Computing Curricula 1991

Report of the

ACM/IEEE-CS Joint Curriculum Task Force

1951-1991

IEEE Computer Society Press

Computing Curricula 1991

Report of the
ACM/IEEE-CS Joint Curriculum Task Force

Allen B. Tucker (Editor and Co-chair), Bruce H. Barnes (Co-chair),
Robert M. Aiken, Keith Barker, Kim B. Bruce, J. Thomas Cain,
Susan E. Conry, Gerald L. Engel, Richard G. Epstein, Doris K. Lidtke,
Michael C. Mulder, Jean B. Rogers, Eugene H. Spafford, and A. Joe Turner

December 17, 1990

ACM Order Number: 201910
ACM ISBN Number: 0-8979-381-7

IEEE Computer Society Press Order Number: 2220
IEEE Computer Society Press ISBN Number: 0-8186-2220-2

Contents

Executive Summary v

I Designing Undergraduate Curricula in Computing 1

1 Introduction and Charter 2

2 Objectives and Overview of this Report 3

3 Relationship with Previous Curricular Recommendations 4

4 Undergraduate Program Goals/Graduate Profiles 6

5 Underlying Principles for Curriculum Design 7
- 5.1 The Nine Subject Areas 8
- 5.2 The Three Processes: Theory, Abstraction, and Design 9
- 5.3 Social and Professional Context 11
- 5.4 Recurring Concepts 12

6 From Principles to Curriculum 15
- 6.1 Assuring Breadth and Depth 16
- 6.2 The Role of Programming 16
- 6.3 The Role of Laboratories 19
- 6.4 Other Educational Experiences 20

7 The Common Requirements 21
- 7.1 Organizing the Common Requirements: The Knowledge Unit 22
- 7.2 Summary of the Common Requirements 23

8 The Advanced and Supplemental Curriculum 25

9 Mathematics and Science Requirements 27
- 9.1 Mathematics 27
- 9.2 Science 28

10 Building a Curriculum **28**

10.1 Overall Design Considerations 28

10.2 Designing Courses from Knowledge Units 29

10.3 Designing Courses from Advanced/Supplemental Material 31

10.4 Integrating the Curriculum into a Course of Study 32

11 Related Concerns **33**

11.1 Faculty and Staff 33

11.2 Laboratory Resources 33

11.3 Service Courses and Joint Degree Programs 34

11.4 Library Support 34

11.5 Relationship with Accreditation, Placement Tests, and Achievement Tests 34

12 Summary **35**

II Details of the Subject Matter **37**

13 Details of the Common Requirements **38**

AL: Algorithms and Data Structures 40

AR: Architecture 44

AI: Artificial Intelligence and Robotics 48

DB: Database and Information Retrieval 50

HU: Human-Computer Communication 51

NU: Numerical and Symbolic Computing 52

OS: Operating Systems 54

PL: Programming Languages 58

PR: Introduction to a Programming Language 64

SE: Software Methodology and Engineering 65

SP: Social, Ethical, and Professional Issues 68

14 Details of the Advanced/Supplemental Material **71**

15 Acknowledgments **76**

A Sample Curricula 79

A.1 Sample Curricula for Preparing Students to Enter the Profession 81

A.2 Sample Curricula with Other Major Goals 136

Executive Summary

This report contains curricular recommendations for baccalaureate programs in the discipline of *computing*, which includes programs with the titles "computer science," "computer engineering," "computer science and engineering," and other similar titles. These recommendations provide a uniform basis for curriculum design across all segments of the educational community—schools and colleges of engineering, arts and sciences, and liberal arts. This report is the first comprehensive undergraduate curriculum report to be endorsed by the two major professional societies in the computing discipline—the Association for Computing Machinery and the Computer Society of the IEEE.

These guidelines provide coverage of new and updated subject matter, including a detailed breakdown of individual lecture and laboratory topics. Fundamental areas of concern in curriculum planning are also addressed: program goals; course design and sequencing; integration of laboratory work; the role of programming; and other related educational experiences. Several sample curricula are illustrated for a variety of academic settings.

The organization of subject matter in this report follows that presented in the 1988 report *Computing as a Discipline* [6], which identifies nine subject areas of the discipline. Each of these areas contains topics that are essential and appropriate for all undergraduate curricula in computing. Those topics are presented here as a set of *common requirements* for all programs.

Undergraduates should also develop an understanding of the historical, social, and ethical context of the discipline and the profession. Therefore, beyond the nine subject areas, these common requirements also contain subject matter under the heading *Social, Ethical and Professional Issues.*

The common requirements do not appear here as a single group of courses. Rather, they appear as smaller packages of closely related topics known as *knowledge units.* Different institutions will combine these knowledge units into courses in different ways to meet their particular needs and priorities. However, all students should cover all of the common requirements sometime during their undergraduate careers. This provision ensures a *broad* coverage of the discipline. *Depth* of study in the discipline is also important for all undergraduates. A separate section of this report addresses the advanced and supplemental material that programs need to provide. Students should achieve depth through an appropriate selection of advanced courses.

These recommendations also recognize that mastery of the discipline includes not only an understanding of basic subject matter, but also an understanding of the three processes, or "points of view," that computing professionals employ and students need to appreciate: *theory, abstraction,* and *design.* Undergraduate programs should include significant study in each of these three processes. Particular attention is paid to the importance of laboratory work as it reinforces student mastery of concepts and their

application to real-world problems.

The cohesiveness of computing is also captured in a collection of fundamental concepts that occur repeatedly across all nine subject areas and all three processes of the discipline. These *recurring concepts* provide a framework for integrating subject matter into courses and complete programs of study. While not all of the recurring concepts are unique to computing, they do tend to summarize fundamental values of the discipline, giving students a means for interrelating diverse topics.

Computing professionals must remain current in the basic subject matter of this rapidly-evolving discipline. Undergraduates should therefore acquire the skills for remaining current and evaluating new ideas. These skills include the ability to read published literature in the field, to attend colloquia and understand the importance of what is being said, to participate in seminar-type discussions, and to work on team projects that address contemporary problems and areas of inquiry. Furthermore, the common requirements, laboratory work, and recurring concepts also provide a basis for remaining current in the discipline of computing after students complete their undergraduate programs.

This report was prepared by the ACM-IEEE-CS Joint Curriculum Task Force, which met several times during the last two years, sponsored presentations at professional conferences, and conducted three rounds of major reviews by a large number of educators, scholars, and practitioners in the discipline of computing. The Task Force has therefore made strong efforts to represent the views of a wide constituency in this report.

Recognizing that curricula evolve rapidly in this discipline, these recommendations are designed with an emphasis on the process of developing undergraduate programs to meet evolving needs. The different sample implementations that appear in the appendix provide detailed evidence of the viability of this report's process for designing contemporary undergraduate programs in 1991. However, as the years pass and these sample implementations become dated, we expect that the fundamental principles of curriculum design that are developed in this report will remain viable for a longer period of time.

Part I

Designing Undergraduate Curricula in Computing

1 Introduction and Charter

In the spring of 1988, the Association for Computing Machinery and the Computer Society of the IEEE formed the Joint Curriculum Task Force. The charter of the Task Force was to present recommendations for the design and implementation of undergraduate (baccalaureate) curricula in the discipline of computing. Throughout this report, the term *computing* is used to encompass the labels *computer science*, *computer science and engineering*, *computer engineering*, *informatics*, and other similar designations for academic programs. Programs in related areas, such as information systems, were not considered by the Task Force.

These recommendations supersede the separate curriculum recommendations [2, 3, 7] of the Association for Computing Machinery and the Computer Society of the IEEE. Since those recommendations were published, significant changes have been made to the introductory courses [10, 11]. The evolution of accreditation guidelines [1, 5, 12], alternative model curricula [9], and the recent report *Computing as a Discipline* [6] all testify to the rapid and fundamental changes that have taken place. A strong motivation for making such a joint effort at this time thus comes from the fact that the discipline and its pedagogy have changed significantly in the last several years.

Another motivation for this work comes from the growing recognition that, despite strong and fundamental differences among institutions offering undergraduate programs in computing, these programs share a substantially large curriculum in common. Any curriculum recommendations that attempt to speak for the entire discipline must not only identify this shared subject matter, but also suggest ways in which it can serve as a basis for building undergraduate programs in different kinds of institutions. This goal has been a particularly challenging and novel aspect of the current Task Force's work, yet it is potentially the most rewarding.

This work thus serves the interests of a wide constituency: in effect, the faculties of all undergraduate programs that offer concentration programs in computing, whether they occur in colleges of arts and sciences, colleges of engineering, liberal arts colleges, or other academic contexts. High quality undergraduate programs in computing exist in all of these contexts. This report provides recommendations for developing curricula that reflect those programs' common interests as well as their differences.

These recommendations have been developed by the Task Force in a series of eight major meetings during the period from May 1988 to August 1990. Earlier drafts of this report have been reviewed by dozens of educators and subject area experts (see acknowledgments starting on page 76 of this report). A revised, complete draft was reviewed in the summer of 1990. This work has also been publicly presented and discussed in panel sessions at the ACM SIGCSE meetings in February 1989 and February 1990, the Computer Society's COMPCON meeting in March 1989, the NECC in June 1989 and June 1990, the IFIP Working Group 2.1 meeting in April 1990, the WCCE in July 1990, as well as other forums. The Task Force has made strong efforts to incorporate

the views and suggestions of all reviewers and discussants into this report.

This report is divided into two parts and an appendix. Part I, the main body of the report, discusses general principles and methodologies that should be used in designing undergraduate curricula in computing. Part II gives a detailed description of the common curricular requirements and advanced/supplemental subject matter. The appendix illustrates sample implementations that reflect these principles, methodologies, common requirements, and advanced subject matter in a wide range of institutional settings and program goals.

2 Objectives and Overview of this Report

This report has the following primary objective: to provide curricular guidance for implementing undergraduate programs in the discipline of computing. The curriculum recommendations are based upon goals for programs and upon a definition of the discipline.

A set of such goals is provided in Section 4, along with a profile for graduates of these programs within the framework of broad institutional objectives.

As a second means for attaining this objective, the report's guidelines are based on a comprehensive definition of the discipline, provided in the report *Computing as a Discipline*. This definition was chosen because:

- it is up-to-date and widely available;
- it was developed by a task force of widely respected scholars in a cooperatively-sponsored effort between the ACM and the IEEE-CS; and
- it provides a detailed and comprehensive specification of the subject matter of the discipline.

The specification of the subject matter of the discipline is summarized in Section 5, along with some additional material and principles to be used in implementing a curriculum.

When used as a basis for curriculum design, this definition provides a conceptual and organizational context from which a common collection of subject matter for all undergraduate programs can be drawn. In the present report, this shared collection of subject matter is called the *common requirements*. That definition also provides the conceptual context from which the advanced and supplementary subject matter can be drawn.

The common requirements serve to introduce the breadth of the discipline, covering all nine subject areas described in the definition, so that students receive a comprehensive view of computing as an integral part of their program of study. However,

because different schools and colleges typically require different balances of discipline-specific coursework as a percentage of the total coursework, this report does not contain a single prescription of courses for all undergraduate programs. Instead it contains the following parts:

- A set of curricular and pedagogical considerations that govern the mapping of the common requirements and advanced/supplemental material into a complete undergraduate degree program. These considerations include the roles of laboratories, programming, mathematics and science, professionalism, a new notion called *recurring concepts*, and other educational experiences that combine to make up an entire undergraduate degree program in computing (see Sections 5, 6, 9, and 10 for more details).
- A collection of subject matter modules called *knowledge units* that comprise the common requirements for all undergraduate programs in the field of computing (see Section 7 and Part II for more details).
- A collection of advanced and supplementary curriculum material that provides depth of study in several of the subject areas. The coverage of this material will vary in accordance with differing overall degree requirements of different types of institutions (see Section 8 and Part II for more details).

Because these curriculum recommendations are intentionally flexible, they do not prescribe a single set of courses for all undergraduate programs in computing. Instead, they provide guidelines that allow individual departments to design their own programs according to local objectives and constraints. Section 10 of this report discusses considerations for building whole curricula out of the common requirements and other elements. In order to provide detailed illustrations of curriculum design, several sample curricula that follow from these considerations are given in the appendix.

Section 11 briefly addresses a number of important practical issues. These include faculty and staff support, laboratories, library and other resources, service courses, and the relationship between these curriculum recommendations and established mechanisms for accreditation, advanced placement, and outcome assessment.

3 Relationship with Previous Curricular Recommendations

In addition to *Computing as a Discipline*, these curricular guidelines are influenced by a number of prior curricular efforts that preceded it, including the ACM guidelines [3], the IEEE-CS guidelines [7], and other alternative guidelines (e.g., [9]). This section summarizes the major similarities and differences between the present report and its predecessors.

Besides providing a contemporary definition of the discipline, the 1988 report *Computing as a Discipline* influenced the present report in the following ways:

- It detailed the integration of laboratory work with classroom lectures.
- It affirmed the importance of design in the curriculum.
- It advocated a breadth-first approach for the introductory courses in the curriculum.

While the present report also promotes broad discipline coverage for all programs, it offers alternative ways of achieving such breadth. Some of these alternatives include a breadth-first approach to the introductory courses while others include a more traditional approach. More generally, the present report has significantly broader overall goals than does the *Computing as a Discipline* report.

The 1983 IEEE-CS curriculum report influenced the present report in the following ways:

- It contained in-depth descriptions of topic areas considered to be important to the computer engineering profession.
- It included well-defined and in-depth laboratory material to support the lecture topics.
- It used modules and submodules as a basis for organizing subject matter and constructing courses.
- It provided a number of sample implementations that adhered to ABET accreditation guidelines.

The present report addresses the curricular needs of a broader range of institutions and programs in computing than does the 1983 IEEE-CS report. Its subject matter is also more current, and the pedagogical alternatives that it offers are more widely applicable to different institutional needs.

The 1978 ACM curriculum report influenced the present report in the following ways:

- It provided detailed course descriptions for undergraduate programs in computer science.
- It was directed primarily to fulfill the needs of emerging programs.
- It gave a prominent role to programming in the curriculum.

- It identified the importance of theory and the study of computers in society by listing advanced electives in these areas.

The present report addresses the curricular needs of a broader range of institutions and programs in computing than does the 1978 ACM report. Its subject matter is also more current, it casts the role and nature of programming in a very different light, and it integrates theory, abstraction, design, and the social context of computing into the curriculum in more compelling ways.

In summary, the present report is influenced by its predecessors in significant ways. Yet the present report is very different in scope and objectives from any one of them. It reflects the rapid and dramatic evolution of the discipline of computing and its pedagogy that has taken place during the last several years. It is holistic, attempting to reach a wider constituency than any of its predecessors. It is intentionally designed to encourage curriculum innovation and evolution, enabling educators to respond in a timely fashion to future changes in the discipline.

4 Undergraduate Program Goals/Graduate Profiles

Undergraduate programs in computing share common attributes, values, and curricula, and so do their graduates. The following discussion reflects the Task Force's view of these shared attributes and values that serve as a basis for the curriculum recommendations that follow.

Undergraduate programs should prepare graduates to understand the field of computing, both as an academic discipline and as a profession within the context of a larger society. Thus, graduates should be aware of the history of computing, including those major developments and trends—economic, scientific, legal, political, and cultural—that have combined to shape the discipline during its relatively short life.

The first goal for undergraduate programs, therefore, is to provide a coherent and broad-based coverage of the discipline of computing. Graduates should develop a reasonable level of understanding in each of the subject areas and the processes that define the discipline, as well as an appreciation for the interrelationships that exist among them.

A second goal for undergraduate programs in computing is to function effectively within the wider intellectual framework that exists within the institutions that house the programs. These institutions vary widely in their respective missions. Some of them emphasize breadth of study over depth, while others emphasize the opposite. Some are rigid in the overall balance between requirements and electives, while others are more flexible. As is the case in other disciplines, undergraduate programs in computing necessarily reflect institutional differences in their respective degree requirements. It follows that graduates of different programs will have received different levels of coverage

in the subject areas of computing, as well as a different balance of emphasis among the processes of theory, abstraction, and design.

Third, different undergraduate programs place different levels of emphasis upon the objectives of preparing students for entry into the computing profession, preparing students for graduate study in the discipline of computing, and preparing students for the more general challenges of professional and personal life. Students enrolled in any undergraduate program in computing should be aware of that program's particular emphasis with regard to these three objectives.

Fourth, undergraduate programs should provide an environment in which students are exposed to the ethical and societal issues that are associated with the computing field. This includes maintaining currency with recent technological and theoretical developments, upholding general professional standards, and developing an awareness of one's own strengths and limitations, as well as those of the discipline itself.

Fifth, undergraduate programs should prepare students to apply their knowledge to specific, constrained problems and produce solutions. This includes the ability to define a problem clearly; to determine its tractability; to determine when consultation with outside experts is appropriate; to evaluate and choose an appropriate solution strategy; to study, specify, design, implement, test, modify, and document that solution; to evaluate alternatives and perform risk analysis on that design; to integrate alternative technologies into that solution; and to communicate that solution to colleagues, professionals in other fields, and the general public. This also includes the ability to work within a team environment throughout the entire problem-solving process.

Finally, undergraduate programs should provide sufficient exposure to the rich body of theory that underlies the field of computing, so that students appreciate the intellectual depth and abstract issues that will continue to challenge researchers in the future. In this light, graduates should be aware of the unusually high rate of change in the technology, the relatively gradual rate of growth in the theory of computing, and the delicate interaction that takes place between these two. They should thus have a strong foundation on which to base lifelong learning and development.

5 Underlying Principles for Curriculum Design

As noted above, the report *Computing as a Discipline* presented a comprehensive and contemporary definition of the discipline of computing. This definition provides a conceptual basis for defining a new teaching paradigm, as well as undergraduate curriculum design guidance for the discipline. The definition includes a specification of the subject matter by identifying nine constituent subject areas and three processes that characterize different working methodologies used in computing research and development. That specification is used in significant ways in this report.

From the subject matter of the nine areas of the discipline, this report identifies a

body of fundamental material called the *common requirements*, which is to be included in every program. The curriculum of each program must also contain substantial emphasis on each of the three processes. In addition, this report identifies a body of subject matter representing the social and professional context of the discipline, also considered to be essential for every program. Finally, a set of concepts that recur throughout the discipline, and that represent important notions and principles that remain constant as the subject matter of the discipline changes, are identified as being an important component of every program.

5.1 The Nine Subject Areas

Nine subject areas are identified in the report *Computing as a Discipline* as comprising the subject matter of the discipline. Each of these areas has a significant theoretical base, significant abstractions, and significant design and implementation achievements. While these subject area definitions cover the entire discipline, they each contain certain fundamental subjects that should be required in all undergraduate programs in computing, and this fundamental subject matter is identified in Section 7 and Part II as the *common requirements* for all programs. On the other hand, certain parts of these subject areas are less central and are therefore not included in the common requirements. These topics are left for the advanced components of the undergraduate or graduate curriculum.

In some cases major components of a subject area that appear in its title are not included in the common requirements. For example, the common requirements do not contain subject matter on symbolic computation. However, the subject area title "Numerical and Symbolic Computation" in *Computing as a Discipline* is retained, both in the interest of continuity and because additional subject matter will appear in the advanced components of computing curricula. For instance, symbolic computation appears among the advanced and supplemental topics that are described in Section 8 and Part II of this report, even though it does not appear among the common requirements.

The nine subject areas are:

Algorithms and Data Structures This area deals with specific classes of problems and their efficient solutions. The performance characteristics of algorithms and the organization of data relative to different access requirements are major components.

Architecture Methods of organizing efficient, reliable computing systems provide a central focus of this area. It includes implementation of processors, memory, communications, and software interfaces, as well as the design and control of large computational systems that are reliable.

Artificial Intelligence and Robotics The basic models of behavior and the building of (virtual or actual) machines to simulate animal and human behavior are

included here. Inference, deduction, pattern recognition, and knowledge representation are major components.

Database and Information Retrieval The area is concerned with the organization of information and algorithms for the efficient access and update of stored information. The modeling of data relationships, security and protection of information in a shared environment, and the characteristics of external storage devices are included in this area.

Human-Computer Communication The efficient transfer of information between humans and machines is the central focus of this area. Graphics, human factors that affect efficient interaction, and the organization and display of information for effective utilization by humans are included.

Numerical and Symbolic Computation General methods for efficiently and accurately using computers to solve equations from mathematical models are central to this area. The effectiveness and efficiency of various approaches to the solution of equations, and the development of high-quality mathematical software packages are important components.

Operating Systems This area deals with control mechanisms that allow multiple resources to be efficiently coordinated during the execution of programs. Included are appropriate services of user requests, effective strategies for resource control, and effective organization to support distributed computation.

Programming Languages The fundamental questions addressed by this area involve notations for defining virtual machines that execute algorithms, the efficient translation from high-level languages to machine codes, and the various extension mechanisms that can be provided in programming languages.

Software Methodology and Engineering The major focus of this area is the specification, design, and production of large software systems. Principles of programming and software development, verification and validation of software, and the specification and production of software systems that are safe, secure, reliable, and dependable are of special interest.

5.2 The Three Processes: Theory, Abstraction, and Design

Because computing is simultaneously a mathematical, scientific, and engineering discipline, different practitioners in each of the above nine subject areas employ different working methodologies, or *processes*, during the course of their research, development, and applications work. One such process, called *theory*, is akin to that found in mathematics, and is used in the development of coherent mathematical theories. It has the following major elements:

- Definitions and axioms
- Theorems
- Proofs
- Interpretation of results

This process is used in developing and understanding the underlying mathematical principles that apply to the discipline of computing.

An undergraduate's first encounter with theory in the discipline often occurs in an introductory mathematics course. Further theoretical material emerges in the study of algorithms (complexity theory), architecture (logic), and programming languages (formal grammars and automata). The present curriculum recommendations contain a significant amount of theory that should be mastered by undergraduates in all programs.

The second process, called *abstraction*, is rooted in the experimental sciences, and has the following elements:

- Data collection and hypothesis formation
- Modeling and prediction
- Design of an experiment
- Analysis of results

When persons engage in abstraction, they are modeling potential algorithms, data structures, architectures, and so forth. They are testing hypotheses about these models, alternative design decisions, or the underlying theory itself.

Undergraduate programs in computing introduce students to the process of abstraction in a variety of ways, both in classes and in laboratories. For example, the basic von Neumann model of a computer is a fundamental abstraction whose properties can be analyzed and compared with other competing models. Students are introduced to this model early in their undergraduate coursework. Undergraduate laboratory experiments that emphasize abstraction stress analysis and inquiry into the limits of computation, the properties of new computational models, and the validity of unproven theoretical conjectures.

The third process, called *design*, is rooted in engineering and is used in the development of a system or device to solve a given problem. It has the following parts:

- Requirements
- Specifications

- Design and Implementation
- Testing and Analysis

When computing professionals are engaged in design, they involve themselves with the conceptualization and realization of systems in the context of real-world constraints.

Undergraduates learn about design both by direct experience and by studying the designs of others. Many laboratory projects are design-oriented, giving students firsthand experience with developing a system or a component of a system to solve a particular problem. These laboratory projects emphasize the synthesis of practical solutions to problems and thus require students to evaluate alternatives, costs, and performance in the context of real-world constraints. Students develop the ability to make these evaluations by seeing and discussing example designs as well as receiving feedback on their own designs.

In all nine subject areas of computing, these three processes of theory, abstraction, and design appear prominently and indispensably. A thorough grounding in each process is thus fundamental to all undergraduate programs in the discipline.

5.3 Social and Professional Context

Undergraduates also need to understand the basic cultural, social, legal and ethical issues inherent in the discipline of computing. They should understand where the discipline has been, where it is, and where it is heading. They should also understand their individual roles in this process, as well as appreciate the philosophical questions, technical problems, and aesthetic values that play an important part in the development of the discipline.

Students also need to develop the ability to ask serious questions about the social impact of computing and to evaluate proposed answers to those questions. Future practitioners must be able to anticipate the impact of introducing a given product into a given environment. Will that product enhance or degrade the quality of life? What will the impact be upon individuals, groups, and institutions?

Finally, students need to be aware of the basic legal rights of software and hardware vendors and users, and they also need to appreciate the ethical values that are the basis for those rights. Future practitioners must understand the responsibility that they will bear, and the possible consequences of failure. They must understand their own limitations as well as the limitations of their tools. All practitioners must make a long-term commitment to remaining current in their chosen specialties and in the discipline of computing as a whole.

To provide this level of awareness, undergraduate programs should devote explicit curricular time to the study of social and professional issues. The subject matter recommended for this study appears under the heading *SP: Social, Ethical, and Professional*

Issues in Part II of this report. Alternative ways of integrating that subject matter into courses of instruction are suggested in the appendix.

5.4 Recurring Concepts

The discussion thus far has emphasized the division of computing into nine subject areas, three processes, and its social and professional context. However, certain fundamental concepts recur throughout the discipline and play an important role in the design of individual courses and whole curricula. *Computing as a Discipline* refers to some of these concepts as *affinity groups* or *basic concerns throughout the discipline.*[1] The Task Force refers to these fundamental concepts as *recurring concepts* in this report.

Recurring concepts are significant ideas, concerns, principles and processes that help to unify an academic discipline at a deep level. An appreciation for the pervasiveness of these concepts and an ability to apply them in appropriate contexts is one indicator of a graduate's maturity as a computer scientist or engineer. Clearly, in designing a particular curriculum, these recurring concepts must be communicated in an effective manner; it is important to note that the appropriate use of the recurring concepts is an essential element in the implementation of curricula and courses based upon the specifications given in this report. Additionally, these concepts can be used as underlying themes that help tie together curricular materials into cohesive courses.

Each recurring concept listed in this report

- Occurs throughout the discipline
- Has a variety of instantiations
- Has a high degree of technological independence

Thus, a recurring concept is any concept that pervades the discipline and is independent of any particular technology. A recurring concept is more fundamental than any of its instantiations. A recurring concept has established itself as fundamental and persistent over the history of computing and is likely to remain so for the foreseeable future.

In addition to the three characteristics given above, most recurring concepts

- Have instantiations at the levels of theory, abstraction and design
- Have instantiations in each of the nine subject areas
- Occur generally in mathematics, science and engineering

These additional points make a strong assertion concerning the pervasiveness and persistence of most of the recurring concepts. Not only do they recur throughout the

[1] Page 9 of [6].

discipline, they do so across the nine subject areas and across the levels of theory, abstraction and design. Furthermore, most are instances of even more general concepts that pervade mathematics, science and engineering.

Below is a list of twelve recurring concepts that we have identified as fundamental to computing. Each concept is followed by a brief description and a characterization in terms of concrete examples. In the remainder of the report, each is explicitly referenced whenever it appears within a curriculum element of the common requirements.

Binding: the processes of making an abstraction more concrete by associating additional properties with it. Examples include associating (assigning) a process with a processor, associating a type with a variable name, associating a library object program with a symbolic reference to a subprogram, instantiation in logic programming, associating a method with a message in an object-oriented language, creating concrete instances from abstract descriptions.

Complexity of large problems: the effects of the nonlinear increase in complexity as the size of a problem grows. This is an important factor in distinguishing and selecting methods that scale to different data sizes, problem spaces, and program sizes. In large programming projects, it is a factor in determining the organization of an implementation team.

Conceptual and formal models: various ways of formalizing, characterizing, visualizing and thinking about an idea or problem. Examples include formal models in logic, switching theory and the theory of computation, programming language paradigms based upon formal models, conceptual models such as abstract data types and semantic data models, and visual languages used in specifying and designing systems, such as data flow and entity-relationship diagrams.

Consistency and completeness: concrete realizations of the concepts of consistency and completeness in computing, including related concepts such as correctness, robustness, and reliability. Consistency includes the consistency of a set of axioms that serve as a formal specification, the consistency of theory to observed fact, and internal consistency of a language or interface design. Correctness can be viewed as the consistency of component or system behavior to stated specifications. Completeness includes the adequacy of a given set of axioms to capture all desired behaviors, functional adequacy of software and hardware systems, and the ability of a system to behave well under error conditions and unanticipated situations.

Efficiency: measures of cost relative to resources such as space, time, money and people. Examples include the theoretical assessment of the space and time complexity of an algorithm, feasibility, the efficiency with which a certain desirable result (such as the completion of a project or the manufacture of a component) can be achieved, and the efficiency of a given implementation relative to alternative implementations.

Evolution: the fact of change and its implications. The impact of change at all levels and the resiliency and adequacy of abstractions, techniques and systems in the face of change. Examples include the ability of formal models to represent aspects of systems that vary with time, and the ability of a design to withstand changing environmental demands and changing requirements, tools and facilities for configuration management.

Levels of Abstraction: the nature and use of abstraction in computing; the use of abstraction in managing complexity, structuring systems, hiding details, and capturing recurring patterns; the ability to represent an entity or system by abstractions having different levels of detail and specificity. Examples include levels of hardware description, levels of specificity within an object hierarchy, the notion of generics in programming languages, and the levels of detail provided in a problem solution from specifications through code.

Ordering in space: the concepts of locality and proximity in the discipline of computing. In addition to physical location, as in networks or memory, this includes organizational location (e.g., of processors, processes, type definitions, and associated operations) and conceptual location (e.g., software scoping, coupling, and cohesion).

Ordering in time: the concept of time in the ordering of events. This includes time as a parameter in formal models (e.g., in temporal logic), time as a means of synchronizing processes that are spread out over space, time as an essential element in the execution of algorithms.

Reuse: the ability of a particular technique, concept or system component to be reused in a new context or situation. Examples include portability, the reuse of software libraries and hardware components, technologies that promote reuse of software components, and language abstractions that promote the development of reusable software modules.

Security: the ability of software and hardware systems to respond appropriately to and defend themselves against inappropriate and unanticipated requests; the ability of a computer installation to withstand catastrophic events (e.g., natural disasters and attempts at sabotage). Examples include type-checking and other concepts in programming languages that provide protection against misuse of data objects and functions, data encryption, granting and revoking of privileges by a database management system, features in user interfaces that minimize user errors, physical security measures at computer facilities, and security mechanisms at various levels in a system.

Tradeoffs and consequences: the phenomenon of trade-offs in computing and the consequences of such trade-offs. The technical, economic, cultural and other effects of selecting one design alternative over another. Trade-offs are a fundamental fact

of life at all levels and in all subject areas. Examples include space-time trade-offs in the study of algorithms, trade-offs inherent in conflicting design objectives (e.g., ease of use versus completeness, flexibility versus simplicity, low cost versus high reliability and so forth), design trade-offs in hardware, and trade-offs implied in attempts to optimize computing power in the face of a variety of constraints.

In constructing curricula from the overall specifications of the Task Force, curriculum designers must be aware of the fundamental role played by recurring concepts. That is, a recurring concept (or a set of recurring concepts) can help to unify the design of a course, a lecture, or a laboratory exercise. From the instructor's perspective (and also from the student's perspective), a course is rarely satisfying unless there is some "big idea" that seems to hold disparate elements together. We see the use of recurring concepts as one method for unifying the material in a course.

At the level of the entire curriculum, the recurring concepts also play a unifying role. They can be used as threads that tie and bind different courses together. For example, in introducing the concept of *consistency* as applied to language design in a programming language course, the instructor might ask students to consider other contexts in which consistency played an important role, such as in a previous software engineering or user interfaces course. By pointing out and discussing the recurring concepts as they arise, the conscientious instructor can help portray computing as a coherent discipline rather than as a collection of unrelated topics.

6 From Principles to Curriculum

An undergraduate curriculum in computing should provide each graduate with a reasonable level of instruction in all of the subject areas identified above. This allows each graduate to achieve a *breadth* of understanding across the entire discipline rather than just in a few of its parts. Breadth is ensured in the present guidelines by way of the *common requirements*.

