility routines, I/O specification statements for allocating space and cataloging files. (5%)

F. *Examinations.* (5%)

2.6 Sample Intermediate Level Courses

Sample versions of three courses at the intermediate level are given to illustrate topics and material which should be required of all computer science majors. This material and the elementary level topics in Section 2.3 constitute the minimum requirements which should be common to all computer science undergraduate programs to achieve the basic objectives of those programs.

Courses which cover the intermediate level material contain a strong emphasis on fundamental concepts exemplified by various types of programming languages, architecture and operating systems, and data structures. Neither theoretical treatments nor case study approaches in and of themselves are adequate or appropriate at this level. Advanced level (elective) courses may be used for predominantly theoretical treatment of topics or for comprehensive case studies.

CS 6. Operating Systems and Computer Architecture I (2-2-3)

Prerequisite: CS 3 and CS 4
(CS 5 recommended)

The objectives of this course are:

(a) to develop an understanding of the organization and architecture of computer systems at the register-transfer and programming levels of system description;

(b) to introduce the major concept areas of operating systems principles;

(c) to teach the inter-relationships between the operating system and the architecture of computer systems.

COURSE OUTLINE

This course should emphasize concepts rather than case studies. Subtleties do exist, however, in operating systems that do not readily follow from concepts alone. It is recommended that a laboratory requiring hands on experience be included with this course.

The laboratory for the course would ideally use a small computer where students could actually implement sections of operating systems and have them fail without serious consequences to other users. This system should have, at a minimum, a CPU, memory, disk or tape, and some terminal device such as a teletype or CRT. The second best choice for the laboratory experience would be a simulated system running on a larger machine.

The course material should be liberally sprinkled with examples of operating system segments implemented on particular computer system architectures. The interdependence of operating systems and architecture should be clearly delineated. Integrating these subjects at an early stage in the curriculum is particularly important because the effects of computer architecture on systems software has long been recognized. Also, modern systems combine the design of operating systems and the architecture.

TOPICS

A. *Review.* Instruction sets. I/O and interrupt structure. Addressing schemes. Microprogramming. (10%)

B. *Dynamic Procedure Activation.* Procedure activation and deactivation on a stack, including dynamic storage allocation, passing value and reference parameters, establishing new local environments, addressing mechanisms for accessing parameters (e.g. displays,

relative addressing in the stack). Implementing nonlocal references. Re-entrant programs. Implementation on register machines. (15%)

C. *System Structure.* Design methodologies such as level, abstract data types, monitors, kernels, nuclei, networks of operating system modules. Proving correctness. (10%)
D. *Evaluation.* Elementary queueing, network models of systems, bottlenecks, program behavior, and statistical analysis. (15%)
E. *Memory Management.* Characteristics of the hierarchy of storage media, virtual memory, paging, segmentation. Policies and mechanisms for efficiency of mapping operations and storage utilization. Memory protection. Multiprogramming. Problems of auxiliary memory. (20%)
F. *Process Management.* Asynchronous processes. Using interrupt hardware to trigger software procedure calls. Process stateword and automatic SWITCH instructions. Semaphores. Ready lists. Implementing a simple scheduler. Examples of process control problems such as deadlock, producer/consumers, readers/writers. (20%)
G. *Recovery Procedures.* Techniques of automatic and manual recovery in the event of system failures. (5%)
H. *Examinations.* (5%)

CS 7. Data Structures and Algorithm Analysis (3-0-3)

Prerequisite: CS 5

The objectives of this course are:
(a) to apply analysis and design techniques to nonnumeric algorithms which act on data structures;
(b) to utilize algorithmic analysis and design criteria in the selection of methods for data manipulation in the environment of a database management system.

COURSE OUTLINE

The material in this outline could be covered sequentially in a course. It is designed to build on the foundation established in the elementary material, particularly on that material which involves algorithm development (P1, P3) and data structures and file processing (D1, D7). The practical approach in the earlier material should be made more rigorous in this course through the use of techniques for the analysis and design of efficient algorithms. The results of this more formal study should then be incorporated into data management system design decisions. This involves differentiating between theoretical or experimental results for individual methods and the results which might actually be achieved in systems which integrate a variety of methods and data structures. Thus, database management systems provide the applications environment for topics discussed in the course.

Projects and assignments should involve implementation of theoretical results. This suggests an alternative way of covering the material in the course, namely to treat concepts, algorithms, and analysis in class and deal with their impact on system design in assignments. Of course, some in-class discussions of this impact would occur, but at various times throughout the course rather than concentrated at the end.

TOPICS

A. *Review.* Basic data structures such as stacks, queues, lists, trees. Algorithms for their implementation. (10%)
B. *Graphs.* Definition, terminology, and property (e.g. connectivity). Algorithms for finding paths and spanning trees. (15%)
C. *Algorithms Design and Analysis.* Basic techniques of design and analysis of efficient algorithms for internal and external sorting/merging/searching. Intuitive notions of complexity (e.g. NP-hard problems). (30%)
D. *Memory Management.* Hashing. Algorithms for dynamic storage allocation (e.g. buddy system, boundary-tag), garbage collection and compaction. (15%)
E. *System Design.* Integration of data structures, sort/merge/search methods (internal and external) and memory media into a simple database management system. Accessing methods. Effects on run time, costs, efficiency. (25%)
F. *Examinations.* (5%)

CS 8. Organization of Programming Languages (3-0-3)

Prerequisite: CS 2 (CS 3 and CS 5 highly recommended)

The objectives of this course are:
(a) to develop an understanding of the organization of programming languages, especially the run-time behavior of programs;
(b) to introduce the formal study of programming language specification and analysis;
(c) to continue the development of problem solution and programming skills introduced in the elementary level material.

COURSE OUTLINE

This is an applied course in programming language constructs emphasizing the run-time behavior of programs. It should provide appropriate background for advanced level courses involving formal and theoretical aspects of programming languages and/or the compilation process.

The material in this outline is not intended to be covered sequentially. Instead, programming languages could be specified and analyzed one at a time in terms of their features and limitations based on their run-time environments. Alternatively, desirable specification of programming languages could be discussed and then exemplified by citing their implementations in various languages. In either case, programming exercises in each language should be assigned to emphasize the implementations of language features.

