Learning Objectives
For courses in
The Department of Computer Science
Oregon State University

April 1, 2002

This document describes the overall learning objectives for courses in the department of computer science at Oregon State University.

Overall Learning Objectives for the Computer Science Degree

The undergraduate program in Computer Science at Oregon State University seeks to prepare graduates to understand the field of computing, both as an academic discipline and as a profession. Achieving this goal involves imparting both a broad comprehension of the discipline as a whole, as well as the acquisition of specific skills used by practitioners of the discipline. It requires an appreciation of both the theory that underlies the field, and the application of that theory to practical problems. And it demands that graduates be aware of the role, importance and impact of computing within the larger society.

Computer Science is simultaneously a mathematical, a scientific, and an engineering discipline. The skills required of those who would work in the field derive from all three traditions. This can be seen in the complex interplay between three significant fundamental concepts at the heart of computer science: theory, abstraction and design.

The concept of *theory* grows out of the tradition of computing as a mathematical science. Like mathematicians, computer scientists will on occasion work in a world of theorems, hypotheses, proofs, and formal models.

The concept of *abstraction* is rooted in the experimental sciences. Like a physicist or a chemist, a computer scientist will often take a complex system and purposely hide, or abstract, certain features in order to make other elements more prominent. Indeed, abstraction is the key tool used in the creation and understanding of complex systems.

Like all engineering disciplines, the concept of *design* is fundamental to computer science. Producing an elegant design is not haphazard, but requires a careful evaluation of requirements, specifications, an appreciation of the various alternatives and their implications, and an evaluation mechanism (testing or analysis) to assess the result.

At a broad level, the general characteristics of our computer science graduates must include the following:[1]

A system-level perspective. While a formal education tends to emphasize fragmented and isolated concepts and skills, the graduate of a computer science program must understand not only the pieces, but also how the pieces fit together to make a whole.

An appreciation of both theory and practice. The graduate must understand both the fundamental concepts and results of theory, and the relevance and application of theory to practical situations.

[1] This list is adapted from Chapter 11 of the Draft ACM Computing Curricula 2001.

Recognition of common themes. There are many themes that occur repeatedly throughout computer science. Examples include recursion and repetition, complexity, modeling, time-space tradeoffs, and abstraction. The student should understand the common features of these themes, and their application to specific situations.

Team project experience. Students must understand that the solution to complex problems almost always involves the combined efforts of many individuals with differing skills and backgrounds. Experience with working in such teams must be an essential part of the student experience.

An awareness of change. Graduates must possess both a solid foundation in Computer Science and an awareness and willingness to deal with rapid changes in the field.

Almost no area of human endeavor experiences change at a rate comparable to that of Computer Science. While many other academic disciplines must confront an explosive growth in new information, typically the transformations wrought by these innovations occur at the edges of their respective fields, well beyond the accepted core of basic knowledge. In contrast, in the field of computer science we continually face revolutionary changes that cut to the heart of our discipline. In just the past decade, for example, we have witnessed the following developments:

- The rise of the Internet as a truly world-wide communication medium

- The development of the world-wide-web as a tool for commerce, education, and knowledge distribution

- Object-oriented programming becoming the dominant programming paradigm

- Computing in embedded systems becoming an intrinsic part of an ever larger number of everyday appliances, and the increasing interconnections between these devices

- An increased awareness on the part of society as a whole in both the benefits and dangers of a networked society, including concerns over security and safety

Because change is fundamental, any description of a curriculum in computer science must necessarily represent only a static snapshot of a constantly evolving scheme. A student graduating with a degree in computer science in 1970 might never have heard the term “structured programming.” The student in 1980 would almost certainly not have been taught object-oriented programming. The graduate of 1990 would not know about the World-Wide-Web. With this history it is clear that the one assurance we can give any student of computer science is the guarantee that at least some of the knowledge they gain during their academic preparation will at some point in their career be superseded by new innovations.

It is therefore clear that the most important skill that we must impart to our graduates is an approach to learning. For a computer scientist learning is not something that is done in college and then applied throughout a lifetime of work; instead learning must be a continual and lifelong process. It is therefore imperative that we teach not only specific information, but we must also impart to the student how to accommodate new information on their own once they leave college.