The undergraduate curriculum should also provide reasonable *depth* of study in some of the nine subject areas of the common requirements. While the particular way in which subject area depth is achieved will vary among different types of programs, it is essential that depth of study be achieved in one way or another. Depth is ensured in the present guidelines by way of the *advanced/supplemental topics*.

Among the three processes—theory, abstraction, and design—it is fair to assume that some undergraduate programs emphasize more theory than design, while others emphasize more design than theory. However, the process of abstraction will normally be prominent in all undergraduate curricula. Theory, abstraction, and design are included throughout the common requirements, and are reinforced by the integration of *laboratory work* with subject matter in a principled and thorough way.

To support the development of maturity in the mathematical and scientific aspects of computing, an undergraduate curriculum should also include certain *mathematics and science* course material to complement the subject matter in the discipline. Similarly, to support the development of maturity in the scientific and engineering aspects of the discipline, the present guidelines recommend that students regularly engage in laboratory work and *other educational experiences.*

An overview of a complete undergraduate program in computing can be summarized as shown in Figure 1. This summary provides a general level of guidance for implementing undergraduate programs in the discipline. There, the need to recognize different institutional contexts and programmatic goals is accommodated by the box labeled *Other Degree Requirements.* Each of the remaining boxes in Figure 1 is discussed in the sections below.

6.1 Assuring Breadth and Depth

To realize the breadth requirements, undergraduate programs should provide a broad selection of topics taken from all nine subject areas of computing. These topics were identified above as the *common requirements.* However, because different institutions will want to fulfill these curricular requirements in different ways, the common requirements are not presented as a single prescribed set of courses. Instead, the common requirements topics are presented as a collection of *knowledge units.* These can be combined in various ways to form different sets of courses, or *implementations*, in different undergraduate settings. Further discussion and examples of constructing courses out of knowledge units appear in Section 7 and the appendix of this report.

To realize the depth requirements, different institutions will offer additional requirements and electives in accordance with their overall educational missions, faculty size, and subject area expertise. Section 8 and Part II of this report describe a number of advanced and supplemental topics that provide depth of study in the nine areas of the discipline of computing.

6.2 The Role of Programming

The term *programming* is understood to denote the entire collection of activities that surround the description, development, and effective implementation of algorithmic solutions to well-specified problems. While this definition is not to be construed to mean simply "coding in a particular programming language or for a particular machine architecture," it does necessarily include the mastery of highly stylized skills in a particular programming language or languages. Thus, fluency in a programming language is an essential attribute of the mastery of programming. On the other hand, programming is not to be construed so broadly as to subsume all of the activities involved in software

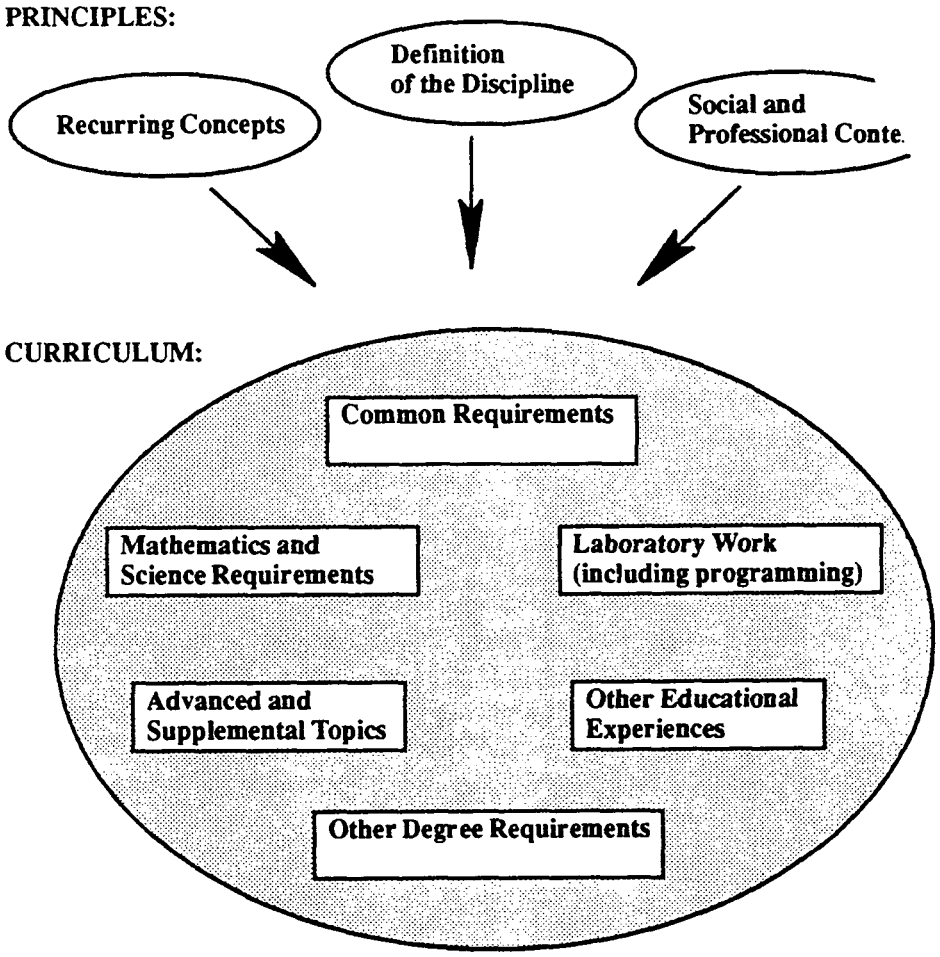

Figure 1: A Complete Curriculum and its Underlying Principles

methodology and engineering. The latter is a much broader notion and includes, for example, the development of specifications and the maintenance of software.

Programming occurs in all nine subject areas in the discipline of computing. It is part of the design process, it is used to implement the models that occur in the abstraction process, and it even occurs sometimes in the process of proving a theoretical result. Thus, the role of programming in the undergraduate curriculum is also multidi-

mensional; students develop programs during the design of software, they exercise and modify programs during other laboratory experiments, and they read programs in the normal course of studying subject matter in textbooks and the published literature. In this latter sense, programming is an extension of the basic communication skills that students and professionals normally use in day-to-day communication.

Mastery of programming can be accomplished in several ways during an undergraduate's career. First, a substantial portion of the common requirements' laboratory work contains programming as an essential part. For example, the knowledge unit *SE1: Fundamental Problem Solving Concepts*[2] requires laboratory work in which students design, implement, and exercise programs in a modern programming language. Second, undergraduate texts in the various subject areas teach and use conventional programming techniques throughout their presentations of subject matter. Lectures in courses throughout the curriculum also use well-established styles of programming to explain or clarify algorithmic concepts. Third, the computing literature is replete with programs and examples of programming methodology, and undergraduates become familiar with this literature during their studies. Instructors should take care to present examples of good programming to their students to study and emulate.

Thus, undergraduate majors should develop an early understanding of programming. They should continue to actively engage in the mastery and use of programming paradigms and languages throughout their coursework, homework, and laboratory exercises.

It should be noted here that the common requirements do not assume that students will have any experience with programming before taking their first course in the discipline. However, increasing numbers of students do gain such experience in secondary school. Many feel that the amount of attention traditionally paid to the syntax of a programming language in the first course is excessive, and ought to be replaced with a more balanced introduction to the discipline.

For these reasons, *PR: Introduction to a Programming Language*[3] is defined as a separate knowledge unit in Part II, but is not a *required* part of the common requirements. That is, most programs will be able to introduce a programming language in conjunction with other knowledge unit material, especially the one named SE1, during the scheduled laboratory periods that accompany the first course. For those programs that do require a more careful treatment of this topic, the optional knowledge unit *PR: Introduction to a Programming Language* can be added to the common requirements to fulfill this need.

[2] See page 65
[3] See page 64

6.3 The Role of Laboratories

The report *Computing as a Discipline* makes a clear statement about the purpose and structure of laboratory work in the undergraduate curriculum; this report supports a similar view. This section of the report expands upon that statement and identifies in more detail how these very important laboratory activities can be integrated into an undergraduate curriculum. An undergraduate curriculum in computing is ideally comprised of an integrated program of lectures and laboratory experiences. The learning process occurs as a result of interaction among students, instructors, and the subject matter.

Laboratories demonstrate the application of principles to the design, implementation, and testing of hardware and software systems. Laboratories also emphasize techniques that utilize contemporary tools and lead to good experimental methods, including the use of measuring instruments, diagnostic aids, software and hardware monitors, statistical analysis of results, and oral and written presentation of findings. The laboratories should augment the instruction that takes place in the lectures by having clearly stated objectives that complement the lectures.

The laboratory exercises included in the *Suggested Laboratories* part of each knowledge unit provide a diverse set of learning experiences. Some involve the execution of hardware, software, or simulators to observe some phenomenon, either by data collection or by visualization. Others are designed to increase student expertise in software methodology through the development of alternative design and implementation techniques. Still other laboratories are similar to science experiments because they involve hypothesis formation and testing. These types of laboratory experiences combine to increase student problem solving ability, analytical skill, and professional judgment.

In the presentation of laboratories among the knowledge units, two distinct types of laboratory work are offered. These are called *open* laboratories and *closed* laboratories. An open laboratory is an unsupervised assignment that may involve the use of a computer, software, or hardware for its completion. Students can complete an open laboratory at their own convenience as it does not require direct supervision. Conventional programming assignments are often done in an open laboratory setting.

A closed laboratory is a scheduled, structured, and supervised assignment that involves the use of computing hardware, software, or instrumentation for its completion. Students complete a closed lab by attending a scheduled session, usually 2–3 hours long, at a specific facility. Supervision is provided by the instructor or a qualified assistant who is familiar with the details of the assignment. The specialized equipment, software, and supervision offered by closed laboratories makes them more desirable than open laboratories in certain situations. Closed laboratories are particularly important in situations where the assignment relies on instructor-student interaction or a team effort among students to complete the work. For example, the use of software tools that facilitate the production of large-scale software, CAD programs, or other packages of

considerable complexity normally require the initial guidance and advice of an expert.

Whether the lab is open or closed, the facility should be staffed by a responsible person during all scheduled laboratory hours. Staffing ensures immediate feedback and guidance during times when students are working independently on laboratory assignments.

The designation *open* or *closed* accompanies many of the suggested laboratory exercises that appear among the common requirements in Part II. These designations, along with the laboratories themselves, should be used as suggestions, or statements of preference by the Task Force, rather than prescriptions for all programs to follow. That is, some laboratories that are designated as *closed* may be done as open laboratories and vice versa. This choice may depend on how the particular lab experience and objectives fit into the curriculum as well as the available facilities in a particular institutional setting.

However, whether it is open or closed, a laboratory assignment should always be carefully planned by the instructor. Descriptions that include clear statements of purpose, methodology, and results should be carefully prepared for the students. Laboratory assignments should be realistically designed, so that an average student can complete the work in the allotted time. It is particularly important to make sure that adequate facilities are available to support the goals of each laboratory assignment.

Completion of a laboratory assignment should be accompanied by a written or oral report by the student. Written reports should reflect a disciplined and mature writing style, in addition to the successful completion of the laboratory work itself. Homework other than laboratory assignments is also expected to occur, and this work should also be well-integrated with the subject matter of the lectures.

6.4 Other Educational Experiences

Beyond the subject matter of the curriculum, undergraduates in computing should have additional experiences that will help them develop the capacity for critical thinking, problem solving, research methods, and professional development. These experiences can be incorporated into the classroom lectures, laboratories, and extracurricular activities of the undergraduate program. These additional experiences generally fall into three categories:

1. working as part of a team
2. communication
3. familiarization with the profession

Team projects normally evolve out of extensive laboratory projects, such as those involving large program design and implementation efforts. Such an experience adds

important breadth, depth, and realism to the curriculum. The team project is also important to the development of written and oral communication skills. For example, a large software project includes the development of program documentation and oral presentations.

Students should be encouraged to develop strong communication skills, both oral and written. Although these skills can be practiced and honed during some of the computing coursework itself, primary instruction in these areas is likely to occur during coursework in disciplines outside of computing. Curriculum designers should therefore ensure that such outside coursework is accessible to all computing majors.

Written and oral communication skills are also developed when a student engages in an independent study or undergraduate research project under the personal tutelage of a faculty member. Sometimes the result is a co-authored publication or conference presentation with the sponsoring faculty member. Opportunities for writing long term papers and making oral presentations are normally most available in advanced elective courses, where the student/faculty ratio is usually favorable to this kind of activity.

Majors should, in the normal course of their studies, also develop appropriate library skills for advanced work in the computing discipline. This includes becoming familiar with the special periodical literature of the discipline, such as that of the ACM and the IEEE Computer Society. Students should eventually be able to discriminate between scholarly work and the popular press, to understand the distinction between refereed and nonrefereed work, and to judge when certain work is appropriate and current.

Finally, students should begin to participate in the local professional activities of the department and appropriate professional societies. This includes attending colloquia, seminars, and student chapter meetings as they become available. Faculty should, in turn, set good examples of professional engagement in the discipline by keeping students well-informed of their own activities, demonstrating good communication skills, participating in team research or design projects, and generally maintaining high professional and scholarly standards themselves.

7 The Common Requirements

The common requirements form the basis for a curriculum in computing by providing a platform of knowledge that is considered essential for all students who concentrate in the discipline. These are not the only topics that should be covered, yet they contain the basic body of knowledge that should be part of every curriculum. The common requirements are expressed below as *knowledge units* rather than complete courses, to allow different programs to package the subject matter in different ways. Such variations will occur because different institutions and types of programs will have different pedagogical priorities, educational goals, and general constraints within which they implement the common requirements.

Subject Area	Tag
Algorithms and Data Structures	AL
Architecture	AR
Artificial Intelligence and Robotics	AI
Database and Information Retrieval	DB
Human-Computer Communication	HU
Numerical and Symbolic Computation	NU
Operating Systems	OS
Programming Languages	PL
Introduction to a Programming Language (optional)	PR
Software Methodology and Engineering	SE
Social, Ethical, and Professional Issues	SP

Figure 2: The Subject Areas and Tags

While every knowledge unit is considered to be essential, the depth and breadth of coverage for each topic therein will not be the same. For example, the knowledge unit *AL6: Sorting and Searching* (see below) recommends that various sorting and searching algorithms be covered in six lecture hours and associated laboratory work. However, this is only about two weeks time. Thus, only a few of the multitude of sorting techniques can be covered in any appreciable depth—perhaps insertion sort, heap sort, and quicksort—while various other sorting techniques can be covered in a more survey-like fashion. Instructors will inevitably make tradeoffs between depth and breadth of coverage, given the constraints of the number of lectures in a knowledge unit, in the same way that they do now with current textbooks. Furthermore, some knowledge units offer only a broad level of coverage of a topic (see, for instance, PL1, OS1, or AI1 in Part II). Additional depth in these subjects can be obtained only by exceeding the coverage recommended by the common requirements, either in the coverage of the knowledge unit, or in an advanced course.

7.1 Organizing the Common Requirements: The Knowledge Unit

In the following discussion, a *knowledge unit* is understood to designate a coherent collection of subject matter that is so fundamental within one of the nine subject areas of the discipline described in Section 5.1, or within the area of social, ethical, and professional issues (SP), that it should occur in every undergraduate curriculum. While the subject matter of a knowledge unit may be related to other knowledge units in the common requirements by way of a prerequisite structure, it can nevertheless be introduced within any of several alternative course structures.

For easy cross-referencing among the knowledge units, Figure 2 gives the two-letter tags used to identify each of the nine subject areas, the additional area of social and professional context, and the optional introduction to a programming language (PR).

The collected knowledge units in Part II are organized by subject area. Each knowledge unit within a subject area is identified by the tag for that subject area. For example, the knowledge units in the area of Algorithms and Data Structures are identified by the tags AL1, AL2, AL3, and so forth.

The following sample knowledge unit, identified in Part II as AL6, illustrates the presentation style for knowledge units.

AL6: Sorting and Searching
Comparison of various algorithms for sorting and searching, with focus on complexity and space versus time trade-offs.

Recurring Concepts: complexity of large problems, consistency and completeness, efficiency, trade-offs and consequences.

Lecture Topics: (six hours minimum)

1. $O(n^2)$ sorting algorithms (e.g., insertion and selection sort); space-time complexity: best, worst cases
2. $O(n \log n)$ sorting algorithms (e.g., quicksort, heapsort, mergesort); space-time complexity: best, worst cases
3. Other sorting algorithms (e.g., Shell sort, bucket sort, radix sort)
4. Comparisons of algorithms
5. Serial search, binary search and binary search tree; space-time complexity: best, worst cases
6. Hashing, collision resolution

Suggested Laboratories: (closed) Corroboration of theoretical complexity of selected sorting and searching algorithms by experimental methods, identifying differences among best, average, and worst case behaviors.

Connections:

Related to: AL5, AL8
Prerequisites: AL4
Requisite for: AI2, OS7, PL9

The meanings of the various components of an individual knowledge unit are given in Part II, Section 13 of this report.

7.2 Summary of the Common Requirements

Below is a complete list of the titles of the knowledge units that comprise the common requirements. While these titles suggest something about the knowledge units' contents, a detailed presentation of each one is given in Part II.

AL: Algorithms and Data Structures (approximately 47 lecture hours)
AL1: Basic Data Structures

AL2: Abstract Data Types
AL3: Recursive Algorithms
AL4: Complexity Analysis
AL5: Complexity Classes
AL6: Sorting and Searching
AL7: Computability and Undecidability
AL8: Problem-Solving Strategies
AL9: Parallel and Distributed Algorithms

AR: Architecture (approximately 59 lecture hours)
AR1: Digital Logic
AR2: Digital Systems
AR3: Machine Level Representation of Data
AR4: Assembly Level Machine Organization
AR5: Memory System Organization and Architecture
AR6: Interfacing and Communication
AR7: Alternative Architectures

AI: Artificial Intelligence and Robotics (approximately nine lecture hours)
AI1: History and Applications of Artificial Intelligence
AI2: Problems, State Spaces, and Search Strategies

DB: Database and Information Retrieval (approximately nine lecture hours)
DB1: Overview, Models, and Applications of Database Systems
DB2: The Relational Data Model

HU: Human-Computer Communication (approximately eight lecture hours)
HU1: User Interfaces
HU2: Computer Graphics

NU: Numerical and Symbolic Computation (approximately seven lecture hours)
NU1: Number Representation, Errors, and Portability
NU2: Iterative Approximation Methods

OS: Operating Systems (approximately 31 lecture hours)
OS1: History, Evolution, and Philosophy
OS2: Tasking and Processes
OS3: Process Coordination and Synchronization
OS4: Scheduling and Dispatch
OS5: Physical and Virtual Memory Organization
OS6: Device Management
OS7: File Systems and Naming
OS8: Security and Protection
OS9: Communications and Networking

OS10: Distributed and Real-time Systems

PL: Programming Languages (approximately 46 lecture hours)
- PL1: History and Overview of Programming Languages
- PL2: Virtual Machines
- PL3: Representation of Data Types
- PL4: Sequence Control
- PL5: Data Control, Sharing, and Type Checking
- PL6: Run-time Storage Management
- PL7: Finite State Automata and Regular Expressions
- PL8: Context-Free Grammars and Pushdown Automata
- PL9: Language Translation Systems
- PL10: Programming Language Semantics
- PL11: Programming Paradigms
- PL12: Distributed and Parallel Programming Constructs

SE: Software Methodology and Engineering (approximately 44 lecture hours)
- SE1: Fundamental Problem-solving Concepts
- SE2: The Software Development Process
- SE3: Software Requirements and Specifications
- SE4: Software Design and Implementation
- SE5: Verification and Validation

SP: Social, Ethical, and Professional Issues (approximately 11 lecture hours)
- SP1: Historical and Social Context of Computing
- SP2: Responsibilities of the Computing Professional
- SP3: Risks and Liabilities
- SP4: Intellectual Property

8 The Advanced and Supplemental Curriculum

A complete curriculum will include not only the common requirements but also certain additional material. This advanced and supplemental material gives each individual student an opportunity to study the subject areas of the discipline in depth. The curriculum should provide depth of study in several of the nine subject areas beyond that provided by the common requirements. Students normally achieve that depth by completing several additional courses in this part of the curriculum. The number of such courses will vary in accordance with institutional norms. The sample implementations in the appendix illustrate these kinds of variations.

The topics in the following list should be considered as areas where courses may be developed to provide in-depth study in advanced undergraduate and graduate courses. Other topics beyond these are important as well, but will vary with the particular

interests and expertise of the faculty in individual programs. However, the topics below tend to be so significant to the discipline at this time that several of them ought to appear among the advanced courses offered by any undergraduate program.

Advanced Operating Systems
* Advanced Software Engineering
Analysis of Algorithms
* Artificial Intelligence
Combinatorial and Graph Algorithms
Computational Complexity
* Computer Communication Networks
* Computer Graphics
Computer-Human Interface
* Computer Security
* Database and Information Retrieval
Digital Design Automation
Fault-Tolerant Computing
Information Theory
Modeling and Simulation
Numerical Computation
* Parallel and Distributed Computing
Performance Prediction and Analysis
Principles of Computer Architecture
Principles of Programming Languages
* Programming Language Translation
Real-Time Systems
Robotics and Machine Intelligence
Semantics and Verification
Societal Impact of Computing
* Symbolic Computation
* Theory of Computation
* VLSI System Design

Several of the topics in this list are marked with an asterisk (*). This denotes that a more complete description is given in Part II of this report. Those descriptions are intended to give more concrete information that will assist in developing courses in these topic areas.

It is important to emphasize that these topics do not necessarily represent courses. Courses can be built from these topics in various ways. As the discipline of computing rapidly evolves, it is likely that the advanced and supplemental courses will change more quickly in subject coverage than will the courses that cover the common requirements.

9 Mathematics and Science Requirements

An understanding of mathematics and science is important for students who concentrate their studies in computing.

9.1 Mathematics

Mathematical maturity, as commonly attained through logically rigorous mathematics courses, is essential to successful mastery of several fundamental topics in computing. Thus, all computing students should take at least one-half year[4] of mathematics courses. These courses should cover at least the following subjects:

Discrete Mathematics: sets, functions, elementary propositional and predicate logic, Boolean algebra, elementary graph theory, matrices, proof techniques (including induction and contradiction), combinatorics, probability, and random numbers.

Calculus: differential and integral calculus, including sequences and series and an introduction to differential equations.

Note that some of the discrete mathematics topics should be treated early in the curriculum, because they are needed for some of the basic knowledge units.

The half-year mathematics requirement should also include at least one of the following subjects:

Probability: discrete and continuous, including combinatorics and elementary statistics.

Linear Algebra: elementary, including matrices, vectors, and linear transformations.

Advanced Discrete Mathematics: a second course covering more advanced topics in discrete mathematics.

Mathematical Logic: propositional and functional calculi, completeness, validity, proof, and decision problems

Many implementations will have additional mathematics requirements beyond this minimum set. For example, professionally-oriented programs will normally require five or six mathematics courses. Students who wish to pursue graduate study in computing are also well-advised to take more mathematics.

[4] By this, we mean the equivalent of one-half academic year of full-time study. Typically, this would be four or five semester-long courses.

9.2 Science

Science is important in computing curricula for three reasons. First, as well-educated scientists and engineers, graduates of computing programs should be able to appreciate advances in science because they have an impact on society and on the field of computing. Second, exposure to science encourages students to develop an ability to apply the scientific method in problem solving. Third, many of the applications students will encounter after graduation are found in the sciences.

For these reasons, all computing curricula should include a component that incorporates material from the physical and life sciences. Ideally, courses in this component are those designed for science majors.

Programs intended to prepare students for entry into the profession should require a minimum of one-half year of science. This normally includes a year-long course in a laboratory science (preferably physics) and additional work in the natural sciences.

10 Building a Curriculum

10.1 Overall Design Considerations

A curriculum for a particular program depends on many factors, such as the purpose of the program, the strengths of the faculty, the backgrounds and goals of the students, instructional support resources, infrastructure support and, where desired, accreditation criteria. Each curriculum will be site-specific, shaped by those responsible for the program who must consider factors such as institutional goals, opportunities and constraints, local resources, and the preparation of the students.

Developing a curriculum involves a design process similar to others with which computer scientists and engineers are familiar. A successful implementation will grow from a well-conceived and well-articulated specification of the purpose of the curriculum, the context in which the curriculum will be delivered, and other external opportunities and constraints. Like systems, the success of the implementation of a curriculum depends on the care with which it is designed.

All curriculum design, including but not limited to computing curricula, should be guided by certain principles. First, the purpose of a curriculum is to educate students. The curriculum design therefore should focus on providing a way for students to gain the desired knowledge, expertise and experience by the time they graduate. The outcome expected for students should drive the curriculum planning.

Second, a curriculum is more than a set of isolated courses. There are unifying ideas and goals that span the whole curriculum. This report has included the discussion of recurring concepts as an indicator of those ideas that should be pervasive in a computing curriculum.

Third, there are many ways for students to learn beyond the standard lecture delivery format. Laboratories, in particular, present many opportunities for innovation. Faculty members should apply creative energy to developing alternative delivery methods.

Fourth, every environment has constraints that must be considered. A curriculum plan must not be overly optimistic nor overly restrictive, but should be realistic, both for students and for faculty. For example, a curriculum plan that requires a student to take four computing courses during the last term of the senior year invites a variety of problems.

Fifth, computing is more than just a collection of facts and algorithms. It is a dynamic, vital discipline that offers many challenging and interesting problems, exciting results, and imaginative applications. The curriculum should try to impart this sense of excitement to the students; the material may be difficult, but it need not be dull.

Designing a computing curriculum adds another set of special concerns. The rapid change in the discipline itself demands that a computing curriculum be dynamic, not static. It must be built so that it can evolve along with the subject matter. All curricula must be evaluated periodically to see that they are meeting their goals; computing curricula must have additional evaluation of content to be sure it is up-to-date. Many departments also face another kind of change caused by changing infrastructure as their organizations, enrollments, and funding levels change.

Another concern in computing curricula comes from the fluid nature of computing's theory. As computing is based on artifacts in addition to physical laws, many of its "fundamentals" are subject to constant reinterpretation and reevaluation. Thus, the theory of computing evolves more rapidly than it does in the other sciences.

Computing, however, has one great advantage for curriculum designers. It provides frequent opportunities for accomplishing multiple goals through one instructional experience. Activities can be leveraged for several purposes. For example, a discussion of loops or recursion can simultaneously introduce the concept of searching by the use of appropriately chosen examples.

10.2 Designing Courses from Knowledge Units

The knowledge units enumerated in Part II specify the scope of topics that all computing students should study. The lecture hours associated with each knowledge unit give an approximate indication of the depth to which these topics should be covered; we intend this to represent the minimum coverage that a typical program should ensure. The prerequisite structure suggests that some sequencing is required in the composition of courses out of knowledge units, but the size of the units allows many organizational options. Additionally, material in a knowledge unit may be split and covered at different times and in different courses, if deemed appropriate.

The knowledge units do not need to be grouped together into what are commonly

called "core courses," as long as they are all covered in one required course or another. Furthermore, some parts may be covered either in a standard class setting or in a laboratory setting. Thus, the knowledge units of the common requirements can be combined in various ways to form courses. In this activity, use the following guidelines:

- Knowledge units should be combined so that the composite subject matter forms a coherent body of topics for an undergraduate. In some cases, one or more unifying concepts can provide the "glue," while in other cases one of the nine subject areas will be the basis for course organization.
- The combined set of courses that comprise an implementation should have a prerequisite structure that is consistent with the prerequisite structure that exists among their constituent knowledge units.
- The combined set of courses that comprise the implementation should cover all of the knowledge units that make up the common requirements. As specified here, these total about 271 lecture hours, which is equivalent to about seven one-semester courses.[5] This total does not include time spent in laboratory exercises, nor does it include the advanced and supplementary coursework that is required for all majors.

Implementations may, of course, exceed this minimum by assigning additional depth of coverage or topics beyond those that are suggested in the common requirements.

Below are two sample courses, an introductory course entitled "Problem-solving, Programs, and Computers" and another course entitled "Data Structures and Algorithms," that result from combining selected knowledge units under the foregoing guidance. The first of these courses has knowledge units from a number of different subject areas, and has "consistency and completeness" as a recurring concept. The second of these courses has all of its knowledge units taken from a single subject area.

Problem Solving, Programs, and Computers

Topic Summary: This course has three major themes: a rigorous introduction to the process of algorithmic problem solving, an introduction to the organization of the computers upon which the resulting programs run, and an overview of the social and ethical context in which the field of computing exists. Problem solving rigor is guaranteed by a commitment to the precise specification of problems and a close association between the problem solving process and that specification. For example, the identification of a loop is closely tied to the discovery of its invariant, which in turn is motivated by the problem specification itself.

[5] When the optional 12-hour knowledge unit *PR: Introduction to a Programming Language* is incorporated into the curriculum, the total number of lecture hours increases to about 283.

Computer organization is introduced by way of a simple von-Neumann machine model and assembler, upon which students can develop and exercise simple programs. Elements of the fetch-execute cycle, runtime data representation, and machine language program structure are thereby revealed.

This course contains 40 lecture hours of KU topics, and is taught as a four credit hour course. A scheduled weekly laboratory is used to teach programming language syntax and machine organization, as well as to support student programming exercises.

Prerequisites: An introduction to logic, as would usually be found at the beginning of a discrete mathematics course (that can be taken concurrently)

Knowledge units: AL3 (3/3), AL6 (2/6), AR3 (2/3), AR4 (7/15), NU1 (1/3), NU2 (2/4), PL1 (2/2), PL3 (2/2), PL4 (1/4), SE1 (11/16), SE5 (4/8), SP1 (3/3)[6]

Data Structures and Analysis of Algorithms

Topic Summary: This course covers data structures and algorithms in some depth. Topics covered include data structures, a more formal treatment of recursion, an introduction to basic problem solving strategies, and an introduction to complexity analysis, complexity classes, and the theory of computability and undecidability. Sorting and searching algorithms are presented in the light of the presentation of problem-solving strategies and complexity issues. Finally, parallel and distributed algorithms are introduced briefly.

This course is taught in three lectures and a two-hour laboratory per week. It contains 33 lecture hours devoted to KU topics and their laboratories.

Prerequisites: "Computing II"

Knowledge units: AL1 (9/13), AL3 (2/3), AL4 (4/4), AL5 (4/4), AL6 (3/6), AL7 (3/6), AL8 (6/6), AL9 (2/3)

For more examples of course descriptions, readers should scan the sample implementations in the appendix.

10.3 Designing Courses from Advanced/Supplemental Material

Every implementation will have required and elective material beyond these common requirements. Individual programs may choose to add topics to courses that primarily contain material from the knowledge units. They may also specify additional required or elective advanced/supplemental courses.

[6] In the list of knowledge units that accompany a particular course description, the notation p/q means that p lecture hours are used out of a total of q hours that are available from the knowledge unit. Thus, when p=q the entire knowledge unit is used in that course.

The advanced/supplemental courses in a curriculum will build upon the common requirements, and will also contain additional topics not specified in the knowledge units. It is quite possible that topics from the common requirements will reappear in advanced courses, where they may be studied in greater depth. Programs can take advantage of advanced courses to guarantee that students receive a balance of theory, abstraction, and design.

The selection of advanced courses will depend on many factors, as described in section 10.1. The strengths and interests of the faculty will be particularly influential in these choices. However, the overriding concern in deciding what advanced courses will be offered should be their relation to the goals of the program. Part II contains descriptions of selected topic areas for advanced courses. Some of the material listed as prerequisite to these topics may well be taught as part of the course itself.

