Computer Science/Mathematics Curriculum
Pepperdine University

Background

Seaver College is the undergraduate liberal arts college of Pepperdine University. This document describes the curriculum for the major in Computer Science/Mathematics. The college does not offer a major in Computer Science apart from Mathematics. Nor does it offer an advanced degree in either discipline. Historically, Seaver College has emphasized quality teaching at the undergraduate level while encouraging scholarly activity by its faculty especially when it can have a positive impact on the undergraduate experience.

Philosophy of the curriculum

The curriculum is based on three themes—abstraction, integration, and languages and paradigms.

Abstraction

Abstraction is based on the concept of layers in which the details of one layer of abstraction are hidden from layers at a higher level. A computer scientist uses abstraction as a thinking tool to understand a system, to model a problem, and to master complexity. The ability to abstract cannot be acquired in a single course, but must be developed over several years. Consequently, all courses in the curriculum emphasize the abstraction process, not only as a framework to understand the discipline but also as a tool to solve problems.

Integration

The curriculum focuses on how well the courses are integrated as opposed to how many courses it has to offer. There are two aspects of integration in the curriculum—integration between courses and the integration of theory and practice. Both aspects of integration are important. Without integration between courses the curriculum becomes simply a collection of unrelated facts with no unity based on fundamental principles. The integration of theory and practice not only serves to re-enforce the students' understanding of abstract concepts but also provides them with insight and appreciation of the practical solutions at hand.

Languages and paradigms

Because of the continued evolution of programming languages and paradigms we would do our students a disservice by emphasizing only one programming language or paradigm throughout the curriculum. Students should be multilingual and should experience multiple paradigms in their undergraduate careers. Our curriculum seeks to strike the proper balance between breadth and depth. Too much breadth will not equip students with the detailed skills necessary to solve realistic problems. Too much depth in one language or paradigm will give students a narrow vision that makes it difficult to consider multiple approaches to a problem.

The curriculum emphasizes in-depth proficiency the first two years and more breadth the last two years. The balance is achieved by choosing one programming language for the first three semesters and another closely related language for the second semester of the second year. Courses in the third and fourth years introduce other programming paradigms based on different languages.

The language choice for the first two years is driven by both pedagogical and practical industry concerns. Pedagogical concerns are important during the first two years, bedcause this is when students begin to form algorithmic thinking patterns and develop problem-solving skills. The criteria are that the programming environment should be simple to learn yet powerful enough to illustrate fundamental concepts of computing. Skill in a practical language is necessary for students to be well equipped for their post graduate careers. The languages for the third and fourth years are chosen for the variety of programming paradigms on which they are based.

Intended outcomes

Upon successful completion of the program, the student should possess

- a mathematical foundation that underpins all scientific endeavors and especially the discipline of computer science. This foundation is sufficient for graduate work in computer science but not in mathematics
- a working knowledge of programming paradigms and software design principles, and programming languages that are used to implement them
- knowledge of fundamental structures in computer science such as computer architecture/organization, operating systems, and computer networks
- knowledge of a variety of other topics in computer science such as databases, artificial intelligence, and computer graphics.

The standard sequence

The courses are divided into a first year core, a second year core, and an upper division curriculum. Following is a list of the courses that are taken in the normal sequence.

First year

Math 220, Formal Methods
CoSc 220, Computer Science I

Math 210, Calculus I
Math 221, Discrete Structures
CoSc 221, Computer Science II

Second year

Math 211, Calculus II
CoSc 320, Data Structures

Math 212, Calculus III
CoSc 330, Computer Systems
Phys 210, Physics I

Third year

Math 330, Linear Algebra
Math 460, Automata Theory; or Math 510, Probability and Statistics I
CoSc 450, Programming Paradigms

CoSc 535, Operating Systems, elective *

Fourth year

Math 460, Automata Theory; or Math 510, Probability and Statistics I
CoSc 475, Computer Networks

CoSc 490, Senior Capstone
CoSc 525, Computer Organization, elective *

*Note: Only one elective required

The Computer Science minor

The Computer Science minor is satisfied by completing a core of five courses plus one elective.