The forgoing must not be construed as asserting that knowledge in computer science is a type of quicksand, a mirage that will vanish and reappear in a different form every few years. Decades of experience as a discipline have shown that there is a fundamental body of knowledge that supports the field of computer science, and any student of computer science should be exposed to this information. One synopsis of this body of knowledge can be found in the curriculum guidelines published by the *Association for Computing Machinery*, the leading professional organization for computer science. In their curriculum guidelines they divide the subject matter appropriate for computer science into several categories, with suggested topics within each category. These categories include the following:

1. Algorithms and Data Structures
2. Architecture
3. Computational Science and Numeric Methods
4. Operating Systems
5. Programming languages
6. Software Methodology and Engineering
7. Social, Ethical and Professional

Within each category a specific set of topics are described. In the following sections we summarize these categories, and illustrate where within our own curriculum each topic is discussed. The number given in square brackets ties the concept to a specific learning objective for the course. For example, CS 160[5] means that the topic is addressed by learning objective number 5 in the course CS 160. These values are omitted for courses from other departments, since we do not yet have detailed learning objectives for these. (A section at the end of this document presents catalog descriptions for courses given in other departments that are part of our program.) A parenthesis around the course means the topic is discussed in the given course, but the course is not required for graduation.

Algorithms and Data Structures

Topic	Discussed in Course
Basic Data Structures	CS 151[5], CS 162[6], CS 261[4]
Abstract Data Types	CS 261[3]
Recursive Algorithms	Math 231, CS 325[1]
Complexity Analysis	CS 162[4], CS 261[3]. CS 325[9]
Complexity Classes	CS 325[6]
Sorting and Searching	CS 162[7], 261[6], 325[5]

Computability and Undecidability	CS 321[5]
Problem-Solving Strategies	CS 161[7], CS 325[7]. CS 361[1]
Parallel and Distributed Algorithms	(CS 475[1])

Architecture

Topic	Discussed in Course
Digital Logic	CS 160[5], ECE 271, ECE 375
Digital Systems	CS 472[1], ECE 271, ECE 375
Machine Level Representation of Data	CS 160[6], CS 151[5], CS 472[4], ECE 271, CS 480[7]
Assembly Level Machine Organization	CS 160[7], CS 472[2], ECE 271
Memory System Organization and Architecture	CS 472[6], ECE 271
Interfacing and Communication	CS 372[2]
Alternative Architectures	CS 472[5], (CS 475[4])

Computational Science and Numeric Methods

Topic	Discussed in Course
Number Representation, Errors, and Portability	CS 151[5], CS 160[6], ECE 271, Math 351
Iterative Approximation Methods	Math 351

Operating Systems

Topic	Discussed in Course
History, Evolution, and Philosophy	CS 311[2]
Tasking and Processes	CS 311[4], CS 411[3]
Process Coordination and Synchronization	CS 311[3], CS 411[3]
Scheduling and Dispatch	CS 311[3], CS 411[3]
Physical and Virtual Memory Organizations	CS 311[2], CS 411[1]
Device Management	CS 311
File Systems and Naming	(CS 312[5]), CS 411[1]
Security and Protection	(CS 372[7]), CS 411[1]
Communications and Networking	CS 311[6], (CS 372[1]), CS 411[1]
Distributed and Real-time Systems	CS 472

Programming Languages

Topic	Discussed in Course

Representation of Data Types	CS 151[5]. CS 160[6], CS 480
Sequence Control	CS 151[3], 161[4], 162[3], CS 381[8]
Data Control, Sharing, and Type Checking	CS 381[3]
Finite State Automata and Regular Expressions	CS 311[7], CS 321[1,2], CS 480[5]
Context-Free Grammars and Pushdown Automata	CS 321[3]
Language Translation Systems	CS 480[6]
Programming Language Semantics	CS 381[9]
Programming Paradigms	CS 381[8]
Distributed and Parallel Programming Constructs	(CS 475[4])

Software Methodology and Engineering

Topic	Discussed in Course
Fundamental Problem-solving Concepts	CS 160[9], CS 325[7]
The Software Development Process	CS 361[1], CS 362[5]
Software Requirements and Specifications	CS 361[3], CS 362[5]
Software Design and Implementation	CS 161[7], CS 361[4], CS 362[6]
Verification and Validation	CS 161[7], CS 361[4], CS 362[1,2]

Social, Ethical, and Professional Issues

Topic	Discussed in Course
Historical and Social Context of Computing	CS 391[1]
Responsibilities of the Computing Professional	CS 160[10], CS 391[2]

Specific Learning Objectives for Courses in Computer Science

In the following sections we describe the specific learning objectives for each course in the Computer Science program. Each description is also accompanied by the Catalog description for the course. Taken together, these represent a concise description of the course, its goals and objectives.