10.4 Integrating the Curriculum into a Course of Study

The computing curriculum will have to be developed to meet institutional requirements and should take advantage of institutional strengths. Ultimately, the computing curriculum will be integrated into a course of study. The appendix of this report gives a number of examples of curricula, including complete courses of study.

Some suggestions for steps to start building a curriculum are:

- *Identify goals of the program*, focusing on student outcomes:
  For what should graduates be prepared?
  Should there be different tracks available to suit different student interests?
  Are there content topics beyond the common requirements that every student should study?

- *Identify strengths of the faculty:*
  What instructional experience do faculty members have?
  Where is their strongest technical expertise?
  What are their interests?

- *Identify constraints of the local situation:*
  What is the institutional philosophy?
  What are the institution's general education requirements?
  What resources are available?
  With what backgrounds do students arrive?

- *Establish a plan and schedule for:*
  design
  implementation
  evaluation

modification
transition

- *Design and implement the curriculum components*, including courses that cover the common requirements, the advanced and supplemental material, and other educational experiences to form complete guidance for students to plan coherent courses of study.

11 Related Concerns

This section summarizes the Task Force recommendations with respect to faculty and staff, laboratory resources, service courses, library support, and the relationship between these curriculum recommendations and other institutions (accreditation of professional programs, secondary school curricula, and evaluation of outcomes).

11.1 Faculty and Staff

The curriculum recommendations in this report cannot be well-implemented without an appropriate complement of full-time faculty whose primary field of expertise is in the discipline of computing. Provision should be made for the faculty to remain current in the discipline.

This means that, in the long term, most of the faculty should have Ph.D.'s in the discipline or the equivalent. Programs that are recruiting new or replacement faculty should aggressively try to hire candidates with Ph.D.'s in computing.

11.2 Laboratory Resources

To support the variety of laboratory work that the curriculum requires, lab equipment and software should be kept current. A rolling plan for laboratory improvement should be kept that includes buying or leasing equipment and software, licenses and contracts, and regular maintenance in a timely fashion. This plan should also provide technical support for hardware and software, ensuring that excessive faculty resources are not required for this purpose.

Laboratories should not only be schedulable for "closed" laboratory sessions, they should also be freely accessible at other times so that students have access to facilities to complete their assignments in a timely fashion. However, the dichotomy between free access to laboratory facilities and the need for protection and safety should also be addressed. The choice between providing an environment for exploring a subject without constraints and restricting the task so that it limits such free exploration should be conscientiously addressed in designing open and closed laboratory assignments.

Laboratories should be staffed with assistants who can not only monitor the appropriate use of equipment and software, but also assist students with the technical aspects of assigned laboratory work.

More information on strategies and concerns for implementing computing laboratories can be found in [4, 8, 13, 14].

11.3 Service Courses and Joint Degree Programs

Recommendations for the design and content of service courses in the discipline of computing are beyond the scope of the present Task Force. However, many of the alternative introductory courses that appear in the example implementations of the appendices may serve well as service courses for non-majors.

Recommendations for the design of joint degree programs, between computing and another discipline, are also beyond the scope of the present report. However, some of the example implementations in the appendix might serve as a starting point for the development of such programs.

11.4 Library Support

Library support is an essential part of any program in computing. Library holdings should contain books, journals, and conference proceedings that are appropriate to the discipline, and be continually refreshed with new publications as they appear. Fundamental to these holdings are the various publications of professional societies, such as the IEEE Computer Society and the ACM.

Appropriate library staff and adequate student access must be available so that students and faculty may use the library's resources conveniently.

11.5 Relationship with Accreditation, Placement Tests, and Achievement Tests

Sample implementations $\mathcal{A}$–$\mathcal{I}$ that appear in the appendix are consistent with the curricular aspects of ABET and CSAB accreditation criteria. However, the accreditation criteria cover various additional program attributes that go beyond the curriculum. Interested readers should consult [1, 5] for more information on accreditation criteria.

Recommendations for the design of secondary school curricula, such as the Advanced Placement (AP) curriculum in computer science, are beyond the scope of the present Task Force. Recommendations for the design of outcome assessment instruments, such as the Graduate Record Examination (GRE) and the Major Field Achievement Test (MFAT) in computer science, are also beyond the scope of the present Task Force.

Future versions of all these instruments will hopefully take the present report's recommendations into consideration.

12 Summary

This report presents a specification for implementing undergraduate programs in computing, as opposed to a single curriculum. This approach is required because of the need to serve a large and diverse constituency. The ultimate success of this approach will therefore depend strongly upon the creative energies of faculty members at the various institutions that support undergraduate programs in computing.

This specification acknowledges the primary importance of the following elements in undergraduate computing curricula:

nine major subject areas
theory, abstraction and design
recurring concepts
the social and professional context
mathematics and science
language and communication
integrated laboratory experience

Successful undergraduate programs that follow from this specification will pay close attention to each of these elements in their implementations.

This report organizes the common subject matter for all programs in the form of *knowledge units*. However, a complete curriculum is far more than a collection of courses constructed out of these knowledge units, as the above list suggests. Each program is expected to require students to cover a substantial amount of material beyond the common requirements in order to ensure depth of study as well as breadth.

The appendix contains several sample curricula that are provided as "proofs of concept" for these specifications. This appendix illustrates that the report provides a basis for building effective curricula at individual institutions, curricula that both embrace the above elements and are consistent with the institutions' own priorities and constraints. Further, the appendix includes both implementations that introduce the material in a breadth-first fashion and implementations that introduce material in a more traditional depth-first fashion. Finally, the appendix presents implementations that are designed to prepare a graduate for immediate entry into the profession as well as implementations that have other educational goals.

As the discipline of computing continues to evolve, so should its undergraduate curricula. This report presents curriculum design guidance in such a way that the discipline and its undergraduate curricula can continue to evolve together for a long period of time following the actual publication of this report.

Part II

Details of the Subject Matter

This part of the Report contains detailed descriptions of the common requirements and advanced/supplemental topics that are used in building complete undergraduate curricula in the discipline of computing.

13 Details of the Common Requirements

The knowledge units are presented in the following pages by subject area, which in turn are arranged alphabetically. No particular order of preference or scheduling is implied by this order of presentation. A full discussion of the mapping of this material to courses is given in the body of the report. Sample mappings of this subject matter into course sequences are given in the appendix.

The number after the "Lecture Topics" part of each knowledge unit indicates the **approximate** number of lecture hours allocated to that set of topics to cover that unit. These approximate numbers of lectures are intended to describe a balance of subject matter coverage across the various topics in the common requirements. The numbers given should be interpreted as the **minimum** number of hours that should be devoted to the material in a **typical** curriculum. Different implementations will treat the various topics in each knowledge unit with varying degrees of depth. These variations reflect differences in individual faculty tastes as well as institutional differences.

Note that individual knowledge units are *not* necessarily indivisible. If it seems appropriate to break the material in a unit across course boundaries, then that can be done. Such subdivision of a knowledge unit is appropriate in cases where a topic is introduced in one course and then covered in more depth later. However, curriculum designers should ensure that the knowledge units so split are not separated in such an arbitrary manner that it hinders the effective presentation of the material.

Finally, we note that the classification of knowledge units among the various subject areas is, in a few cases, arbitrary. For instance, the knowledge unit "AI2: Problems, State Spaces, and Search Strategies" could equally have been classified within the AL area rather than the AI area. In most cases, the classifications are reasonable; these exceptions therefore should not hinder the overall clarity of this style of presentation for the common requirements.

Each individual knowledge unit (KU) in this appendix has the following style of presentation:

KU Tag#: Knowledge Unit Name
Paragraph describing the goals and general subject matter coverage of the knowledge unit.

Recurring Concepts: List of recurring concepts that are appropriate to this knowledge unit.

Lecture Topics: ((minimum number of lecture hours) hours minimum)

1. lecture topic
2. lecture topic

⋮

Suggested Laboratories: (open or closed) List of typical laboratory exercises and goals.

Connections:

Related to: List of KU tags for units that overlap this one
Prerequisites: List of KU tags for prerequisite knowledge units
Requisite for: List of KU tags for knowledge units that follow this one

Here, the "KU Tag" is the two-letter subject area tag (see Figure 2) under which the knowledge unit is classified and cross-referenced.

AL: Algorithms and Data Structures

There are approximately 47 hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Algorithms and Data Structures emphasize the following topics: basic data structures, abstract data types, recursive algorithms, complexity analysis, sorting and searching, computability and undecidability, problem-solving strategies, and parallel and distributed algorithms.

AL1: Basic Data Structures

Introduction to the definition, implementation, and applications of the basic data structures and associated operators that are found in computer science. These include lists, arrays, tables, stacks, queues, trees, and graphs.

Recurring Concepts: conceptual and formal models, consistency and completeness, efficiency, levels of abstraction, reuse, trade-offs and consequences.

Lecture Topics: (13 hours minimum)

1. Definitions of the basic data structures
2. Use of the basic data structures
3. Contiguous and linked implementations; customization to fit problem requirements, including space vs. time trade-offs

Suggested Laboratories: (open) Use of stacks, queues, and other data structures in typical computer science applications, such as backtracking, parsing, discrete simulations, and so forth. Examination of trade-offs among different implementation strategies, such as linked lists vs. arrays. Design of new data structures and their implementations. Students gain familiarity with using appropriate data structures in solving problems, and creating alternative implementations of data structures.

Connections:

Related to: AL2, PL3
Prerequisites: SE1, Discrete Mathematics
Requisite for: OS7, PL6

AL2: Abstract Data Types

Introduction to the concept of abstract data type (ADT), and motivation as a methodology to separate implementation details from applications.

Recurring Concepts: complexity of large problems, conceptual and formal models, levels of abstraction, reuse.

Lecture Topics: (two hours minimum)

1. Purpose of ADT's
2. Implementation of ADT's in a higher-order language, with examples

Suggested Laboratories: (open or closed) Develop a program using an ADT provided by the instructor, and then run it using alternative implementations. Students gain an

appreciation for abstraction, modularity and substitutability of different implementations.

Connections:

Related to: AL1, PL2, PL3, PL5
Prerequisites: SE1
Requisite for: PL11, SE2, SE3

AL3: Recursive Algorithms

Introduction to the foundations and use of recursive algorithms in problem solving.

Recurring Concepts: conceptual and formal models, consistency and completeness, levels of abstraction, trade-offs and consequences.

Lecture Topics: (three hours minimum)

1. Introduction to recursive algorithms
2. Connection with mathematical induction
3. Comparison of iterative and recursive algorithms

Suggested Laboratories:

1. (closed) Trace execution of a recursive program, viewing changes of values of local variables and parameters. Student gains a better understanding of recursion.
2. (open) Develop a recursive solution to a programming problem. Student gains experience with using recursion as a problem-solving tool.

Connections:

Related to:
Prerequisites: SE1, Discrete Math (Proof by Induction)
Requisite for: AL4, AL7, PL1, PL6, PL7

AL4: Complexity Analysis

Introduction to the notion of computational complexity (time and space), and its use in the analysis of algorithms.

Recurring Concepts: complexity of large problems, efficiency, trade-offs and consequences.

Lecture Topics: (four hours minimum)

1. Asymptotic analysis (big "O", little "o") of upper and average bounds including iterative and recursive algorithms
2. Time vs. Space trade-offs in algorithms

Suggested Laboratories: (closed) Corroboration of theoretical complexity by experimental methods, identifying differences among best, average, and worst case behaviors. Students will gain familiarity with predicting and measuring time and space requirements of common algorithms.

Connections:

Related to:
Prerequisites: AL3, Discrete Math (Recurrence Relations)
Requisite for: AL5, AL6, AL8

AL5: Complexity Classes

General overview of complexity classes P and NP, including upper, average, and lower bound analysis.

Recurring Concepts: complexity of large problems, conceptual and formal models, efficiency.

Lecture Topics: (four hours minimum)

1. Complexity classes P, NP, P-Space; tractable and intractable problems, existence of methods for obtaining approximate solutions to intractable problems
2. Lower bound analysis (e.g., for sorting)
3. NP-completeness

Suggested Laboratories: (closed) Timing of selected algorithms from various complexity classes, in order to see difference in running times for small and large problems. Students will gain an understanding of complexity classes, and a further appreciation for asymptotic measurements of complexity and the difference between average and worst case analysis.

Connections:
Related to: AL6
Prerequisites: AL4
Requisite for:

AL6: Sorting and Searching

Comparison of various algorithms for sorting and searching, with focus on complexity and space versus time trade-offs.

Recurring Concepts: complexity of large problems, consistency and completeness, efficiency, trade-offs and consequences.

Lecture Topics: (six hours minimum)

1. $O(n^2)$ sorting algorithms (e.g., insertion and selection sort); space-time complexity: best, worst cases
2. $O(n \log n)$ sorting algorithms (e.g., quicksort, heapsort, mergesort); space-time complexity: best, worst cases
3. Other sorting algorithms (e.g., Shell sort, bucket sort, radix sort)
4. Comparisons of algorithms
5. Serial search, binary search and binary search tree; space-time complexity: best, worst cases
6. Hashing, collision resolution

Suggested Laboratories: (closed) Corroboration of theoretical complexity of selected sorting and searching algorithms by experimental methods, identifying differences among best, average, and worst case behaviors.

Connections:

Related to: AL5, AL8
Prerequisites: AL4
Requisite for: AI2, OS7, PL9

AL7: Computability and Undecidability

Models of computable functions and undecidable problems. Includes discussion of total and partial recursive functions, Church's thesis, and universal machines.

Recurring Concepts: complexity of large problems, conceptual and formal models, levels of abstraction.

Lecture Topics: (six hours minimum)

1. Models of computable functions selected from Turing machines, RAM, (partial) recursive functions, lambda calculus, imperative languages; Church's thesis
2. Universal machines (e.g., universal Turing machine)
3. Decision problems; recursive and recursively enumerable problems
4. Undecidable problems (e.g., the halting problem)

Suggested Laboratories: (open) Using a simulator, have the student write programs for a Turing machine or other abstract machine.

Connections:

Related to: PL7, PL8
Prerequisites: AL3, Discrete Math (Logic and Proofs)
Requisite for:

AL8: Problem-Solving Strategies

Introduction to different strategies for constructing algorithmic problem solutions. Examples from graph algorithms, matrix operations, etc.

Recurring Concepts: complexity of large problems, conceptual and formal models, consistency and completeness, efficiency, trade-offs and consequences.

Lecture Topics: (six hours minimum)

1. Greedy algorithms; definition, examples, correctness, complexity
2. Divide-and-conquer algorithms; definition, examples, correctness, complexity
3. Backtracking algorithms; definition, examples, correctness, complexity

Suggested Laboratories: (open) Design and implement solutions to problems using greedy, divide-and-conquer, and backtracking paradigms (implementation may not be required in all cases). Students will develop an appreciation for these important problem-solving paradigms and making good choices when designing algorithms.

Connections:

Related to: AL6, AI2
Prerequisites: AL4, Discrete Mathematics
Requisite for:

AL9: Parallel and Distributed Algorithms

Introduction to the development of algorithms for parallel and distributed architectures, illustrating how parallelism can yield significant speed-up in comparison with sequential execution. Examples can be taken from areas such as parallel MAX, MIN, AND, and OR, as well as parallel adders.

Recurring Concepts: complexity of large problems, conceptual and formal models, consistency and completeness, efficiency, ordering in time.

Lecture Topics: (three hours minimum)

1. Models of parallel architecture
2. Sample parallel algorithms

Suggested Laboratories: (open) Using either a parallel computer or a simulator of a parallel computer, have the student solve a problem previously solved on a sequential system, and compare the performance.

Connections:
Related to: AR7, OS3, PL12
Prerequisites:
Requisite for:

AR: Architecture

There are approximately 59 hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Architecture emphasize the following topics: digital logic, digital systems, machine level representation of data, assembly level machine organization, memory system organization and architecture, interfacing and communication, and alternative architectures.

AR1: Digital Logic

The idea of simple building blocks implemented in different technologies; different levels of integration. Physical considerations such as delays, fan-in, fan-out. The use of a Medium Scale Integrated (MSI) device (Programmable Logic Device, or PLD) to implement complex functions in a single chip. Common flipflop types. Representation of sequential synchronous circuits and their operation, as described by state diagrams and tables. Clocked operation and ripple-through effects, MSI devices, and their use in making many of the basic logic functions. Interconnection of large units.

Recurring Concepts: complexity of large problems, conceptual and formal models, consistency and completeness, levels of abstraction, ordering in space, ordering in time, reuse, trade-offs and consequences.

Lecture Topics: (12 hours minimum)

1. Basic logic elements and switching theory; minimization and implementation of functions
2. Propagation delays and hazards
3. Technologies; types of flipflop
4. Devices (e.g., demultiplexers, multiplexers, decoders, encoders, adders, subtractors, comparators, shift registers, counters, PLD-type devices)
5. Memories (e.g., ROM, PROM, EPROM, EAROM, RAM)
6. Analysis and synthesis of synchronous circuits, asynchronous vs. synchronous circuits

Suggested Laboratories: (closed) Design simple logic circuits and implement them with SSI, e.g., parity generation and checking and code conversions. Other design exercises should include the use of multiplexers to perform complex logic on a single chip, and the use of adders and 2's complement addition and subtraction. PLD-type devices can be designed and burned as well as PROMs and EPROMs. Sequential circuits should be designed using individual flipflops and registers. Both serial and parallel data transfer. should be included. Students learn how logic functions are implemented in combinational and sequential circuits. Comparison of implementations show reliability, e.g., hazard-free operation and trade-offs should be seen.

Connections:

Related to: PL7
Prerequisites: Discrete Mathematics
Requisite for: AR2

AR2: Digital Systems

The transfer of information from one storage device to another and the means of controlling data flow. The electronic functions of tristate devices, the bus structures and data control concepts. Various ways for describing designs.

Recurring Concepts: complexity of large problems, consistency and completeness, levels of abstraction, ordering in time, reuse, trade-offs and consequences.

Lecture Topics: (six hours minimum)

1. Register transfer notation, conditional and unconditional
2. Algorithmic state machines, steering networks, load transfer signals
3. Tristates and bus structures
4. Iteration, top down/bottom up, divide and conquer
5. Decomposition, trade-offs, economics
6. Block diagrams, timing diagrams, transfer language

Suggested Laboratories: (closed) These exercises show data transfer in algorithmic state machines. Students use tristates, including timing diagrams, to route data. Design is stressed, so that students develop a proper attitude toward design for reliability.

Connections:

Related to: SE4
Prerequisites: AR1, SE1
Requisite for: AR4, AR5

AR3: Machine Level Representation of Data

Basic machine representations of numeric and non-numeric data.

Recurring Concepts: binding, consistency and completeness, reuse.

Lecture Topics: (three hours minimum)

1. Numeric data representation; e.g., binary, octal, hexadecimal, fixed point, 1's and 2's complement, signed, floating point, decimal, BCD, XS3
2. Non-numeric data; e.g., alphanumeric, ASCII, ISO

Connections:

Related to: NU1, PL3
Prerequisites:
Requisite for: AR4, SE4

AR4: Assembly Level Machine Organization

Comparisons of different types of instruction sets and corresponding addressing modes. Emphasis on the relationships among instruction sets, fetch and execute operations, and the underlying architecture. Introduction to the concept of interrupts, as well as the purpose and specifications of a control unit with respect to logic operations. Hardwired and microprogrammed control units, their respective advantages and disadvantages. Vertical and horizontal microcoding. General methods for designing for maintenance, such as breaking up the design for easy maintenance and adding extra hardware for easier access to special registers.

Recurring Concepts: binding, consistency and completeness, ordering in space, ordering in time, trade-offs and consequences.

Lecture Topics: (15 hours minimum)

1. Basic organization; von Neumann, block diagram, data paths, control path, functional units (e.g., ALU, memory, registers), instruction cycle
2. Instruction sets and types
3. Assembly/machine language
4. Addressing modes (e.g., direct, indirect, register, displacement, indexing)
5. Control unit; instruction fetch and execution, operand fetch
6. I/O and interrupts
7. Hardwired realization
8. Microprogrammed realization; formats and coding

Suggested Laboratories: (closed) Exercises include programming at the assembly language level, detecting errors, and using a debugger. Ideally, students should have access to an independent laboratory, in order to minimize interference with other activities. Some of this work can be done with simulators. Students should appreciate the challenge of producing efficient and correct code, as well as observe the relationship between assembly/machine level languages and the architecture.

Connections:

Related to: OS6, PL2
Prerequisites: AR2, AR3
Requisite for: OS1, PL9

AR5: Memory System Organization and Architecture

Consideration of the physical implementation of large memory systems, together with the techniques of data storage and checking. Overall concepts of virtual memory, cache memory, and the consequences of multiprocessor/multicache architectures. Detailed discussion of the DMA process, as well as techniques for fault handling and factors affecting reliability.

Recurring Concepts: binding, consistency and completeness, efficiency, reuse, trade-offs and consequences.

Lecture Topics: (13 hours minimum)

1. Storage systems and technology
2. Coding, data compression, data integrity
3. Space allocation, hierarchy
4. Main memory organization, bus operations, cycle times for selection and addressing
5. Cache memory, read/write
6. Virtual memory
7. Bussing systems, control, DMA
8. Fault handling, reliability

Connections:

Related to: OS5, OS6, OS7, OS10, PL2
Prerequisites: AR2
Requisite for: AR6, AR7

AR6: Interfacing and Communication

Input/output control and how it is achieved. Techniques for interrupt handling.

Recurring Concepts: binding, consistency and completeness, ordering in time, trade-offs and consequences.

Lecture Topics: (five hours minimum)

1. Input/output control methods, interrupts
2. Interrupt acknowledgement

3. Synchronization, open loop, handshaking
4. External storage, physical organization and drives

Suggested Laboratories: (closed) Experiments on standalone equipment can be devised to demonstrate synchronization and handshaking protocols. Students will gain a better understanding of synchronization and the interrupt process.

Connections:

Related to: HU1, OS3, OS6, OS10
Prerequisites: AR5
Requisite for: OS9

AR7: Alternative Architectures

Comparison of stack, array, vector, multiprocessor, hypercube, RISC, and CISC machines. Introduction to the general topic of parallel architectures.

Recurring Concepts: complexity of large problems, conceptual and formal models, consistency and completeness, efficiency, evolution, ordering in time, security, trade-offs and consequences.

Lecture Topics: (five hours minimum)

1. Comparisons
2. CISC, RISC
3. Parallel architectures (e.g., VLIW, SISD, MISD, SIMD, MIMD)
4. Tight coupling

Connections:

Related to: AL9, OS3, OS10, PL12
Prerequisites: AR5
Requisite for:

AI: Artificial Intelligence and Robotics

There are approximately nine hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Artificial Intelligence and Robotics emphasize the following topics: history and applications of artificial intelligence; problems, state spaces and search strategies. While this coverage is minimal, it does provide a sufficient introduction to artificial intelligence that will allow students to decide whether or not to pursue further studies in this area.

AI1: History and Applications of Artificial Intelligence

Introduction to the basic history, premises, goals, social impact, and philosophical implications of artificial intelligence. Capabilities and limitations; applications in expert systems, natural language, robotics, planning, speech, and vision.

Recurring Concepts: conceptual and formal models, evolution, levels of abstraction, trade-offs and consequences.

Lecture Topics: (three hours minimum)

1. History, scope, and limits of artificial intelligence; the Turing test; games
2. Social, ethical, legal, and philosophical aspects
3. Expert systems and shells; effective applications
4. Natural language
5. Speech and vision
6. Robotics and planning

Suggested Laboratories: (closed)

1. Interaction with an existing expert system; observing its behavior, capabilities, and limitations.
2. Construction of a small expert system using an expert system shell. Students will assess the advantages and disadvantages of using such a shell, compared with solving a similar problem in an AI language such as LISP or PROLOG.

Connections:

Related to: HU1, PL1, SP1, SP2, SP3
Prerequisites:
Requisite for: AI2

AI2: Problems, State Spaces, and Search Strategies

Identification of fundamental classes of algorithms for artificial intelligence. Methods for developing appropriate state space representations for problems (where appropriate), and implementation of various search algorithms that are appropriate to each representation. Evaluation of candidate representations and search strategies in terms of their appropriateness and efficiency.

Recurring Concepts: conceptual and formal models, consistency and completeness, efficiency, levels of abstraction, ordering in space, ordering in time.

Lecture Topics: (six hours minimum)

1. Problems and state spaces, knowledge representation
2. Basic control strategies (e.g., depth-first, breadth-first)
3. Forward and backward reasoning
4. Heuristic search (e.g., generate and test, hill climbing, breadth-first search, means-ends analysis, graph search, minimax search)

Suggested Laboratories:

1. (open) Implementation of several of the search strategies discussed in lectures, using a suitable AI language.
2. (closed) Observation of the behavior of several heuristic search implementations applied to a particular problem or a set of problems. The goal of this lab is to allow students to collect data on the performance of various heuristic search algorithms.

Connections:
Related to: AL8, PL11
Prerequisites: AL6, AI1
Requisite for:

DB: Database and Information Retrieval

There are approximately nine hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Database and Information Retrieval emphasize the following topics: overview and applications of database systems, conceptual modeling, and the relational data model.

DB1: Overview, Models, and Applications of Database Systems

Introduction to the basic goals, functions, models, components, applications, and social impact of database systems.

Recurring Concepts: complexity of large problems, conceptual and formal models, evolution, trade-offs and consequences.

Lecture Topics: (four hours minimum)

1. History and motivation for database systems
2. Components of database systems; data, dictionary, database management system, application programs, users, administration
3. conceptual modeling (e.g., entity-relationship, object-oriented)
4. Functions supported by a typical database system; access methods, security, deadlock and concurrency problems, fourth generation environments
5. Recent developments and applications (e.g., hypertext, hypermedia, optical disks)
6. Social impact of database systems; security and privacy

Suggested Laboratories:

1. (closed) Interaction with a database management system. Students create a small database and evaluate how the system supports functions introduced in lectures.
2. (closed) Work in small teams on a conceptual modeling problem. Students evaluate the advantages and disadvantages of alternative solutions.

Connections:
Related to: OS8, SE4, SP1, SP2, SP3
Prerequisites:
Requisite for: DB2

DB2: The Relational Data Model

Introduction to the relational data model and the concept of a non-procedural query language. Mapping a conceptual model to a relational schema.

Recurring Concepts: complexity of large problems, conceptual and formal models, consistency and completeness, levels of abstraction, trade-offs and consequences.

Lecture Topics: (five hours minimum)

1. Relational data model terminology; mapping conceptual schema to a relational schema
2. Representing relationships, entity and referential integrity
3. Overview of relational algebra
4. Representing database relationships
5. Database query language; data definition, query formulation, update sublanguage, expressing constraints, referential integrity, embedding in a procedural language

Suggested Laboratories: (closed) Using a procedural language, students will implement the join operation of the relational algebra. At least two diverse implementations will be attempted in order to demonstrate the relative efficiency of the various techniques. The goal of this lab is to introduce students to the computational aspects of relational algebra.

Connections:

Related to: OS7
Prerequisites: DB1, SE1, Discrete Mathematics
Requisite for:

HU: Human-Computer Communication

There are approximately eight hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirement for the subject area of Human Computer Communication are directed toward providing the student with knowledge of user interfaces and fundamentals of computer graphics. The following topics are emphasized: input/output devices, use and construction of interfaces, and basic graphics concepts. Since students will gain significant experience with a variety of computing systems and their user interfaces throughout their education, HU knowledge units do not cover this subject area extensively.

HU1: User Interfaces

Introduction to the hardware/software interfaces, including their physical characteristics, that mediate human-computer communication. Characteristics of command interpreters and shells, novice vs. expert level interfaces, and on-line assistance are discussed. Features of toolkits for interface design are treated.

Recurring Concepts: binding, conceptual and formal models, consistency and completeness, levels of abstraction, ordering in space, ordering in time, trade-offs and consequences.

Lecture Topics: (five hours minimum)

1. Devices; VDTs, pointing devices, keyboards and function keys, tablets and printers, speech recognition and generation
2. Interfaces; menu systems, command languages, direct manipulation
3. Features of common interface toolkits.

Suggested Laboratories:

1. (open) Compare various large scale user interface strategies. This exercise would involve interaction using two or more system interfaces. The goal is one of making computer scientists informed consumers of user interface designs.
2. (open or closed) Features of one or more high level toolkits are explored so that the student can better appreciate the process of creating software that incorporates a good user interface

Connections:

Related to: AR6, AI1, OS6, OS9, HU2, PL2
Prerequisites:
Requisite for:

HU2: Computer Graphics

Fundamentals of computer graphics, including devices, operations (including primitives and attributes, making primitives, etc.), and software systems.

Recurring Concepts: binding, conceptual and formal models, levels of abstraction, ordering in space, trade-offs and consequences.

Lecture Topics: (three hours minimum)

1. Graphics output devices and their properties: vector devices, raster devices, frame buffers and image stores, canvases, event handling
2. Graphics primitives and attributes; 2-D and 3-D primitives, text primitives
3. Graphics software systems; general graphics standards (e.g., GKS, PHIGS)

Suggested Laboratories:

1. (open) Graphics programming exercises implemented in conjunction with this knowledge unit serves to strengthen the student's appreciation for these topics.

Connections:

Related to: HU1
Prerequisites: SE1, Linear Algebra
Requisite for:

NU: Numerical and Symbolic Computing

There are approximately seven hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Numerical and Symbolic Computation emphasize the following topics: number representation, errors, portability, and iterative approximation methods. While this coverage is surely minimal, many additional topics in the Numerical and Symbolic Computation subject area will naturally occur in other parts of the curriculum, especially in mathematics.

NU1: Number Representation, Errors, and Portability

Introduction to the main concerns of mathematical and scientific programming; the detection and control of errors in computer arithmetic; implications for portability of mathematical software.

Recurring Concepts: consistency and completeness, reuse.

Lecture Topics: (three hours minimum)

1. Finite precision of integer and floating point number representation
2. Errors in computer arithmetic and related portability issues
3. Well-conditioned and ill-conditioned problems

Connections:

Related to: AR3, PL3
Prerequisites: SE1
Requisite for: NU2

NU2: Iterative Approximation Methods

An introduction to standard methods, such as Newton's method, for iteratively approximating solutions to mathematical problems. A survey of applications for such methods in mathematics, the sciences, and engineering, including differentiation and integration.

Recurring Concepts: conceptual and formal models, consistency and completeness.

Lecture Topics: (four hours minimum)

1. Iterative approximation to mathematical problems (e.g., Newton's method)
2. Error classification; computational, representational, and methodological distinctions
3. Overview of applications in the sciences and engineering

Suggested Laboratories: (open or closed) Develop a program that solves a series of linear equations, using Gaussian elimination. Exercise it with well-conditioned and ill-conditioned sets of data.

Connections:

Related to:
Prerequisites: Calculus, NU1
Requisite for:

OS: Operating Systems

There are approximately 31 hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Operating Systems emphasize the following topics: history, evolution, and philosophies; tasking and processes; process coordination and synchronization; scheduling and dispatch; physical and virtual memory organization; device management; file systems and naming; security and protection; communications and networking; distributed operating systems; and real-time concerns.

OS1: History, Evolution, and Philosophy

An insight into operating system development, including historical information about the development of architectural support for changes in software, and the economic and technical forces that drive operating system development. An overview of structuring methods, such as monolithic, layered, and object-oriented. Case histories of significant systems, such as the following: Multics, OS/VM, Unix, Atlas, Hydra.

Recurring Concepts: complexity of large problems, consistency and completeness, evolution, trade-offs and consequences.