Minor core

Math 220, Formal Methods
CoSc 220, Computer Science I
Math 221, Discrete Structures
CoSc 221, Computer Science II
CoSc 320, Data Structures

Minor elective

CoSc 330, Computer Systems
Math 460, Automata Theory
CoSc 450, Programming Paradigms

The contract major

Seaver College offers an alternative to the established majors for students with at least 30 units of college credit and a minimum grade point average of 2.5. The individualized major is established by faculty approval of a contract that specifies the courses that are to be taken for the contract major. The contract typically specifies courses from two separate disciplines, and includes study in each discipline with more depth than is usual for a minor. The most common contract with computer science as one of its components includes business as its other component, but disciplines as varied as religion, philosophy, art, and music have been included with computer science in the past. The student must complete at least 45 units at Seaver College following the signing of the contract for the individualized major.

The computer science part of the contract major is satisfied by completing a core of seven courses plus one elective.

Contract major core

Math 220, Formal Methods
CoSc 220, Computer Science I
Math 221, Discrete Structures
CoSc 221, Computer Science II
CoSc 320, Data Structures
CoSc 330, Computer Systems
CoSc 450, Programming Paradigms

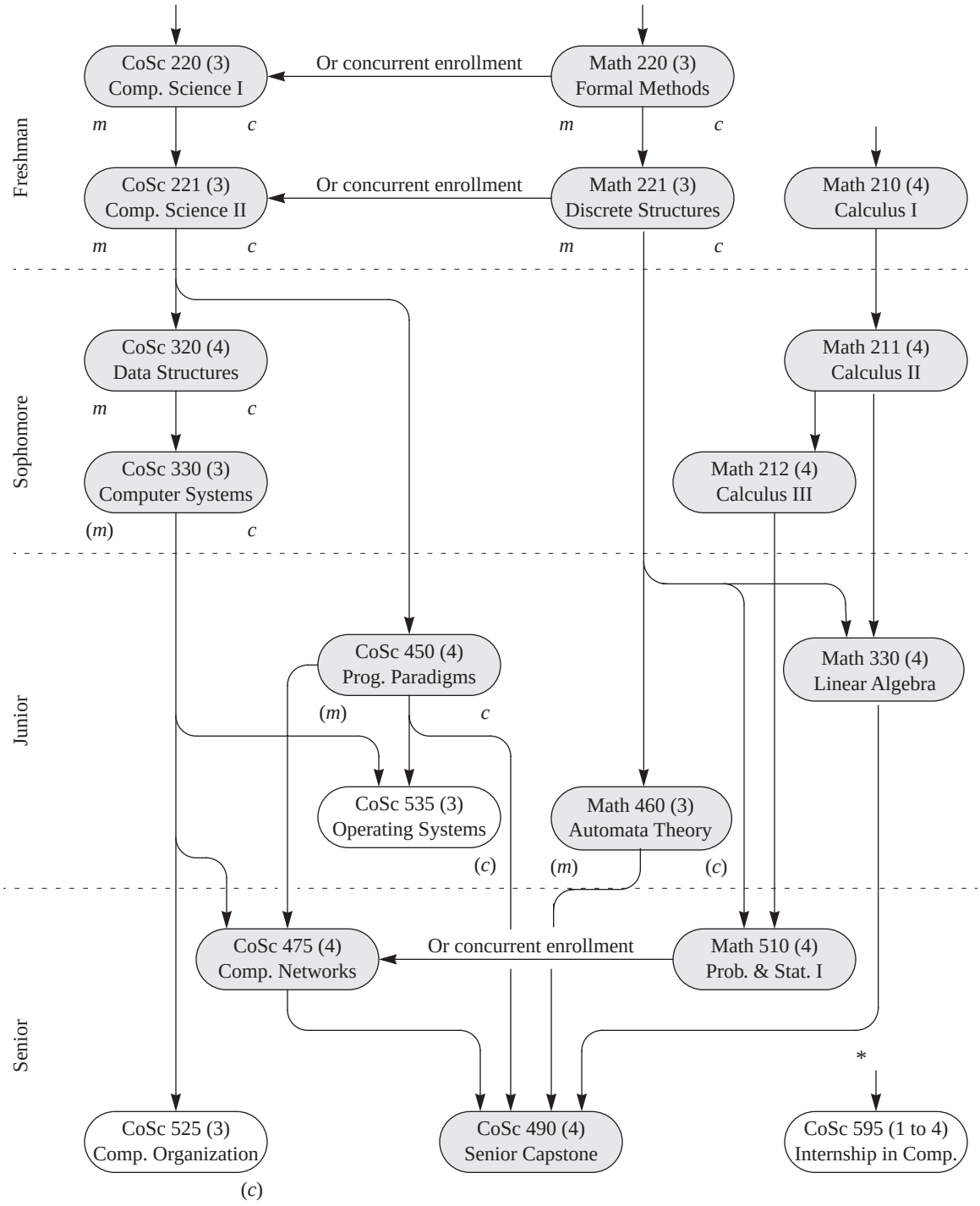
Contract major elective

Math 460, Automata Theory
CoSc 525, Computer Organization
CoSc 535, Operating Systems

The courses for the other component of the contract major are established by the faculty from the discipline of the other component.

Prerequisites

The following diagram shows the prerequisite structure of the curriculum. Semester hours are in parentheses. Required courses are shaded with one additional computer science elective plus physics required for the major. The letter *m* indicates courses required for a computer science minor and the letter *c* the courses for the computer science part of a contract major. One additional course must be chosen from the set of courses labeled (*m*) or (*c*).



*Note: Prerequisite for Internship in Computing is 90 units completed with a minimum 3.0 GPA and committee approval.

The first year

Fall semester

Math 220, Formal Methods (3)
CoSc 220, Computer Science I (3)

The courses in the first year of a computer science curriculum must serve the purpose of laying the foundation for the curriculum as a whole. Formal Methods is one such course. It is designed to teach a sound understanding of proof and a skill in formal manipulation. Its immediate application is to programming methodology. Students will discover that to know how and why a program works, they need to view a program beyond its operational semantics. Understanding how and why a program works is equivalent to proving its correctness. Proof of correctness of a program requires the program itself to be defined as a mathematical entity. Formal Methods provides students with a methodology to set up a program as a mathematical object, build it, prove its correctness, and in the process understand it. It also teaches students to think rigorously, a skill that is valuable in all courses.