TOPICS

A. *Language Definition Structure.* Formal language concepts including syntax and basic characteristics of grammars, especially finite state, context-free, and ambiguous. Backus-Naur Form. A language such as Algol as an example. (15%)

B. *Data Types and Structures.* Review of basic data types, including lists and trees. Constructs for specifying and manipulating data types. Language features affecting static and dynamic data storage management. (10%)

C. *Control Structures and Data Flow.* Programming language constructs for specifying program control and data transfer, including DO . . . FOR, DO . . . WHILE, REPEAT . . . UNTIL, BREAK, subroutines, procedures, block structures, and interrupts. Decision tables, recursion. Relationship with good programming style should be emphasized. (15%)

D. *Run-time Consideration.* The effects of the run-time environment and binding time on various features of programming languages. (25%)

E. *Interpretative Languages.* Compilation vs. interpretation. String processing with language features such as those available in SNOBOL 4. Vector processing with language features such as those available in APL. (20%)

F. *Lexical Analysis and Parsing.* An introduction to lexical analysis including scanning, finite state acceptors and symbol tables. An introduction to parsing and compilers including push-down acceptors, top-down and bottom-up parsing. (10%)

G. *Examinations.* (5%)

3. Computer Science Electives

3.1 Introduction

In this section a variety of computer science electives will be considered which are appropriate at the elementary and advanced levels. Elective courses at the elementary level, while enhancing the program of a student, normally should not be used to meet the requirements of the major program. Elective courses at the advanced level should be selected to meet major requirements as well as to allow the student to explore particular areas of computer science in more detail.

3.2 Elementary Level

At the elementary level it would be highly desirable to provide a mechanism for offering courses in specific programming languages such as APL, Cobol, LISP, or PL/I which could be taken as electives by computer science majors or majors in other disciplines. The extent of the course, the number of credits offered and the prerequisites would depend on the language offered and the purpose for offering it. One convenient way to achieve this goal would be to include in the curriculum a Programming Language Laboratory for variable credit (i.e. one to three semester hours). The prerequisite could be designated in general as "consent of instructor" or more specifically as CS 1 or CS 2 and the laboratory could be taken for repeated credit provided that different languages were taught. In addition to its function as an elective, the laboratory could be offered in conjunction with an intermediate or advanced course, thus enabling an instructor to require students to learn a specific language at the same time they take a course (e.g. LISP in the laboratory along with CS 7—Data Structures and Algorithm Analysis).

3.3 Advanced Level

Ten advanced level elective courses are specified. Computer Science departments should offer as many as possible of these courses on a regular basis, but few departments are expected to have sufficient resources to offer all, or even a large majority, of them. Possible additional courses which could be offered as special topics are listed in Section 3.4.

CS 9. Computers and Society (3-0-3)

Prerequisite: elementary core material

The objectives of this course are:

(a) to present concepts of social value and valuations;

(b) to introduce models which describe the impact of computers on society;

(c) to provide a framework for professional activity that involves explicit consideration of and decisions concerning social impact;

(d) to present tools and techniques which are applicable to problems posed by the social impact of computers.

Much debate surrounds the role of this course in the curriculum. While few will disagree that professional computer scientists should be instructed to evaluate social issues regarding that which they do, it has been argued that such a course is not a computer science course, but rather should be in the area of the social sciences. Another argument is presented which states that this material is so important that it should not merely be covered in a single course, but instead should be integrated throughout the curriculum. Although this latter argument has validity, it is difficult to insure sufficient coverage of topics when they are scattered throughout a number of courses. As a result it is recommended that this course be considered at least as a strongly recommended elective. If, in fact, the material to meet the above objectives is not covered in the other intermediate and advanced level courses in this program, then this course should be required.

A computer science major taking an advanced level computers and society course would be expected to be familiar with the elementary material described in the previous section. All of that material, however, is not necessarily prerequisite for such a course. The prerequisite should, in fact, be chosen in such a manner that nonmajors would also be able to take the course. A mixture of majors in such a course would provide broadening

interchange and would benefit both the computer science students and the other majors. The course should be taught by the computer science faculty, but team-teaching with faculty from other disciplines should be encouraged. The course could be general and treat a number of computer impact topics, or specific, and treat in depth one of the topics (such as legal issues in computing). This recommendation is conditioned on the assumption that instructors who present material on societal impact, whether as an entire course or as part of other courses, will try to include both sides of or approaches to issues without instilling their own philosophical leanings on complex societal issues. For example, certain topics contain political overtones which should be discussed, but which, if not done carefully, can give the material a political science flavor it does not deserve.

A strict outline is not given. The number of topics and extent of coverage as well as the instructional techniques used can vary considerably and still meet the objectives of the course. A term project involving computer applications that are manifested in the local community is strongly recommended. Possible topics, but certainly not an exhaustive list, that could be included in such a course are as follows:

A. History of computing and technology
B. The place of the computer in modern society
C. The computer and the individual
D. Survey of computer applications
E. Legal issues
F. Computers in decision-making processes
G. The computer scientist as a professional
H. Futurists' views of computing
I. Public perception of computers and computer scientists

CS 10. Operating Systems and Computer Architecture II (2-2-3)

Prerequisite: CS 6; Corequisite: a course in statistics

COURSE OUTLINE

This course continues the development of the material in CS 6. Emphasis should be on intrasystem communication.

TOPICS

A. *Review.* I/O and interrupt structure. Addressing schemes. Memory management. (10%)
B. *Concurrent Processes.* Concepts of processes in parallel. Problems associated with determinancy, freedom from deadlock, mutual exclusion, and synchronization. (15%)
C. *Name Management.* Limitations of linear address space. Implementation of tree-structured space of objects for the support of modular programming. (15%)
D. *Resource Allocation.* Queueing and network control policies. Concepts of system balancing and thrashing. Job activation/deactivation. Process scheduling. Multiprogramming systems. (25%)
E. *Protection.* Contraints for accessing objects. Mechanism to specify and enforce access rules. Implementation in existing systems. (15%)
F. *Advanced Architecture and Operating Systems Implementations.* Pipelining and parallelism. User interface considerations. Introduction to telecommunications, networks (including minicomputers) and distributed systems. (15%)
G. *Examinations.* (5%)

CS 11. Database Management Systems Design (3-0-3)

Prerequisites: CS 6 and CS 7

COURSE OUTLINE

This course should emphasize the concepts and structures necessary to design and implement a database management system. The student should become acquainted with current literature on the subject and should be given an opportunity to use a database management system if possible.

During the course the student should gain an understanding of various physical file organization and data organization techniques. The concept of data models should be covered and the network, relational, and hierarchical data models should be explored. Examples of specific database management systems should be examined and related to the data models discussed. The student should become familiar with normalized forms of data relations including canonical schema representations. Techniques of systems design and implementation should be discussed and practiced. Data integrity and file security techniques should be explored. The major experience of the course should be the design and implementation of a simple database management system that would include file security and some form of query into the system.