Lecture Topics: (three hours minimum)

1. Hardware evolution
2. Economic forces and constraints
3. Structuring methods; layered model, object-server model
4. Application needs and significant case histories

Connections:

Related to: PL2, SP1
Prerequisites: AR4
Requisite for: OS2

OS2: Tasking and Processes

The development of multitasking and its areas of usefulness. High-level views of preemptive scheduling and time-sharing. The concept of process registers and hardware-defined contexts. Models of process creation and activation.

Recurring Concepts: binding, conceptual and formal models, efficiency, levels of abstraction, ordering in time, trade-offs and consequences.

Lecture Topics: (two hours minimum)

1. Tasks, processes
2. Structures; ready list, process control blocks, etc.
3. Dispatching, context switches
4. Role of interrupts

Suggested Laboratories: (open) Design and implementation of a simple context switcher and multiple tasks, using a timer to cause context switch. Done either in a high-level language or on available simulator or machine. Student gains understanding of process context and the idea of context switch.

Connections:

Related to:
Prerequisites: OS1
Requisite for: OS3, OS4

OS3: Process Coordination and Synchronization

The idea of concurrent execution; race conditions, Bernstein's conditions, and precedence graphs. Conditions for deadlock, and deadlock avoidance, prevention, and resolution. Particular models and mechanisms for synchronization.

Recurring Concepts: conceptual and formal models, consistency and completeness, ordering in time, trade-offs and consequences.

Lecture Topics: (four hours minimum)

1. Concurrent execution
2. Sharing access, race conditions
3. Deadlock: causes, conditions, prevention
4. Models and mechanisms (e.g., busy waiting, spin locks, Dekker's algorithm, semaphores, mutex locks, regions, monitors)

Suggested Laboratories: (open) Using a simulator or an actual system, explore the consequences of shared access under different timings. Develop mechanisms to synchronize access and prove lack of conflict. Observe a deadlock and decide how to prevent it from happening. Students gain an appreciation of problems of synchronization and race conditions.

Connections:

Related to: AL9, AR6, AR7, OS4, PL12
Prerequisites: OS2
Requisite for: OS5, OS6, OS10

OS4: Scheduling and Dispatch

Preemptive and non-preemptive scheduling strategies (e.g., SJN, SJF, FIFO, LJN, round-robin, priority scheduling, and hybrid schemes). Analysis of strategies. Three levels of scheduler—short-term, medium-term, long-term.

Recurring Concepts: consistency and completeness, efficiency, ordering in time, security, trade-offs and consequences.

Lecture Topics: (three hours minimum)

1. Preemptive and non-preemptive scheduling
2. Schedulers and policies

Suggested Laboratories: (open) Run various job mixes in a simulator or actual system under various kinds of scheduling, and then analyze the results. Students learn how to analyze a scheduling policy, and the effects of different scheduling policies.

Connections:

Related to: OS3, OS6
Prerequisites: OS2
Requisite for: OS10

OS5: Physical and Virtual Memory Organization

Simple binding to physical memory. Fence registers, offset registers, partitions, pages, segments, swapping, overlays. More complex binding. Paging and segmentation, separately and together. Caching and associative buffers. Dirty and used bits. Fetch, placement and replacement policies and their effects (e.g., LIFO, FIFO, LRU, best-fit, first-fit). The idea of a working set. Thrashing.

Recurring Concepts: binding, complexity of large problems, evolution, levels of abstraction, security, trade-offs and consequences.

Lecture Topics: (four hours minimum)

1. Physical memory and registers
2. Overlays, swapping, partitions
3. Pages and segments
4. Placement and replacement policies
5. Thrashing, working sets

Suggested Laboratories: (open) Analysis of access times, delays, I/O operations to manage various job mixes under various algorithms, largely with a simulator. Adjustment of page size, page-ahead, victim policies, penalties, etc. to observe behavior of system. Students gain an understanding of memory management schemes.

Connections:

Related to: AR5, PL2
Prerequisites: OS3
Requisite for: OS8

OS6: Device Management

Dedicated processes. Double buffering, disk striping and staggering, latency timings. Free list management, caching, recovery.

Recurring Concepts: binding, consistency and completeness, ordering in space, ordering in time, security, trade-offs and consequences.

Lecture Topics: (two hours minimum)

1. Free lists, layout
2. Servers, interrupts
3. Recovery from failures

Connections:

Related to: AR4, AR5, AR6, HU1, OS4, OS7
Prerequisites: OS3
Requisite for: OS9

OS7: File Systems and Naming

Indexing and directories. Contiguous vs. block allocation, trade-offs. Support for backups and administration.

Recurring Concepts: binding, consistency and completeness, levels of abstraction, ordering in space, ordering in time, trade-offs and consequences.

Lecture Topics: (four hours minimum)

1. File layout (e.g., indexed, contiguous)
2. Directories, contents and structure
3. Naming, searching, access, backups
4. Fundamental file concepts; basic file organizations, basic file manipulations, blocking and buffering
5. Sequential files
6. Nonsequential files (e.g., hashed files, tree-structured files, B-trees, multiple-key files)

Suggested Laboratories: (open) Experiments (possibly with a simulator) on the effect of file size and transfer latencies, to gain an impression of how file systems behave. By examining the amount of disk space used and the number of accesses, students learn to retrieve and evaluate performance data that occurs out of various possible organizations of files and directories.

Connections:

Related to: AR5, DB2, OS6, OS8
Prerequisites: AL1, AL6
Requisite for:

OS8: Security and Protection

Capabilities and access lists, privacy, covert channels, physical security, authentication mechanisms (passwords, challenges, keys), formalisms. Library call, interface, argument validation and translation. Memory protection. Recovery management. Integrity and privacy. Auditing.

Recurring Concepts: conceptual and formal models, security, trade-offs and consequences.

Lecture Topics: (three hours minimum)

1. Overview of system security, with examples
2. Security methods and devices; protection, access, authentication
3. Memory protection
4. Recovery management

Connections:

Related to: DB1, OS7, SP3
Prerequisites: OS5
Requisite for:

OS9: Communications and Networking
ISO layers. TCP/IP Internet Protocol Suite. Logical connections and services. Datagram vs. connections. Internetworking and routing.

Recurring Concepts: binding, complexity of large problems, conceptual and formal models, consistency and completeness, levels of abstraction, ordering in time, security, trade-offs and consequences.

Lecture Topics: (three hours minimum)
1. Protocol suites
2. Streams and datagrams
3. Internetworking and routing
4. Servers, services

Connections:
Related to: HU1, OS10, SP3
Prerequisites: AR6, OS6
Requisite for:

OS10: Distributed and Real-time Systems
Location of control. Servers vs. distributed services. Authentication. Synchronization and deadlocks in a distributed system. Failures and recovery. Special concerns surrounding deadlines, I/O, loading. Recovery and risk control in real-time systems.

Recurring Concepts: binding, conceptual and formal models, consistency and completeness, levels of abstraction, ordering in time, trade-offs and consequences.

Lecture Topics: (three hours minimum)
1. Synchronization and timing
2. Failures, risks, and recovery
3. Special concerns in real-time systems

Connections:
Related to: AR5, AR6, AR7, OS9, PL12, SP3
Prerequisites: OS3, OS4
Requisite for:

PL: Programming Languages

There are approximately 46 hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Programming Languages emphasize the following topics: history; virtual machines; representation of data types; sequence control; data control, sharing, and type checking; run-time storage management; finite state automata and regular expressions; context-free grammars and pushdown automata; language translation systems; semantics; programming paradigms; and distributed and parallel programming constructs.

PL1: History and Overview of Programming Languages

A brief historical survey of major developments in programming languages, beginning with the evolution of procedural high-level languages. An overview of contemporary programming paradigms and their related languages, including procedural, functional, logic, object-oriented, and parallel programming.

Recurring Concepts: conceptual and formal models, evolution.

Lecture Topics: (two hours minimum)

1. Early languages; Algol, Fortran, Cobol
2. The evolution of procedural languages (e.g., the Algol, PL/1, Pascal, Euclid, Modula-2, and Ada chains of development)
3. Non-procedural paradigms and languages; functional (e.g., Lisp), logic (e.g., Prolog), object-oriented (e.g., Smalltalk), and parallel (e.g., Occam)

Connections:

Related to: AI1, SP1
Prerequisites: AL3
Requisite for: PL2

PL2: Virtual Machines

Actual vs. virtual computers. The understanding of programming languages in terms of their corresponding virtual machines (regardless of the actual architecture on which they run). Language translation understood conceptually as an implementation on a virtual machine, followed by a sequence of translations through a hierarchy of virtual computers. Binding time as an important notion in understanding the semantics of languages.

Recurring Concepts: binding, conceptual and formal models, levels of abstraction.

Lecture Topics: (two hours minimum)

1. Virtual computers for programming languages
2. Hierarchy of virtual machines presented to the user through the program, the translator, the operating system, etc.
3. Consequences of different binding times for translation

Connections:

Related to: AL2, AR4, AR5, HU1, OS1, OS5
Prerequisites: PL1
Requisite for: PL4, PL9, PL11, SE4

PL3: Representation of Data Types
Elementary and structured data types. Creation of user-defined data types and their applications.

Recurring Concepts: conceptual and formal models, levels of abstraction, ordering in space.

Lecture Topics: (two hours minimum)
1. Choice and representation of elementary data types; integer, real, Boolean, character.
2. Specification and representation of structured data types; arrays, records, sets, pointers, dynamic storage allocation.

Suggested Laboratories: (open) Solve a programming problem requiring a structured, dynamically-allocated variable. Student gains experience with language support for structured, dynamic data types.

Connections:
Related to: AL1, AL2, AR3, NU1
Prerequisites: SE1
Requisite for: PL9

PL4: Sequence Control
Flow of control in programming languages in evaluating expressions and executing statements. User-defined expressions and statements.

Recurring Concepts: binding, levels of abstraction, ordering in time, security.

Lecture Topics: (four hours minimum)
1. Expressions, order of evaluation, and side-effects
2. Statements; simple and compound
3. Subprograms and coroutines as abstractions of expressions and statements
4. Exception handling

Connections:
Related to: SE5
Prerequisites: PL2
Requisite for: PL5

PL5: Data Control, Sharing, and Type Checking
Methods of sharing and restricting access to data in programming languages. Type checking and type inference.

Recurring Concepts: binding, complexity of large problems, levels of abstraction, ordering in space, security.

Lecture Topics: (four hours minimum)
1. Mechanisms for sharing and restricting access to data (e.g., block structure, COMMON, ADT's, aliasing)

2. Static vs. dynamic scope, lifetimes, visibility
3. Parameter-passing mechanisms; reference, value, name, result, etc.
4. Varieties of type checking disciplines and their mechanics; static vs. dynamic vs. untyped, explicit vs. implicit, polymorphism vs. overloading

Suggested Laboratories:

1. (closed) Exercise the same program in languages with dynamic and static scoping, to see different effects.
2. (open) Develop a program in a dynamically typed language, illustrating flexibility over static typing.

Connections:

Related to: AL2, SE3, SE5
Prerequisites: PL4
Requisite for:

PL6: Run-time Storage Management

Allocation, recovery, and reuse of storage during program execution.

Recurring Concepts: binding, levels of abstraction, ordering in space, reuse, security.

Lecture Topics: (four hours minimum)

1. Static allocation
2. Stack-based allocation and its relationship with recursion
3. Heap-based allocation

Connections:

Related to:
Prerequisites: AL1, AL3
Requisite for: PL9

PL7: Finite State Automata and Regular Expressions

Finite state automata (fsa) as restricted models of computation and acceptors of regular expressions. Application of regular expressions to programming language analysis.

Recurring Concepts: conceptual and formal models, levels of abstraction.

Lecture Topics: (six hours minimum)

1. Deterministic and non-deterministic fsa
2. Equivalence of deterministic and non-deterministic fsa
3. Regular expressions
4. Equivalence of fsa and regular expressions
5. Applications of regular expressions

Suggested Laboratories: (closed) Use an emulator to build and run a finite state automaton that will accept a given language. Students gain practice in designing fsa.

Connections:

Related to: AL7, AR1

Prerequisites: AL3
Requisite for: PL8

PL8: Context-Free Grammars and Pushdown Automata
Use of context-free grammars (cfg, or BNF) as a formal description device for programming language syntax. Equivalence of cfg and pushdown automata (pda). Use of pushdown automata in parsing programming languages.

Recurring Concepts: conceptual and formal models, levels of abstraction.

Lecture Topics: (four hours minimum)
1. Context-free grammars
2. Nondeterministic pushdown automata
3. Equivalence of cfg's and pda's
4. Application to table-driven and recursive descent parsing

Suggested Laboratories: (open) Design and exercise a table-driven parser for a simple context-free language. Student gains comfort with parsing and its strong relationship with an underlying grammar.

Connections:
Related to: AL7
Prerequisites: PL7
Requisite for: PL10

PL9: Language Translation Systems
An overview of the language translation process, encompassing the range from compilers to interpreters.

Recurring Concepts: binding, conceptual and formal models, consistency and completeness, levels of abstraction, ordering in space, ordering in time.

Lecture Topics: (three hours minimum)
1. Comparison of pure interpreters vs. compilers; operation and use
2. Lexical Analysis and Parsing
3. Symbol table handling
4. Code generation
5. Optimization

Suggested Laboratories: (open) Develop a simple parser (e.g., recursive descent) for arithmetic expressions that returns an expression tree.

Connections:
Related to:
Prerequisites: AL6, AR4, PL2, PL3, PL6
Requisite for:

PL10: Programming Language Semantics

Use of formal and informal models to describe programming language semantics.

Recurring Concepts: conceptual and formal models, levels of abstraction.

Lecture Topics: (two hours minimum)

1. Informal semantics
2. Formal semantics (e.g., axiomatic, denotational, operational)

Connections:

Related to: SE3, SE5
Prerequisites: PL8
Requisite for:

PL11: Programming Paradigms

Introduction to alternative programming paradigms (e.g. functional, logic, and object-oriented) and languages (e.g. Lisp, Prolog, and Smalltalk, respectively). Program construction using at least two of these paradigms. Advantages and disadvantages vs. the procedural programming paradigm.

Recurring Concepts: conceptual and formal models, levels of abstraction, trade-offs and consequences.

Lecture Topics: (10 hours minimum)

1. Overview of functional, logic, and object-oriented paradigms and languages
2. Designing programs with these paradigms; run-time environment, flow of control
3. Example programs and applications
4. Advantages (e.g., referential transparency) and disadvantages (e.g., efficiency on sequential architectures) of alternative programming paradigms vs. procedural programming; applications in artificial intelligence, database, and software design

Suggested Laboratories: (open) Choose from the following, in accordance with the particular paradigms that are stressed in the lectures:

1. Develop or modify several short programs in a functional (applicative) language (e.g., Miranda, ML, FP, or a pure subset of Scheme or Lisp). Student gains familiarization with the functional programming paradigm.
2. Develop several short programs in a logic-based language (e.g., Prolog). Student gains familiarization with the logic programming paradigm.
3. Develop one or more programs in an object-oriented language (e.g. Simula 67, Smalltalk, C++, or Eiffel). Student gains familiarization with the object-oriented programming paradigm.

Connections:

Related to: AI2, SE4
Prerequisites: AL2, PL2
Requisite for: PL12

PL12: Distributed and Parallel Programming Constructs

Description of alternative realizations of asynchronous and parallel constructs in programming languages.

Recurring Concepts: conceptual and formal models, consistency and completeness, efficiency, levels of abstraction.

Lecture Topics: (three hours minimum)

1. Addition of parallel constructs (e.g. tasks, monitors, parallel loops, coroutines) to procedural programming languages
2. Problems involving scheduling and contention for resources
3. Promise of functional, logic, object-oriented or other special-purpose languages on highly parallel or distributed architectures

Suggested Laboratories: (open) Develop one or more parallel programs in a programming language that supports parallelism (e.g., Ada, Concurrent Pascal, Occam, Parlog). Student gains familiarity with parallelism and its use at this language level.

Connections:

Related to: AL9, AR7, OS3, OS10
Prerequisites: PL11
Requisite for:

PR: Introduction to a Programming Language

There are approximately 12 optional hours of lectures recommended for this set of knowledge units.

The following knowledge unit may optionally be added to the common requirements by programs that need additional time to introduce programming, beyond the time allotted to that purpose by the common requirements themselves.

PR: Introduction to a Programming Language

An introduction to the syntactic and execution characteristics of a modern programming language, along with its use in the construction and execution of complete programs that solve simple algorithmic problems.

Recurring Concepts: conceptual and formal models, efficiency, evolution, reuse, tradeoffs and consequences.

Lecture Topics: (12 hours minimum)

1. Basic type declarations (e.g., integer, real, boolean, char, string)
2. Arithmetic operators and assignment
3. Conditional statements
4. Loops and recursion
5. Procedures, functions, and parameters
6. Arrays and records

7. Overall program structure

Suggested Laboratories: (open or closed) Students should develop and run three or four programs that solve elementary algorithmic problems. Experience with compiling, finding and correcting syntax errors, and executing programs will be gained.

Connections:

Related to: SE1[7]
Prerequisites:
Requisite for:

SE: Software Methodology and Engineering

There are approximately 44 hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Software-Methodology and Engineering emphasize the following topics: fundamental problem solving concepts, the software development process, software specifications, software design and implementation, verification, and validation.

SE1: Fundamental Problem-Solving Concepts

Introduction to the basic ideas of algorithmic problem solving and programming, using principles of top-down design, stepwise refinement, and procedural abstraction. Basic control structures, data types, and input/output conventions.

Recurring Concepts: conceptual and formal models, consistency and completeness, levels of abstraction.

Lecture Topics: (16 hours minimum)

1. Procedural abstraction; parameters
2. Control structures; selection, iteration, recursion
3. Data types (e.g., numbers, strings, booleans) and their uses in problem solving
4. The software design process; from specification to implementation; stepwise refinement; graphical representation

Suggested Laboratories:

1. (closed) Exercise a given program that solves a simple prespecified problem. Trace the values of selected variables and answer specific questions about the program's behavior. Students will become familiar with the elements of program behavior and the correspondence between the steps of its execution and the problem that it solves.
2. (open) Design a program that solves a simple prespecified problem.
3. (open) Design procedures that realize the individual steps reflected in the pseudocode solution to a more intricate problem. Each procedure should be

[7] PR can either precede SE1 or be intermixed with it.

documented by way of precise specifications and implemented with appropriate parameters.

4. (closed) Implement a search problem, first using iteration and then using recursion. Trace the execution of both implementations and contrast their run-time characteristics. Determine which is best and under what circumstances.
5. (open) Design and implement a simple sorting program (e.g., insertion or bubble sort), and demonstrate that it meets the specifications of sorting.

Connections:

Related to: PR[8]
Prerequisites:
Requisite for: AL1, AL2, AL3, AR2, DB2, HU2, NU1, PL3, SE2

SE2: The Software Development Process

Introduction to models and issues concerned with the development of high quality software. Use of tools and environments that facilitate the design and implementation of large software systems. The role and use of standards.

Recurring Concepts: complexity of large problems, conceptual and formal models, consistency and completeness, levels of abstraction.

Lecture Topics: (eight hours minimum)

1. Software life-cycle models (e.g., waterfalls, prototyping, iterative development)
2. Software design objectives
3. Documentation
4. Configuration management and control
5. Software reliability issues; safety, responsibility, risk assessment
6. Maintenance
7. Specification and design tools, implementation tools

Suggested Laboratories:

1. (open) Implement a prototype for a given specification.
2. Given a software design and an intermediate implementation in an iterative development, complete the next iterative implementation.
3. Criticize a given set of documentation for a software product.
4. Given the code and specifications for a software implementation, along with a modified set of specifications, modify the code to conform to the new specifications. Describe documentation that would have been useful in performing the modification.

Connections:

Related to: SE3, SP3
Prerequisites: AL2
Requisite for: SE4

[8] SE1 can either follow PR or be intermixed with it.

SE3: Software Requirements and Specifications

Introduction to the development of formal and informal specifications for defining software system requirements.

Recurring Concepts: complexity of large problems, conceptual and formal models, levels of abstraction, reuse.

Lecture Topics: (four hours minimum)

1. Informal specifications
2. Formal specifications; preconditions and postconditions, algebraic specifications for ADT's

Suggested Laboratories:

1. Given an informal problem statement, or a user group with an implementation request, perform a requirements analysis for the implementation project and produce a requirements analysis document.
2. Given a set of informal specifications, produce a set of formal specifications for the same problem.

Connections:

Related to: PL5, PL10, SE2
Prerequisites: AL2
Requisite for: SE4, SE5

SE4: Software Design and Implementation

Introduction to the principal paradigms that govern the design and implementation of large software systems.

Recurring Concepts: complexity of large problems, conceptual and formal models, consistency and completeness, reuse.

Lecture Topics: (eight hours minimum)

1. Functional/process-oriented design
2. Bottom-up design; support for reuse
3. Implementation strategies (e.g., top-down, bottom-up, teams)
4. Implementation issues; performance improvement, debugging, antibugging

Suggested Laboratories: (open)

1. Do an object-based design for a given set of specifications.
2. Implement the design from the above lab assignment.
3. Given a problem specification and a set of working modules with specifications, do a bottom-up design for the problem, applying as much reuse as possible.
4. Do a top-down implementation for a given software design.
5. Given a design and a partial top-down implementation, perform the next step in the implementation.

Connections:

Related to: AR2, DB1, PL11

Prerequisites: SE2, SE3, PL2
Requisite for:

SE5: Verification and Validation

Introduction to methods and techniques for verification and validation of software systems.

Recurring Concepts: consistency and completeness, efficiency, trade-offs and consequences.

Lecture Topics: (eight hours minimum)

1. Using pre- and postconditions, invariants, elementary proofs of correctness
2. Code and design reading, structured walkthroughs
3. Testing (e.g., test plan generation, acceptance testing, unit testing, integration testing, regression testing)

Suggested Laboratories: (open) Given a software specification, a piece of software, and a suite of test data, use verification and validation techniques to test the software and record findings.

Connections:
Related to: PL4, PL5, PL10
Prerequisites: SE3, Discrete Mathematics
Requisite for: SP3

SP: Social, Ethical, and Professional Issues

There are approximately 11 hours of lectures recommended for this set of knowledge units.

The knowledge units in the common requirements for the subject area of Social and Ethical Issues emphasize the following topics: historical and social context, professional responsibilities, risks and liabilities, and intellectual property.

Note: While there are no laboratories listed for knowledge units SP1–SP4, the following kinds of activities should accompany their coverage in a course of instruction:

1. Write a short expository paper, with references, that demonstrates understanding of the historical or social context of some specific aspect of computing (e.g., computers in medicine, computerization and work, information access and privacy, public interest in computer records, the societal role of research).

2. Write a short paper, with references, that discusses an incidence of misuse of computers or information technology.

3. Discuss particular aspects of professionalism in a seminar setting. Draw conclusions about ethical and societal dimensions of the profession.

4. Write or discuss a short paper discussing methods of risk assessment and reduction, and their role in the design process.

5. Present a case study in copyright or patent violation as a seminar discussion, with an accompanying writing assignment that demonstrates student understanding of the principles.

SP1: Historical and Social Context of Computing

An introduction to the larger social context with which the discipline of computing exists, with an emphasis on how the discipline interacts with and serves the interests of society. The history of the discipline and the profession serves as a starting point for this study.

Special attention is paid here to the general goals and methods of computing professionals, emphasizing activities, methods, and values that they share in common with colleagues in the other scientific and engineering disciplines. The uniqueness of computing as a discipline, such as uncommon levels of access to information and processing power, is also noted with respect to its special requirements for professional integrity. Finally, an overview of the use of computer technology by various government and private industry professions is given. Here, the limitations imposed by the absence of non-technical skills, as well as the notion of limits of the technology itself, should be introduced.

Recurring Concepts: complexity of large problems, consistency and completeness, evolution, trade-offs and consequences.

Lecture Topics: (three hours minimum)

1. Historical and social context of computing
2. Definition of its subject areas and professional activities
3. Similarities and differences with other scientific and professional disciplines
4. Uses, misuses, and limits of computer technology

Connections:

Related to: AI1, DB1, OS1, PL1
Prerequisites:
Requisite for: SP2, SP3, SP4

SP2: Responsibilities of the Computing Professional

The various social and ethical responsibilities of the computing professional are emphasized. These include professional ethics, concern for information security and privacy, whistleblowing, the role of professional societies, social responsibilities, knowing one's own limitations, the continuity of professional advancement, the role of the professional in educating society, and technical consultancy in public policy issues.

Recurring Concepts: complexity of large problems, consistency and completeness, evolution, security, trade-offs and consequences

Lecture Topics: (three hours minimum)

1. Professional societies and their role
2. Social responsibilities (e.g., security and privacy)
3. Professional growth
4. Ethical choices (e.g., whistleblowing)

Connections:

Related to: AI1, DB1, SP3, SP4
Prerequisites: SP1
Requisite for:

SP3: Risks and Liabilities

A survey of the kinds of risks that can accompany a computing application, and the considerations that a system designer can make to account for these risks. Such risks include software and hardware bugs, unforeseen user interactions, security risks, privacy violations, unethical uses, user misunderstandings, and misapplications of a system. Discussion of losses and associated questions of liability.

Recurring Concepts: consistency and completeness, security, trade-offs and consequences

Lecture Topics: (three hours minimum)

1. Types of risks; bugs, security, privacy, misuse, etc.
2. Types of losses
3. Losses and liability; personal, institutional, legal aspects

Connections:

Related to: AI1, DB1, OS8, OS9, OS10, SE2, SP2, SP4
Prerequisites: SE5, SP1
Requisite for:

SP4: Intellectual Property

Identification of the main forms of intellectual property, the means for protecting it, and the penalties for violating it. Focus is placed on the reasons for the notion of intellectual property within the larger framework of society. Legal difficulties raised by modern forms for representing intellectual property, such as novel software systems and hardware designs. The differences between patents, copyrights, trade secrets, trademarks and other means of protection, and their respective penalties for violation, both national and international.

Recurring Concepts: evolution, levels of abstraction, security, trade-offs, and consequences.

Lecture Topics: (two hours minimum)

1. Definition of intellectual property; motivation and applications in computing
2. Means of protection; patent, copyright, trade secret, trademark
3. Infringement and penalties

Connections:

Related to: SP2, SP3
Prerequisites: SP1
Requisite for:

14 Details of the Advanced/Supplemental Material

The following descriptions are intended to give detailed guidance for the design of advanced and supplemental courses in certain topic areas of the discipline. Each one contains a topic summary, an overview of laboratory work (if appropriate), and the prerequisites from the common requirements and other subjects that should precede advanced/supplemental study of the topic. (Recall from section 8 that these descriptions cover only 11 of the 28 advanced and supplemental topics that were listed there. Advanced and supplemental courses for the remaining topic areas in that list can also be offered in undergraduate programs.)

Advanced Software Engineering

Topic Summary: This topic area is devoted to methods and tools that increase the quality and decrease the cost of developing and maintaining complex software systems. It includes activities that cover the whole spectrum of the software development life-cycle.

Subtopics include: process and life-cycle models; specification methods, notations, and tools; validation and verification; debugging and program understanding; quality assurance; testing paradigms (i.e., unit, regression); testing strategies (e.g., white box, functional); metrics; tools (CASE); prototyping; version control; configuration management; end-user considerations; standards and international issues; documentation; maintenance; reuse; safety; reliability; portability; and project organization (e.g., cost estimates, schedules, economic models).

Suggested Laboratories: A software design and implementation project, preferably done as a group project, is an effective laboratory component of this topic area. Possible projects might include analysis, specification, and high-level design; modifying and updating an existing piece of software; testing and integrating separate components; or doing a full design and implementation task. Students should also see and use a variety of contemporary software design tools.

Prerequisites: AL1, AL2, NU1, PL1–PL6, PL11, SE (all), SP2–SP4, Discrete Mathematics.

Artificial Intelligence

Topic Summary: A selective survey of key concepts and applications of artificial intelligence, and an in-depth experience with a language commonly used for building AI systems (e.g., Lisp or Prolog).

Subtopics include knowledge representation, state space/searching, heuristic search, expert systems, expert system shells, natural language processing, propositional logic, learning and cognitive models, and vision.

Suggested Laboratories: Students will implement, modify, or enhance several AI systems using an AI language and associated tools (e.g., expert system shells, knowledge acquisition tools).

Prerequisites: AL1–AL3, AI1, AI2, PL11, SE1, SE2, Discrete Mathematics.

Computer Communication Networks

Topic Summary: This topic gives students a foundation in the study of computer networks. Current methods and practices in the use of computer networks to enable communication are covered. Also covered are the physical and architectural elements and information layers of a communication network, along with diagnostic, design, operational, and performance measurement tools that are used to implement, operate, and tune such a network. Different network architectures are contrasted, and compared with traditional mainframe and time-shared computer models.

Important subtopics include network architecture and communication protocols, network elements, data link, switching and routing, end-to-end protocols, LANs, and data security.

Suggested Laboratories: Case studies of existing network architectures and protocols (e.g., Ethernet, Wangnet, and FDDI) provide valuable laboratory work. Hardware communication devices and performance measurement tools should also be used in directed laboratory exercises.

Prerequisites: Significant coverage of the common requirements in the AR area (i.e., AR5-AR7), the OS area (i.e., OS3, OS4, OS7–OS11), and the PL area (i.e., PL5, PL6, PL12).

Computer Graphics

Topic Summary: An overview of the principles and methodologies of computer graphics, including the representation, manipulation, and display of two- and three-dimensional objects. Subtopics include characteristics of display devices (e.g., raster, vector); representing primitive objects (lines, curves, surfaces) and composite objects; two- and three-dimensional transformations (translation, rotation, scaling); hidden lines and surfaces; shading and coloring; interactive graphics and the user interface; animation techniques.

Suggested Laboratories: Students should have access to a suite of graphics software tools and a high quality color display. Exercises will provide experience with the design, implementation, and evaluation of program that manipulate and display graphic objects.

Prerequisites: HU2, SE1, SE2, Calculus, Linear Algebra, Discrete Mathematics.

Computer Security

Topic Summary: This area addresses the problem of how to secure computer systems, networks, and data from unauthorized or accidental access, modification, and denial of service. Courses designed for this area will draw material from the following subtopics: formal definitions of security, privacy, and integrity; risk assessment and management; information theory; information flow and covert channels; coding and cryptography.

Additional subtopics include: authentication methods; capabilities, access lists, and protection domains; standards; malicious software (e.g., viruses, logic bombs); audit and control methods; legal factors; database and inference control; security kernels; and verification methods.

Suggested Laboratories: None specifically recommended; however, this area is especially well-suited to the use of case studies to reinforce various principles covered above.

Prerequisites: AL4, AL5, AL7, DB (all), OS (all), SE5, SP3, Calculus, Linear Algebra.

Database and Information Retrieval

Topic Summary: This topic includes the material normally found in a first course in database systems, as well as several advanced subtopics. Basic coverage includes data models (E-R, relational, and object-oriented); query languages (relational algebra, relational calculus); the data dictionary; implementation of a relational database kernel; and case studies of commercial database languages and systems.

Additional subtopics include: query optimization; theory of normal forms and database design; transaction processing, concurrency control, and recovery from failure; security and integrity; distributed database systems; language paradigms and database languages; user interfaces and graphical query languages; advanced study of physical database organization; emerging database technologies (e.g., hypertext and knowledge-based systems); and logic as a data model.

Suggested Laboratories: Open and closed laboratories can be assigned for students to gain experience with relational database kernel implementation using an imperative language (e.g., C++ or Ada), and with implementing parts of a particular database management system. Other similar labs can be designed to cover various other subtopics listed above.

Prerequisites: DB1–DB2, HU1, OS7–OS10, SE1, SE2, Discrete Mathematics.