A combination of operational and mathematical thinking is needed to solve programming problems. In order for the Formal Methods course to be effective, it is necessary to complement it with a course in programming where students learn the operational aspect of programs. This role is fulfilled by the Computer Science I course, which will make use of programming methodology and will serve to re-enforce students' understanding of formal methods. The content and purpose of Computer Science I and Formal Methods are integrated. Such integration is designed to provide students with a concrete framework on which they can build and develop their problem solving skills.

Currently, the programming environment for Computer Science I and II is the Qt 4 object-oriented class library based on the C++ language. The Qt 4 class library has features that reinforce the ideas from the Formal Methods course. In particular, the library is well documented and includes the Qt 4 assert macro that supports the implementation of pre- and post-conditions for Hoare triples from Formal Methods. The library permits students to begin with the procedural approach to programming, continue through successively higher levels of abstraction, and culminate with the object-oriented paradigm. It therefore satisfies our goal of students learning multiple paradigms in the curriculum. Qt 4 also provides a powerful graphical user interface designer. With this tool, students construct programs that look like professionally developed commercial software. In addition, the Qt 4 system is provided as a free open-source system that runs on MSWindows, MacOS, and Linux computers.

Spring semester

Math 210, Calculus I (4)
Math 221, Discrete Structures (3)
CoSc 221, Computer Science II (3)

Calculus is a fundamental mathematical tool for the sciences. The calculus course begins a three-semester sequence whose goal is proficiency in using this tool.

Discrete Structures presents the abstract mathematical view of fundamental data structures that form the building blocks of an algorithm. The course uses the methodology of its prerequisite course, Formal Methods, to perform complexity analysis and proof of correctness on the algorithms that manipulate data structures. It complements the concurrent Computer Science II course, which teaches the concrete implementation of data structures and algorithms in a specific programming language.

The purpose of Computer Science II is to make the transition from the procedural paradigm to the object-oriented paradigm. The steps include recursion, dynamic storage allocation, inheritance, and polymorphism. The course culminates with the implementation of abstract classes using the state design pattern as an application of object-oriented programming. Consequently, students learn abstraction by experiencing layers of abstraction, starting at the lower procedural level and progressing to the higher object-oriented level.