TOPICS

A. *Introduction to Database Concepts.* Goals of DBMS including data independence, relationships, logical and physical organizations, schema and subschema. (5%)
B. *Data Models.* Hierarchical, network, and relational models with a description of the logical and data structure representation of the database system. Examples of implementations of the various models. (15%)
C. *Data Normalization.* First, second, and third normal forms of data relations. Canonical schema. Data independence. (5%)
D. *Data Description Languages.* Forms, applications, examples, design strategies. (10%)
E. *Query Facilities.* Relational algebra, relational calculus, data structures for establishing relations. Query functions. Design and translation strategies. (15%)

F. *File Organization.* Storage hierarchies, data structures, multiple key systems, indexed files, hashing. Physical characteristics. (25%)
G. *Index Organization.* Relation to files. Inverted file systems. Design strategies. (5%)
H. *File Security.* Authentication, authorization, transformation, encryptions. Hardware and software techniques. Design strategies. (10%)
I. *Data Integrity and Reliability.* Redundancy, recovery, locking, and monitoring. (5%)
J. *Examinations.* (5%)

CS 12. Artificial Intelligence (3-0-3)

Prerequisite: CS 7

COURSE OUTLINE

This course introduces students to basic concepts and techniques of artificial intelligence, or intelligent systems, and gives insights into active research areas and applications. Emphasis is placed on representation as a central and necessary concept for work in intelligent systems. Strategies for choosing representations as well as notational systems and structures should be discussed. Students should understand, for example, that the selection of a programming language is really a basic representational choice and that an important component of that choice is whether the programming language is really the basic representational mode or whether it is a translator/interpreter of an intermediate representational mode such as the predicate calculus or other notational system (e.g. modal or fuzzy logics).

Other issues of importance in this course are natural language, vision systems, search strategies, and control. The extent and type of coverage will vary. The use of natural language and vision systems in applications of intelligent systems research to other disciplines should be emphasized. Search strategies should be seen as being implicit in representation and control. General issues related to control should be discussed and illustrated by examples of existing systems. A variety of applications could be mentioned at the beginning of the course as motivation for studying intelligent systems. These applications could then be elaborated on at appropriate times throughout the course or at the end.

Students could profit from a background in LISP because of its widespread use in artificial intelligence work. A Programming Language Laboratory as described in Section 3.2 could be used to provide this background either concurrently or with CS 7. If neither alternative is possible, then an introduction to LISP could be included in the course during the discussion of representation, but there would not be enough time for an in-depth treatment of the language.

TOPICS

A. *Representation.* Constraints and capabilities of notational systems such as logics and programming languages. Notational structures such as trees, networks, statistical representations, and frames. Strategies for choosing representations (e.g. exploiting natural constraints in data, representation of similar patterns as in analogies). Introduction to LISP. (40%)
B. *Search Strategies.* Tree and graph searches (e.g. depth and breadth first, minimax, alpha-beta). Heuristics. (15%)
C. *Control.* General characteristics of production and procedurally oriented systems. Parallel vs. serial processing. Existing systems to illustrate issues (e.g. HEARSAY II, DENDRAL, MYCIN). (20%)
D. *Communication and Perception.* Introduction to concepts related to current research in natural language and in vision systems. Use of tactility in intelligent systems. (10%)
E. *Applications.* Sampling of current work in such areas as psychology, medicine, science, architecture, and such machines as industrial robots. (10%)
F. *Examinations.* (5%)

CS 13. Algorithms (3-0-3)

Prerequisites: CS 7 and CS 8

COURSE OUTLINE

This course should develop students' abilities as writers and critics of programs by exposing students to problems and their algorithmic solution. As programming is both art and science, student programmers can benefit considerably from analysis of case studies in a wide variety of areas. All options for presenting algorithms in a very high level language should be considered, without regard for whether a processor exists for that language. Translation of each algorithm to a more machine-readable form can be given separately, if necessary. Careful choice of the level of abstraction appropriate to a given problem should be made as a means of adjusting students' load in the course.

Domain independent techniques should emerge during the course as algorithm-rich topics are presented from various areas. One convenient classification of topics into areas to ensure breadth of coverage is: combinatorics, numerical analysis, systems programming, and artificial intelligence. Algorithms from a majority of these areas should be analyzed, although not necessarily in the order indicated in the outline. The percentage ranges are intended to give instructors flexibility in choosing areas and topics.

TOPICS

A. *Combinatorics.* Algorithms for unordered and ordered sets, graphs, matrices (within the semi-ring paradigm), bit vectors. (10-25%)
B. *Numerical Analysis.* Algorithms for integer arithmetic (fast multiplication, prime testing, sieves, factoring, greatest common denominator, linear Diophantine equations), real arithmetic (Taylor series, how various calculators work), polynomial arithmetic, random numbers, matrix operations (inversion, determinants). (10-25%)

C. *Systems Programming.* Algorithms in text processors (pattern matching) language processors (parsing, storage management), operating systems (scheduling, synchronization), database management (sorting, searching). (10-25%)
D. *Artificial Intelligence.* Algorithms in natural language processing (concordances, context-free parsers), robotics (vision, manipulator operation), theorem proving and problem solving (decision methods, search heuristics). (10-25%)
E. *Domain Independent Techniques.* Divide-and-conquer. Solution of recurrence equations. Dynamic programming. (15%)
F. *Examinations.* (5%)

CS 14. Software Design and Development (3-0-3) or (2-2-3)
Prerequisites: CS 7 and CS 8

COURSE OUTLINE

This course presents a formal approach to state-of-the-art techniques in software design and development and provides a means for students to apply the techniques. An integral part of the course is the involvement of students working in teams in the organization, management, and development of a large software project. The team project aspect can be facilitated either by scheduling separate laboratories or by using some of the lecture periods to discuss practical aspects of the team projects.

TOPICS

A. *Design Techniques.* Formal models of structured programming. Demonstrations of code reading and correctness. Stepwise refinement and reorganization. Segmentation. Top-down design and development. Information hiding. Iterative enhancement. Structured design. Strength and coupling measures. (50%)
B. *Organization and Management.* Milestones and estimating. Chief programmer teams. Program libraries. Walk-throughs. Documentation. (15%)
C. *Team Project.* Organization, management, and development of a large scale software project by students working in teams. (30%)
D. *Examinations.* (5%)

CS 15. Theory of Programming Languages (3-0-3)
Prerequisite: CS 8

COURSE OUTLINE

This is a course in the formal treatment of programming language translation and compiler design concepts. Course material builds on the background established in CS 8, specifically on the introduction to lexical analysis, parsing, and compilers. Emphasis should be on the theoretical aspects of parsing context-free languages, translation specifications, and machine-independent code improvement. Programming projects to demonstrate various concepts are desirable, but extensive projects to write compilers, or major components of compilers, should be deferred to a special topics course on compiler writing.