Parallel and Distributed Computing

Topic Summary: This topic involves the design, structure, and use of systems having interacting processors. It includes concepts from most of the nine subject areas of the discipline of computing. Concepts from AL, PL, AR, OS, and SE are important for the basic support of parallel and distributed systems, while concepts from NU, DB, AI, and HU are important in many applications.

Subtopics include concurrency and synchronization; architectural support; programming language constructs for parallel computing; parallel algorithms and complexity; messages vs. remote procedure calls vs. shared memory models; structural alternatives (e.g., master-slave, client-server, fully distributed, cooperating objects); coupling (tight vs. loose); naming and binding; verification, validation, and maintenance issues; fault tolerance and reliability; replication and avoidability; security; standards and protocols; temporal concerns (persistence, serializability); data coherence; load balancing and scheduling; appropriate applications.

Suggested Laboratories: Programming assignments should ideally be developed on a multiprocessor or simulated parallel processing architecture.

Prerequisites: AL9, AR6, AR7, OS (all), PL11, PL12, SE3, SE5.

Programming Language Translation

Topic Summary: This topic is an in-depth study of the principles and design aspects of programming language translation. The major components of a compiler are discussed; lexical analysis, syntactic analysis, type checking, code generation, and optimization. Alternative parsing strategies (e.g., top-down, LR, recursive descent) are presented and compared with respect to space and time tradeoffs.

Subtopics include ambiguity, data representation, recovery, symbol table design, binding, compiler generation tools (e.g., LEX and YACC), syntax directed editors, linkers, loaders, incremental compiling, and interpreters.

Suggested Laboratories: Laboratory exercises will assist students in reinforcing concepts by designing and implementing components of a compiler for a small but representative language. Alternative parsing strategies will be implemented and their performance compared. Laboratory work for this course is well suited to team projects.

Prerequisites: AR3, AR4, PL2–PL10, SE2.

Symbolic Computation

Topic Summary: This topic provides coverage of the foundations and uses of algebraic systems, as well as insights into current methods for effectively using computers to do symbolic computation. Students should be able to understand basic symbolic computations and their underlying data structures and algorithms. Using a currently available system, students will be able to solve mathematical problems symbolically. The role of symbolic computation in the discipline of computing and related disciplines, as well as its strengths and limitations, should also be taught.

Subtopics include computer algebraic systems; data representations; fundamental algorithms (e.g., matrix calculations, Taylor series, differentiation); polynomial simplification; advanced algorithms (e.g. modular methods for GCD, matrix inversion, polynomial factorization); formal integration.

Suggested Laboratories: Exercises should be given so that students can use a contempo-

rary symbol manipulation system (e.g. MACSYMA, REDUCE, Mathematica) to solve problems.

Prerequisites: AL1, AL4, AL8, AR3, AI2, NU1, NU2, PL3, PL4, SE1, Discrete Mathematics, Calculus, Linear Algebra.

Theory of Computation

Topic Summary: Continuation of the study of formal models of computation, including finite automata, pushdown automata, linear-bounded automata, and Turing machines (deterministic and nondeterministic). From the formal language perspective, regular, context-free, context-sensitive, and unrestricted grammars will be studied and shown to be equivalent to the corresponding machine models. Church's thesis will be discussed and the equivalence of various models of computation (e.g., Turing machines, random-access machines, lambda calculus, and recursive functions) is also included. These models provide a basis for the study of computability, including effectively enumerable and undecidable problems.

Suggested Laboratories: Optionally, students will design and implement simple Turing machines or automata using a simulator (e.g., Turing's World).

Prerequisites: Discrete Mathematics, AL5, AL7, PL7, PL8, SE5.

VLSI System Design

Topic Summary: This material is intended to provide students with an understanding of the design and implementation of VLSI logic devices. This includes issues of casting digital system architectures into silicon devices and the associated design alternatives. Examples of practical design methodologies (e.g., CAD tools) and the attributes of available technologies are compared.

Following an introduction to VLSI design, this topic covers integrated circuit technologies, design methodologies for VLSI, semicustom and custom MOS circuit design, and support technologies. Further subtopics include high-speed VLSI and application-specific systems, systolic arrays, and wafer scale integration.

Suggested Laboratories: A current equipped laboratory for the design and simulation of VLSI logic devices should be available. This will enable students to develop a working knowledge of the tools and experiment with different organizations. A significant VLSI design should be the capstone experience for this topic. (Fabrication and testing of the design are not necessary.)

Prerequisites: All of the AR common requirements are a prerequisite for this topic. It is also recommended that the OS and SE common requirements be covered before studying this topic.

15 Acknowledgments

The Task Force wishes to thank the following persons who generously served as reviewers for this work in its various stages:

Narenda Ahuja, Donald J. Bagert, Jr., James C. Bezdek, Nathaniel Borenstein, Richard J. Botting, Donald Bouldin, Albert W. Briggs, Jr., J. Glenn Brooksheer, Wai-Kai Chen, Lucio Chiaraviglio, Neal S. Coulter, Steve Cunningham, Ruth Davis, Peter J. Denning, Scot Drysdale, Larry A. Dunning, Peter Durato, J. Philip East, Adel S. Elmaghraby, John W. Fendrich, Gary A. Ford, Edwin C. Foudriat, Donald L. Gaitros, David Garnick, Judith L. Gersting, Sakti P. Ghosh, Ratan K. Guha, Stephen T. Hedetniemi, Thomas T. Hewett, Jane Hill, Stuart Hirshfield, James A. Howard, John Impagliazzo, Greg Jones, Charles Kelemen, Willis King, Robert L. Kruse, Yedidyah Langsam, Eugene Lawler, Burt Leavenworth, R. Rainey Little, Dennis Martin, Fred J. Maryanski, Jeffrey J. McConnell, Daniel D. McCracken, Catherine W. McDonald, John McPherson, David G. Meyer, Victor Nelson, Chris Nevison, Robert Noonan, Jeffrey Parker, Margaret Peterson, Paul Purdom, Arthur Riehl, David C. Rine, Rockford J. Ross, Richard Salter, G. Michael Schneider, Greg Scragg, Mary Shaw, Daniel P. Siewiorek, David L. Soldan, Harry W. Tyrer, Annelieses von Mayrhauser, Z. G. Vranesic, Henry Walker, F. Garnett Walters, John Werth, Terry Winograd, Charles W. Winton, Charles T. Wright, Jr., and Grace Chi-Dak N. Yeung.

Additional thanks to Kathleen A. Heaphy for proofreading and editorial comments on early drafts of the report.

References

[1] Accreditation Board for Engineering and Technology, Inc. Criteria for accrediting programs in engineering in the United States. Technical report, December 1988.

[2] ACM Curriculum Committee on Computer Science. Curriculum 68: Recommendations for the undergraduate program in computer science. *Communications of the ACM*, 11(3):151–197, March 1968.

[3] ACM Curriculum Committee on Computer Science. Curriculum 78: Recommendations for the undergraduate program in computer science. *Communications of the ACM*, 22(3):147–166, March 1979.

[4] A. W. Bennett. A position paper on guidelines for electrical and computer engineering education. *IEEE Transactions on Education*, E-29(3):175–177, August 1986.

[5] Computing Sciences Accreditation Board. Criteria for accrediting programs in computer science in the United States. Technical report, January 1987.

[6] Peter J. Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen B. Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Communications of the ACM*, 32(1):9–23, January 1989.

[7] Educational Activities Board. The 1983 model program in computer science and engineering. Technical Report 932, Computer Society of the IEEE, December 1983.

[8] Educational Activities Board. Design education. Technical report, Computer Society of the IEEE, October 1986.

[9] Norman E. Gibbs and Allen B. Tucker. Model curriculum for a liberal arts degree in computer science. *Communications of the ACM*, 29(3):202–210, March 1986.

[10] Elliot P. Koffman, Philip L. Miller, and Caroline E. Wardle. Recommended curriculum for CS1: 1984 a report of the ACM curriculum task force for CS1. *Communications of the ACM*, 27(10):998–1001, October 1984.

[11] Elliot P. Koffman, David Stemple, and Caroline E. Wardle. Recommended curriculum for CS2, 1984: A report of the ACM curriculum task force for CS2. *Communications of the ACM*, 28(8):815–818, August 1985.

[12] Michael C. Mulder and John Dalphin. Computer science program requirements and accreditation—an interim report of the ACM/IEEE computer society joint task force. *Communications of the ACM*, 27(4):330–335, April 1984.

[13] Ware Myers. The crisis in higher education: solutions. *IEEE Computer*, 18(7):69–79, July 1985. Report of the COMPSO 85 Panel Session.

[14] The National Advisory Group of Sigma Xi. An exploration of the nature and quality of undergraduate education in science, mathematics and engineering. Technical report, Sigma Xi, The Scientific Research Society, New Haven, CT, January 1986.

A Sample Curricula

This appendix contains a variety of sample curricula that illustrate effective implementations that follow from the specifications in the main body of the report. These implementations exemplify how various kinds of undergraduate programs can be effectively implemented in different institutional settings, and hence serve a wide variety of educational goals.

The sample curricula are presented in two separate sections, as identified in the two tables below. These tables can serve as a guide to identifying sample curricula that are most relevant to particular institutional needs and priorities.

The curricula in the first section of this appendix have as a common goal the preparation of graduates for entry into the computing profession. These are identified as follows.

Implementation	Title	Page
Implementation $\mathcal{A}$:	A Program in Computer Engineering	82
Implementation $\mathcal{B}$:	A Program in Computer Engineering (Breadth-First)	89
Implementation $\mathcal{C}$:	A Program in Computer Engineering (Minimal Number of Credit-Hours)	96
Implementation $\mathcal{D}$:	A Program in Computer Science	103
Implementation $\mathcal{E}$:	A Program in Computer Science (Breadth-First)	109
Implementation $\mathcal{F}$:	A Program in Computer Science (Theoretical Emphasis)	115
Implementation $\mathcal{G}$:	A Program in Computer Science (Software Engineering Emphasis)	121
Implementation $\mathcal{H}$:	A Liberal Arts Program in Computer Science (Breadth-First)	127
Implementation $\mathcal{I}$:	A Program in Computer Science & Engineering	133

The curricula in the second section of this appendix have other goals than the preparation of graduates for entry into the profession. These curricula are identified as follows.

Implementation	Title	Page
Implementation $\mathcal{J}$:	A Liberal Arts Program in Computer Science	137
Implementation $\mathcal{K}$:	A Liberal Arts Program in Computer Science (Breadth-First)	143
Implementation $\mathcal{L}$:	A Program in Computer Science (Theoretical Emphasis)	149

All of the curricula that appear in both sections of this appendix are presented in a common format, with four major parts:

1. A set of goals for the program of study and an explanation of any institutional or curricular constraints that are assumed;
2. A summary of degree requirements, in tabular form, to indicate the curricular content in its entirity;
3. A set of course descriptions for those courses in the computing component of the curriculum;
4. A sample four year schedule that a typical student might follow.

In order to clarify the identification of courses, levels, and implementations, each course is numbered in a way that identifies the implementation in which it appears and the level at which it is normally taught. Thus, a course numbered C_x200 is a course in implementation x that is ordinarily taught at the sophomore level, while course C_y409 is a course in implementation y that is usually taught at the senior level.

To provide additional clarity, all of the sample implementations contain courses based on the semester system, in which a semester provides 14 weeks of lecture and lab time exclusive of final examinations, vacations, and reading periods.

A.1 Sample Curricula for Preparing Students to Enter the Profession

All of the sample curricula in this section have as a major goal the preparation of graduates for entry into the computing profession. There are many ways of building a curriculum whose graduates are well educated computer scientists and engineers. To emphasize this point, there are nine distinct programs of study outlined in this section. These programs differ in their emphasis and in the institutional constraints that are assumed.

Each of the implementations in this section has been designed to satisfy the curricular requirements of the relevant professional accreditation body (EAC/ABET or CSAC/CSAB). However, professional accreditation addresses much more than just curriculum, and readers who are interested in accreditation should consult the relevant EAC/ABET or CSAC/CSAB criteria ([1] or [5]) for complete accreditation criteria.

The sample programs shown in implementations $\mathcal{A}$–$\mathcal{C}$ in this section are designed to satisfy the quantitative curricular requirements of current EAC/ABET criteria for computer engineering programs. With respect to curriculum, each of these three implementations meets or exceeds the EAC/ABET minimum requirements in science and mathematics, computing (i.e., "computer science and engineering"), other engineering subjects, humanities and social sciences, and the minimum design experience. The specific topic distribution illustrated in the Typical Student Schedule of each implementation also meets the EAC/ABET topic requirements.

The sample programs shown in implementations $\mathcal{D}$–$\mathcal{H}$ in this section are designed to satisfy the quantitative curricular requirements of current CSAC/CSAB criteria. With respect to curriculum, each of these five implementations meets or exceeds the CSAC/CSAB minimum requirements in science and mathematics, computing (i.e., "computer science"), and humanities and social sciences.

The sample program shown in implementation $\mathcal{I}$ satisfies both the quantitative curricular aspects of current EAC/ABET criteria and the quantitative curricular aspects of current CSAC/CSAB criteria.

Implementation $\mathcal{A}$: A Program in Computer Engineering

Goals and Features of this Program: This program leads to the B.S. in Computer Engineering, providing a balanced treatment of hardware and software principles. The common requirements are spread widely across a range of courses, allowing the subject matter to be revisited and spiral learning to take place. The curriculum contains sufficient flexibility to support various areas of specialization. (Though three areas of specialization are illustrated in this example, more or fewer can be offered, depending on such factors as faculty size, faculty interests, and other considerations.) This program structure also allows a course of study that is broadly based, providing an unconstrained selection from among the professional electives. In all cases, the capstone design experience takes place after sufficient depth of coverage in the major subject areas has been completed.

This program's goals are to prepare students for a lifetime career in computing by establishing a foundation for lifelong learning and development. It also provides a platform for further work leading to careers in such fields as business, law, and medicine.

Summary of Requirements: This program of study is built around a set of twelve required courses in computer science and engineering, comprising 38 credit-hours of study. Flexibility is incorporated through judicious choice of electives in computer science and engineering as well as in other areas. The computer engineering segment of the curriculum, including professional electives, comprises 56 credit-hours of study. This curriculum utilizes a relatively traditional course structure and content.

This program requires 42 courses, with credit-hours distributed as follows:

15	hours of physics, chemistry, and science electives
22	hours of mathematics
24	hours of humanities and social sciences (to include English composition)
38	hours of required computer science and engineering
18	hours of computer science and engineering electives
9	hours of other engineering courses
9	hours of free electives
135	

Course Descriptions

$C_{\mathcal{A}}$101, 102 Introduction to Computing I, II

Topic Summary: This is a two semester course that introduces algorithmic problem solving. In the context of a modern programming language, such topics as problem

solving strategies, basic data structures, data and procedural abstraction, and algorithm complexity are discussed. Examples involving searching and sorting techniques and elementary numerical analysis are used to motivate students. Students hone their problem solving skills in the programming laboratory associated with the course. They also gain experience in the laboratory with the user interfaces of a variety of systems.

C_A101 and C_A102 are each three credit-hour courses taught in two lectures and one two hour laboratory-sized recitation per week. Over the two semesters, there are 56 available lectures, of which 46 are identified as pertaining to specific KU's and their laboratories. Thus there is ample time for adding the PR knowledge unit if this is the first programming experience for students.

Prerequisites: Calculus and/or Discrete Mathematics concurrently

Knowledge units: AL1 (5/13), AL2 (1/2), AL3 (1/3), AL4 (4/4), AL6 (3/6), AL8 (3/8), HU1 (2/5), NU1 (3/3), NU2 (4/4), PL1 (2/2), PL4 (2/4), SE1 (6/16), SE2 (1/8), SE3 (1/4), SE4 (2/8), SE5 (1/8), SP1 (2/3), SP2 (2/3), SP4 (1/2)

C_A201 Introduction to Computer Engineering

Topic Summary: This is a first course in digital systems, including a treatment of logic and digital circuits as well as design using register level components. Data representation, device characteristics, and register transfer notation are covered in a manner that stresses application of basic problem solving techniques to both hardware and software design. Requirements specification, the design process, and issues associated with use of graphical interfaces are also discussed.

C_A201 is a three credit-hour course containing 29 lecture hours of KU topics and their laboratories.

Prerequisites: C_A 101, 102

Knowledge units: AR1 (12/12), AR2 (6/6), AR3 (3/3), HU1 (3/5), SE1 (2/16), SE2 (1/8), SE3 (1/4), SE5 (1/8)

C_A202 Analysis and Design of Algorithms

Topic Summary: A course in algorithms, C_A202 treats such topics as appropriate choice of data structures, recursive algorithms, complexity issues, and issues associated with computability and decidability. Intractable problems, such as those found in artificial intelligence, are discussed. An introduction to parallel algorithms is also included.

C_A202 is a three credit-hour course, containing 33 lecture hours of KU topics and their laboratories.

Prerequisites: C_A201, Discrete Mathematics

Knowledge units: AL1 (8/13), AL2 (1/2), AL3 (2/3), AL5 (4/4), AL6 (3/6), AL7 (6/6), AI1 (3/3), AI2 (6/6)

C_A301 Hardware System Design

Topic Summary: This course involves topics associated with the design and implementation of small systems. Design of hardwired control units as well as a treatment of memory system organization is integrated with a significant laboratory component to provide the student with hands on design experience.

C_A301 is a four credit-hour course containing 15 lecture hours of KU topics. The course is taught as a three hour lecture with accompanying three hour laboratory session each week. This course goes significantly beyond the common requirements in hardware design and laboratory experience.

Prerequisites: C_A201

Knowledge units: AR4 (10/15), AR5 (5/13)

C_A302 Software Systems

Topic Summary: A course in software systems, C_A302 stresses problem solving strategies and concepts applied in the context of issues associated with the design and implementation of software systems. Students gain an appreciation for intractable problems as well as an exposure to concurrent systems. Levels of abstraction are emphasized in data modeling and mapping to storage structures, and a treatment of user interfaces is included.

C_A302 is a four credit-hour course with a significant software development laboratory. Students spend a large portion of their time in design and implementation of small and medium sized software systems, and in gaining experience with various environments. The course is taught in three hours of lecture and three hours of laboratory per week, and 25 lecture hours are associated with specific KU topics.

Prerequisites: C_A202

Knowledge units: AL8 (3/6), AL9 (3/3), DB1 (3/4), DB2 (5/5), HU2 (3/3), PL2 (2/2), PL3 (2/2), SE2 (1/8), SE3 (1/4), SE4 (1/8), SE5 (1/8)

C_A303 Software Engineering

Topic Summary: This course in software engineering treats topics associated with the design and implementation of large software systems. A continued emphasis on problem solving concepts is integrated with a treatment of the software life cycle, requirements specifications, and verification and validation issues. Social and ethical issues faced by the computing professional are discussed in the context of software engineering.

C_A303 is a three credit-hour course containing 30 hours of KU topics and their laboratories. As it is taught after the students have considerable exposure to the design and implementation of small and medium scale software projects, the software engineering concepts are presented in the context of their prior experience, enabling the students to appreciate them more fully.

Prerequisites: C_A302

Knowledge units: SE1 (8/16), SE2 (5/8), SE3 (1/4), SE4 (5/8), SE5 (5/8), SP1 (1/3), SP2 (1/3), SP3 (3/3), SP4 (1/2)

C_A304 Programming Languages

Topic Summary: C_A304 is a course in programming languages that treats language design issues and language translators.

C_A304 is a three credit-hour course taught at the junior level, after students have had exposure to a variety of programming languages and problem solving paradigms. There are 42 lecture hours available in the semester, of which 37 are allocated to specific KU topics and their laboratories.

Prerequisites: C_A101, C_A102

Knowledge units: PL4 (2/4), PL5 (4/4), PL6 (4/4), PL7 (6/6), PL8 (4/4), PL9 (3/3), PL10 (2/2), PL11 (10/10), PL12 (3/3)

C_A305 Operating Systems

Topic Summary: This is a course in systems software that is largely concerned with operating systems. Such topics as process management, device management, and memory management are discussed, as are relevant issues associated with security and protection, networking, and distributed operating systems.

C_A305 is a three credit-hour course taught in three lecture hours per week. There are 34 lecture hours allocated to specific KU topics and their laboratories.

Prerequisites: C_A301, C_A302

Knowledge units: DB1(1/4), HU1 (2/5), OS1 (3/3), OS2 (2/2), OS3 (4/4), OS4 (3/3), OS5 (4/4), OS6 (2/2), OS7 (4/4), OS8 (3/3), OS9 (3/3), OS10 (3/3)

C_A306 Computer Architecture

Topic Summary: This is a course that deals with design alternatives in computer architecture. Assembly level machine organization, memory subsystem organization, interfacing concepts and issues arising in managing communication with the processor are covered, as are a number of alternative computer architectures.

C_A306 is a three credit-hour course, containing 23 lecture hours of KU topics and their laboratories.

Prerequisites: C_A301, C_A302

Knowledge units: AR4 (5/15), AR5 (8/13), AR6 (5/5), AR7 (5/5)

C_A401-402 Design Laboratory

Topic Summary: This is a two semester sequence of courses that constitutes a capstone design experience in computer engineering. Typically, it would require the student to build on the aggregated knowledge gained in previous years. A design experience involving the integration of hardware and software components as well as analysis of alternatives and trade-offs is ordinarily provided.

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Chem 101	4	Chemistry I	Phys 101	4	Physics I
Math 101	4	Calculus I	Math 102	4	Calculus II
C_A101	3	Introduction to Computing I	C_A102	3	Introduction to Computing II
Engl 100	3	English Composition	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
CSE Seminar	0		CSE Seminar	0	
	17			17	

Sophomore Year

First Semester			Second Semester		
Phys 201	4	Physics II	Sci Elect	3	Science Elective
Math 201	4	Calculus III	Math 203	4	Linear Algebra and Diff. Equations
C_A201	3	Introduction to Computer Engineering	C_A202	3	Design & Analysis of Algorithms
Math 202	3	Discrete Math	EE 201	3	Basic Circuits and Electronics
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
CSE Seminar	0		CSE Seminar	0	
	17			16	

Junior Year

First Semester			Second Semester		
C_A301	4	Hardware Systems	C_A303	3	Software Engineering
C_A302	4	Software Systems	C_A304	3	Programming Languages
EE 301	3	Linear Systems	ES 301	3	Engineering Science (e.g., Thermo, Fields)
Math 301	3	Probability & Statistics	Prof. El.	3	Professional Elective
HU/SS	3	HU/SS Elective	HU/SS	3	Professional Ethics
CSE Seminar	0		CSE Seminar	0	
	17			15	

Senior Year

First Semester			Second Semester		
C_A305	3	Operating Systems	C_A306	3	Computer Architecture
C_A401	3	Design Laboratory	C_A402	3	Design Laboratory
Prof. El.	3	Professional Elective	Prof. El.	3	Professional Elective
Prof. El.	3	Professional Elective	Prof. El.	3	Professional Elective
Prof. El.	3	Professional Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
CSE Seminar	0		CSE Seminar	0	
	18			18	

The Professional Electives found in the table below are selected to develop an area of concentration in one of the subfields of computer science and engineering. The courses mentioned in the table are typical of what might be selected, but the specific elective courses offered by any institution will be a function of the current interests of the faculty associated with the program.

	Software Engineering	Computer System Design	Knowledge-Based Systems
Prof. Elective 1	Performance Analysis	Advanced Electronics	Database Systems
Prof. Elective 2	Database Systems	Computer Communications	Computer Graphics
Prof. Elective 3	Theory of Computing	Design Automation	Artificial Intelligence
Prof. Elective 4	Compiler Construction	Performance Analysis	Expert Systems
Prof. Elective 5	Verification and Validation	VLSI Design	Distributed Databases
Prof. Elective 6	Adv. Topics in Softw. Eng'g.	Fault Tolerant Computing	Natural Language

Implementation $\mathcal{B}$: A Program in Computer Engineering (Breadth-First)

Goals and Features of this Program: This program leads to the B.S. in Computer Engineering in a way similar to Implementation $\mathcal{A}$, but with one major exception: the common requirements are presented in such a way that the breadth of the discipline is covered early in an undergraduate's program of study rather than late. Specifically, the common requirements are completely covered in a set of seven courses—the same seven courses that are the basis for three other implementations in this appendix. Furthermore, after students have completed the first four courses among these seven, they will have had some exposure to all nine subject areas of the discipline along with the area of social and professional issues.

This program's goals are to prepare students for a lifetime career in computing by establishing a foundation for lifelong learning and development. It also provides a platform for further work leading to careers in such fields as business, law, and medicine.

Summary of Requirements: This program of study is built around a set of nine required courses in computer science and engineering, comprising 34 credit-hours of study. This program of study incorporates a course structure that covers the breadth of the discipline in the first four Computer Engineering courses. Depth of coverage and the requisite design experiences are provided in the balance of the program. Flexibility is incorporated through judicious choice of electives in computer science and engineering as well as in other areas. The computer science and engineering segment of the curriculum (including professional electives) consists of 61 credit-hours of study.

This program of study requires 42 courses, with credit-hours distributed as follows:

12	hours of physics, chemistry, and science electives
21	hours of mathematics
24	hours of humanities and social sciences (to include English composition)
40	hours of required computer science and engineering
18	hours of computer science and engineering electives
9	hours of other engineering courses
12	hours of free electives
136	

Course Descriptions

$C_{\mathcal{B}}101$ Problem Solving, Programs, and Computers

Topic Summary: This course has three major themes: a rigorous introduction to the process of algorithmic problem solving, an introduction to the organization of the computers upon which the resulting programs run, and an overview of the social and ethical

context in which the field of computing exists. Problem solving rigor is guaranteed by a commitment to the precise specification of problems and a close association between the problem solving process and that specification. For example, the identification of a loop is closely tied to the discovery of its invariant, which in turn is motivated by the problem specification itself.

Computer organization is introduced by way of a simple von Neumann machine model and assembler, upon which students can develop and exercise simple programs. Elements of the fetch-execute cycle, runtime data representation, and machine language program structure are thereby revealed.

C_B101 contains 40 lecture hours of KU topics. The course is taught as a four credit-hour course. A scheduled weekly laboratory is used to teach the necessary high-level syntax and machine organization, as well as to support student programming exercises.

Prerequisites: An introduction to logic, as would usually be found at the beginning of a discrete mathematics course (that can be taken concurrently)

Knowledge units: AL3 (3/3), AL6 (2/6), AR3 (2/3), AR4 (7/15), NU1 (1/3), NU2 (2/4), PL1 (2/2), PL3 (2/2), PL4 (1/4), SE1 (11/16), SE5 (4/8), SP1 (3/3)

C_B102 Abstraction, Data Structures, and Large Software Systems

Topic Summary: This course introduces the notion of a large software system and the principles and methodologies that accompany its effective realization. A simple operating system model is used throughout the course as a motivational case study.

The idea of an abstract data type (ADT) is first introduced, and then reinforced through the characterization of fundamental data structures in the discipline—stacks, queues, and trees. Computational complexity is introduced and motivated through the design and analysis of two or three sorting algorithms. Other design issues are also explore, such as the use of dynamic storage for implementing an ADT. An overview of a compiler as a large software system is also given as a case study.

C_B102 is a four credit-hour course that covers 41 lecture hours associated with knowledge units. Laboratories for the course are of two types: closed sessions in which students work individually and open sessions in which students work in teams to solve a larger software problem than those which were introduced in course C_B101.

Prerequisites: C_B101

Knowledge units: AL1 (8/13), AL2 (2/2), AL4 (1/4), AL6 (4/6), HU1 (5/5), OS1 (3/3), OS2 (2/2), PL2 (2/2), PL5 (2/4), PL6 (4/4), PL9 (3/3), SE2 (2/8), SE5 (1/8), SP4 (2/2)

C_B203 Levels of Architecture, Languages, and Applications

Topic Summary: This course emphasizes the variety of levels from which the discipline

of computing can be viewed. Levels of architecture are unfolded through the introduction of finite automata, digital logic, and microprogramming. Levels of languages are revealed through an examination of sequence control, type checking, runtime storage management, and nonprocedural (functional or logic) programming paradigms. Levels of applications are treated through a general introduction to the areas of database systems and artificial intelligence.

C_B203 is a four credit-hour course consisting of 37 lecture hours of knowledge units and related laboratories.

Prerequisites: C_B102 and Discrete Mathematics

Knowledge units: AR1 (4/12), AR2 (2/6), AI1 (3/3), AI2 (4/6), DB1 (4/4), PL4 (3/4), PL5 (2/4), PL11 (10/10), SE1 (5/16)

C_B204 Algorithms, Concurrency, and the Limits of Computation

Topic Summary: This course emphasizes theoretical aspects of computer science. A review of graph theory is followed by a study of basic graph algorithms—search and traversal, shortest path, connectedness, etc.

Computational complexity levels beyond $O(n^2)$, including the ideas of tractable, intractable, and unsolvable problems are introduced and illustrated. Complexity classes P, NP, and NC are also characterized along with the notions of Turing machines and Turing computability. Concurrency is also introduced here, along with its applications in algorithms, architectures, operating systems, distributed computing, and networks.

C_B204 is a four credit-hour course containing 39 lecture hours of KU topics and related laboratories.

Prerequisites: C_B203

Knowledge units: AL1 (5/13), AL4 (3/4), AL5 (4/4), AL7 (6/6), AL8 (6/6), AL9 (3/3), OS3 (3/4), OS4 (3/3), PL12 (3/3), SP3 (3/3)

C_B305 Language Formalisms and Software Methodology

Topic Summary: This course combines a study of language syntax and semantics with an in-depth treatment of the software design process. The latter emphasizes an object-oriented approach that is introduced and contrasted with traditional design methodologies. The design of a syntax directed interactive editor is used as a case study throughout this course.

C_B305 is a four credit-hour course that covers 36 lecture hours of KU topics and laboratories.

Prerequisites: C_B203

Knowledge units: OS3 (1/4), OS6 (2/2), PL7 (6/6), PL8 (4/4), PL10 (2/2), SE2 (6/8), SE3 (4/4), SE4 (8/8), SE5 (3/8)

C_B306 Intermediate Computer Architecture

Topic Summary: This course provides an in-depth treatment of computer architecture, including digital logic, digital systems, memory system organization, interfacing and communication, and alternative architectures.

C_B306 is best taught as a four credit-hour course taught in four lecture hours per week, and comprises 36 lecture hours of KU topics and laboratories.

Prerequisites: C_B204

Knowledge units: AR1 (8/12), AR2 (4/6), AR4 (8/15), AR5 (5/13), AR7 (5/5), OS9 (3/3), OS10 (3/3)

C_B307 Data Representation, Files, and Database Systems

Topic Summary: This course is a study of contemporary models and methodologies for representing, storing, and retrieving large quantities of information stored on external devices. Alternative views of data are seen from the perspectives of the system, the human interface, and the applications.

C_B307 is a four credit-hour course; 42 lecture hours are allocated to specific KU topics and their related laboratories.

Prerequisites: C_B203

Knowledge units: AR3 (1/3), AR5 (8/13), AR6 (5/5), AI2 (2/6), DB2 (5/5), HU2 (3/3), NU1 (2/3), NU2 (2/4), OS5 (4/4), OS7 (4/4), OS8 (3/3), SP2 (3/3)

C_B308 Software Engineering

Topic Summary: This course in software engineering gives the student significant experience in pragmatic software engineering. Students, working in teams, design, implement, and present a substantial software project. The course extends the fundamental treatment of software methodology found in C_B305 and grounds the methodologies used in practical experience.

Prerequisites: Prerequisites: C_B305

C_B309 Advanced Architecture and Operating Systems

Topic Summary:

This course focuses on alternative and special purpose architectures and operating systems. Parallel architectures are treated in depth, as are parallel algorithms and

distributed operating systems. In addition, special purpose architectures such as those developed for signal processing applications are discussed.