CS101: **Computers: Applications and Implications**

Catalog Description

> The varieties of computer hardware and software. The effects, positive and negative, of computers on human lives. Ethical implications of information technology. Hands-on experience with a variety of computer applications, including multimedia and Internet communication tools. (4 credits)

On completion of the course, students must demonstrate the ability to:

1. Describe the functions of the hardware components,
2. Describe different computer systems and their purposes,
3. Describe different types of software and their purposes,
4. Describe networking and telecommunication concepts and protocols,
5. Demonstrate general understanding in several computing fields such as computer security, artificial intelligence, software system design, and robotics.
6. Describe social and ethical implications of computers in different aspects of our society and life.
7. Demonstrate familiarity and ability to use:
   - Word processing application
   - Spreadsheet application
   - Presentation graphics application
   - Image editing application
   - Web browser and search applications
   - Web authoring application
   - Email application

CS 151: **Introduction to Programming in C**

Catalog Description

Thorough treatment of the basic elements of C, bitwise operations, flow of control, input/output, functions, arrays, strings, and structures. (4 credits)

Upon completion of this course students will be able to understand and demonstrate:

1. Rules for defining valid identifiers in the C language. Create identifiers for of variables and functions. Recognise invalid identifiers.

2. Rules for creating and evaluating expressions using the arithmetic, relational, logical and reference operators in the C language. Evaluate expressions and create appropriate C language expressions from human language statements.

3. Behaviour of C as a procedural language and a program as a linear sequence of steps. Evaluate conditional, selection, and repetitive control statements. Create appropriate conditional, selection, or repetitive control statements as required to accomplish a programming task.

4. Function use and definition. Use pre-defined functions. Prototype functions. Write functions. Demonstrate and use scope rules for local identifiers. Distinguish the role and use of formal and actual arguments.

5. Role of data types. Explain the distinction between integer and floating point data types. Select the appropriate data type based on the relative range and or accuracy of the data to be represented. Use single dimensional arrays as an ordered collection of data of a single type. Use multiple dimensional arrays of a single data type. Use structures, unions, and enumerated data types. Use pointers, references and de-references to data. Demonstrate the use of NULL. Use the typdef keyword.

Submitted by Mike Johnson, March 2001

CS 160: **Computer Science Orientation**

Catalog Description

> Introduction to the computer science field and profession. Team problem solving. Social and ethical issues surrounding use of computers. (3 credits)

On completion of the course, students must demonstrate the ability to:

1. Set up a user account on the College of Engineering network. Describe and use the basic features of an email program and a Web browser.

2. Use the library's electronic catalog to find a book's Library of Congress call number, given its title or its author. Use the library's web-based database to locate magazine and journal articles matching multiple subject criteria.

3. Plan a course of study that satisfies all requirements for a B.S. in computer science.

4. Work as a member of a team to complete written homework assignments and a programming project.

5. Determine the output of simple logical circuits containing AND, OR, XOR, and NOT gates.

6. Represent, add, and subtract integers and recognize overflow. Represent, add, and subtract floating-point numbers and recognize overflow and round-off error. Represent ASCII characters and machine language instructions.

7. List the four basic computer hardware functional units (CPU, memory, input, and output) and describe how they interact.

8. Write simple machine language programs and describe and three phases in the execution of a machine language instruction.

9. Describe what an algorithm is, represent simple algorithms in pseudocode, perform simple worst case analysis of algorithms, and describe a problem for which there is no algorithm.

10. Evaluate ethical problems using both ends-oriented and means-oriented reasoning.

11. Describe several career options available to B.S. graduates in computer science

CS 161: **Introduction to Computer Science**

Catalog Description

> Overview of fundamental concepts of computer science. Introduction to problem solving, software engineering and object-oriented algorithm development and programming. PREREQ: CS 151 or equivalent. COREQ: MTH 231.

Upon completion of this course students will be able to understand and demonstrate:

1. Demonstrate the rules for defining valid identifiers in a programming language by creating identifiers for variables, methods or functions, classes, and objects.

2. Demonstrate the rules for creating and evaluating arithmetic, relational, and logical expressions by evaluating and creating valid, appropriate expressions from human language statements.

3. Create appropriate control statements as required to accomplish a program task, including the interception and handling of error conditions.

4. Recognize and understand the importance of rudimentary software engineering design principles and software quality factors.

5. Recognize and understand the relationship between the software engineering design principles and software quality assurance and testing.

CS 162: **Introduction to Computer Science II**

Catalog Description

> Basic data structures. Computer programming techniques and application of software engineering principles. Introduction to analysis of programs. PREREQ: CS 161, MTH 231.

On completion of the course, students must demonstrate the ability to:

1. Demonstrate an understanding of abstraction, modularity, separation of concerns by decomposing a process into an appropriate sequence of functions.