TOPICS

A. *Review.* Grammars, languages, and their syntax and semantics. Concepts of parsing and ambiguity. BNF description of Algol. (15%)
B. *Scanners.* Finite state grammars and recognizers. Lexical scanners. Implementation of symbol tables. (20%)
C. *Parsers.* Theory and examples of context-free languages and push-down automata (PDA). Context-free parsing techniques such as recursive descent, LL(k), precedence, LR(k), SLR (k). (40%)
D. *Translation.* Techniques of machine-independent code generation and improvement. Inherited and synthesized attributes. Syntax directed translation schema. (20%)
E. *Examinations.* (5%)

CS 16. Automata, Computability, and Formal Languages (3-0-3)
Prerequisites: CS 8 and MA 4 (see Sect. 4.1)

COURSE OUTLINE

This course offers a diverse sampling of the areas of theoretical computer science and their hierarchical interconnections. Basic results relating to formal models of computation should be introduced. Stress should be given to developing students' skills in understanding rigorous definitions in computing environments and in determining their logical consequences. In this regard strong emphasis should be placed on problem assignments and their evaluations.

Material need not be presented in the order specified, but it is important to give nearly equal emphasis among the major areas. Topics within each area can be covered in greater depth in appropriate special topics courses.

TOPICS

A. *Finite State Concepts.* Acceptors (including nondeterminism). Regular expressions. Closure properties. Sequential machines and finite state transducers. State minimization. (30%)
B. *Formal Grammars.* Chomsky hierarchy grammars, pushdown acceptors and linear bounded automata. Closure properties and algorithms on grammars. (35%)
C. *Computability and Turing Machines.* Turing machine as acceptor and transducer. Universal machine. Computable and noncomputable functions. Halting problem. (30%)
D. *Examinations.* (5%)

CS 17. Numerical Mathematics: Analysis (3-0-3)
Prerequisites: CS 1 and MA 5

COURSE OUTLINE

This course with CS 18 forms a one-year introduction to numerical analysis. The courses are intended to

be independent of each other. Students should be expected not only to learn the basic algorithms of numerical computation, but also to understand the theoretical foundations of the algorithms and various problems related to the practical implementations of the algorithms. Thus each topic implies a discussion of the algorithm, the related theory, and the benefits, disadvantages, and pitfalls associated with the method. Programming assignments should be given to illustrate solutions of realistic problems rather than just the coding of various algorithms. Topics such as convergence and error analysis for specific algorithms should be treated in a theoretical manner. Floating point arithmetic and use of mathematical subroutine packages are included in both courses because they should be discussed throughout the courses as they relate to specific problems. All other topics in each course should be covered sequentially. The depth to which topics are treated may vary, but most, if not all, topics should be discussed.

TOPICS

A. *Floating Point Arithmetic.* Basic concepts of floating point number systems. Implications of finite precision. Illustrations of errors due to roundoff. (15%)
B. *Use of Mathematical Subroutine Packages.* (5%)
C. *Interpolation.* Finite difference calculus. Polynomial interpolation. Inverse interpolation. Spline interpolation. (15%)
D. *Approximation.* Uniform approximation. Discrete least-squares. Polynomial approximation. Fourier approximation. Chebyshev economization. (10%)
E. *Numerical Integration and Differentiation.* Interpolatory numerical integration. Euler-McLauren sum formula. Gaussian quadrature. Adaptive integration. Fast Fourier transform. Richardson extrapolation and numerical differentiation. (15%)
F. *Solution of Nonlinear Equations.* Bisection. Fixed point iteration. Newton's method. Secant method. Muller's method. Aitken's process. Rates of convergence. Efficient evaluation of polynomials. Bairstow's method. (15%)
G. *Solution of Ordinary Differential Equations.* Taylor series methods. Euler's method, with local and global error analysis. Runge-Kutta methods. Predictor-corrector methods. Automatic error monitoring—change of step size and order. Stability. (20%)
H. *Examinations.* (5%)

CS 18. Numerical Mathematics: Linear Algebra (3-0-3)

Prerequisites: CS 1 and MA 5

COURSE OUTLINE

The same remarks apply to this course as to CS 17.

TOPICS

A. *Floating Point Arithmetic.* Basic concepts of floating point number systems. Implications of finite precision. Illustrations of errors due to roundoff. (15%)
B. *Use of Mathematical Subroutine Packages.* (5%)
C. *Direct Methods for Linear Systems of Equations.* Gaussian elimination. Operational counts. Implementation, including pivoting and scaling. Direct factorization methods. (20%)
D. *Error Analysis and Norms.* Vector norms and matrix norms. Condition numbers and error estimates. Iterative improvement. (15%)
E. *Iterative Methods.* Jacobi's method. Gauss-Seidel method. Acceleration of iterative methods. Overrelaxation. (15%)
F. *Computation of Eigenvalues and Eigenvectors.* Basic theorems. Error estimates. The power method. Jacobi's method. Householder's method. (15%)
G. *Related Topics.* Numerical solution of boundary value problems for ordinary differential equations. Solution of nonlinear systems of algebraic equations. Least-squares solution of overdetermined systems. (10%)
H. *Examinations.* (5%)

3.4 Special Topics

The special topics courses should be offered whenever departmental resources are sufficient to do so. Thus content and prerequisites may vary each time they are offered because the available material is changing rapidly and different faculty members may have widely differing opinions of what should be included in a course. Most importantly, the material should be current and topical. In time, some of the material should be integrated into courses previously specified or may replace entire courses in the curriculum. Monitoring of this phase of the program should be a continuing activity of individual departments and C³S.