The second year

Fall semester

Math 211, Calculus II (4)
CoSc 320, Data Structures (4)

Data Structures continues the object-oriented paradigm from Computer Science II. The course presents data structures that are essential to programming, such as hash tables, stacks, queues, trees, and graphs, as systems of cooperating objects. This approach gives more depth to object-oriented design principles. The course continues to use an industry mainstream language, C++.

Spring semester

Math 212, Calculus III (4)
CoSc 330, Computer Systems (3)
Phys 210, Physics I (5)

Computer Systems presents a unified picture of system architecture based on four layers of abstraction: high-order, assembly, operating system, and machine. Each layer has its own language: Java, assembly language, operating system calls, and machine language respectively. The course emphasizes the relationship between the layers by exploring the translation process using finite state machines. A software project to do a translation using the Java programming language enhances students' programming skill. The switch from C++ to Java in Computer Systems furthers our goal of students learning multiple languages in the curriculum.

Physics I introduces students to the most fundamental of the experimental sciences and to the scientific method. The laws of physics provide an understanding of how physical systems work and are the basis of computer modeling of the physical world.

The third year

Fall semester

Math 330, Linear Algebra
Math 460, Automata Theory
CoSc 450, Programming Paradigms

Linear algebra presents matrix algebra, vector spaces, and linear transformations. Its goal is to provide students with the skill to apply these mathematical tools to problems in computer science such as graphics, computer modeling, and numerical methods.

The goal of Automata Theory is to answer the question, What is computability, and what are its limits? Students learn the abstract models that help to answer these questions—finite automata, pushdown automata, and Turing Machines. The course prepares students for more advanced work in theoretical computer science and serves as a basis for applied work in language design and compiler construction.

Programming Paradigms introduces three major programming models that complement the procedural and object-oriented approaches: the functional, declarative, and concurrent models. Each model is presented in the context of an associated programming language: Lisp, Prolog, and Java respectively. This course furthers the goal of providing experiences in multiple paradigms and languages in the undergraduate curriculum.

Spring semester

CoSc 535, Operating Systems

Operating Systems is a continuation in more depth of the topic introduced in the Computer Systems course. Operating systems are presented as multi-layer components that hide the details of hardware implementation in order to master the complexity of resource management. The course furthers the theme of abstraction in the curriculum and serves as an example of the power of abstraction in software design.

The fourth year

Fall semester

Math 510, Probability and Statistics I
CoSc 475, Computer Networks

Probability and Statistics I introduces continuous and discrete probability distributions. The course provides the student with the mathematical tools to solve problems in computer modeling and performance analysis of computer networks and systems.

Computer Networks presents data communication and networking principles based on the seven-layer OSI reference model. It provides students with software and hardware tools to simulate and test various networking environments. It includes the use of mathematical tools from the probability course to construct queueing models for performance analysis of network protocols, furthering the goal of integration between courses in the curriculum. That Computer Networks is a required course in the curriculum is a reflection of the growing importance of networks in our society as exemplified by the ubiquity of the Internet.

Spring semester

CoSc 490, Senior Capstone
CoSc 525, Computer Organization
CoSc 595, Internship in Computing

The Senior Capstone course centers on a large team-based software project that requires students to integrate their knowledge and experience from previous courses. One component of the course will be content-based whose topic will vary from year to year. Its purpose is to give students an experience that represents what they are likely to encounter in the computer industry.

Computer Organization is a continuation in more depth of the topic introduced in the Computer Systems course. The hardware organization of computers is presented in multiple layers of abstraction and includes the logic gate layer, the microprogramming layer, and the machine layer. The course is yet another example of the pervasive theme of abstraction throughout the curriculum.

Internship in Computing provides students with a practicum in industry. It serves as a bridge between the academic world and the professional world.

Math 220
Formal Methods

Catalog course description

MATH 220. Formal Methods (3)
Formal logic as a tool for mathematical proofs. Propositional calculus—Boolean expressions, logic connectives, axioms, and theorems. Predicate calculus—universal and existential quantification, modeling English propositions. Application to program specification, verification, and derivation.