2. Understand abstract data types, classes, objects, inheritance, and encapsulation.

3. Describe the characteristics of an algorithm.

4. Describe the asymptotic execution time of simple algorithms (big O notation).

5. Understand software testing techniques.

6. Demonstrate an understanding of basic linear structures and describe instances appropriate for their use.

7. Compare and contrast iterative and recursive algorithms and demonstrate an understanding conditions appropriate for their use.

CS 252: **User Interface Design**

Catalog Description

> Introduction to the basic principles of user interface design and evaluation. Includes the use of interactive devices, layout, symbolism, color and other interface characteristics, tools and methods for evaluating interfaces, and related topics from human factors and usability engineering. (4 credits) Prereq: CS 161 or CS 295.

Upon completion of this course, students will be able to understand and demonstrate:

1. The basic principles of human-computer interactions and how they affect the usability of software interfaces.

2. The ability to apply guidelines based on those principles in the design of user interfaces.

3. The ability to design, build, and evaluate computer interfaces using methods from usability engineering.

4. The adherence to the standard coding practices employed in software companies.

5. Productivity and cooperation on team-based software engineering projects.

CS 261: **Data Structures**

Catalog Description

> Complexity analysis. Approximation Methods. Trees and graphs. File processing. Binary search trees. Hashing. Storage management. (4 credits) Prereq: CS 162, Math 232.

On completion of the course, students must demonstrate the ability to:

1. Demonstrate an understanding of abstraction, modularity, separation of concerns, classes, objects, inheritance, correctness, reliability, robustness, ease of use, efficiency, modifiability, and encapsulation by their considered implementation in programs in an appropriate programming language.

2. Determine the asymptotic complexity of an algorithm and be able to compare the execution time of competing algorithms based on their asymptotic complexity, space utilization, and other factors.

3. Demonstrate an understanding of abstract data types, such as stacks, queues, priority queues, tree structures, and hash tables, by implementing them in appropriate language.

4. Demonstrate an understanding of and the ability to implement software testing techniques.

5. Demonstrate an understanding of recursion by its appropriate use in software engineering assignments.

CS 262: **Programming Projects in C++**

Catalog Description

> Learning a second computer programming language. Elements of C++. Object-oriented programming. Experience team work on a large programming project. Prereq: CS 261.

On completion of this course, the students should have demonstrated:

1. Use of good programming practices and a consistent programming style.
2. Proficiency with builtin types and control structures in C++.
3. Proficiency with basic container types in the standard template library.
4. Proficiency in using streams for basic input/output operations.
5. Proficiency in designing concrete and abstract classes.
6. Proficiency in using pointers.
7. Proficiency with basic memory management in C++.
8. Ability to use inheritance and virtual functions.
9. Ability to design simple generic types and functions.

CS 271: **Computer Architecture and Assembly Language**

Catalog Description

Introduction to functional organization and operation of digital computers. Coverage of Assembly language; addressing, stacks, argument passing, arithmetic operations, decisions, macros, modularization, linkers and debuggers. PREREQ: CS 161, MTH 231.

CS 295: **Introduction to Web Authoring**

Catalog Description

> Techniques and tools for designing and publishing on the World Wide Web; hypertext and HTML; site and page design; media integration; issues raised by Internet publishing. 4 credits.

By the completion of this course, students will be expected to:

1. Formulate the basic HTML and web design principles and apply them to web page creation, including Tags, lists, tables, images, links, fonts

2. Understand the limitations associated with the web media and be able to articulate the trade off between lowest common denominator and high end web presentations

3. Experience working in a group to present materials. And develop intra group communication skills

4. Understand the life cycle of Web page development.

5. have the basic graphic art skills to create and manipulate images for web sites

6. Clearly understand how to communicate a story via web pages

7. Understand the dynamics of web based commerce.

8. Be able to create web pages which include pre-coded programs to enhance a web site, this may include cgi, javascript, or other dynamic page generation languages.

9. Have experienced several (more than 2) web creation tools to be able to compare and contrast the ease of use of the tools.

10. Have added sound and/or video to a web page as a way to enhance the page. And learning the limitations around multi-media presentation.

CS 311: **Operating Systems I**

Catalog Description

Introduction to operating systems using UNIX as the case study. System calls and utilities, fundamentals of processes and interprocess communication. (4 credits) Prereq: CS 151, CS 261, ECE 271 or CS 271,

By the completion of this course, students will be expected to:

1. Demonstrate the ability to develop programs using the following programming tools:
   - development environments
   - libraries
   - revision control systems
   - scripting languages

2. Demonstrate a knowledge of the overview of operating systems including history, structure, and implementation.

3. Demonstrate an understanding of processes including data structures, states, state transitions, and synchronization.

4. Demonstrate the ability to do concurrent programming.

5. Demonstrate the ability to write code that provides mutual exclusion for processes sharing variables or other resources.