Examples of special topics courses include:

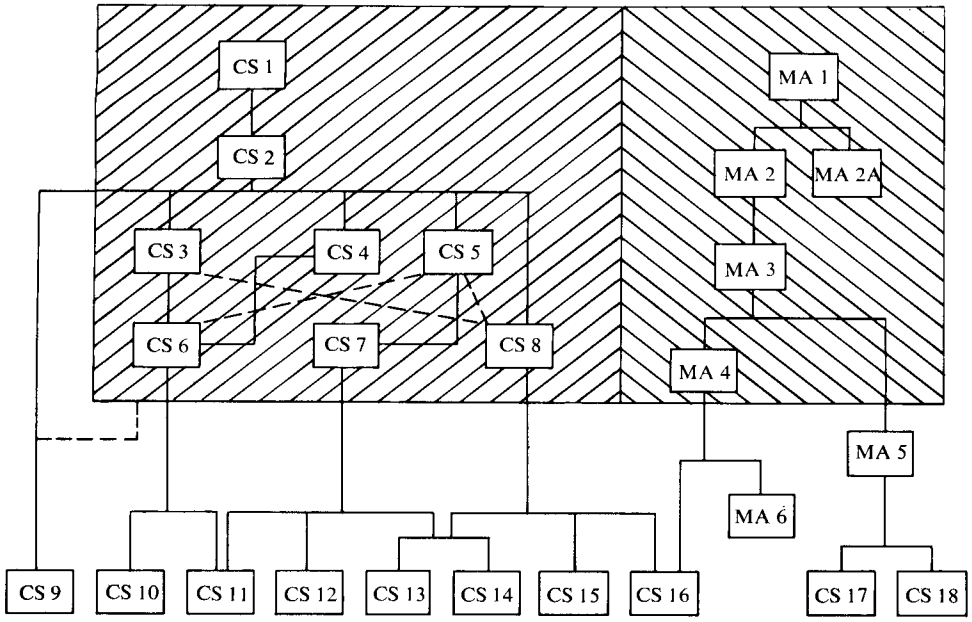
A. Microcomputer Laboratory
B. Minicomputer Laboratory
C. Performance Evaluation
D. Telecommunications/Networks/Distributed Systems
E. Systems Simulation
F. Advanced Systems Programming
G. Graphics
H. Compiler Writing Laboratory
I. Structured Programming
J. Topics in Automata Theory
K. Topics in Computability
L. Topics in Formal Language Theory
M. Simulation and Modeling

4. The Undergraduate Program

4.1 Introduction

Outlines of eighteen computer science courses are included in previous sections. Eight of the courses indicate one of the ways in which the core material might be presented. Ten courses along with thirteen topics courses

Fig. 2. Recommended computer science and mathematics courses.



illustrate the kind of elective material to be offered at an advanced level.

The eighteen computer science courses are as follows:

CS 1. Computer Programming I
CS 2. Computer Programming II
CS 3. Introduction to Computer Systems
CS 4. Introduction to Computer Organization
CS 5. Introduction to File Processing
CS 6. Operating Systems and Computer Architecture I
CS 7. Data Structures and Algorithm Analysis
CS 8. Organization of Programming Languages
CS 9. Computers and Society
CS 10. Operating Systems and Computer Architecture II
CS 11. Database Management Systems Design
CS 12. Artificial Intelligence
CS 13. Algorithms
CS 14. Software Design and Development
CS 15. Theory of Programming Languages
CS 16. Automata, Computability, and Formal Languages
CS 17. Numerical Mathematics: Analysis
CS 18. Numerical Mathematics: Linear Algebra

The structure of these courses is given in Figure 2. The following set of mathematics courses is included in the structure for completeness and because of its relevance to an undergraduate program in computer science:

MA 1. Introductory Calculus
MA 2. Mathematical Analysis I
MA 2A. Probability
MA 3. Linear Algebra
MA 4. Discrete Structures
MA 5. Mathematical Analysis II
MA 6. Probability and Statistics

Their role and the extent to which they conform to the needs of a computer science major are discussed in Section 4.3.

Solid and dashed lines represent, respectively, absolute and recommended prerequisites. The shaded area depicts the core curriculum in computer science and required mathematics courses.

4.2 Computer Science Requirements and Electives

The computer science major will consist of the eight courses of the core material plus four additional courses selected from the recommended computer science advanced electives with no more than two in any one specific subfield of the disciplines. Within the requirements for the four elective courses, the special topics courses specified in Section 3.4 should also be considered as possible electives for the major.

It should be noted that as students proceed through the computer science portion of the program, they begin at a very practical level and as they progress the work becomes more conceptual and theoretical. At the junior level the program is strongly conceptual while in the senior year the program may be fully theoretical, or involve a significant amount of theory supplemented with laboratory activities.

4.3 Mathematics Requirements

An understanding of and the capability to use a number of mathematical concepts and techniques are vitally important for a computer scientist. Analytical and algebraic techniques, logic, finite mathematics, aspects of linear algebra, combinatorics, graph theory, optimization methods, probability, and statistics are, in various ways, intimately associated with the development of computer science concepts and techniques. For example, probability and statistics develop the required tools for measure-

ment and evaluation of programs and systems, two important aspects of computer science. Analysis, as commonly contained in calculus courses, gives the mathematical bases for important concepts such as sets, relations, functions, limits, and convergence. Discrete structures provides the bases for semigroups, groups, trees, graphs, and combinatorics, all of which have applications in algorithms analysis and testing, as well as in data structure design. Thus mathematics requirements are integral to a computer science curriculum even though specific courses are not cited as prerequisites for most computer science courses. Unfortunately, the kind and amount of material needed from these areas for computer science usually can only be obtained, if at all, from the regular courses offered by departments of mathematics for their own majors.

Ideally, computer science and mathematics departments should cooperate in developing courses concentrating on discrete mathematics which are appropriate to the needs of computer scientists. Such courses, however, if offered by mathematics departments, would substantially increase their service course load and would constitute a heavy additional commitment of their resources. On the other hand, these course offerings could constitute an applied mathematics component which, in turn, might provide attractive alternatives for some mathematics departments. Suitable computer oriented mathematics course offerings constitute an important topic which should be explored more thoroughly both on local (i.e. individual institutions) and national levels. Specific course recommendations, however, are outside the domain of this report.

Until such time as suitable courses become readily available, it will be necessary to rely on the most commonly offered mathematics courses for the mathematical background needed by computer science majors. One set of such courses was recommended in 1965 by the Committee on Undergraduate Programs in Mathematics (CUPM) of the Mathematical Association of America. Courses MA 1, 2, 2A, 3, 5, and 6 in the structure included in Section 4.1 are intended to be CUPM recommended courses. Details on course contents can be found in the CUPM report [5].

MA 4 represents a more advanced course in discrete structures than that given in "Curriculum '68". The course will build on concepts developed by the study of calculus and linear algebra and will emphasize applications of discrete mathematics to computer science. In particular, if techniques in probability are not included in an earlier course, some emphasis should be given to them in this course. A number of examples of suitable outlines for this course have appeared in the literature, primarily in the *SIGCSE Bulletin* [6, 7, 8, 9, 10].

If courses of the type cited above are the only kind of mathematics courses available, then MA 1, MA 2, MA 2A, MA 3, and MA 4 should be required of all computer science majors. In addition, MA 5 or MA 6 may be required depending on which advanced level computer science electives are selected. If more appropriate courses are provided as a result of interaction between computer science and mathematics departments, then the specification of required mathematics courses and the prerequisite structure should be reconsidered.