Prerequisites: Prerequisites: C_B306 and C_B307

C_B401–402 Design Laboratory

Topic Summary:

This two-course sequence represents a capstone design experience in the curriculum. Student laboratory work involves the specification, design, implementation, testing, and documentation of a system intended to solve a substantive problem. Appropriate laboratories generally involve development of a computing system with both hardware and software components. Written and oral reports are required.

Prerequisites: C_B305, C_B306, C_B307

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Chem 101	3	Chemistry I	Phys 101	3	Physics I
Math 101	4	Calculus I	Math 102	4	Calculus II
C_S101	4	Prob Solving, Programs, & Computers	C_S102	4	Abstraction, Data Struct, & Lg Softw Syst
Engl 100	3	English Composition	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	17			17	

Sophomore Year

First Semester			Second Semester		
Phys 201	3	Physics II	Sci El.	3	Science Elective
Math 201	3	Calculus III	Math 203	4	Linear Algebra and Diff. Equations
C_S203	4	Levels of Arch, Languages, & Applications	C_S204	4	Algorithms, Concurrency, & Limits of Comp.
Math 202	3	Discrete Structures	EE 201	3	Basic Circuits and Electronics
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	16			17	

Junior Year

First Semester			Second Semester		
C_S305	4	Language Formalisms, & Softw Methodology	C_S307	4	Data Representation Files, & Database Sys
C_S306	4	Intermediate Comp Arch	ES 301	3	Engineering Science (e.g., Thermo, Fields)
EE 301	3	Linear Systems	C_S308	3	Software Engineering
Math 301	3	Probability & Statistics	C_S309	3	Adv. Architecture & Operating Systems
Elect	3	Elective	HU/SS	3	Professional Ethics
	17			16	

Senior Year

First Semester			Second Semester		
C_S401	3	Design Laboratory	C_S402	3	Design Laboratory
HU/SS	3	HU/SS Elective	Prof. El.	3	Professional Elective
Prof. El.	3	Professional Elective	Prof. El.	3	Professional Elective
Prof. El.	3	Professional Elective	Prof. El.	3	Professional Elective
Prof. El.	3	Professional Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
	18			18	

The Professional Electives found in the table below are selected to develop an area of concentration in one of the subfields of computer science and engineering. The courses mentioned in the table are typical of what might be selected, but the specific elective courses offered by any institution will be a function of the current interests of the faculty associated with the program.

	Software Engineering	Computer System Design	Knowledge-Based Systems
Prof. Elective 1	Performance Analysis	Advanced Electronics	Database Systems
Prof. Elective 2	Database Systems	Computer Communications	Computer Graphics
Prof. Elective 3	Theory of Computing	Design Automation	Artificial Intelligence
Prof. Elective 4	Compiler Construction	Performance Analysis	Expert Systems
Prof. Elective 5	Verification & Validation	VLSI Design	Distributed Databases
Prof. Elective 6	Adv. Softw. Engineering	Fault Tolerant Computing	Decision Support Systems

Implementation $\mathcal{C}$: A Program in Computer Engineering (Minimal Number of Credit-Hours)

Goals and Features of this Program: This curriculum leads to the B.S. in Computer Engineering by packaging the common requirements and the overall degree requirements in a different way. Specifically, a variety of programming paradigms and problem solving strategies are covered earlier in this program of study than in the other implementations. In addition, it differs from Implementations $\mathcal{A}$ and $\mathcal{B}$ because it is designed to fit into an institutional environment that requires no more than 120 credit-hours for all programs.

This program's goals are to prepare students for a lifetime career in computing by establishing a foundation for lifelong learning and development. It also provides a platform for further work leading to careers in such fields as business, law, and medicine.

Summary of Requirements: This program of study is built around a set of twelve required courses in computer science and engineering, comprising 35 credit-hours of study. Flexibility is incorporated through judicious choice of electives in computer science and engineering as well as in other areas. The computer science and engineering segment of the curriculum (including professional electives) consists of 50 credit-hours of study.

This program of study requires 40 courses, with content credit-hours distributed as follows:

15	hours of physics and chemistry
19	hours of mathematics
21	hours of humanities and social sciences (to include English)
35	hours of required computer science and engineering
15	hours of computer science and engineering electives
9	hours of other engineering courses
6	hours of free electives
120	

Course Descriptions

C_C101 Introduction to Problem Solving using a Computer

Topic Summary: This course is an introduction to problem solving using a computer. If this is a first course in "computing", elementary language syntax should be integrated with a treatment of fundamental problem solving concepts. An introduction to the software development process is also included so that students begin to develop a

methodical approach to problem solving using a computer. Ethical issues are discussed in the context of personal responsibility and intellectual property.

C_C101 contains 32 lecture hours of KU topics. C_C101 is a three credit-hour course that can be taught in two lectures and one two hour laboratory-sized recitation per week. Over the semester, there are 28 available lectures, and 28 available scheduled laboratory hours. The laboratory hours may be used to introduce and handle problems encountered in any "first programming" course and also to treat some of the KU topics themselves. Thus there should be adequate time for inclusion of a treatment of syntactic issues if this is the first "programming course" for the students.

Prerequisites: None.

Knowledge units: AL1 (2/13), AL2 (1/2), HU1 (1/5), SE1 (16/16), SE2 (8/8), SP1 (1/3), SP2 (1/3), SP3 (1/3), SP4 (1/2)

C_C102 Introduction to Software Engineering

Topic Summary: This is a first course in software engineering, stressing the software life cycle, proper selection of data structures and algorithms, and availability and choice of programming paradigms and language features for appropriate design and implementation of well engineered software. In keeping with the philosophy that treatment of ethical issues should be incorporated in multiple courses, C_C102 includes discussion of these issues in the context of software engineering concepts.

C_C102 contains 31 lecture hours of KU topics, and is taught as a three credit-hour course. An open laboratory and significant programming experiences are an integral component of the course.

Prerequisites: C_C101

Knowledge units: AL1 (3/13), HU1 (1/5), PL1 (1/2), PL2 (1/2), PL3 (2/2), PL4 (3/4), PL5 (2/4), PL11 (3/10), SE3 (3/4), SE4 (8/8), SP1 (1/3), SP2 (2/3), SP3 (1/3)

C_C201 Numeric Computation and Problem Solving

Topic Summary: A course in numerical methods and software engineering, C_C201 treats such topics as iterative approximation methods, development of numerical algorithms, and program verification. The procedural programming paradigm is utilized, appropriate documentation procedures are stressed, and a significant portion of the course is devoted to the introduction to numerical analysis and linear algebra that is required. There is ample time available to introduce the requisite background in logic for a proper treatment of verification issues at this level.

C_C201 is a three credit-hour course, containing 21 lecture hours of KU topics and associated laboratories.

Prerequisites: C_C102 and Calculus

Knowledge units: NU1 (3/3), NU2 (4/4), PL11 (3/10), SE5 (8/8), SP1 (1/3), SP3 (1/3), SP4 (1/2)

C_C202 Computer Organization and Assembly Language Programming

Topic Summary: This course is a first course in computer organization and assembly language programming. Students are exposed to the register level architecture of a modern processor and gain experience programming in the assembly language for that processor. Topics associated with data representation, I/O devices, and bus transactions are treated in this course.

C_C202 is a three credit-hour course containing 29 lecture hours of KU topics and laboratories.

Prerequisites: C_C102

Knowledge units: AR1 (2/12), AR2 (6/6), AR3 (3/3), AR4 (13/15), AR5 (3/13), AR6 (2/5)

C_C203 Abstraction and System Modeling

Topic Summary: C_C203 is a course that uses symbolic computation as a vehicle for introducing issues related to data and procedural abstractions and modeling concepts. Students are exposed to the functional and object oriented programming paradigms, linguistic features that support these paradigms, and analysis of algorithms in the context of AI applications.

C_C203 is a three credit-hour course containing 37 lecture hours of KU topics and laboratories.

Prerequisites: C_C201

Knowledge units: AL1 (4/13), AL2 (1/2), AL3 (3/3), AL4 (2/4), AL5 (2/4), AI1 (3/3), AI2 (6/6), PL1 (1/2), PL2 (1/2), PL4 (1/4), PL5 (2/4), PL6 (4/4), PL11 (4/10), PL12 (3/3)

C_C301 Logic and Digital Circuits

Topic Summary: C_C301 is a traditional course in logic and digital circuits covering analysis and design of combinational and sequential circuits.

C_C301 is a three credit-hour course taught in three lecture hours per week. It contains 14 hours of KU topics and laboratories, but it goes significantly beyond the "common requirements" in its treatment of digital circuits.

Prerequisites: C_C202 and some background in electronics

Knowledge units: AR1 (10/12), PL7 (4/6)

C_C302 Design and Analysis of Algorithms

Topic Summary: C_C302 treats the design and analysis of algorithms, incorporating a discussion of complexity and decidability issues along with a further development of problem solving strategies and amplification of previous treatment of basic data structures. An introduction to database systems is included to provide a context for some of these issues.

C_C302 is a three credit-hour course. It is taught at the junior level, after the students have exposure to a variety of programming languages and problem solving paradigms so that some of the more theoretical issues can be treated more easily. The course treats 35 lecture hours of KU topics and associated laboratories.

Prerequisites: C_C203

Knowledge units: AL1 (4/13), AL4 (2/4), AL5 (2/4), AL6 (6/6), AL7 (6/6), AL8(6/6), DB1 (4/4), DB2 (5/5)

C_C303 Computer Architecture

Topic Summary: This is a course in computer architecture that treats alternatives in instruction set architectures, memory system organization, control design, parallel processing (architectures and algorithms), and operating systems concepts that are related to architectural principles.

C_C303 is a three credit-hour course taught in three lecture hours per week. There are 42 lecture hours available in the semester, of which 29 are allocated to specific KU areas and associated laboratories.

Prerequisites: C_C202, C_C203

Knowledge units: AL9 (3/3), AR4 (2/15), AR5 (10/13), AR7 (5/5), OS1 (2/3), OS2 (1/2), OS3 (2/4), OS5 (4/4)

C_C304 Software Systems

Topic Summary: C_C304 is a course that treats system software, including a discussion of language translators and operating systems.

C_C304 is a three credit-hour course taught in three lecture hours per week. There are 42 lecture hours available in the semester, of which 29 are allocated to specific KU areas and laboratories.

Prerequisites: C_C203, C_C303

Knowledge units: OS1 (1/3), OS2 (1/2), OS3 (2/4), OS4 (3/3), OS6 (2/2), OS7 (4/4), OS8 (3/3), OS10 (2/3), PL7 (2/6), PL8 (4/4), PL9 (3/3), PL10 (2/2)

C_C305 Computer Engineering Laboratory

Topic Summary: C_C305 is a three credit-hour laboratory course in which students perform exercises under supervision that are relevant to and reinforce concepts found in the following KU's: AR1, AR2, AR4, AR6, OS3, OS6, and OS7. Typically, these exercises would involve designing, building, and testing digital circuits using SSI and MSI components. Also included are the design and implementation of various operating system utilities, generally written in a Unix environment.

The course is taught in a four hour scheduled laboratory and a one hour lecture each week.

Prerequisites: C_C301, C_C303, and C_C304 (at least concurrently)

Knowledge units: exercises relevant to AR1, AR2, AR4, AR6, OS3, OS6, and OS7

C_C401 - 402 Digital Systems Design and Laboratory

Topic Summary: C_C401 and 402 provide a course in small system design and its associated laboratory. Students generally work from the requirements stage through system specification, design, and implementation. Upon completion of these two courses, students have designed and implemented and tested the hardware and software components of an embedded system.

C_C401 and 402 constitute a six credit-hour course, containing 14 lecture hours of KU topics and labooratories. It can be taught in one semester to give the students a very intense design experience, or it can be spread over two semesters to allow more time to complete the design. Students work in teams, with formal reports required at regular intervals.

Prerequisites: C_C301, 304

Knowledge units: AR6 (3/5), HU1 (3/5), HU2 (3/3), OS9 (3/3), OS10 (1/3), SE3 (1/4)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Chem 101	3	Chemistry I	Chem 103	3	Chemistry II
Phys 101	3	Physics I	Phys 102	3	Physics II
Math 101	3	Calculus I	Math 102	3	Calculus II
C_C100	3	Intro to Prob Solv w/ Comput	C_C101	3	Intro to Software Engr
Engl 100	3	English I	HU/SS	3	HU/SS Elective
	15			15	

Sophomore Year

First Semester			Second Semester		
Math 201	3	Diff Equ.	Math 202	3	Calculus III
EE 201	3	Basic Circuit Theory	EE 202	3	Instrumentation Laboratory
C_C201	3	Numeric Comp & Prob Solving	EE 204	3	Analog & Digital Circuits
C_C202	3	Comp Org & Assy Language	Phys 201	3	Modern Physics of Devices
HU/SS	3	HU/SS Elective	C_C203	3	Abstractions & System Modeling
	15			15	

Junior Year

First Semester			Second Semester		
Math 301	3	Discrete Math	Math 302	3	Probability & Statistics
C_C301	3	Logic & Digital Circuits	C_C304	3	Software Systems
C_C302	3	Design & Analysis of Algorithms	C_C305	3	Comp Eng Lab
C_C303	3	Comp Architecture	Prof El.	3	Professional Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	15			15	

Senior Year

First Semester			Second Semester		
C_C401	3	Digital Systems Design	Prof El.	3	Professional Elective
C_C402	3	Digital Systems Design Lab	Prof. El.	3	Professional Elective
Prof El.	3	Professional Elective	HU/SS El.	3	HU/SS Elective
Prof El.	3	Professional Elective	Elect	3	Elective
HU/SS	3	HU/SS Elective	Elect	3	Elective
	15			15	

The Professional Electives found in the table below are selected to develop an area of concentration in one of the subfields of computer science and engineering. The courses mentioned in the table are typical of what might be selected, but the specific elective courses offered by any institution will be a function of the current interests of the faculty associated with the program.

	Software Engineering	Computer System Design	Knowledge-Based Systems
Prof. Elective 1	Performance Analysis	Advanced Electronics	Database Systems
Prof. Elective 2	Database Systems	Computer Communications	Computer Graphics
Prof. Elective 3	Theory of Computing	Design Automation	Artificial Intelligence
Prof. Elective 4	Compiler Construction	Performance Analysis	Expert Systems
Prof. Elective 5	Verification & Validation	VLSI Design	Distributed Databases
Prof. Elective 6	Adv. Softw. Engineering	Fault Tolerant Computing	Decision Support Systems

Implementation $\mathcal{D}$: A Program in Computer Science

Goals and Features of this Program: This program leads to the B.S. in Computer Science through a design-oriented selection of courses. It incorporates an early emphasis on software and hardware design followed by later courses in programming languages, operating systems, and computer architecture. Examples of alternative tracks for advanced specialization are provided in the areas of software engineering, knowledge-based systems, and multiprocessing.

This program prepares students for a career in computing, providing a foundation for lifelong learning through professional development and/or graduate study.

Summary of Requirements: This program of study is built around a set of ten (10) required courses in computer science , comprising 30 credit-hours of study. Flexibility is incorporated through judicious choice of electives in computer science as well as in other areas. The computer science segment of the curriculum (including professional electives) consists of 48 credit-hours of study.

This program of study requires 40 courses, with credit-hours distributed as follows:

14	hours of science
20	hours of mathematics
30	hours of humanities and social sciences (to include English composition)
30	hours of required computer science
18	hours of computer science electives
12	hours of free electives
124	

Course Descriptions

$C_{\mathcal{D}}$101,102 Introduction to Computing I, II

Topic Summary: This set of two courses comprise a two semester course that introduces algorithmic problem solving. In the context of a modern programming language, such topics as problem solving strategies, basic data structures, data and procedural abstraction, and algorithm complexity are discussed. Examples involving searching and sorting techniques and elementary numerical analysis are used to motivate students. Students hone their problem solving skills in the programming laboratory associated with the course. They also gain experience in the laboratory with the user interfaces of a variety of systems.

$C_{\mathcal{D}}$101-102 contains 39 lecture hours of KU topics. Each of $C_{\mathcal{D}}$101 and $C_{\mathcal{D}}$102 is a three credit-hour courses taught in two lectures and one two hour laboratory-sized

recitation per week. Thus there is ample time for addition of the PR knowledge unit if this is the first programming course for students.

Prerequisites: none; Calculus and/or Discrete Mathematics concurrently

Knowledge units: AL1 (5/13), AL2 (1/2), AL3 (1/3), AL4 (1/4), AL6 (3/6), AL8 (2/6), HU1 (2/5), NU1 (3/3), NU2 (4/4), PL1 (2/2), PL4 (2/4), SE1 (4/16), SE2 (1/8), SE3 (1/4), SE4 (2/8), SE5 (1/8), SP1 (2/3), SP2 (1/3), SP4 (1/2)

C_D201 Introduction to Computer Systems

Topic Summary: This is a first course in digital systems, including a treatment of logic and digital circuits as well as design using register level components. Data representation, device characteristics, and register transfer notation are covered in a manner that stresses application of basic problem solving techniques to both hardware and software design. Requirements specification, the design process, and issues associated with use of graphical interfaces are also discussed.

C_D201 is taught as a three credit-hour course, in which 25 hours of lectures and laboratories are identified with specific KU areas.

Prerequisites: C_D102

Knowledge units: AR1 (12/12), AR2 (4/6), AR3 (2/3), HU1 (2/5), SE1 (2/16), SE2 (1/8), SE3 (1/4), SE5 (1/8),

C_D202 Analysis and Design of Algorithms

Topic Summary: A course in algorithms, C_D203 treats such topics as appropriate choice of data structures, recursive algorithms, complexity issues, and issues associated with computability and decidability. Intractable problems, such as those found in artificial intelligence, are discussed. An introduction to parallel algorithms is also included.

C_D202 is taught as a three credit hour course, in which 34 lecture hours are identified with specific KU areas.

Prerequisites: C_D201, Discrete Mathematics

Knowledge units: AL1 (8/13), AL2 (1/2), AL3 (2/3), AL4 (3/4), AL5 (4/4), AL6 (3/6), AL7 (3/6), AL8 (1/6), AI1 (3/3), AI2 (6/6)

C_D301 Computer Organization and Assembly Language Programming

Topic Summary: This course is a first course in computer organization and assembly language programming. Students are exposed to the register level architecture of a modern processor and gain experience programming in the assembly language for that

processor. Topics associated with data representation, I/O devices, and bus transactions that have been previously mentioned are reinforced and amplified.

C_D301 is a three credit-hour course containing 21 lecture hours of KU topics and associated laboratories.

Prerequisites: C_D201

Knowledge units: AR2 (2/6), AR3 (1/3), AR4 (12/15), AR5 (6/13)

C_D302 Software Systems

Topic Summary: A course in software systems, C_D302 stresses problem solving strategies and concepts applied in the context of issues associated with the design and implementation of software systems. Students gain an appreciation for intractable problems as well as an exposure to concurrent systems. Levels of abstraction are emphasized in data modeling and mapping to storage structures, and a treatment of user interfaces is included.

C_D302 is a three credit-hour course with a significant software development component. Students spend a large portion of their time designing and implementing small and medium sized software systems, and gaining experience with various environments. The course has 30 lecture hours and laboratories associated with specific KU topics.

Prerequisites: C_D202

Knowledge units: AL7 (3/6), AL8 (3/6), AL9 (3/3), DB1 (3/4), DB2 (5/5), HU2 (3/3), PL2 (2/2), PL3 (2/2), SE1 (1/16), SE2 (1/8), SE3 (1/4), SE4 (1/8), SE5 (1/8)

C_D303 Software Engineering

Topic Summary: This course in software engineering treats topics associated with the design and implementation of large software systems. A continued emphasis on problem solving concepts is integrated with a treatment of the software life cycle, requirements specifications, and verification and validation issues. Social and ethical issues faced by the computing professional are discussed in the context of software engineering.

C_D303 is a three credit-hour course taught in three lecture hours per week. As it is taught after the students have considerable exposure to the design and implementation of small and medium scale software projects, the software engineering concepts are presented in the context of their prior experience, enabling the students to appreciate them more fully. Specific KU areas account for 32 lecture hours and associated laboratories.

Prerequisites: C_D302

Knowledge units: SE1 (9/16), SE2 (5/8), SE3 (1/4), SE4 (5/8), SE5 (5/8), SP1 (1/3), SP2 (2/3), SP3 (3/3), SP4 (1/2)

C_D304 Programming Languages

Topic Summary: C_D304 is a course in programming languages that treats language design issues and language translators.

C_D304 is a three credit-hour course taught in three lecture hours per week. It is taught after students have had exposure to a variety of programming languages and problem solving paradigms, so that linguistic issues and programming paradigms can be treated at a more advanced level. It has 38 lecture hours and laboratories allocated to specific KU areas.

Prerequisites: C_D301

Knowledge units: PL4 (2/4), PL5 (4/4), PL6 (4/4), PL7 (6/6), PL8 (4/4), PL9 (3/3), PL10 (2/2), PL11 (10/10), PL12 (3/3)

C_D305 Operating Systems

Topic Summary: This is a course in systems software that is largely concerned with operating systems. Such topics as process management, device management, and memory management are discussed, as are relevant issues associated with security and protection, networking, and distributed operating systems.

C_D305 is a three credit-hour course taught in three lecture hours per week, with 33 lecture hours and laboratories allocated to specific KU areas.

Prerequisites: C_D301, C_D302

Knowledge units: DB1(1/4), HU1 (1/5), OS1 (3/3), OS2 (2/2), OS3 (4/4), OS4 (3/3), OS5 (4/4), OS6 (2/2), OS7 (4/4), OS8 (3/3), OS9 (3/3), OS10 (3/3)

C_D306 Computer Architecture

Topic Summary: This is a course that deals with design alternatives in computer architecture. Instruction set architectures, memory subsystem organization, interfacing concepts and issues arising in managing communication with the processor are covered, as are a number of alternative computer architectures.

C_D306 is a three credit-hour course taught in three lecture hours per week, with 20 lecture hours and laboratories allocated to specific KU areas.

Prerequisites: C_D301, C_D302

Knowledge units: AR4 (3/15), AR5 (7/13), AR6 (5/5), AR7 (5/5)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Phys 101	4	Physics I	Phys 102	4	Physics II
Math 101	4	Calculus I	Math 102	4	Calculus II
C_D101	3	Introduction to Computing I	C_D102	3	Introduction to Computing II
Engl 100	3	English Composition	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	17			17	

Sophomore Year

First Semester			Second Semester		
Sci El.	3	Science Elective	Sci El.	3	Science Elective
Math 201	3	Calculus III	Math 202	3	Linear Algebra
C_D201	3	Introduction to Computer Systems	C_D202	3	Design & Analysis of Algorithms
Math 203	3	Discrete Math	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	15			15	

Junior Year

First Semester			Second Semester		
C_D301	3	Computer Org and Assembly Lang	C_D303	3	Software Engineering
C_D302	3	Software Systems	C_D304	3	Programming Languages
Math 301	3	Probability & Statistics	SO 301	3	Professional Ethics
CS El.	3	CS Elective	CS El.	3	CS Elective
Elect	3	Elective	HU/SS	3	HU/SS Elective
	15			15	

Senior Year

First Semester			Second Semester		
C_D305	3	Operating Systems	C_D306	3	Computer Architecture
CS El.	3	CS Elective	CS El.	3	CS Elective
CS El.	3	CS Elective	CS El.	3	CS Elective
HU/SS	3	HU/SS Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
	15			15	

The computer science electives found in the table below are selected to develop an area of concentration in one of the subfields of computer science. The courses mentioned in the table are typical of what might be selected, but the specific elective courses offered by any institution will be a function of the current interests of the faculty associated with the program.

	Software Engineering	Knowledge-Based Systems	Multiprocessing
CS Elective 1	Performance Analysis	Database Systems	Adv. Operating Systems
CS Elective 2	Database Systems	Computer Graphics	Computer Comm. Networks
CS Elective 3	Theory of Computing	Artificial Intelligence	Computer Security
CS Elective 4	Compiler Construction	Natural Language	Fault-Tolerant Computing
CS Elective 5	Verification and Validation	Distributed Databases	Parallel & Dist. Computing
CS Elective 6	Software Eng'g Design	Expert Systems Design	Computer Syst. Design

Implementation $\mathcal{E}$: A Program in Computer Science (Breadth-First)

Goals and Features of this Program: This program leads to the B.S. in Computer Science in a way similar to Implementation $\mathcal{D}$, but with one major exception: the common requirements are presented in such a way that the breadth of the discipline is covered early in an undergraduate's program of study rather than late. Specifically, the common requirements are completely covered in a set of seven courses, and the first four courses provide exposure to all nine subject areas of the discipline along with the area of social and professional issues. While these are the same seven courses that appear in implementation $\mathcal{B}$, the balance of this degree program has fewer computer science requirements and more free electives than implementation $\mathcal{B}$.

This program prepares students for a career in computing, providing a foundation for lifelong learning through professional development and/or graduate study.

Summary of Requirements: This program of study is built around a set of seven (7) required courses in computer science, comprising 28 credit-hours of study. Flexibility is incorporated through judicious choice of electives in computer science as well as in other areas.

This program requires 39 courses, with credit-hours distributed as follows:

12	hours of science
21	hours of mathematics
36	hours of humanities and social sciences (to include English composition)
28	hours of required computer science
15	hours of computer science electives
15	hours of free electives
127	

Course Descriptions

C$\mathcal{E}$101 Problem Solving, Programs, and Computers

Topic Summary: This course has three major themes: a rigorous introduction to the process of algorithmic problem solving, an introduction to the organization of the computers upon which the resulting programs run, and an overview of the social and ethical context in which the field of computing exists. Problem solving rigor is guaranteed by a commitment to the precise specification of problems and a close association between the problem solving process and that specification. For example, the identification of a loop is closely tied to the discovery of its invariant, which in turn is motivated by the problem specification itself.

Computer organization is introduced by way of a simple von Neumann machine model and assembler, upon which students can develop and exercise simple programs. Elements of the fetch-execute cycle, runtime data representation, and machine language program structure are thereby revealed.

Cε101 contains 40 lecture hours of KU topics. The course is taught as a four credit-hour course. A scheduled weekly laboratory is used to teach the necessary high-level syntax and machine organization, as well as to support student programming exercises.

Prerequisites: An introduction to logic, as would usually be found at the beginning of a Discrete Mathematics course (that can be taken concurrently)

Knowledge units: AL3 (3/3), AL6 (2/6), AR3 (2/3), AR4 (7/15), NU1 (1/3), NU2 (2/4), PL1 (2/2), PL3 (2/2), PL4 (1/4), SE1 (11/16), SE5 (4/8), SP1 (3/3)

Cε102 Abstraction, Data Structures, and Large Software Systems

Topic Summary:

This course introduces the notion of a large software system and the principles and methodologies that accompany its effective realization. A simple operating system model is used throughout the course as a motivational case study.

The idea of an abstract data type is first introduced, and then reinforced through the characterization of fundamental data structures in the discipline – stacks, queues, and trees. Computational complexity is introduced and motivated through the design and analysis of two or three sorting algorithms. Other design issues are also explore, such as the use of dynamic storage for implementing an ADT. An overview of a compiler as a large software system is given as a case study.

Cε102 is a four credit-hour course that covers 41 lecture hours associated with knowledge units. Laboratories for the course are of two types: closed sessions in which students work individually and open sessions in which students work in teams to solve a larger software problem than those which were introduced in course Cε101.

Prerequisites: Cε101

Knowledge units: AL1 (8/13), AL2 (2/2), AL4 (1/4), AL6 (4/6), HU1 (5/5), OS1 (3/3), OS2 (2/2), PL2 (2/2), PL5 (2/4), PL6 (4/4), PL9 (3/3), SE2 (2/8), SE5 (1/8), SP4 (2/2)

Cε203 Levels of Architecture, Languages, and Applications

Topic Summary: This course emphasizes the variety of levels from which the discipline of computing can be viewed. Levels of architecture are unfolded through the introduction of finite automata, digital logic, and microprogramming. Levels of languages are revealed through an examination of sequence control, type checking, runtime storage

management, and nonprocedural (functional or logic) programming paradigms. Levels of applications are treated through a general introduction to the areas of database systems and artificial intelligence.

C$\mathcal{E}$203 is a four credit-hour course consisting of 37 lecture hours of knowledge units and their laboratories.

Prerequisites: C$\mathcal{E}$102 and Discrete Mathematics

Knowledge units: AR1 (4/12), AR2 (2/6), AI1 (3/3), AI2 (4/6), DB1 (4/4), PL4 (3/4), PL5 (2/4), PL11 (10/10), SE1 (5/16)

C$\mathcal{E}$204 Algorithms, Concurrency, and the Limits of Computation

Topic Summary: This course emphasizes theoretical aspects of computer science. A review of graph theory is followed by a study of basic graph algorithms—search and traversal, shortest path, connectedness, etc.

Computational complexity levels beyond $O(n^2)$, including the ideas of tractable, intractable, and unsolvable problems are introduced and illustrated. Complexity classes P, NP, and NC are also characterized along with the notions of Turing machines and Turing computability.

Concurrency is also introduced here, along with its applications in algorithms, architectures, operating systems, distributed computing, and networks.

C$\mathcal{E}$204 is a four credit-hour course containing 39 lecture hours of KU topics and associated laboratories.

Prerequisites: C$\mathcal{E}$203

Knowledge units: AL1 (5/13), AL4 (3/4), AL5 (4/4), AL7 (6/6), AL8 (6/6), AL9 (3/3), OS3 (3/4), OS4 (3/3), PL12 (3/3), SP3 (3/3)

C$\mathcal{E}$305 Language Formalisms and Software Methodology

Topic Summary: This course combines a study of language syntax and semantics with an in-depth treatment of the software design process. The latter emphasizes an object-oriented approach which is introduced and contrasted with traditional design methodologies. The design of a syntax directed interactive editor is used as a case study throughout this course.

C$\mathcal{E}$305 is a four credit-hour course that is taught in three lecture hours and one scheduled laboratory per week. The course treats 36 lecture hours of KU topics.

Prerequisites: C$\mathcal{E}$203

Knowledge units: OS3 (1/4), OS6 (2/2), PL7 (6/6), PL8 (4/4), PL10 (2/2), SE2 (6/8), SE3 (4/4), SE4 (8/8), SE5 (3/8)

$C_{\mathcal{E}}306$ Intermediate Computer Architecture

Topic Summary: This course provides an in-depth treatment of computer architecture, including digital logic, digital systems, memory system organization, interfacing and communication, and alternative architectures.

$C_{\mathcal{E}}306$ is best taught as a four credit-hour course taught in four lecture hours per week. It comprises 36 lecture hours of KU topics and their laboratories.

Prerequisites: $C_{\mathcal{E}}204$

Knowledge units: AR1 (8/12), AR2 (4/6), AR4 (8/15), AR5 (5/13), AR7 (5/5), OS9 (3/3), OS10 (3/3)

$C_{\mathcal{E}}307$ Data Representation, Files, and Database Systems

Topic Summary: This course is a study of contemporary models and methodologies for representing, storing, and retrieving large quantities of information stored on external devices. Alternative views of data are seen from the perspectives of the system, the human interface, and the applications.

$C_{\mathcal{E}}307$ is a four credit-hour course containing 42 lecture hours of specific KU topics and their laboratories.