6. Demonstrate the ability to write socket based client/server systems.

7. Demonstrate the ability to program with regular expressions.

Submitted by Mike Johnson and Jon Herlocker, March 2001

CS 312: **System Administration**

Catalog Description

> Introduction to UNIX system administration. Network administration and routing. Internet services. Security issues. (4 credits) Prereq: CS 311 or instructor approval

By the completion of this course, students will be able to:

1. Install an OS and configure a system for a specific task-partitioned disk space, add and remove network services, connect to a network. Add new packages to a system.

2. Add/delete users, check disk space usage, change passwords. Manipulae and change user permissions on files and directories.

3. List processes, kill rogue processes, detect high use processes, move processes from foreground to background. Change process priority.

4. Shut down and reboot a system safely.

5. Write programs to automate simple system administration tasks.

6. Demonstrate the ability to research a topic and present a clear articulation of the topic issue.

7. To work with a team to expand the amount of work that is possible to do through delegation, organization and sharing of resources.

Submitted by John Sechrest, March 2001

CS 321: **Theory of Computation**

Catalog Description

> Survey of models of computation including finite automata, formal grammars, and Turing machines. Prereq: CS 261, Math 231

On completion of the course, students will be able to:

1. Describe the properties of regular and context-free languages. Be able to determine if a language is regular or context-free.

2. Design finite state automata (both deterministic and nondetermanistic), regular grammars and regular expressions for regular languages and convert one description to another.

3. Design grammars and push down automata for context-free languages and convert one to the other.

4. Explain the concept of a Turing Machine and write simple programs for aTuring Machine.

5. Explain the concept of undecidability and give examples of undecidable problems.

Submitted by Paul Cull, March 2001

CS 325: **Analysis of Algorithms**

Catalog Description

> Recurrence relations, combinatorics, recursive algorithms, proofs of correctness. (4 credits) Pereq: CS 261, Math 232.

On completion of this course, students will be able to demonstrate an understanding of the following topics:

1. Recursive algorithms.
2. Using difference equations
3. Inductive proofs of correctness
4. Timing algorithms
5. Search algorithms
6. NP completeness
7. Divide and conquer algorithms
8. Approximation for optimisation
9. Big Oh notation and asymptotics

CS 361: **Software Engineering I**

Catalog Description

> Introduction to software engineering beginning with an overview of the software lifecycle, followed by a focus on the "front end" of that lifecycle including requirements analysis, and specification and design techniques. (4 credits) Prereq: CS 261. WIC course.

On completion of the course, students will be able to:

1. Describe three process models of the software life cycle and discuss their phases, advantages, and disadvantages.

2. Gather requirements for a realistic software system by interacting with a user or user group; write a requirements specification document.

3. Model system requirements using one or more semi-formal notation such as: UML, dataflow diagrams, entity-relationship diagrams, or state diagrams.

4. Design software systems at the architectural level, and at lower levels using one or more techniques such as: object-oriented or structured design. Express designs in design specification documents.

5. Validate requirements and designs by reviewing specifications with the user or user group; adjust the specification or design as necessary.

6. Work effectively in teams.

7. Prepare effective, professional software-related documents.

Submitted by Curt Cook and Gregg Rothermel, March 2001

CS 362: **Software Engineering II**

Catalog Description:

> Continued introduction to software engineering, focusing on the "back end" of the software lifecycle, including implementation, software verification and validation, and software evolution and maintenance. (4 credits) Prereq: CS 361

On completion of the course, students will be able to:

1. Describe several techniques for validating and measuring the quality of software, including both formal and semi-formal techniques.

2. Apply testing techniques including black-box and white-box techniques, at the unit, integration, system, and regression testing phases.

3. Actively participate in a software inspection or review.

4. Use a debugger and other applicable techniques to locate faults.

5. Describe advantages of, and cost-benefit tradeoffs inherent in, the use of automated tools for building software and automating configuration management, such as make and CVS, and use such tools in a realistic setting.

6. Describe and justify a set of common programming guidelines and procedures.

7. Describe several methods of estimating the cost and developing a schedule for a programming project.

8. Work effectively in teams.

Submitted by Curt Cook and Gregg Rothermel, March 2001

CS 372: **Computer Networks**

Catalog Description:

> Introduction to principles, organization and implementation of computer networks. Basic converage of fundamentals, architecture, topography, and application issues. 4 Credits. Prereq: CS 261, CS 311, Math 231.