4.4 Other Requirements and Electives

As specified in this report, the minimum requirements are 36 semester hours in computer science and 15 semester hours in mathematics. This is certainly less than half of the required hours of a typical undergraduate degree program.

Additional requirements and electives will vary with the requirements of the individual institutions and hence only the most general of recommendations can be given.

It is certainly recognized that writing and communication skills must be emphasized throughout the program. This must be accomplished by requiring appropriate courses in the humanities, and also by emphasis on these skills in the courses within the computer science program itself. Surveys of employers stress the need for these skills as a requirement for employment.

Science and engineering departments represent fruitful areas for support of a computer science program. For those institutions with access to an engineering program, courses such as switching circuits and digital logic should be utilized. Within the science departments, a number of options are available to meet general university requirements. In addition to courses in fields such as physics, it should be noted that the increasing emphasis on computing in the biological and environmental sciences offers additional options for students.

A large portion of the job market involves work in business oriented computer fields. As a result, in those cases where there is a business school or related department, it would be most appropriate to take courses in which one could learn the technology and techniques appropriate to this field. For those students choosing this path, business courses develop the background necessary to function in the business environment.

The general university requirements in the social sciences, with careful advising, will generally be adequate, although it should be recognized that increasing use of computers in these fields may make it appropriate for some students to devise a minor in such an area if that is within their interests.

In consideration of this entire area of general requirements and electives, it must be recognized that a person who is going into the computer job market at the bachelor's level will, in all likelihood, initially be a systems, scientific, engineering, or business programmer. As a result, the student is well advised to work out a program with an advisor that will provide a meaningful and thorough background in the area of the student's interest. The general liberal arts requirements of the institution will give the necessary breadth to the program. A well developed concentration in an area other than com-

puter science will put the student in a position to develop and grow in that area as well as in computer science.

5. Service Courses

5.1 Introduction

There is a great need and demand for computer science material by students who do not intend to major in computer science. Faculty of computer science departments must be willing to offer different courses for those students than for majors when that is appropriate. Service courses should be offered by computer science faculty rather than by faculty in other departments. This, of course, implies that the courses must be made appealing by providing appropriate computer science content in a manner that is attuned to the needs, levels, and backgrounds of the students taking such courses.

There is some possibility that certain courses can be team-taught by faculty from computer science and from one or more other disciplines, but it must be recognized that this approach is difficult. Heads of departments must make difficult decisions regarding how much of the department's teaching resources is to be used for majors and how much is to be used for students in other disciplines. In making these decisions, it is essential that the department and institution properly acknowledge and reward faculty who are working in this area, if the courses are to maintain a high level of excellence.

A variety of service courses must be considered to satisfy the diverse needs of groups of students. Among the categories of undergraduate level courses are the following: (a) liberal arts or general university requirements; (b) supporting work for majors in other disciplines; and (c) continuing education.

5.2 General Service Courses

Students taking a course to satisfy a requirement such as a general university requirement may come from any discipline other than computer science. Some of the science, engineering, or mathematics oriented students may profit most by taking the same first course recommended for computer science students (CS 1). This has an immediate advantage for students who become interested enough in computing to want additional computer science courses. They will have the prerequisite for the second (and subsequent) courses for the computer science major. Those students who stop after one or two of these courses at least have excellent basic programming techniques to apply to computer oriented work in their discipline.

Other students will require more specialized study than that listed in CS 1. For many of these students the courses listed in the section on elementary computer science electives may be more appropriate.

It must still be recognized that a different course (or courses) must be provided for majors in the fields mentioned above as well as for majors in business oriented fields, social sciences, education, and humanities. Service courses for these students normally should include a combination of computer appreciation, programming, applications, and societal impact. Different mixes of these broad areas should be considered for different groups, and the amount of each is best determined by each institution. Topics within each area should be as pertinent to the group served as possible, especially in the language chosen to illustrate programming. To meet this goal, feedback from students is important and communication between computer science and other departments, including periodic review of the courses, is essential. The course should have no prerequisites and it should be made clear to the students that the course is not intended for those who want additional work in computer science. If local conditions warrant, the material could be presented in two semesters rather than one.

Though as indicated, full specification of such courses is impossible, an example can be given to illustrate the kind of course under consideration:

CSS 1. Computer Applications and Impact (3-0-3)

COURSE OUTLINE

A survey of computer applications in areas such as file management, gaming, CAI, process control, simulation, and modeling. Impact of computers on individuals and society. Problem solving using computers with emphasis on analysis, formulation of algorithms, and programming. Projects chosen from various application areas of student interest.

TOPICS (percentages dependent on local situations)

A. *Computer Systems:* Batch and interactive, real time, information management, networks. Description of each system, how it differs from the others, and kinds of applications for which each system is best suited.
B. *Databases:* Establishment and use. Data definition and structures.
C. *Errors:* Types, effects, handling.
D. *Social Implications:* Human-machine interface. Privacy. Moral and legal issues.
E. *Future Social Impact:* Checkless society. CAI. National data banks.
F. *Languages:* As appropriate, introduction to a business oriented language, a symbol manipulation language, and/or a procedure oriented language. Brief exposition of characteristics which make these languages appropriate for particular classes of problems.
G. *Concepts and Techniques Used in Solving Problems:* Selected from appropriate application areas such as CAI, data management, gaming, information retrieval, and simulation.
H. *Projects and Examinations.*

5.3 Supporting Areas

A number of students will choose computer science as a supporting (or minor) area. Various possibilities

for sets of courses should be available. One of the ways to achieve this by using the same courses as taken by a computer science major is to require courses CS 1 and CS 2; at least two of the courses CS 3, CS 4, CS 5; and at least two of the courses CS 6, CS 7, CS 8. Additional courses could then be taken as student interest and program requirements would allow. Computer science faculty should communicate with faculty from other departments to determine the needs of the other departments and to indicate how certain courses or course combinations might satisfy the needs.

In those cases where existing courses are not appropriate as supporting work for other majors, new courses should be created, probably to be offered as upper division level courses. Two alternatives for establishing sets of courses for use as supporting work are as follows: (a) CS 1 and CS 2, one course combining material from CS 5 and CS 7, and one course combining material from CS 3, CS 4, and CS 6; and (b) CS 1 and CS 2, one course combining material from CS 3 and CS 5, and one course combining material from CS 4, CS 6, and CS 7. Alternative (a) attempts to combine similar topics from different levels while alternative (b) attempts to combine different topics from similar levels. It should be recognized that students who complete either of the latter two alternatives may not be well enough prepared to take a more advanced computer science course for which any of the courses CS 6, CS 7, or CS 8 are prerequisite.