Prerequisites: $C_{\mathcal{E}}203$

Knowledge units: AR3 (1/3), AR5 (8/13), AR6 (5/5), AI2 (2/6), DB2 (5/5), HU2 (3/3), NU1 (2/3), NU2 (2/4), OS5 (4/4), OS7 (4/4), OS8 (3/3), SP2 (3/3)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Chem 101	3	Chemistry I	Phys 101	3	Physics I
Math 101	3	Calculus I	Math 102	3	Calculus II
$C_{\mathcal{E}}$101	4	Prob Solving, Programs, & Computers	$C_{\mathcal{E}}$102	4	Abstraction, Data Structures, & Lg Soft Syst
Engl 100	3	English Composition	Math 103	3	Discrete Struct. I
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	16			16	

Sophomore Year

First Semester			Second Semester		
Phys 201	3	Physics II	Sci El.	3	Science Elective
Math 201	3	Calculus III	Math El.	3	Math Elective
$C_{\mathcal{E}}$203	4	Levels of Arch, Languages, & Applications	$C_{\mathcal{E}}$204	4	Algorithms, Concurrency, & Limits of Comp.
Math 202	3	Discrete Struct. II & Linear Algebra	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	16			16	

Junior Year

First Semester			Second Semester		
$C_{\mathcal{E}}$305	4	Language Formalism & Softw. Methodology	$C_{\mathcal{E}}$307	4	Data Representation, Files, Database Syst.
$C_{\mathcal{E}}$306	4	Intermediate Comp Architecture	CS Elect	3	CS Elective
Math 301	3	Probability & Statistics	SO 301	3	Ethics
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	17			16	

Senior Year

First Semester			Second Semester		
CS Elec.	3	CS Elective	CS Elec.	3	CS Elective
CS Elec.	3	CS Elective	CS Elec.	3	CS Elective
HU/SS	3	HU/SS Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
	15			15	

The computer science electives found in the table below are selected to develop an area of concentration in one of the subfields of computer science. The courses mentioned in the table are typical of what might be selected, but the specific elective courses offered by any institution will be a function of the current interests of the faculty associated with the program.

	Software Engineering	Knowledge-Based Systems	Multiprocessing
CS Elective 1	Performance Analysis	Database Systems	Adv. Operating Systems
CS Elective 2	Database Systems	Computer Graphics	Computer Comm. Networks
CS Elective 3	Theory of Computing	Artificial Intelligence	Computer Security
CS Elective 4	Compiler Construction	Expert Systems	Fault-Tolerant Computing
CS Elective 5	Verification & Validation	Distributed Databases	Parallel & Dist. Computing
CS Elective 6	Software Eng'g Design Project	Decision Support Systems	Performance Prediction & Analysis

Implementation $\mathcal{F}$: A Program in Computer Science (Theoretical Emphasis)

Goals and Features of this Program: This program of study has a more theoretical focus, especially in its intermediate and elective courses, than Implementations $\mathcal{A}$–$\mathcal{E}$. It leads to the B.S. in Computer Science through a balanced treatment of theory, abstraction, design, and the social and professional context of the discipline. For example, the capstone course "Software Systems and Professional Practice" ties together the formal concerns of software design and the social and ethical issues confronting computing professionals.

This program prepares students for a career in computing, providing a foundation for lifelong learning through professional development and/or graduate study.

Summary of Requirements: This program is built around a set of nine required courses in Computer Science, comprising 27 credit-hours of study. In addition, students are required to take five electives in Computer Science plus six courses in mathematics, including Discrete Mathematics and a course on methods of proof (inductive proofs, constructive proofs, diagonalization arguments, etc.). Furthermore, students are required to take three four credit laboratory science courses.

This program of study requires 120 credit-hours, which are distributed as follows:

12	hours of science
23	hours of mathematics
33	hours of humanities and social sciences (including English composition)
27	hours of required computer science
15	hours of computer science electives
12	hours of free electives
122	

Course Descriptions

$C_{\mathcal{F}}101$ Computing I

Topic Summary: This course introduces the student to the basic concepts of the discipline of computing, emphasizing elementary facts concerning computer architecture, programming languages, software methodology, and algorithms. Considerable time is devoted to learning how to solve problems using an appropriate programming language. Basic principles of program design and implementation are introduced. Abstract data types, sorting and searching are treated at an elementary level.

$C_{\mathcal{F}}101$ is a three credit-hour course taught in two lectures and one two hour lab per week. $C_{\mathcal{F}}101$ contains 42 lecture hours of which 36 are devoted to KU topics.

Prerequisites: None

Knowledge units: AL2 (1/2), AL6 (3/6), AR4 (3/15), PL3 (1/2), PR (12/12), SE1 (16/16)

$C_{\mathcal{F}}102$ Computing II

Topic Summary: This course continues the development of basic topics in computer organization, programming languages and software methodology. It introduces some elementary numerical methods as an application of the above. It introduces the society and professionalism subject area. Specific topics include further exposure to basic data structures (such as linked lists), an introduction to recursion and the machine level representation.

$C_{\mathcal{F}}102$ is a three credit-hour course taught in three lectures and one two hour lab per week. $C_{\mathcal{F}}102$ contains 42 lecture hours of which 35 are devoted to KU topics.

Prerequisites: $C_{\mathcal{F}}101$

Knowledge units: AL1 (4/13), AL2 (1/2), AL3 (1/3), AR3 (3/3), NU1 (3/3), NU2 (4/4), PL1 (2/2), PL2 (2/2), PL3 (1/2), PL4 (1/4), PL5 (2/4), SE2 (8/8), SP1 (3/3)

$C_{\mathcal{F}}203$ Computer Architecture

Topic Summary: This course introduces the fundamentals of digital logic, digital systems and digital systems design. Beginning with the fundamental building blocks of digital systems at the digital logic level, this course emphasizes different levels of integration. A considerable amount of time is devoted to memory system organization and architecture, including the physical implementation of large memory systems.

$C_{\mathcal{F}}203$ is a three credit-hour course, in which 31 lecture hours are devoted to KU topics and laboratories.

Prerequisites: $C_{\mathcal{F}}102$

Knowledge units: AR1 (12/12), AR2 (6/6), AR5 (13/13)

$C_{\mathcal{F}}204$ Data Structures and Analysis of Algorithms

Topic Summary: This course covers data structures and algorithms in some depth. Topics covered include data structures (developed in more depth than in $C_{\mathcal{D}}101$ and $C_{\mathcal{F}}102$), a more formal treatment of recursion, an introduction to basic problem solving strategies, an introduction to complexity analysis complexity classes and an introduction to the theory of computability and undecidability. Sorting and searching algorithms are

presented in the light of the presentation of problem-solving strategies and complexity issues. Finally, parallel and distributed algorithms are introduced briefly.

$C_{\mathcal{F}}204$ is a three credit-hour course taught in three lectures and one two hour lab per week. $C_{\mathcal{F}}204$ contains 42 lecture hours of which 33 are devoted to KU topics and laboratories.

Prerequisites: $C_{\mathcal{F}}102$

Knowledge units: AL1 (9/13), AL3 (2/3), AL4 (4/4), AL5 (4/4), AL6 (3/6), AL7 (3/6), AL8 (6/6), AL9 (2/3)

$C_{\mathcal{F}}305$ Computer Systems and Interfaces

Topic Summary: This course introduces assembly level machine organization and assembly language programming. Topics include instruction sets and types, addressing modes, input/output and interrupts, modularity, partitioning and redundancy. This is followed by the treatment of interfacing and communications and alternate architectures. The user interface and elementary topics in computer-human interaction are then discussed.

$C_{\mathcal{F}}305$ is a three credit-hour course taught in two lectures per week. In addition, there is one two hour lab per week. $C_{\mathcal{F}}305$ contains 28 lecture hours, of which 27 are devoted to KU topics and laboratories.

Prerequisites: $C_{\mathcal{F}}203$

Knowledge units: AR4 (12/15), AR6 (5/5), AR7 (5/5), HU1 (5/5)

$C_{\mathcal{F}}306$ Principles of Programming Languages

Topic Summary: This course introduces a wide range of topics relating to programming languages with an emphasis on abstraction and design. Design issues relevant to the implementation of programming languages are discussed, including a review and more in-depth treatment of mechanisms for sequence control, data structure implementation and run-time storage management. The major programming paradigms are surveyed, including important paradigms used in artificial intelligence, along with a discussion of their applications in artificial intelligence. Finally, language constructs that support distributed and parallel computing are introduced and these constructs are applied to problems in parallel and distributed computing.

$C_{\mathcal{F}}306$ is a three credit-hour course taught in three lectures per week. $C_{\mathcal{F}}306$ contains 42 lecture hours of which 30 are devoted to KU topics.

Prerequisites: $C_{\mathcal{F}}204$

Knowledge units: AL9 (1/3), AI1 (1/3), AI2 (6/6), PL4 (3/4), PL5 (2/4), PL6 (4/4), PL11 (10/10), PL12 (3/3)

C$\mathcal{F}$307 Operating Systems & File Organization

Topic Summary: This course provides a thorough introduction to operating systems and file organization. Operating systems topics include tasking and processes, processes coordination and synchronization, physical and virtual memory organization, communications and networking, distributed operating systems and real-time concerns. File systems are introduced, leading to a treatment of database systems with an emphasis on file organization techniques.

C$\mathcal{F}$307 is a three credit-hour course taught in three lectures per week. C$\mathcal{F}$307 contains 42 lecture hours of which 37 are devoted to KU topics and laboratories.

Prerequisites: C$\mathcal{F}$204, C$\mathcal{F}$305

Knowledge units: DB1 (3/4), DB2 (5/5), OS1 (3/3), OS2 (2/2), OS3 (4/4), OS4 (3/3), OS5 (4/4), OS6 (2/2), OS7 (4/4), OS8 (1/3), OS9 (3/3), OS10 (3/3)

C$\mathcal{F}$308 Theory of Computation

Topic Summary: This course introduces the theory of computability, including important results from the study of automata and formal languages. The course begins with a discussion of automata and their relationship to regular, context free and context-sensitive languages. General theories of computability are presented, including Turing machines, recursive functions and lambda calculus. Notions of decidability and undecidability are discussed and this is related to complexity analysis. Finally, approaches to formal program semantics are presented and analyzed, leading to a brief introduction to the topic of formal program verification.

C$\mathcal{F}$308 is a three credit-hour course taught in three lectures per week. It contains 42 lecture hours, 21 of which are devoted to KU topics.

Prerequisites: C$\mathcal{F}$204, Discrete Mathematics

Knowledge units: AL7 (3/6), PL7 (6/6), PL8 (4/4), PL9 (3/3), PL10 (2/2), SE5 (3/8)

C$\mathcal{F}$309 Software Systems and Professional Practice

Topic Summary: This course provides a rigorous introduction to software methodology and engineering. The emphasis is on rigorous techniques for software specification and design, verification and validation. An ambitious application is introduced, according to the interests of the instructor. This application should also serve as a bridge to a discussion of ethical and societal issues, which are discussed in depth. Additional ethical and society issues are brought out in a discussion of the nature of artificial intelligence. Operating system and database issues relating to security and protection are introduced.

C$\mathcal{F}$309 is a three credit-hour course taught in three lectures per week. It contains 42 lecture hours of which 33 are devoted to KU topics and laboratories.

Prerequisites: C$\mathcal{F}$306, C$\mathcal{F}$307

Knowledge units: AI1 (2/3), DB1 (1/4), HU2 (3/3), OS8 (2/3), SE3 (4/4), SE4 (8/8), SE5 (5/8), SP2 (3/3), SP3 (3/3), SP4 (2/2)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Math 101	4	Discrete Math	Math 102	4	Methods of Proof
$C_{\mathcal{D}}101$	3	Computing I	$C_{\mathcal{D}}102$	3	Computing II
HU/SS	3	HU/SS Elective	Sci. El.	3	Science Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
Elect	3	Elective	Elect	3	Elective
	16			16	

Sophomore Year

First Semester			Second Semester		
Math 201	3	Calculus I	Math 202	3	Calculus II
$C_{\mathcal{F}}203$	3	Computer Architecture	$C_{\mathcal{F}}305$	3	Computer Systems & Interfaces
$C_{\mathcal{F}}204$	3	Data Structures & Analysis of Algorithms	$C_{\mathcal{F}}306$	3	Principles of Programming Languages
Sci. El.	3	Science Elective	Sci. El.	3	Science Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	15			15	

Junior Year

First Semester			Second Semester		
Math 301	3	Linear Algebra	Math 302	3	Prob & Statistics
$C_{\mathcal{F}}307$	3	Operating Systems & File Organization	$C_{\mathcal{F}}308$	3	Theory of Computation
CS El.	3	CS Elective	$C_{\mathcal{F}}309$	3	Software Systems & Professional Practice
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
Sci. El.	3	Science Elective	HU/SS	3	HU/SS Elective
	15			15	

Senior Year

First Semester			Second Semester		
CS El.	3	CS Elective	CS El.	3	CS Elective
CS El.	3	CS Elective	CS El.	3	CS Elective
Math El.	3	Mathematics Elective	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
Elect	3	Elective	Elect	3	Elective
	15			15	

Advanced Courses Any of the following courses can be offered as advanced electives to complete the major.

- Advanced Operating Systems
- Advanced Software Design
- Artificial Intelligence
- Database Principles
- Semantics and Verification
- Programming Language Translation
- Networks and Distributed Computing
- Computer Graphics
- Parallelism and Concurrency
- Simulation
- Numerical and Symbolic Computation

Implementation $\mathcal{G}$: A Program in Computer Science (Software Engineering Emphasis)

Goals and Features of this Program: This program leads to the B.S. in Computer Science in such a way that students concentrate their studies on the principles and applications of software engineering. The foundation of this curriculum is a sequence of fundamental courses in software engineering. The rest of the curriculum emphasizes software development, large software systems design, programming paradigms, operating systems, and a capstone software project.

This program prepares students for a career in computing, providing a foundation for lifelong learning through professional development and/or graduate study.

Summary of Requirements: This program of study requires 40 courses, with credit-hours distributed as follows:

12	hours of science electives
18	hours of mathematics
36	hours of humanities and social sciences (including 12 hours of English)
34	hours of required computer science
9	hours of computer science electives
21	hours of free electives
135	

Course Descriptions

The introductory sequence of two courses, Introduction to Software Engineering and Software Methodology, act as the cornerstone for this curriculum. Within these courses, students will be introduced to programming in the context of solid software engineering practices. No assumptions about prior programming experience are made, although prior experience with editors and program execution is useful.

Historically, the introductory course sequence has introduced a procedural programming language, although functional languages have also been popular. For this program, however, departments might choose to use an object-oriented language. In the first course, one might use an object-oriented analysis style, introduce the message-passing model of computation, and use internal implementation of objects to show data and procedures. This would leave inheritance until the second course, where students can begin using system-defined objects such as lists and arrays as the basis for developing their own ADT's.

C$\mathcal{G}$101- Introduction to Software Engineering

Topic Summary: This course introduces programming and software engineering. The

methodology may be based on functional analysis or object-oriented analysis as described above. Discussion of fundamental algorithms and data structures is included, focusing on ADT's throughout. User interfaces should be covered in the specification of programming tasks. The historical and social context of computing will be discussed, integrated with other course material.

This four credit-hour course contains 35 KU lecture hours and associated laboratories.

Prerequisites: None

Knowledge units: AL1 (7/13), AL2 (2/2), AL3 (3/3), AL6 (2/6), HU1 (2/5), SE1 (16/16), SP1 (3/3)

C_G102 Software Methodology

Topic Summary: Continuing in software engineering methodology with focus on lifecycles, analysis, specification, documentation of the process as well as documentation of the product, this course again will include ADT's and the beginning of careful analysis of some algorithms. The design of human/program interfaces continues in this course and performance concerns are introduced. An important social/professional topic is the discussion of intellectual property rights.

This course is a four credit-hour course, with 37 KU lecture hours and associated laboratories.

Prerequisites: C_G101

Knowledge units: AL1 (6/13), AL4 (4/4), AL6 (2/6), AR3 (3/3), AR5 (6/13), HU1 (3/5), HU2 (3/3), SE2 (8/8), SP4 (2/2)

C_G203 Software Development

Topic Summary: This course expands the software engineering instruction to larger projects, particularly addressing the problems of scaling up process and product. Software evolution and maintenance concerns are also emphasized. In this context, students should study the impact of calculation intensive computation, using and discussing particularly numerical analysis packages. Search strategies and other artificial intelligence techniques are introduced as large-problem approaches, along with other discussion the problem solving strategies from an algorithm analysis point of view.

This is a four credit-hour course, with 40 KU lecture hours and laboratories.

Prerequisites: C_G102

Knowledge units: AL8 (4/6), AI1 (3/3), AI2 (6/6), NU1 (3/3), NU2 (4/4), SE3 (4/4), SE4 (8/8), SE5 (8/8)

C_C204 Programming Paradigms

Topic Summary: Various programming languages are studied from three points of view:

1. the paradigms and models they express (e.g., the von Neumann machine)
2. the levels of abstraction they represent (e.g., assembly language, fourth generation)
3. the way they are defined and implemented (e.g., translators)

This course contains 39 KU lecture hours and associated laboratories.

Prerequisites: C_C102

Knowledge units: AR4 (6/15), PL1 (2/2), PL2 (2/2), PL3 (2/2), PL4 (4/4), PL5 (4/4), PL6 (4/4), PL9 (3/3), PL10 (2/2), PL11 (10/10)

C_C305 Software Systems

Topic Summary: This course looks at large scale applications problems, such as database problems, large expert systems, real-time, distributed and concurrent system problems. As much of the material on distributed and concurrent systems will presented in survey form and as a result, not include lab assignments, instructors can use this course for other lab experiences, such as practice in porting between systems or configurations.

This course contains 32 KU lecture hours and laboratories.

Prerequisites: C_C203

Knowledge units: AL9 (3/3), AR2 (2/6), AR5 (4/13), AR6 (3/5), AR7 (5/5), DB1 (4/4), DB2 (5/5), OS10 (3/3), PL12 (3/3)

C_C306 Formalisms and Computation

Topic Summary: The use of representation and manipulation and the application of formal systems in computation are the focus of this course. Logic, sets, algebra, complexity and computability will be discussed and exemplified. An introduction to digital logic and elementary digital systems is integrated with the theoretical treatment of these issues.

This course contains 40 KU lecture hours and laboratories.

Prerequisites: C_C204, Discrete Mathematics

Knowledge units: AL5 (4/4), AL6 (2/6), AL7 (6/6), AL8 (2/6), AR1 (12/12), AR2 (4/6), PL7 (6/6), PL8 (4/4)

C_G307 Operating Systems

Topic Summary: Organization and implementation of operating systems, including time and space management, control and robustness of a variety of systems are studied. While the assembly language used in C_G204 may have been simulated, in this course students should use a real assembly language and program real machine interaction.

This course contains 42 KU lecture hours and laboratories.

Prerequisites: C_G204

Knowledge units: AR4 (9/15), AR5(3/13), AR6 (2/5), OS1 (3/3), OS2 (2/2), OS3 (4/4), OS4 (3/3), OS5 (4/4), OS6 (2/2), OS7 (4/4), OS8 (3/3), OS9 (3/3)

C_G407, 408 Capstone Project Sequence

Topic Summary: This two-semester course presents the student with a strong experience in software engineering. Students, working in teams, investigate, design, implement and present to their classmates a significant software project. The project should solve a significant, complex and hopefully generalizable problem, dealing with constraints and trade-offs in the solution. The course includes study of a certain amount of project management concerns (planning, scheduling, assessing progress, recognizing hard parts of a problem, but not personnel management or specific leadership training). While students are working on their project, instructors will want to want to use lecture time to cover issues of professional responsibility, risks and liabilities. They also may want to use lecture time to cover such technical topics as international concerns, metrics and measurement methods, quality assurance processes and time-to-market economics.

This is a six credit-hour course, containing 6 KU lecture hours.

Prerequisites: C_G306

Knowledge units: SP2 (3/3), SP3 (3/3)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Engl 101	3	English I	Engl 102	3	English II
Math 101	4	Discrete Math I	Math 102	4	Discrete Math II
$C_{\mathcal{C}}$101	4	Introduction to Software Engineering	$C_{\mathcal{C}}$102	4	Software Methodology
Sci El.	3	Science Elective	Sci El.	3	Science Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	17			17	

Sophomore Year

First Semester			Second Semester		
Sci El.	3	Science Elective	Sci El.	3	Science Elective
Math 201	3	Math Topics for Computer Science	Math 203	4	Calculus
$C_{\mathcal{C}}$203	4	Software Development	$C_{\mathcal{C}}$204	4	Programming Paradigms
Engl El.	3	English Elective	Engl El.	3	English Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	16			17	

Junior Year

First Semester			Second Semester		
$C_{\mathcal{C}}$305	4	Software Systems	$C_{\mathcal{C}}$307	4	Operating Systems
$C_{\mathcal{C}}$306	4	Formalisms & Computation	CS El.	3	Computer Science Elective
Math El.	3	Math Elective	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
	17			16	

Senior Year

First Semester			Second Semester		
$C_{\mathcal{C}}$407	3	Software Eng'g Project I	$C_{\mathcal{C}}$408	3	Software Eng'g Project II
CS El.	3	CS Elective	CS El.	3	CS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
Elect	3	Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
	15			15	

Advanced Courses Any of the following courses can be offered as advanced electives to complete the major.

- Artificial Intelligence
- Database Principles
- Semantics and Verification
- Compiler Design
- Networks and Distributed Computing
- Computer Graphics
- Parallelism and Concurrency
- Simulation
- Numerical and Symbolic Computation

Implementation $\mathcal{H}$: A Liberal Arts Program in Computer Science (Breadth-First)

Goals and Features of this Program: This program leads to the B.S. in Computer Science by providing professional education within a liberal arts context. This program also covers the common requirements in such a way that the breadth of the discipline is covered early in an undergraduate's program of study rather than late.

Because of its liberal arts context, this program provides a well-rounded educational experience. It also prepares students for a career in computing, providing a foundation for lifelong learning through professional development and/or graduate study.

Summary of Requirements: This program of study is built around a set of seven (7) required courses in computer science, comprising 28 credit-hours of study. Flexibility is incorporated through judicious choice of electives. The computer science segment of the curriculum (including computer science electives) consists of 40 credit-hours of study. This program differs from implementation $\mathcal{E}$ because it has more electives in the humanities and social sciences, in keeping with its liberal arts orientation.

This program of study requires 32 courses, with credit-hours distributed as follows:

16	hours of science
16	hours of mathematics
40	hours of humanities and social sciences (to include English)
28	hours of required computer science
12	hours of computer science electives
16	hours of free electives
128	

Course Descriptions

$C_{\mathcal{H}}101$ Problem Solving, Programs, and Computers

Topic Summary: This course has three major themes: a rigorous introduction to the process of algorithmic problem solving, an introduction to the organization of the computers upon which the resulting programs run, and an overview of the social and ethical context in which the field of computing exists. Problem solving rigor is guaranteed by a commitment to the precise specification of problems and a close association between the problem solving process and that specification. For example, the identification of a loop is closely tied to the discovery of its invariant, which in turn is motivated by the problem specification itself.

Computer organization is introduced by way of a simple von Neumann machine model and assembler, upon which students can develop and exercise simple programs. Elements of the fetch-execute cycle, runtime data representation, and machine language program structure are thereby revealed.

$C_{\mathcal{H}}101$ contains 40 lecture hours of KU topics, and is taught as a four credit-hour course. A scheduled weekly laboratory is used to teach the necessary high-level syntax and machine organization, as well as to support student programming exercises.

Prerequisites: An introduction to logic, as would usually be found at the beginning of a discrete mathematics course (which can be taken concurrently)

Knowledge units: AL3 (3/3), AL6 (2/6), AR3 (2/3), AR4 (7/15), NU1 (1/3), NU2 (2/4), PL1 (2/2), PL3 (2/2), PL4 (1/4), SE1 (11/16), SE5 (4/8), SP1 (3/3)

$C_{\mathcal{H}}102$ Abstraction, Data Structures, and Large Software Systems

Topic Summary: This course introduces the notion of a large software system and the principles and methodologies that accompany its effective realization. A simple operating system model is used throughout the course as a motivational case study.

The idea of an abstract data type (ADT) is first introduced, and then reinforced through the characterization of fundamental data structures in the discipline—stacks, queues, and trees. Computational complexity is introduced and motivated through the design and analysis of two or three sorting algorithms. Other design issues are also explored, such as the use of dynamic storage for implementing an ADT. An overview of a compiler as a large software system is given as a case study.

$C_{\mathcal{H}}102$ is a four credit-hour course that covers 41 lecture hours associated with knowledge units. Laboratories for the course are of two types: closed sessions in which students work individually and open sessions in which students work in teams to solve a larger software problem than those which were introduced in course $C_{\mathcal{H}}101$.

Prerequisites: $C_{\mathcal{H}}101$

Knowledge units: AL1 (8/13), AL2 (2/2), AL4 (1/4), AL6 (4/6), HU1 (5/5), OS1 (3/3), OS2 (2/2), PL2 (2/2), PL5 (2/4), PL6 (4/4), PL9 (3/3), SE2 (2/8), SE5 (1/8), SP4 (2/2)

$C_{\mathcal{H}}203$ Levels of Architecture, Languages, and Applications

Topic Summary: This course emphasizes the variety of levels from which the discipline of computing can be viewed. Levels of architecture are unfolded through the introduction of finite automata, digital logic, and microprogramming. Levels of languages are revealed through an examination of sequence control, type checking, runtime storage

management, and nonprocedural (functional or logic) programming paradigms. Levels of applications are treated through a general introduction to the areas of database systems and artificial intelligence.

$C_{\mathcal{H}}203$ is a four credit-hour course consisting of 37 lecture hours of knowledge units and laboratories.

Prerequisites: $C_{\mathcal{H}}102$ and Discrete Mathematics

Knowledge units: AR1 (4/12), AR2 (2/6), AI1 (3/3), AI2 (4/6), DB1 (4/4), PL4 (3/4), PL5 (2/4), PL11 (10/10), SE1 (5/16)

$C_{\mathcal{H}}204$ Algorithms, Concurrency, and the Limits of Computation

Topic Summary: This course emphasizes theoretical aspects of computer science. A review of graph theory is followed by a study of basic graph algorithms—search and traversal, shortest path, connectedness, etc.

Computational complexity levels beyond $O(n^2)$, including the ideas of tractable, intractable, and unsolvable problems are introduced and illustrated. Complexity classes P, NP, and NC are also characterized along with the notions of Turing machines and Turing computability.

Concurrency is also introduced here, along with its applications in algorithms, architectures, operating systems, distributed computing, and networks.

$C_{\mathcal{H}}204$ is a four credit-hour course containing 39 lecture hours of KU topics and laboratories.

Prerequisites: $C_{\mathcal{H}}203$

Knowledge units: AL1 (5/13), AL4 (3/4), AL5 (4/4), AL7 (6/6), AL8 (6/6), AL9 (3/3), OS3 (3/4), OS4 (3/3), PL12 (3/3), SP3 (3/3)

$C_{\mathcal{H}}305$ Language Formalisms and Software Methodology

Topic Summary: This course combines a study of language syntax and semantics with an in-depth treatment of the software design process. The latter emphasizes an object-oriented approach which is introduced and contrasted with traditional design methodologies. The design of a syntax directed interactive editor is used as a case study throughout this course.

$C_{\mathcal{H}}305$ is a four credit-hour course that treats 36 lecture hours of KU topics and associated laboratories.

Prerequisites: $C_{\mathcal{H}}203$

Knowledge units: OS3 (1/4), OS6 (2/2), PL7 (6/6), PL8 (4/4), PL10 (2/2), SE2 (6/8), SE3 (4/4), SE4 (8/8), SE5 (3/8)

$C_{\mathcal{H}}306$ Intermediate Computer Architecture

Topic Summary: This course provides an in-depth treatment of computer architecture, including digital logic, digital systems, memory system organization, interfacing and communication, and alternative architectures.

$C_{\mathcal{H}}306$ is a four credit-hour course, with 36 lecture hours of KU topics and laboratories.

Prerequisites: $C_{\mathcal{H}}204$

Knowledge units: AR1 (8/12), AR2 (4/6), AR4 (8/15), AR5 (5/13), AR7 (5/5), OS9 (3/3), OS10 (3/3)

$C_{\mathcal{H}}307$ Data Representation, Files, and Database Systems

Topic Summary: This course is a study of contemporary models and methodologies for representing, storing, and retrieving large quantities of information stored on external devices. Alternative views of data are seen from the perspectives of the system, the human interface, and the applications.

$C_{\mathcal{H}}307$ is a four credit-hour course, with 42 lecture hours allocated to specific KU topics and laboratories.

Prerequisites: $C_{\mathcal{H}}203$

Knowledge units: AR3 (1/3), AR5 (8/13), AR6 (5/5), AI2 (2/6), DB2 (5/5), HU2 (3/3), NU1 (2/3), NU2 (2/4), OS5 (4/4), OS7 (4/4), OS8 (3/3), SP2 (3/3)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Math 101	4	Calculus I	Math 102	4	Calculus II
$C_{\mathcal{H}}101$	4	Prob Solving, Programs, & Computers	$C_{\mathcal{H}}102$	4	Abstraction, Data Struct, & Lg Softw Syst
Sci El.	4	Science Elective	Sci El.	4	Science Elective
Engl 100	4	Freshman English	HU/SS	4	HU/SS Elective
	16			16	

Sophomore Year

First Semester			Second Semester		
Sci El.	4	Science Elective	Sci El.	4	Science Elective
Math 201	4	Discrete Structures	Math 202	4	Probability & Statistics
$C_{\mathcal{H}}203$	4	Levels of Arch, Languages, & Applications	$C_{\mathcal{H}}204$	4	Algorithms, Concurrency, & Limits of Comp.
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
	16			16	

Junior Year

First Semester			Second Semester		
$C_{\mathcal{H}}305$	4	Language Formalisms & Softw Methodology	$C_{\mathcal{H}}307$	4	Data Representation Files, & Database Sys
$C_{\mathcal{H}}306$	4	Intermediate Comp Arch	CS El.	4	Computer Science Elective
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Senior Year

First Semester			Second Semester		
CS El.	4	CS Elective	CS El.	4	CS Elective
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Advanced Courses Any of the following courses can be offered as advanced electives to complete the major.

- Artificial Intelligence
- Database Principles
- Semantics and Verification
- Compiler Design
- Networks and Distributed Computing
- Computer Graphics
- Parallelism and Concurrency
- Simulation
- Numerical and Symbolic Computation

Implementation $\mathcal{I}$: A Program in Computer Science and Engineering

Goals and Features of this Program: This program leads to the B.S. in computer science and engineering in a way that reflects a slight modification of either Implementation $\mathcal{A}$ or Implementation $\mathcal{D}$. That is, minor alterations can be made to either implementation so that the result simultaneously satisfies the curricular requirements of current EAC/ABET and CSAC/CSAB accreditation criteria.

One possible modification of Implementation $\mathcal{A}$ so that its structure simultaneously satisfies both sets of accreditation requirements is shown in the Typical Student Schedule on page 134. There, a HU/SS Elective is inserted in the second semester of each of the junior and senior years, in place of a free elective and an engineering science course. These changes are reflected in the following altered Summary of Requirements.

15	hours of physics, chemistry, and science electives
22	hours of mathematics
30	hours of humanities and social sciences (to include English composition)
38	hours of required computer science and engineering
18	hours of computer science and engineering electives
6	hours of other engineering courses
6	hours of free electives
135	

To modify Implementation $\mathcal{D}$ so that it meets both requirements, the Typical Student Schedule could be altered as shown on page 135. There, a third semester science elective becomes a required chemistry course, a fourth semester mathematics course becomes a 4-credit course, a fourth semester HU/SS elective becomes an engineering science course, and an eighth semester free elective becomes a HU/SS elective. It should also be noted that CS Elective 6 in each track of the table of electives for Implementation $\mathcal{D}$ must be implemented so that it meets the EAC/ABET design criteria. The following altered Summary of Requirements reflects these changes.

