Computing Curricula 1991

Collected Knowledge Units and
Selected Implementations J and K

DRAFT

of May 26, 2008

Contents

1 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.

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.

1.1 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 **??** 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 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

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 1: The Subject Areas and Tags

tags AL1, AL2, AL3, and so forth.

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

1.2 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

1.3 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:

1.4 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:

1.5 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:

1.6 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:

1.7 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:

1.8 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:

1.9 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:

1.10 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:

1.11 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, trade-offs 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[1]
Prerequisites:
Requisite for:

1.12 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

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

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 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[2]
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.

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

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

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

1.13 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:

1.13.1 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	

1.13.2 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_{\mathcal{J}}$305 Theory of Computation

Topic Summary: Formal models of computation such as finite state automata, push-down 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_{\mathcal{J}}$102 and Discrete Mathematics

Knowledge units: AL5 (2/4), AL7 (6/6), PL7 (6/6), PL8 (4/4)

$C_{\mathcal{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_{\mathcal{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_{\mathcal{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)

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.

1.13.3 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	

1.13.4 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_{\mathcal{K}}$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.

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_{\mathcal{K}}$101. This course contains 41 lecture hours of KU topics.

Prerequisites: $C_{\mathcal{K}}$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{K}}$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, 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_{\mathcal{K}}$102

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{K}}$204 Algorithms, Concurrency, and the Limits of Computation

Topic Summary: While courses $C_{\mathcal{K}}$101–$C_{\mathcal{K}}$203 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_{\mathcal{K}}$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{K}}$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.

This course covers 36 hours of KU topics and related laboratories.

Prerequisites: $C_{\mathcal{K}}$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{K}}$306 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_{\mathcal{K}}$204

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

$C_{\mathcal{K}}307$ 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 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: $C_{\mathcal{K}}102$ and $C_{\mathcal{K}}203$

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	Discrete Math	Math 102	4	Calculus I
C$_{\mathcal{K}}$101	4	Problem Solving, Programs, & Computers	C$_{\mathcal{K}}$102	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$_{\mathcal{K}}$203	4	Levels of Arch, Lang, & Applic	C$_{\mathcal{K}}$204	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$_{\mathcal{K}}$305	4	Lang Formalisms & Softw Meth	C$_{\mathcal{K}}$306	4	Intermediate Comp Architecture
Elect	4	Elective	C$_{\mathcal{K}}$307	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.