5.4 Continuing Education

Continuing education is an area which has grown so rapidly and includes such a large variety of interests that it is virtually impossible to specify course possibilities. Nevertheless, computer science departments must address the needs appropriate to their local situations. Some of the possibilities which should be considered are: (a) adult education courses, probably versions of the courses suggested to meet general university requirements; (b) professional development seminars, usually consisting of one day to several weeks devoted to a specific topical area (e.g. minicomputers, database management systems); and (c) courses offered in the evenings or on weekends (on or off campus), possibly regular course offerings or modifications of them primarily for employed persons who need to acquire or enhance their computer science background. The latter possibility would include full-scale baccalaureate or master's degree programs.

6. Other Considerations

6.1 Introduction

Implementation of the computer science curriculum recommendations given in this report implies more than the development of a coherent program of courses. Articulation with other educational institutions and with employers of graduates of such programs must be given serious attention, and a commitment must be made to provide and maintain these resources. In most cases, such commitments go well beyond the boundaries of computer science departments.

Specific requirements involving such areas as staff, equipment, and articulation will vary among institutions depending on such things as size, location, capability, and mission of the school and program. As a result, specific recommendation in these areas cannot be given. However, in this section, general guidelines for implementation in these areas are discussed.

6.2 Facilities

In order to implement the full set of recommendations contained in this report, a wide range of computing facilities will be required. Equipment such as data entry devices, microcomputers, minicomputers, and medium or large-scale computer systems all play separate and important roles in the development of the computer scientist.

Data entry devices such as card punches, teletypewriters, and display terminals should be provided for program preparation and communication between student and computer. Such equipment should be conveniently located and in a large enough area for both easy and convenient student access and use. This equipment may be provided and maintained by the central computing facility at the institution for general student and faculty use, or, if enrollments in the computer science program and demands for service warrant, the equipment may be located and maintained by the department with some restriction on the use by other departments. To implement successfully an adequate program that insures easy and ready access to such facilities, close cooperation and planning is necessary that will involve the computer science department, the computer center, and, perhaps, other departments which use these computer facilities.

Microcomputers are quite desirable in teaching details of computer architecture previously only attainable by extensive programming of "hypothetical computers," simulators, or textbook discussions. They have provided a relatively inexpensive and highly versatile resource which can be used in a variety of ways including combining several such units into reasonably sophisticated and powerful computer systems. Their use is becoming so widespread that in addition to using microcomputers in a systems course, under some circumstances, consideration may be given to offering a laboratory course in which each student, or a group of students in the course, would purchase a suitable kit and construct a computer.

The availability of one or more minicomputers in a department allows the students to obtain "hands-on" experience as well as the opportunity to utilize interactive systems and programming languages which may not be available, or practical, on a medium or large-scale computer system. This kind of equipment also allows the

student to work on software development projects, and other projects that might not be possible due to restrictions on the use of the central facility. It is desirable that the department maintain and schedule such minicomputer facilities in such a way that student usage and software development can proceed in an orderly fashion through laboratory course work and individual projects.

A medium or large-scale computer, normally operated and maintained as a central facility at the institution for use by all departments, should provide appropriate hardware and software support for the major program. Auxiliary memory is required in order to store files so that access methods specified in the core courses can be implemented and tested. Suitable input/output devices and system facilities are needed so that rapid turnaround of student jobs is possible, interactive computing is available, and programming languages used in the curriculum are supported.

Regardless of what specific items of computer equipment are available to support a curriculum in computer science, effective teaching and research in the field require laboratory facilities. Computer science is in part an empirical science which involves implementing procedures as well as studying theoretically based processes. Because systems, algorithms, languages, and data structures are created, studied, and measured via combinations of hardware and software, it is essential that appropriate laboratory facilities be made available that are comparable to those necessary in the physical and biological sciences and engineering disciplines. This implies that appropriate laboratory facilities are available for student and faculty use, and may imply that additional laboratory space is required by certain faculty and students for special purposes. The initial budgetary support for establishing these laboratories may be substantial, and continuing regular budgetary support is essential for successful implementation of a program.

While we have thus far stressed the hardware facilities necessary for the recommended curriculum, equal attention must also be given to software. In order for the student to master the material in the core and elective courses, sufficient higher level languages must be available. Additionally, special purpose systems such as statistical systems, database management systems, information storage and retrieval systems, and simulation systems should be available for student use. It must be recognized in planning that many of these systems require a significant initial and continuing investment on the part of the institution. Where possible, fast turnaround or interactive systems should be considered in order to provide as much access as possible for the student.

In addition to the computer related facilities required for the recommended curriculum, there is also a requirement for those resources of a university that are normally associated with any discipline. Adequate library facilities, including significant holdings of periodicals are absolutely necessary, and the implementor of this report is referred to the basic library list [4] for a basis of establishing a library collection to support the instructional program.

While traditional library support is essential to the computer science program, it must be recognized that the field requires some additional resources that may not be necessary in other disciplines. Specifically, the student of computer science must have available, in some form, language, programming, and systems manuals as well as documentation for programs and other materials directly related to the development and use of systems. This material must be easily and conveniently available to the student at all times.

6.3 Staff

Insofar as it is possible, the vast majority of faculty members in departments offering the curriculum that has been recommended in this report should have their primary academic training in computer science. At the same time, it remains the case that demand exceeds supply for these individuals and it is often necessary, and in some cases desirable, to acquire faculty with degrees in other disciplines, but who have experience in computing through teaching or employment in government, business, or industry.

The size of the department will depend on available resources, required teaching loads, commitments to offering service courses, and commitments to continuing education programs. Approximately six full-time equivalent faculty members are necessary to offer a minimal program that would include the core courses as well as a selection of elective and service courses. Most of these faculty members should be capable of offering all of the core courses in addition to elective courses in their areas of specialization. Additional continuing instructional support may be available from the computer center, and from other departments such as mathematics which may offer numerical analysis or other applied mathematics courses that could be cross-listed by both departments. In addition, adjunct faculty from local government, business, or industry are valuable additions in many cases. Such individuals are often able to bring a different perspective to the program; however, care must be taken to insure that the program does not become overly dependent on individuals who may be unable to perform continuing service.

Because of the rapid growth of this field, consideration must be given to providing ongoing opportunities for faculty development, such as a sabbatical leave program, opportunities to attend professional development seminars, and interchange programs with industry.