On completion of the course, students will be able to to:

1. Describe the basic principles involved in network hardware and software design.
2. Describe the fundamental physical layer design methods, which include data transmission, transmission media, data encoding and data communication interface.
3. Describe the basic data link layer issues; error control techniques and data link protocols.
4. Describe the basic medium-access sublayer issues and basic multiple access protocols.
5. Describe the basic network layer design issues and associated routing algorithms.
6. Describe the basic transport layer issues and simple protocols.
7. Describe the basic application layer design issues, including simple security design methods.

Submitted by Bella Bose, March 2001

CS 381: **Programming Language Fundamentals**

Catalog Description

> An introduction to the concepts found in a variety of programming languages. Programming languages as tools for problem solving. A brief introduction to languages from a number of different paradigms. (4 credits) Prereq: CS 261

On completion of the course, students will be able to:

1. Describe and work problems that accurately predict program behavior under static versus dynamic scoping mechanisms.

2. Describe and work problems that accurately predict program behavior under static versus dynamic typing mechanisms, with or without the use of type constraints.

3. Describe and work problems that accurately predict program behavior under the by-value, by-reference, by-constant, by-result, by-value-result, and by-name mechanisms of parameter passing.

4. Describe the contents of the run-time stack as it stands at any moment in program execution.

5. Describe the extent of polymorphism facilitated by dynamic typing versus inheritance versus overloading.

6. Understand exception handling mechanisms and be able to implement exception handlers in Java and in 2-3 other programming languages.

7. Explain the essential differences between the imperative, functional, dataflow, and object-oriented paradigms. Create programs in several paradigms.

8. Explain how programming language semantics are defined (axiomatic, denotational, operational semantics).

Submitted by Margaret Burnett, March 2001

CS 391: **Social and Ethical Issues in Computer Science**

Catalog Description

> In-depth exploration of the social, psychological, political and ethical issues surrounding the computer industry and the evolving information society. (3 credits) Prereq: C 101

On completion of the course, students will be able to:

1. Identify and evaluate the social impact of current and future computer-related technologies.

2. Explain the roles and responsibilities of today's computer professional.

3. Interpret state, federal, and international laws with respect to computer activity.

4. Identify and evaluate risks associated with computer technologies.

5. Identify and evaluate the benefits of computer technologies.

6. Prepare and present informational and engaging presentations about issues facing computer professionals.

CS 395: Interactive Multimedia

Catalog Description

> Technological, aesthetic, and pedagogical issues of communication using interactive multimedia and hypermedia; techniques for authoring interactive multimedia projects using a variety of digital media tools. PREREQ: CS 101, ART 120.

On completion of the course, students will be able to:

1. understand the relationship between digital media and multimedia.
2. digitize photographs, sound clips, and video clips, edit them, apply digital effects, and optimize them for delivery on the Web or CD-ROM.
3. create animated interactive presentations using a cross-platform multimedia authoring tool.
4. use a multimedia authoring tool to integrate audio, video, graphics, and animation into interactive multimedia projects for delivery on CD-ROM and the Web.
5. apply basic graphic design principles to create aesthetically pleasing multimedia projects.
6. apply basic user-interface design principles to create multimedia projects that are easy to understand and to use.
7. understand the nature of the multimedia design team and the kinds of work done by team members.
8. coordinate a multimedia project from the earliest design phase through the storyboard and prototype phases to the final tested product.
9. discuss the fundamental ethical and legal issues related to interactive multimedia design.

CS 411: **Operating Systems II**

Catalog Description

Principles of computer operating systems; concurrent processes, mutual exclusion, memory management, file systems, I/O systems, performance evaluation, multiprocessor systems, and distributed systems. (4 credits) Prereq: CS 311, ECE 375 or CS 271.

Students who take Computer Science 411 must:

Demonstrate an understanding of the fundamental objects, functions, issues, algorithms, design issues, and tradeoffs involved in …

1. process and thread management
2. process synchronization
3. CPU scheduling
4. memory management, including paging and virtual memory systems
5. file systems and file management
6. I/O management
7. operating system protection & security

8. Implement and test algorithms for operating system functions such as CPU scheduling, multiprogramming, virtual memory, and disk scheduling.