14	hours of science
21	hours of mathematics
30	hours of humanities and social sciences (to include English composition)
30	hours of required computer science
18	hours of computer science electives
3	hours of engineering science
9	hours of free electives
125	

Typical Student Schedule All courses listed below are understood to have the same course descriptions as their counterparts in Implementation $\mathcal{A}$. For the remaining details, readers should consult page 82.

Freshman Year

First Semester			Second Semester		
Chem 101	4	Chemistry I	Phys 101	4	Physics I
Math 101	4	Calculus I	Math 102	4	Calculus II
$C_{\mathcal{I}}101$	3	Introduction to Computing I	$C_{\mathcal{I}}102$	3	Introduction to Computing II
Engl 100	3	English Composition	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
CSE Seminar	0		CSE Seminar	0	
	17			17	

Sophomore Year

First Semester			Second Semester		
Phys 201	4	Physics II	Sci Elect	3	Science Elective
Math 201	4	Calculus III	Math 203	4	Linear Algebra and Diff. Equations
$C_{\mathcal{I}}201$	3	Introduction to Computer Engineering	$C_{\mathcal{I}}202$	3	Design & Analysis of Algorithms
Math 202	3	Discrete Math	EE 201	3	Basic Circuits and Electronics
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
CSE Seminar	0		CSE Seminar	0	
	17			16	

Junior Year

First Semester			Second Semester		
$C_{\mathcal{I}}301$	4	Hardware Systems	$C_{\mathcal{I}}303$	3	Software Engineering
$C_{\mathcal{I}}302$	4	Software Systems	$C_{\mathcal{I}}304$	3	Programming Languages
EE 301	3	Linear Systems	HU/SS	3	HU/SS Elective
Math 301	3	Probability & Statistics	Prof. El.	3	Professional Elective
HU/SS	3	HU/SS Elective	HU/SS	3	Professional Ethics
CSE Seminar	0		CSE Seminar	0	
	17			15	

Senior Year

First Semester			Second Semester		
$C_{\mathcal{I}}305$	3	Operating Systems	$C_{\mathcal{I}}306$	3	Computer Architecture
$C_{\mathcal{I}}401$	3	Design Laboratory	$C_{\mathcal{I}}402$	3	Design Laboratory
Prof. El.	3	Professional Elective	Prof. El.	3	Professional Elective
Prof. El.	3	Professional Elective	Prof. El.	3	Professional Elective
Prof. El.	3	Professional Elective	HU/SS	3	HU/SS Elective
Elect	3	Elective	Elect	3	Elective
CSE Seminar	0		CSE Seminar	0	
	18			18	

Typical Student Schedule All courses listed below are understood to have the same course descriptions as their counterparts in Implementation $\mathcal{D}$. For the remaining details, readers should consult page 103.

Freshman Year

First Semester			Second Semester		
Phys 101	4	Physics I	Phys 102	4	Physics II
Math 101	4	Calculus I	Math 102	4	Calculus II
$C_{\mathcal{T}}101$	3	Introduction to Computing I	$C_{\mathcal{T}}102$	3	Introduction to Computing II
Engl 100	3	English Composition	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	17			17	

Sophomore Year

First Semester			Second Semester		
Chem 101	3	Chemistry I	Sci El.	3	Science Elective
Math 201	3	Calculus III	Math 202	4	Linear Algebra & Dif. Eq'ns.
$C_{\mathcal{T}}201$	3	Introduction to Computer Systems	$C_{\mathcal{T}}202$	3	Design & Analysis of Algorithms
Math 203	3	Discrete Math	ES 202	3	Electrical Circuits
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	15			16	

Junior Year

First Semester			Second Semester		
$C_{\mathcal{T}}301$	3	Computer Org and Assembly Lang	$C_{\mathcal{T}}303$	3	Software Engineering
$C_{\mathcal{T}}302$	3	Software Systems	$C_{\mathcal{T}}304$	3	Programming Languages
Math 301	3	Probability & Statistics	SO 301	3	Professional Ethics
CS El.	3	CS Elective	CS El.	3	CS Elective
Elect	3	Elective	HU/SS	3	HU/SS Elective
	15			15	

Senior Year

First Semester			Second Semester		
$C_{\mathcal{T}}305$	3	Operating Systems	$C_{\mathcal{T}}306$	3	Computer Architecture
CS El.	3	CS Elective	CS El.	3	CS Elective
CS El.	3	CS Elective	CS El.	3	CS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
Elect	3	Elective	Elect	3	Elective
	15			15	

A.2 Sample Curricula with Other Major Goals

The sample implementations in this section generally have as their principal goal the education of well rounded individuals who are not necessarily being trained as professionals in the discipline of computing. The goals of these programs often include "preparation for a lifetime of learning," "breadth of education," "preparation for graduate study," and so forth. Departments offering degree programs such as these are found in Schools and Colleges of Arts and Sciences and Liberal Arts Colleges.

It is not unusual for computer science majors in these programs to double-major in another discipline such as Economics, Mathematics, English, Japanese, or Psychology. Because of this educational breadth, graduates have a variety of career options available. Graduates desiring entry into the computing profession normally take more than the minimum number of required courses, or else pursue further studies, in order to complete their preparation. To support the breadth of a liberal arts education, the number of courses required for a computer science major is often limited to a maximum of 9 four-credit computer science courses in a 32-course degree. Additionally, a small number of related courses are normally required, but the total number of required courses often does not exceed 12 or 13.

Implementation $\mathcal{J}$: A Liberal Arts Program in Computer Science

Goals and Features of this Program: This program of study leads to the liberal arts degree in Computer Science in such a way that theory and abstraction are emphasized early in the curriculum, while applications appear more prominently in the advanced electives. The total number of required courses in computer science is limited because students need also to cover a broad range of courses in the humanities and social sciences in order to complete their degree requirements.

This program provides students with an undergraduate major in computer science within a liberal arts context. Students acquire a foundation for lifelong personal and professional growth through a variety of career paths.

Summary of Requirements: This program of study requires 32 courses, with credit-hours distributed as follows:

16	hours of mathematics
16	hours of humanities and social science
28	hours of required computer science
8	hours of computer science electives
60	hours of free electives
128	

Course Descriptions

C$_\mathcal{J}$101 Fundamentals of Computing I

Topic Summary: This course introduces students to fundamental aspects of the field of computing, focusing on problem-solving and software design concepts and their realization as computer programs. Topics include procedural abstraction, control structures, iteration, recursion, data types and their representation, and iterative approximation methods. An introduction to a high-level language, for the purpose of gaining mastery of these principles, will be provided in lectures and in closely-coordinated laboratory experiences.

This four credit-hour course incorporates 42 KU hours and laboratory work.

Prerequisites: Topics in Discrete Mathematics concurrently

Knowledge units: AL3 (3/3), NU1 (3/3), NU2 (4/4), PR (8/12), SE1 (16/16), SE2 (5/8), SP1 (3/3)

C$_\mathcal{J}$102 Fundamentals of Computing II

Topic Summary: This course moves students into the domain of software design, introducing principles that are necessary for solving large problems. Here, the classical

software design process serves as a basis for treating such topics as abstract data types, specifications, complexity analysis, and file organization. Basic data structures (queues, stacks, trees, and graphs) and transformations (sorting and searching), are introduced as representative of the fundamental tools that are used to aid in this process. Principles of file and database organization are also introduced here.

This four credit-hour course covers 45 lecture hours of KU material and associated laboratory work.

Prerequisites: C$_\mathcal{J}$101

Knowledge units: AL1 (13/13), AL2 (2/2), AL4 (4/4), AL6 (2/6), DB1 (4/4), DB2 (5/5), SE2 (3/8), SE3 (4/4), SE4 (8/8)

C$_\mathcal{J}$203 Computer Organization

Topic Summary: Principles of computer architecture are introduced from a layered point of view, beginning at the level of data representation and progressing through the machine language execution cycle, addressing modes, and symbolic assembly level of language. Interfacing and communication, as well as fundamental notions of an operating system (including tasking and processes), are also introduced. Coordinated laboratory exercises will allow students to experiment with program behavior and machine elements at each of these levels.

This four credit-hour course introduces 42 lecture hours of KU topics and laboratory work.

Prerequisites: C$_\mathcal{J}$101

Knowledge units: AR1 (7/12), AR2 (6/6), AR3 (3/3), AR4 (15/15), AR5 (2/13), AR6 (4/5), OS1 (3/3), OS2 (2/2)

C$_\mathcal{J}$204 Algorithms

Topic Summary: This course continues the study of the design and analysis of algorithms, particularly those for handling complex data structures and non-numeric processes. Introduction to algorithm design techniques, including greedy algorithms, divide-and-conquer, and dynamic programming. Lower and upper bounds of program complexity are analyzed. A brief introduction to AI is given, focusing on data representation and heuristic search methods. Introduction to algorithm verification and the impact of parallel computation on algorithms, operating systems (process synchronization and coordination, etc), and architectures (SIMD, etc).

This is a four credit-hour course, incorporating 41 lecture hours of KU topics.

Prerequisites: C$_\mathcal{J}$102, C$_\mathcal{J}$203, and Discrete Math

Knowledge units: AL5 (2/4), AL6 (4/6), AL8 (6/6), AL9 (3/3), AR7 (5/5), AI1 (3/3), AI2 (6/6), OS3 (4/4), SE5 (8/8)

C$_J$305 Theory of Computation

Topic Summary: Formal models of computation such as finite state automata, pushdown automata and Turing machines will be studied, along with the corresponding elements of formal languages (including regular expressions, context-free languages, and recursively enumerable languages). These models will be used to provide a mathematical basis for the study of computability, and to provide an introduction to the formal theory behind compiler construction. The study of Church's thesis and universal Turing machines will lead to the study of undecidable problems.

This course includes material significantly beyond the common requirements in topics associated with complexity classes (AL5), computability (AL7), and theoretical models of computation (PL7 and PL8). It contains approximately 18 hours of the KU treatment of these topics.

Prerequisites: C$_J$102 and Discrete Mathematics

Knowledge units: AL5 (2/4), AL7 (6/6), PL7 (6/6), PL8 (4/4)

C$_J$306 Programming Languages: Principles and Paradigms

Topic Summary: A short history of programming languages and styles precedes the study of an important collection of programming paradigms. This material includes data types, data control, sequence control, run-time storage, language translation, and semantics. The paradigms and their languages include procedural, functional, logic, and object-oriented programming. Language features which support parallel and distributed computing are also introduced. The utility of various interactive tools and environments, as well as very high level languages, to complement these paradigms in the programming domain, is also examined.

This four credit-hour course contains approximately 44 lecture hours of KU topics and associated laboratory work.

Prerequisites: C$_J$203

Knowledge units: HU1 (5/5), HU2 (3/3), PL1 (2/2), PL2 (2/2), PL3 (2/2), PL4 (4/4), PL5 (4/4), PL6 (4/4), PL9 (3/3), PL10 (2/2), PL11 (10/10), PL12 (3/3)

C$_J$307 Architecture and Operating Systems

Topic Summary: A more in-depth treatment of computer architecture, technological choices, and the operating system interface with the hardware, the application, and the system user. Contemporary social and professional issues, such as intellectual property,

risks and liabilities, and system security, are also discussed in the context of architecture and operating system design.

This four credit-hour course covers 47 hours of KU topics and laboratories.

Prerequisites: C$_{\mathcal{J}}$203

Knowledge units: AR1 (5/12), AR5 (11/13), AR6 (1/5), OS4 (3/3), OS5 (4/4), OS6 (2/2), OS7 (4/4), OS8 (3/3), OS9 (3/3), OS10 (3/3), SP2 (3/3), SP3 (3/3), SP4 (2/2)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Math 102	4	Discrete Math	Math 101	4	Calculus I
C$\mathcal{J}$101	4	Fundamentals of Computing I	C$\mathcal{J}$102	4	Fundamentals of Computing II
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Sophomore Year

First Semester			Second Semester		
Math 201	4	Calculus II	Math 202	4	Linear Algebra
C$\mathcal{J}$203	4	Computer Organization	C$\mathcal{J}$204	4	Algorithms,
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
HU/SS	4	HU/SS Elective	Elect	4	Elective
	16			16	

Junior Year

First Semester			Second Semester		
C$\mathcal{J}$305	4	Theory of Computation	C$\mathcal{J}$307	4	Architecture and Operating Systems
C$\mathcal{J}$306	4	Programming Languages	HU/SS	4	HU/SS Elective
Elect	4	Elective	Elect	4	Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Senior Year

First Semester			Second Semester		
CS El.	4	CS Elective	CS El.	4	CS Elective
Elect	4	Elective	Elect	4	Elective
Elect	4	Elective	Elect	4	Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Advanced Courses Any of the following courses can be offered as advanced electives to complete the major.

- Artificial Intelligence
- Database Principles
- Compiler Design
- Networks and Distributed Computing
- Computer Graphics
- Parallelism and Concurrency
- Simulation
- Numerical and Symbolic Computation
- Object-oriented Software Design
- Semantics and Verification
- Topics

The course "Topics" is intended to serve the interests of special or new topics that emerge in the discipline, as well as the special needs of faculty who would develop the curriculum for such topics.

Implementation $\mathcal{K}$: A Liberal Arts Program in Computer Science (Breadth-First)

Goals and Features of this Program: This program leads to the liberal arts degree in Computer Science in a way similar to Implementation $\mathcal{J}$, but with one major exception: the common requirements are presented in such a way that the breadth of the discipline is covered early in an undergraduate's program of study rather than late. Specifically, the first four courses provide exposure to all nine subject areas of the discipline along with the area of social and professional issues.

This program provides students with an undergraduate major in computer science within a liberal arts context. Students acquire a foundation for lifelong personal and professional growth through a variety of career paths.

Summary of Requirements: This program of study requires 32 courses, with credit-hours distributed as follows:

16	hours of mathematics
16	hours of humanities and social science
28	hours of required computer science
8	hours of computer science electives
60	hours of free electives
128	

Course Descriptions

$C_{\mathcal{K}}101$ Problem-solving, Programs, and Computers

Topic Summary: This course has three major themes: a rigorous introduction to the process of algorithmic problem-solving, an introduction to the organization of the computers upon which the resulting programs run, and an overview of the societal and ethical context in which the field of computing exists. Problem-solving rigor is guaranteed by a commitment to the precise specification of problems and a close association between the problem-solving process and that specification.

Computer organization is introduced by way of a simulated von Neumann machine model and assembler, upon which students can develop and exercise simple programs. Elements of the fetch-execute cycle, runtime data representation, and machine language program structure are thereby revealed.

A scheduled weekly laboratory is used to teach the necessary high level language syntax and machine organization, as well as to support student programming exercises. This four-hour course contains 40 lecture hours of KU topics.

Prerequisites: an introduction to logic, as found at the beginning of a discrete mathematics course, which can be taken concurrently.

Knowledge units: AL3 (3/3), AL6 (2/6), AR3 (2/3), AR4 (7/15), NU1 (1/3), NU2 (2/4), PL1 (2/2), PL3 (2/2), PL4 (1/4), SE1 (11/16), SE5 (4/8), SP1 (3/3)

C_K102 Abstraction, Data Structures, and Large Software Systems

Topic Summary: This course introduces the notion of a large software system and the principles and methodologies that accompany its effective realization. A simple operating system model is used throughout the course as a motivational case study.

The idea of an abstract data type (ADT) is first introduced, and then reinforced through the characterization of fundamental data structures in the discipline–stacks, queues, and trees. Computational complexity is introduced and motivated through the design and analysis of two or three sorting algorithms. Other design issues are also explored, such as the use of dynamic storage for implementing an ADT. An overview of a compiler as a large software system is given as a case study.

Laboratories for this course are of two types: closed sessions in which students work individually, and open sessions in which students work in teams to solve a larger software problem than those which were introduced in course C_K101. This course contains 41 lecture hours of KU topics.

Prerequisites: C_K101

Knowledge units: AL1 (8/13), AL2 (2/2), AL4 (1/4), AL6 (4/6), HU1 (5/5), OS1 (3/3), OS2 (2/2), PL2 (2/2), PL5 (2/4), PL6 (4/4), PL9 (3/3), SE2 (2/8), SE5 (1/8), SP4 (2/2)

C_K203 Levels of Architecture, Languages, and Applications

Topic Summary: This course emphasizes the variety of levels from which the discipline of computing can be viewed. Levels of architecture are unfolded through the introduction of finite automata, digital logic, and microprogramming. Levels of languages are revealed through an examination of sequence control, type checking, run time storage management, and nonprocedural (functional or logic) programming paradigms. Levels of applications are treated through a general introduction to the areas of database systems and artificial intelligence.

This four semester-hour course contains 37 hours of KU topics and related laboratories.

Prerequisites: C_K102

Knowledge units: AR1 (4/12), AR2 (2/6), AI1 (3/3), AI2 (4/6), DB1 (4/4), PL4 (3/4), PL5 (2/4), PL11 (10/10), SE1 (5/16)

C_K204 Algorithms, Concurrency, and the Limits of Computation

Topic Summary: While courses C_K101–C_K203 emphasize the processes of abstraction and design in the discipline of computing, this courses emphasizes the process of theory. A review of graph theory precedes a study of basic graph algorithms—search and traversal, shortest path, connectedness, etc.

Computational complexity levels beyond $O(n^2)$, including the ideas of tractable, intractable, and unsolvable problems, are introduced and illustrated. Complexity classes P, NP, and NC are also characterized, along with the notion of Turing machine and Turing computability. Concurrency is also introduced here, along with its applications in algorithms, architectures, operating systems, distributed computing, and networks.

This course incorporates 39 lecture hours of KU topics.

Prerequisites: C_K203

Knowledge units: AL1 (5/13), AL4 (3/4), AL5 (4/4), AL7 (6/6), AL8 (6/6), AL9 (3/3), OS3 (3/4), OS4 (3/3), PL12 (3/3), SP3 (3/3)

C_K305 Language Formalisms and Software Methodology

Topic Summary: This course combines a study of language syntax and semantics with an in-depth treatment of the software design process. The latter emphasizes an object-oriented approach, which is introduced and contrasted with traditional design methodologies. The design of a syntax- directed interactive editor is used as a case study throughout this course.

This course covers 36 hours of KU topics and related laboratories.

Prerequisites: C_K203

Knowledge units: OS3 (1/4), OS6 (2/2), PL7 (6/6), PL8 (4/4), PL10 (2/2), SE2 (6/8), SE3 (4/4), SE4 (8/8), SE5 (3/8)

C_K306 Intermediate Computer Architecture

Topic Summary: This course is a more in-depth treatment of computer architecture, including digital logic, digital systems, memory system organization, interfacing and communication, and alternative architectures.

This course covers 36 lecture hours of KU topics and related laboratories.

Prerequisites: C_K204

Knowledge units: AR1 (8/12), AR2 (4/6), AR4 (8/15), AR5 (5/13), AR7 (5/5), OS9 (3/3), OS10 (3/3)

C_K307 Data Representation, Files, and Database Systems

Topic Summary: This course is a study of contemporary models and methodologies for representing, storing, and retrieving large quantities of information stored on external devices. Alternative views of data are seen from the different perspectives of architecture, operating systems, data base systems, the human interface, and the applications.

This four semester-hour course introduces 42 lecture hours of KU topics and related laboratory work.

Prerequisites: C_K102 and C_K203

Knowledge units: AR3 (1/3), AR5 (8/13), AR6 (5/5), AI2 (2/6), DB2 (5/5), HU2 (3/3), NU1 (2/3), NU2 (2/4), OS5 (4/4), OS7 (4/4), OS8 (3/3), SP2 (3/3)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Math 101	4	Discrete Math	Math 102	4	Calculus I
C_K101	4	Problem Solving, Programs, & Computers	C_K102	4	Abstraction, Data Structures, & Lg Softw Syst
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Sophomore Year

First Semester			Second Semester		
Math 201	4	Calculus II	Math 202	4	Probability & Statistics
C_K203	4	Levels of Arch, Lang, & Applic	C_K204	4	Algorithms, Concur., & Limits of Comp
HU/SS	4	HU/SS Elective	HU/SS	4	HU/SS Elective
HU/SS	4	HU/SS Elective	Elect	4	Elective
	16			16	

Junior Year

First Semester			Second Semester		
C_K305	4	Lang Formalisms & Softw Meth	C_K306	4	Intermediate Comp Architecture
Elect	4	Elective	C_K307	4	Data Rep, Files, & Database Systems
Elect	4	Elective	Elect	4	Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Senior Year

First Semester			Second Semester		
CS El.	4	CS Elective	CS El.	4	CS Elective
Elect	4	Elective	Elect	4	Elective
Elect	4	Elective	Elect	4	Elective
Elect	4	Elective	Elect	4	Elective
	16			16	

Advanced Courses Any of the following courses can be taken as advanced electives to complete the major.

- Theory of Computation
- Algorithms
- Artificial Intelligence
- Database Principles
- Compiler Design
- Networks and Distributed Computing
- Computer Graphics
- Parallelism and Concurrency
- Simulation
- Numerical and Symbolic Computation
- Object-oriented Software Design
- Semantics and Verification
- Topics

The course "Topics" can serve the interests of special or new topics that emerge in the discipline, and in response to the special interests of faculty who would develop the curriculum for such topics. At least one of these electives should include a significant software design project.

Implementation $\mathcal{L}$: A Program in Computer Science (Theoretical Emphasis)

Goals and Features of this Program: This curriculum leads to the B.S. in Computer Science in such a way that theory and abstraction are emphasized early in the curriculum, while applications appear more prominently in the advanced electives. Of special note in this program are its required courses in algorithms, theory of computation, and computational science. This program therefore offers in-depth treatment of theoretical foundations and formal methods in computer science, preparing students especially well for graduate study in these areas.

This curriculum provides students with an undergraduate major in computer science within an arts and sciences context. Students acquire a foundation for lifelong personal and professional growth in the discipline or in a related field.

Summary of Requirements: This program is built around a set of nine (9) required courses in Computer Science, comprising 27 credit-hours of study. In addition, this program assumes that students are required to take five (5) electives in Computer Science (depending upon the institutional setting) plus five (5) courses in mathematics, including Discrete Mathematics and a course on methods of proof (inductive proofs, constructive proofs, diagonalization arguments, etc.).

This program of study requires 40 courses, with credit-hours distributed as follows:

15	hours of mathematics
8	hours of science
24	hours of humanities and social science (including English composition)
33	hours of required computer science
15	hours of computer science electives
33	hours of free electives
128	

Course Descriptions

$C_{\mathcal{L}}101$ Introduction to Computer Science I

Topic Summary: This course introduces software design methodology using a modern programming language. Students are introduced to the process of designing and implementing a program to solve an algorithmic problem. Data types are covered, and sorting and searching techniques are introduced to help motivate the methodological issues. Students are introduced to control structures, data types and procedural abstraction. The course also introduces the history and social impact of computing and the nature of intellectual property.

$C_{\mathcal{L}}101$ is a four credit-hour course, and 40 lecture hours are devoted to KU topics. Availability of dedicated laboratory time is essential for this course.

Prerequisites: Discrete Mathematics (concurrently).

Knowledge units: AL1 (5/13), AL6 (2/6), AR3 (1/3), PL3 (2/2), PR (7/12), SE1 (16/16), SP1 (3/3), SP2 (1/3), SP3 (1/3), SP4 (2/2)

$C_{\mathcal{L}}102$ Introduction to Computer Science II

Topic Summary: This course continues the development of fundamental ideas in software design and development. Students are introduced to the concept of abstract data types. That concept is applied to the implementation of various data structures, including stacks, queues, and binary trees. Sorting and searching algorithms using these structures are introduced. Other topics include recursion, the software lifecycle, requirements specifications, and an introduction to program verification.

$C_{\mathcal{L}}102$ is a four credit-hour course, containing 19 lecture hours devoted to KU topics. Availability of dedicated laboratory time is essential for this course.

Prerequisites: $C_{\mathcal{L}}101$, Methods of Proof (concurrently).

Knowledge units: AL1 (8/13), AL2 (2/2), AL3 (3/3), AL6 (2/6), SE2 (1/8), SE3 (1/4), SE4 (1/8), SE5 (1/8)

$C_{\mathcal{L}}203$ Analysis of Algorithms

Topic Summary: This course gives an in-depth treatment of complexity analysis. Various sorting and searching algorithms, as well as state-space searching methods in artificial intelligence and file structures used in database systems, are used to help motivate and develop this topic. Complexity classes are discussed, as well as the nature of NP-completeness and intractability. This leads to a discussion of computability and to a discussion of the universal Turing machine. Special problems associated with parallel algorithms are discussed.

$C_{\mathcal{L}}203$ is a three credit-hour course, with 24 lecture hours devoted to KU topics. Innovative laboratory material can be helpful in this course.

Prerequisites: $C_{\mathcal{L}}102$.

Knowledge units: AL4 (4/4), AL5 (4/4), AL6 (2/6), AL7 (3/6), AL8 (6/6), AL9 (3/3), AI2 (2/6)

$C_{\mathcal{L}}204$ Theory of Computation

Topic Summary: This course introduces the theory of computability, including important results from the study of automata and formal languages. The course begins with

a discussion of automata and their relationship to regular, context free and context-sensitive languages. General theories of computability are presented, including Turing machines, recursive functions and lambda calculus. Notions of decidability and undecidability are discussed and this is related to complexity analysis. Finally, approaches to formal program semantics are presented and analyzed, leading to a brief introduction to the topic of formal program verification.

$C_{\mathcal{L}}204$ is a three credit-hour course, with 21 lecture hours devoted to KU topics.

Prerequisites: $C_{\mathcal{L}}203$

Knowledge units: AL7 (3/6), PL7 (6/6), PL8 (4/4), PL9 (3/3), PL10 (2/2), SE5 (3/8)

$C_{\mathcal{L}}205$ Design of Digital Systems

Topic Summary: This course introduces digital systems, their mathematical models and their realization in hardware. Topics include logic elements, minimization techniques, memory system organization and architecture, interfacing and data communications, and alternative architectures, including multiprocessor and hypercube machines.

$C_{\mathcal{L}}205$ is a four credit-hour course, with 41 lecture hours devoted to KU topics. Availability of dedicated lab time is essential for this course.

Prerequisites: $C_{\mathcal{L}}102$

Knowledge units: AR1 (12/12), AR2 (6/6), AR5 (13/13), AR6 (5/5), AR7 (5/5)

$C_{\mathcal{L}}206$ Operating Systems and Database Systems

Topic Summary: This course begins with an a fairly in-depth treatment of operating systems and concludes with a brief introduction to database systems. Operating systems topics include tasks, processes, process synchronization, physical and virtual memory organization, distributed operating systems, security and privacy. In support of these operating systems topics, programming language constructs that support concurrency are introduced. Database topics include physical database organization and the relational data model.

$C_{\mathcal{L}}206$ is a four credit-hour course, with 43 lecture hours devoted to KU topics. Innovative laboratory material can be helpful in this course.

Prerequisites: $C_{\mathcal{L}}205$

Knowledge units: DB1 (4/4), DB2 (5/5), OS1 (3/3), OS2 (2/2), OS3 (4/4), OS4 (3/3), OS5 (4/4), OS6 (2/2), OS7 (4/4), OS8 (3/3), OS9 (3/3), OS10 (3/3), PL12 (3/3)

$C_{\mathcal{L}}307$ Programming Languages and Paradigms

Topic Summary: This course introduces the student to the nature of contemporary programming languages by beginning with an in-depth treatment of assembly language

and ending with an in-depth treatment of a true object-oriented language (such as Smalltalk or Eiffel). In between, the evolution of imperative languages (FORTRAN, Algol, PL/I, Pascal, Ada) and functional languages (Lisp, Scheme, ML) are discussed. In addition, fundamental design and implementation concepts for high-level programming languages are introduced, including the concepts of binding, type checking and run-time storage management.

$C_{\mathcal{L}}307$ is a four credit-hour course with 43 lecture hours devoted to KU topics and associated laboratories.

Prerequisites: $C_{\mathcal{L}}204$

Knowledge units: AR3 (2/3), AR4 (15/15), PL1 (2/2), PL2 (2/2), PL4 (4/4), PL5 (4/4), PL6 (4/4), PL11 (10/10)

$C_{\mathcal{L}}308$ Software Engineering and User Interfaces

Topic Summary: This course presents an in-depth treatment of many software engineering topics, including software engineering paradigms, requirements specification, functional and object-oriented design, software verification and maintenance. Software environments and software engineering tools are discussed and students are introduced to their use. Societal implications such as cost of failure and professional responsibilities are considered. The course also includes an introduction to computer-human interaction and a discussion of user interfaces from a software engineering perspective. This course requires that students participate in a team project.

$C_{\mathcal{L}}308$ is a four credit-hour course, and contains 33 lecture hours of KU topics. Availability of dedicated laboratory time is essential for this course.

Prerequisites: $C_{\mathcal{L}}204$, $C_{\mathcal{L}}206$

Knowledge units: HU1 (5/5), HU2 (3/3), SE2 (7/8), SE3 (3/4), SE4 (7/8), SE5 (4/8), SP2 (2/3), SP3 (2/3)

$C_{\mathcal{L}}309$ Computational Science

Topic Summary: This course introduces a variety of algorithms that are useful in scientific computing and in artificial intelligence. Numerical methods are treated in some depth, with special emphasis on linear systems, numerical integration, error analysis and the concept of stability. Important algorithms in artificial intelligence and neural networks are introduced. Other topics might include the Fourier transform, computational geometry and other algorithms of special interest to the instructor.

$C_{\mathcal{L}}309$ is a three credit-hour course with 14 lecture hours of KU topics.

Prerequisites: $C_{\mathcal{L}}204$, Calculus and Linear Algebra.

Knowledge units: AI1 (3/3), AI2 (4/6), NU1 (3/3), NU2 (4/4)

Typical Student Schedule

Freshman Year

First Semester			Second Semester		
Math 101	3	Discrete Math	Math 102	3	Methods of Proof
$C_{\mathcal{L}}101$	4	Introduction to CS I	$C_{\mathcal{L}}102$	4	Introduction to CS II
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
Sci. El.	4	Science Elective	Sci. El.	4	Science Elective
Elect	3	Elective	Elect	3	Elective
	17			17	

Sophomore Year

First Semester			Second Semester		
Math 103	3	Calculus I	Math 104	3	Calculus II
$C_{\mathcal{L}}203$	3	Analysis of Algorithms	$C_{\mathcal{L}}204$	3	Theory of Computation
$C_{\mathcal{L}}205$	4	Design of Digital Systems	$C_{\mathcal{L}}206$	4	Operating Systems & Database Syst
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	HU/SS	3	HU/SS Elective
	16			16	

Junior Year

First Semester			Second Semester		
Math 103	3	Linear Algebra	$C_{\mathcal{L}}309$	3	Computational Science
$C_{\mathcal{L}}307$	4	Prog Lang & Paradigms	CS El.	3	CS Elective
$C_{\mathcal{L}}308$	4	Softw Eng'g	HU/SS	3	HU/SS Elective
HU/SS	3	HU/SS Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
	17			15	

Senior Year

First Semester			Second Semester		
CS El.	3	CS Elective	CS El.	3	CS Elective
CS El.	3	CS Elective	CS El.	3	CS Elective
Elect	3	Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
Elect	3	Elective	Elect	3	Elective
	15			15	

Advanced Courses Any of the following courses can be taken as advanced electives to complete the major.

- Advanced Algorithms
- Artificial Intelligence
- Compiler Design
- Networks and Distributed Computing
- Computer Graphics
- Parallelism and Concurrency
- Simulation
- Numerical and Symbolic Computation
- Object-oriented Software Design
- Semantics and Verification
- Topics

The course "Topics" can serve the interests of special or new topics that emerge in the discipline, and in response to the special interests of faculty who would develop the curriculum for such topics. At least one of these electives should include a significant software design project.