Topics

Hours*	Topic
	Preliminaries (3 hours)
1.5	Textual substitution
1.0	Leibniz rule and function evaluation
0.5	The assignment statement
	Boolean expressions (5 hours)
1.5	Syntax and evaluation of boolean expressions
0.5	Equality vs equivalence
1.0	Satisfiability, validity, and duality
2.0	Modeling English propositions
	Propositional calculus (6 hours)
1.0	Equational logic and inference rules
0.5	Equivalence
1.0	Negation and inequivalence
2.0	Disjunction and conjunction
1.5	Implication
	Proof techniques and applications (4 hours)
1.0	Monotonicity, deduction theorem, case analysis
0.5	Proof by contradiction
0.5	Proof by contrapositive
2.0	Solving word problems
	Quantification (4 hours)
0.5	Types
1.0	Syntax and interpretation of general quantification
1.5	General quantification axioms
1.0	Quantification range theorems
	Predicate calculus (5 hours)
1.5	Universal quantification
1.5	Existential quantification
2.0	Formalizing English statements
	Predicates and programming (8 hours)
2.0	Specification of programs
2.0	Reasoning about the assignment statement
2.0	Calculating parts of assignments
2.0	Conditional statements and expressions

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 220
Computer Science I

Catalog course description

COSC 220. Computer Science I (3)
Introduction to programming with an object-oriented library using C++. Input/output—graphical user interfaces based on the model/view/controller paradigm. Programming constructs—sequential, conditional, iterative. Data abstraction—abstract data structures, stacks and lists as abstract data types. Procedural abstraction—proper procedures, function procedures. Basic algorithms and applications—random numbers, iterative array searching and sorting. Prerequisite: MATH 220 or concurrent enrollment.

Topics

Hours*	Topic
	The BlackBox framework (3 hours)
0.5	Text editor
2.5	Languages and grammars, EBNF
	Graphical user interface (5 hours)
0.5	The output Log as an abstract data structure
0.5	Sequential statements
1.0	Numeric types as abstract data types
0.5	String types as abstract data types
0.5	The assignment statement
1.0	RECORD types
1.0	Interactive input/output with dialog boxes
	Data abstraction (4 hours)
1.0	Using stacks as abstract data types
2.0	Applications to prefix, infix, and postfix expressions
1.0	Using lists as abstract data types
	Conditional statements (5 hours)
0.5	Boolean expressions
1.0	IF statements
1.0	Boolean types as abstract data types and check boxes
1.0	Nested IF statements
0.5	ASSERT statements
1.0	CASE statements and radio buttons
	Text input/output (3 hours)
0.5	The model/view/controller paradigm
1.5	Text input from the focus window
1.0	Text output to a window
	Loop statements (5 hours)
1.5	WHILE statements
1.5	FOR statements
2.0	Nested loops
	Procedural abstraction (4 hours)
1.5	Writing function procedures
2.5	Writing proper procedures
	Basic algorithms and applications (6 hours)
1.0	Random numbers
1.0	One-dimensional arrays, stack and list implementations
2.0	Iterative array searching and sorting
2.0	Two-dimensional arrays

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

Math 221
Discrete Structures

Catalog course description

MATH 221. Discrete Structures (3)
Application of formal methods to discrete analysis—mathematical induction, the correctness of loops, relations and functions, combinatorics, analysis of algorithms. Application of formal methods to the modeling of discrete structures of computer science—sets, binary trees. Prerequisite: MATH 220.

Topics

Hours*	Topic
	Set Theory (5 hours)
1.0	Set comprehension and membership
2.0	Set cardinality, subset, complement, union, intersection, difference, power set
1.5	Properties of set operations
0.5	Families of sets, axiom of choice, paradoxes
	Mathematical induction (6 hours)
1.5	Induction over the natural numbers
1.5	Inductive definitions
1.0	Binary trees
2.0	Correctness of loops
	Relations and functions (11 hours)
1.0	Tuples and cross products
2.0	Relations, domain, range, product, inverse
1.5	Properties—reflexive, irreflexive, symmetric, antisymmetric, asymmetric, transitive
1.5	Equivalence relations, equivalence classes
1.5	Functions—determinate, total, one-to-one, onto
1.5	Function inverse
2.0	Partial orders, posets, glb, lub
	Combinatorial analysis (4 hours)
2.0	Sum and product rules, permutations and combinations
2.0	Pigeonhole principle, applications
	Recurrence relations (3 hours)
1.5	Homogeneous difference equations
1.5	Closed solutions of inductive definitions
	Analysis of algorithms (6 hours)
2.0	Space/time complexity
4.0	Asymptotic behavior: big oh, big omega, big theta

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 221
Computer Science II

Catalog course description

COSC 221. Computer Science II (3)
Introduction to object-oriented programming. Recursion—basic algorithms, array searching and sorting. Dynamic storage allocation—pointer types, linked lists and binary search trees as abstract data types. Classes—objects, abstract classes, inheritance and polymorphism, linked lists and binary trees as classes. Prerequisites: MATH 221 or concurrent enrollment and COSC 220.