A department which operates its own laboratory facilities should consider obtaining a full-time staff member to maintain such systems, be responsible for necessary documentation and languages, and coordinate other activities connected with the laboratory. Such a staff

member would provide continuity in the development of the laboratory resource.

The field is still developing rapidly, and as was indicated earlier, is at least in part empirical in nature. As a result faculty will be required to devote a great deal of time to course development, software development, development of laboratory resources, and development of service offerings. To provide for continuing excellence in these areas, it must be recognized that they are essential contributions to the program and profession, and as such should be considered within the context of the reward structure of the institution.

6.4 Articulation

It is imperative that departments offering computer science programs keep in close contact with secondary schools, community and junior colleges, graduate schools, and prospective employers of their graduates. This requires a continuing, time consuming effort. Primary responsibility for this effort could be placed with one faculty member, whose teaching load should then be reduced. Experience has shown that person-to-person contact on a continuing basis is necessary for successful articulation.

Usually, a central office in a four-year institution has direct contact with secondary schools. With computing becoming more prevalent at that level, however, it is highly useful and appropriate for a departmental representative to maintain contact with those local secondary schools which offer, or desire to offer, courses in computing.

Articulation agreements exist in many areas between four-year institutions and community and junior colleges. These agreements need to be updated frequently as programs or courses change, and personal contact between departments is necessary to keep abreast of these changes. Transfer programs in community and junior colleges are often geared to programs at four-year institutions. As a result, proposed changes in the four-year program which influence transfer programs should be promulgated as soon as possible so that the community and junior colleges can incorporate such changes, thereby reducing the lag between programs to the benefit of transfer students.

Some of the graduates of the recommended program will continue academic work in computer science in graduate school, but most will seek employment upon graduation. Departments must be aware of the graduate school requirements so that their programs prepare students adequately for advanced work in the field, but they must also maintain communication with employers in order to know what job requirements exist so that the faculty can advise students more effectively. Feedback from recent graduates of the program is quite useful in this regard and should be encouraged as much as possible. In order to most effectively implement this aspect of the program, faculty members should have available to them graduate school brochures, Civil Service Commission documents, and whatever else can come from personal contacts with employees in government and industry, as well as from the professional societies.

References

1. Curriculum Committee on Computer Science (C³S). Curriculum '68, recommendations for academic programs in computer science. *Comm. ACM 11*, 3 (March 1968), 151-197.
2. Austing, R.H., Barnes, B.H., and Engel, G.L. A survey of the literature in computer science education since Curriculum '68. *Comm. ACM 20*, 1 (Jan. 1977), 13-21.
3. Education Committee (Model Curriculum Subcommittee) of the IEEE Computer Society. A curriculum in computer science and engineering. Committee Report, IEEE Pub. EH0119-8, January 1977.
4. Joint Committee of the ACM and the IEEE Computer Society. A library list on undergraduate computer science–computer engineering and information systems. Committee Report, IEEE Pub. EH0131-3, 1978.
5. Committee on the Undergraduate Program in Mathematics. A general curriculum in mathematics for colleges. Rep. to Math. Assoc. of America, CUPM, Berkeley, Calif., 1965.
6. Special Interest Group on Computer Science Education. SIGCSE Bulletin, (ACM) *5*, 1 (Feb. 1973).
7. Special Interest Group on Computer Science Education. SIGCSE Bulletin, (ACM) *6*, 1 (Feb. 1974).
8. Special Interest Group on Computer Science Education. SIGCSE Bulletin, (ACM) *7*, 1 (Feb. 1975).
9. Special Interest Group on Computer Science Education. SIGCSE Bulletin, (ACM) *8*, 1 (Feb. 1976).
10. Special Interest Group on Computer Science Education. SIGCSE Bulletin, (ACM) *8*, 3 (Aug. 1976).

Appendix

Contributors to the C³S Report

Robert M. Aiken, University of Tennessee
Michael A. Arbib, University of Massachusetts
Julius A. Archibald, SUNY at Plattsburgh
William Atchison, University of Maryland
Richard Austing, University of Maryland
Bruce Barnes, National Science Foundation
Victor R. Basili, University of Maryland
Barry Bateman, Southern Illinois University
Della T. Bonnette, University of Southwestern Louisiana
W.P. Buckley, Aluminum Company of America
Frank Cable, Pennsylvania State University
Gary Carlson, Brigham Young University
B.F. Caviness, Rensselaer Polytechnic Institute
Donald Chand, Georgia State University
Sam Conte, Purdue University
William Cotterman, Georgia State University
Daniel Couger, University of Colorado
John F. Dalphin, Indiana University–Purdue University at Fort Wayne
Gene Davenport, John Wiley and Sons
Charles Davidson, University of Wisconsin
Peter Denning, Purdue University
Ed Desautels, University of Wisconsin
Benjamin Diamant, IBM
Karen A. Duncan, MITRE Corporation
Gerald Engel, Old Dominion University
Michael Faiman, University of Illinois
Patrick Fischer, Pennsylvania State University
Arthur Fleck, University of Iowa
John Gannon, University of Maryland
Norman Gibbs, College of William and Mary
Malcolm Gotterer, Florida International University
David Gries, Cornell University

(Appendix continued on next page)

Computer Systems — D. Siewiorek, Editor

FOCUS Microcomputer Number System

Albert D. Edgar and Samuel C. Lee
The University of Oklahoma

FOCUS is a number system and supporting computational algorithms especially useful for microcomputer control and other signal processing applications. FOCUS has the wide-ranging character of floating-point numbers with a uniformity of state distributions that give FOCUS better than a twofold accuracy advantage over an equal word length floating-point system. FOCUS computations are typically five times faster than single precision fixed-point or integer arithmetic for a mixture of operations, comparable in speed with hardware arithmetic for many applications. Algorithms for 8-bit and 16-bit implementations of FOCUS are included.

Key Words and Phrases: number representation, logarithmic arithmetic, computational speed, computational accuracy, microcomputer applications
CR Categories: 3.24, 3.80, 4.0, 4.22, 5.11

I. Introduction

Sophistication of requirements in systems design has pressed digital systems into real-time signal processing. In addition to the usual software techniques [14], many hardware techniques are actively pursued [4, 7, 8, 10, 17]. Besides, methods of representing a quantity and critical peripheral information, such as significant arithmetic, unnormalized arithmetic, error estimation in computer calculation, etc., are of vital concern in this field [1, 6, 11–13]. This paper presents a digital number system providing speed and accuracy for digital signal processing using a microcomputer.

Authors' address: School of Electrical Engineering and Computer Science, University of Oklahoma, 202 West Boyd, Room 219, Norman, OK 73019.