Submitted by Jon Herlocker, April 2002

CS 420: **Graph Theory with Computer Science Applications**

Catalog Description

> Directed and undirected graphs; paths, circuits, trees, coloring, planar graphs, partitioning; computer representation of graphs and graph algorithms. Applications in software complexity metrics, program testing, and compiling. (3 credits) Prereq: CS 325, Math 232

On completion of this course, a student will be able to:

1. Demonstrate an understanding of major concepts in graph theory including paths, distance, connectedness, isomorphism, matching, planarity, coloring, independence, covering, factoring, and spanning trees.

2. Demonstrate an understanding of the fundamental results in graph theory including characterizations of trees, Euler graphs, planar graphs, and n-connected graphs, degree sequences of graphs, planar graph coloring, Hamiltonian graphs and network flows.

3. Understand basic graph algorithms including depth-first and breadth-first search, minimal spanning trees, maximal matching, and shortest path.

4. Describe computer science applications in the areas of software testing, software complexity, network topologies, and data structures.

Submitted by Curt Cook, March 2001

CS 430: **AI Programming Techniques**

Catalog Description

> Symbols and symbolic programming. List basics: eval, recursion, variable binding and scoping, macros. Representation and problem solving in Lisp, advanced topics: alternative data representations, generators, data-driven control, agendas, AI Programming paradigms (4 credits) Prereq: CS 325, CS 381

On completion of the course, students will be able to:

1. Define the dimensions along which agents and environments vary, along with the key functions that must be implemented in a general agent.

2. Implement agents using various search methods, including uninformed and informed, deterministic and stochastic search.

3. Explain and apply the basics of logical and structured (e.g. Bayesian Network) probabilistic inference and decision evaluation.

4. Compare and contrast learning methods including supervised and unsupervised learning, the role of generalization in learning and the kinds of generalization that takes place in various learning methods.

Submitted by Prasad Tadepalli, march 2001

CS 440: **Database Management Systems**

Catalog Description

Purpose of database systems, levels of data representation. Entity-relationship model. Relational systems: data definition, data manipulation, query language (SQL), relational calculus and algebra, data dependencies and normal forms. DBTG network model. Query optimization, recovery, concurrency control.(4 credits) Prereq: CS 261

On completion of the course, students will be able to:

1. Describe the fundamental principles and goals of integrated data management.
2. Design and implement a relational database and formulate SQL queries.
3. Explain the principles of data modelling, create an ER diagram or UML class diagram, and generate a relational schema from such a diagram.
4. Understand the mechanisms of Web-based database access and create a simple Web-based database application.

Submitted by Toshi Minoura, March 2001

CS 450: **Introduction to Computer Graphics**

Catalog Description

Display devices, graphics software, interactive graphics, three-dimensional graphics. (4 credits) Prereq: CS 311, CS 325

On completion of the course, students will be able to:

1. Describe the use of an API for 2D graphics programming.

2. Explain the concepts of windows and viewports and perform coordinate conversions for them.

3. Understand the mechanisms for 2D and 3D transformations with homogeneous transformation matrices.

4. Perform view transformations, local transformations and projections.

5. Explain the mechanisms for backface removal and hidden surface elimination.

6. Explain various shading methods and use them.

7. Explain the concepts for ray tracing.

8. Create animated 3D scenes.

Submitted by Toshi Minoura, March 2001

CS 472: **Computer Architecture**

Catalog Description

> Computer architecture using processors memories, I/O devices and I/O interfaces as building blocks. Study of instruction set design and implementation, processor implementation, pipelining and memory hierarchies. Issues and tradeoffs involved in the design of Reduced Instruction Set Computer (RISC) architectures. (4 credits) Prereq: ECE 375 (Colisted as ECE 472)

On completion of the course, the students will be able to:

1. Describe the performance of a computer system in terms of the number of instructions, clock cycles per instruction and clock period.

2. Describe the basic instruction sets for a simple computer system.

3. Design a simple data path using a subset of instructions.

4. Describe the basic concepts of 1's and 2's complement number systems and floating point number systems.

5. Describe basic concepts of pipeline architectures and the design issues.

6. Describe the architecture of memory systems, and simple parallel interconnection networks.

CS 475: **Introduction to Parallel Computing**

Catalog Description

> Theoretical and practical survey of parallel processing, including a discussion of parallel architectures, parallel programming languages, and parallel algorithms. Programming one or more parallel computers in a higher-level parallel language. (4 credits) Prereq: CS 325, CS 472 or ECE 472.

On completion of the course, the students will be able to:

1. Define and compute speedup, parallelizability, and efficiency.

2. State Amdahl's Law and use it to predict the maximum speedup achievable from a parallel version of a sequential program, given its execution profile.