Topics

Hours*	Topic
	Recursion (10 hours)
2.0	The run-time stack and call tree for procedures
1.0	The run-time stack and call tree for functions
3.0	Recursive functions—factorial, summation, binomial coefficient
4.0	Recursive procedures—towers of hanoi, binary search, quick sort
	Dynamic storage allocation (9 hours)
2.0	Pointer types—the NEW procedure, pointer assignment
1.0	Linked nodes
3.0	Linked lists as abstract data types—iterative and recursive implementations
3.0	Binary search trees as abstract data types—iterative and recursive implementations
	Classes (16 hours)
2.0	Objects and methods—using lists and trees as objects
2.0	Building simple objects and writing methods
4.0	Building simple abstract classes, inheritance and polymorphism
4.0	The state design pattern implementation of linked lists
4.0	The state design pattern implementation of binary search trees

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 320
Data Structures

Catalog course description

COSC 320. Data Structures (4)
Abstract data types, classes, and design patterns with C++. Sorting algorithms—insertion sort, merge sort, heapsort, quicksort. Linear data structures—stacks, queues, linked lists. Hash tables. Trees—binary search trees, 2-3 trees, B-trees, abstract syntax trees. Disjoint sets. Graphs—search algorithms, spanning trees, Kruskal's and Dijkstra's algorithms. Prerequisite: COSC 221.

Topics

Hours*	Topic
	Introduction (6 hours)
2.0	Procedural features of C++
2.0	Object-oriented features of C++
2.0	Review of asymptotic behavior and recurrences
	Sorting algorithms (5 hours)
1.0	Insertion sort
1.0	Merge sort
2.0	Heapsort, priority queues
1.0	Quicksort
	Linear data structures (10 hours)
1.0	Array implementation of stack and queue ADTs
2.0	Pointer implementation of linked lists
1.0	Linked implementation of stack and queue ADTs
2.0	State design pattern for linked lists
4.0	Strategy design pattern for stack and queue ADTs
	Hash tables (3 hours)
2.0	Collision techniques
1.0	Hashing functions
	Trees (10 hours)
0.5	Mathematical definitions
1.0	Binary search tree ADTs
3.0	State design pattern for binary search trees
2.0	2-3 trees
1.0	B-trees
1.0	Abstract syntax trees
1.5	Recursive composition design pattern
	Disjoint sets (2 hours)
2.0	Union and set membership algorithms
	Graphs (11 hours)
0.5	Mathematical definitions
1.5	Adjacency list and adjacency matrix representations
2.0	Breadth-first and depth-first searches
1.0	Minimum spanning trees
2.0	Kruskal's algorithm
2.0	The shortest path problem
2.0	Dijkstra's algorithm

Total: 47.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 330
Computer Systems

Catalog course description

COSC 330. Computer Systems (3)
A study of computers as multi-level systems. The machine level—binary representations, instruction sets, von Neumann machines. The assembly level—addressing modes, compiling to the assembly level, language translation principles. The operating system level—loaders, interrupts. Prerequisite: COSC 320.

Topics

Hours*	Topic
	The machine level (9 hours)
1.5	Unsigned binary representation
1.5	Two's complement binary representation
1.0	Hexadecimal and character representation
2.0	von Neumann machines
1.5	Character input/output and direct addressing
1.5	Programming in machine language
	The assembly level (13 hours)
2.0	Assemblers
1.0	Decimal input/output and immediate addressing
1.0	Symbols
1.0	Assignment statements
3.0	Branching instructions and flow of control
3.0	Stack-relative addressing and procedure calls
2.0	Indexed addressing and arrays
	Language translation principles (9 hours)
2.0	Languages, grammars, and parsing
2.0	Finite state machines
2.0	Implementing finite state machines
3.0	Code generation
	The operating system level (4 hours)
2.0	Loaders
2.0	Interrupts

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

Math 460
Automata Theory

Catalog course description

MATH 460. Automata Theory (3)
Theoretical models of computation. Finite automata—regular expressions, Kleene's theorem, regular and nonregular languages. Pushdown automata—context-free grammars, Chomsky normal form, parsing. Turing machines—the halting problem. NP-complete problems. Prerequisite: MATH 221.

Topics

Hours*	Topic
	Finite automata (12 hours)
2.0	Regular expressions
2.0	Deterministic finite automata
2.0	Kleene's theorem
2.0	Nondeterministic finite automata
2.0	Regular and nonregular grammars
2.0	Decidability—the pumping lemma
	Pushdown automata (12 hours)
3.0	Context-free grammars
2.0	Chomsky normal form
3.0	Pushdown automata
2.0	Parsing
2.0	Decidability
	Turing machines (9 hours)
3.0	Turing machine models
3.0	Reduction
3.0	Decidability—the halting problem
	Computational complexity (2 hours)
2.0	NP-complete problems

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 450
Programming Paradigms

Catalog course description

COSC 450. Programming Paradigms (4)
A study of three programming paradigms and their associated languages: the functional paradigm with Common Lisp, the logical/declarative paradigm with Prolog, and the concurrent processing paradigm with Java. Prerequisite: COSC 221.

Topics

Hours*	Topic
	Functional paradigm with Common Lisp (16 hours)
4.0	Common Lisp syntax, lists, S-expressions, recursion
4.0	Functions, lambda-expressions, closures
4.0	Function mapping, control blocks, lexical scoping, dynamic scoping
4.0	Input/Output, macros, symbolic processing
	Logical/declarative paradigm with Prolog (15 hours)
3.0	Prolog syntax, unification and variable instantiation, back-tracking
1.0	Declarative and procedural semantics
2.0	Static and dynamic predicates
1.0	Input/Output
4.0	Lists and structures
4.0	Application to artificial intelligence
	Concurrent processing paradigm with Java (16 hours)
4.0	Java syntax, Java threads
1.0	Process synchronization
2.0	Deadlock, live-lock, and indefinite postponement
2.0	Semaphores
2.0	Monitors
3.0	Synchronization design patterns
2.0	Java event-driven programming

Total: 47.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 475
Computer Networks

Catalog course description

COSC 475. Computer Networks (4)
The theory of computer networks and its applications. Network layers and protocols for the OSI reference model. TCP/IP and the Internet. Network programming using Java. Rudiments of queueing theory. Prerequisites: MATH 510 or concurrent enrollment, COSC 450, and COSC 330.

Topics

Hours*	Topic
	Introduction (4 hours)
1.0	The OSI reference model
3.0	The Java network programming model
	The physical layer (4 hours)
2.0	Network hardware
2.0	Theoretical basis for data communication
	The data link layer (6 hours)
3.0	Error detection and correction algorithms
3.0	Data link protocols
	The medium access sublayer (4 hours)
2.0	Multiple access protocols
2.0	IEEE standards and bridges
	The network layer (6 hours)
1.5	Routing algorithms
1.5	Congestion control algorithms
1.0	Internetworking
1.0	The network layer in the Internet
1.0	Fragmentation methods
	The transport layer (5 hours)
2.0	Transport protocols
3.0	The Internet transport protocols (TCP/IP and UDP)
	The application layer (7 hours)
2.0	Network security
2.0	Domain Name System (DNS)
3.0	Simple Network Management Protocol (SNMP)
	Distributed programming (7 hours)
3.0	CORBA programming
4.0	Java RMI programming
	Queueing models (4 hours)
1.5	Homogeneous birth-death processes
2.5	The M/M/1 system

Total: 47.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 490
Senior Capstone

Catalog course description

COSC 490. Senior Capstone (4)
A large software team project based on a topic that may vary from year to year and which builds on one or more of the prerequisites. Possible topics include but are not limited to database, computer graphics, artificial intelligence, compiler construction, distributed computing. Oral presentation required. Prerequisites: COSC 475, COSC 450, MATH 330, and MATH 460.