3. Identify parallelism occurring in a program as pipelining, control (task) parallelism, or data parallelism.

4. Define Flynn's taxonomy and give examples of SISD, SIMD, and MIMD computers.

5. Draw the most common interconnection networks (mesh, perfect shuffle, butterfly, and hypercube) and explain the relative advantages and disadvantages of each with respect to diameter, bisection width, and number of edges/node.

6. Explain the advantages and disadvantages of constructing parallel computers using commodity off-the-shelf components.

7. Write SPMD-style parallel programs using either a standard message passing library or threads package. The collection of programs produced should exhibit local communications, collective communications, barrier synchronizations, and file I/O. They should demonstrate both block and cyclic allocation of array elements to processors.

CS 480: **Translators**

Catalog Description

> An introduction to compilers; grammars, syntax-directed translation, parsers, semantic analysis, and optimisation. (4 Credits).
> Prereq: CS 321, ECE 375 or CS 271.

On completion of the course, students will be able to:

1. Describe characteristics common to a wide variety of different forms of translators (XML->html, Tex->dvi, hardware description languages, and so on)

2. Describe the phases of a compiler

3. Understand the difference between the compile time and run time representation of a program, and the structures common to each. (For example, symbol tables and activation records).

4. Use regular expressions and context free languages to define a language syntax.

5. Perform manipulations on a grammar for a simple context free language (for example, removal of recursion, factoring, making a grammar LL(1)).

6. Implement a lexical analyser to recognise tokens defined by regular expressions

7. Implement a parser, using either top down (recursive descent) or bottom up (LR) techniques

8. Explain the purpose and importance of types in programming languages, including internal representation issues and type checking.

9. Generate working target language for simple programming constructs.

Submitted by Tim Budd and Martin Erwig, march 2001

Catalog Descriptions for Related Courses

We do not have explicit learning objectives for those courses taught in other departments. However, the catalog descriptions for these courses give some indication of the topics discussed in each course.

ECE 271: Digital Logic Design

A first course in digital logic design using small and medium scale integrated circuits. 3 Credits.

ECE 375: Computer Structures and Assembly Language Programming

Introduction to the Von Neuman computer architecture and assembly language programming. 4 Credits.

Math 231, Math 232: Elements of Discrete Mathematics

Elementary logic, mathematical induction, functions and sequences, finite and infinite sets, counting techniques, basic matrix algebra, relations, graphs, trees, semigroups. 4 Credits each.

Math 251: Differential Calculus

Differential calculus for engineers and scientists. Rates of change: the derivative, velocity, and acceleration. The algebraic rules of differential calculus and derivatives of polynomial, rational, and trigonometric functions. Maximum-minimum problems, curve sketching, and other applications. Antiderivatives and simple motion problems. PREREQ: MTH 112. (Bacc Core Course)

Math 252: Intergral Calculus

Definite integrals, elementary applications to area, force, and work. Integral tables and basic techniques of integration, calculus of logarithmic and exponential functions, polar coordinates, applications to areas, volumes, force, work, and growth and decay problems. PREREQ: MTH 251. (Bacc Core Course)

Math 253: Infinite Series and Sequences

Indeterminate forms. Sequences and series, especially Taylor's formula and power series. Applications to numerical estimation with error analysis. Series with complex terms and the Euler identities. Brief introduction to functions of several variables, partial derivatives, the chain rule, and double integrals in rectangular coordinates. PREREQ: MTH 252. (Bacc Core Course)

Math 254: Vector Calculus

Vectors and vector functions. Surfaces, partial derivatives, gradients, and directional derivatives. Multiple integrals with applications. Related matrix and linear algebra concepts. PREREQ: MTH 252. (Bacc Core Course)

Math 351: Introduction to Numerical analysis

Introduction to the computation of approximate solutions to mathematical problems that cannot be solved by hand; analysis of errors; rootfinding for nonlinear equations in one variable; interpolation of functions; numerical integration.

Physics 211, 212, 213: General Physics with Calculus

A comprehensive introductory survey course intended primarily for students in the sciences and engineering. Topics include mechanics, wave motion, thermal physics, electromagnetism, and optics. Elementary calculus is used. Laboratory work accompanies the lectures. Concurrent enrollment in a recitation section is strongly recommended. PREREQ: MTH 251 for PH 211; MTH 252 and PH 211 for PH 212; MTH 254 and PH 212 for PH 213. COREQ: MTH 252 for PH 211, MTH 254 for PH 212. Lec/lab. (Bacc Core Course)

Statistics 314: Introduction to statistics for Engineers

Probability, common probability distributions, sampling distributions, estimation, hypothesis testing, control charts, regression analysis, experimental design. PREREQ: MTH 253.