Topics

Hours*	Topic
	Topic (23 hours)
23.0	A current topic in computer science that builds on one or more of the prerequisites. The topic is the basis of the team project in the other part of the course. The topic will be presented in sufficient depth to be appropriate for a senior level course. It will integrate students' prior knowledge and extend their understanding of a subfield of computer science.
	Team project (23 hours)
23.0	Teams will be organized in groups of two or three students. The scope of the project will be large enough that it cannot be completed by a single person during one semester. Most of the project activities will take place outside the classroom, with class time used for discussion of technical or theoretical questions that arise in the development process and regular progress reports by team members. The final product will include written documentation of the software design and a user manual.
	Program assessment (1 hour)
1.0	Assessment of computer science/mathematics undergraduate program

Total: 47.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 525
Computer Organization

Catalog course description

COSC 525. Computer Organization (3)
Hardware organization and design. The logic gate level—combinational and sequential circuits and devices. The microprogramming level—microarchitecture, microprograms. The machine level—CPU designs, instruction formats, addressing modes, floating point formats. Parallel architectures. Occasional laboratory sessions. Prerequisite: COSC 330.

Topics

Hours*	Topic
	The logic gate level (13 hours)
2.0	Boolean algebra and logic gates
2.0	Combinational analysis
2.0	Combinational design
2.0	Combinational devices
2.0	Latches and clocked flip-flops
2.0	Sequential analysis
1.0	Sequential design
	The microprogramming level (12 hours)
1.0	The data section of the CPU
1.0	The control section of the CPU
2.0	Microinstructions—timing, sequencing
2.0	Macroinstructions
1.0	Microassembly language
2.0	Microprograms, nanoprograms
1.0	Pipelining
2.0	Cache memory
	The machine level (7 hours)
2.0	Instruction formats
2.0	Addressing modes
1.0	Flow of control
2.0	IEEE floating point standards
	Parallel architectures (3 hours)
1.0	Taxonomy of parallel computers
1.0	Single instruction, multiple data architectures
1.0	Multiple instruction, multiple data architectures

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 535
Operating Systems

Catalog course description

COSC 535. Operating Systems (3)
Operating Systems design and implementation—process management, device management, memory management, file management, protection and security. Prerequisites: COSC 330 and COSC 450.

Topics

Hours*	Topic
	Overview of operating system concepts (3 hours)
1.5	System calls
1.5	OS structures
	Process management (7 hours)
2.0	Process synchronization
2.0	Inter-process communication
2.0	Process scheduling policies
1.0	Examples
	Device management (7 hours)
2.0	Hardware—input/output, direct memory access
2.0	Software—device drivers
2.0	Deadlocks
1.0	Examples
	Memory management (7 hours)
2.0	Virtual memory
2.5	Paging algorithms
1.5	Segmentation
1.0	Examples
	File management (7 hours)
2.0	Low-level files
2.0	Structured files
3.0	Database management systems
	Protection and security (4 hours)
1.0	Authentication
1.0	Authorization
2.0	Encryption

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours

CoSc 105
Introduction to Programming

Catalog course description

COSC 105. Introduction to Programming (3)
Introduction to programming with C++. Data types—numeric, character, the string class, boolean. Input/output stream classes—interactive I/O, file I/O. Programming constructs—sequential, conditional, iterative. Functions—parameter passing mechanisms, function libraries. Arrays—one-dimensional arrays, searching and sorting, two-dimensional arrays. Introduction to classes.

Topics

Hours*	Topic
	The computing environment (1 hour)
0.5	Text editor
0.5	Compiler
Data types and input/output (6 hours)	
1.0	Numeric types—int, long, float, double
0.5	The assignment statement
1.0	Operations on numeric types
1.0	Numeric input/output using cin and cout
1.0	Character type and the string class
1.0	Operations on character and string types
0.5	Character and string input/output using cin and cout
	Programming constructs (12 hours)
1.0	Boolean expressions and the boolean type
1.5	IF statements
1.5	Nested IF statements
1.0	The switch statement
3.0	The while loop
2.0	File input/output using fstream
2.0	The for loop
	Functions (7 hours)
1.0	Functions that return void
2.5	Parameters—passing by value, passing by reference
1.5	Functions that return a value
2.0	Function libraries, separate compilation and reuse
	Arrays (7 hours)
1.0	One-dimensional arrays
2.0	Search algorithms—sequential search, binary search
2.0	Iterative sorting algorithms
2.0	Two-dimensional arrays, applications to vectors and matrices
	Data abstraction (2 hours)
2.0	Classes

Total: 35.0 hours, excluding holidays, review sessions, and exams

*Fifty-minute class hours