Computer Science III Computing

Description

This course is the first in a three-course sequence that provides students with a foundation in computer science. Students develop fundamental programming skills using a language that supports an object-oriented approach, incorporating security awareness, human-computer interactions and social responsibility.

Among the associate-degree transfer programs, the Computer Science I-II-III coursework provides a common "core" body of knowledge in computing. The complete course sequence is designed in such a manner that students progress in knowledge, proficiency and professional maturity in several specific areas, including software engineering principles and professional and ethical conduct.

The progression of software engineering topics across CS I-II-III originates in CS I, where there is an emphasis on using a cyclic approach for program development by iterating through designing, coding, and testing program modules. Complemented by algorithm analysis, students are encouraged to think abstractly about problems and to begin developing processes for decomposing problems into organized parts. Encouraging clear documentation, good naming conventions and consistent secure coding style contribute to a disciplined approach to writing programs.

The progression of software engineering topics across CS I-II-III continues in CS II, where greater emphasis is placed on abstraction and sound software design principles, engaging students in the development of secure software components that solve a wide range of related problems and can be reused. The students determine the necessary elements of simple ADTs (such as a counter or a date) and then construct them; by their very nature, these components must be well-documented to encourage reuse. Additionally the students write assertions such as pre-conditions and post-conditions describing each class method, thereby encouraging students to think deeply about a simple problem before coding. After coding, the components must be well-tested, and therefore the use of test plans and test drivers are practiced. These activities reinforce the notion of constructing software from well-defined, independent pieces and complement the study of using existing library classes and APIs in software solutions.

The progression of software engineering topics across CS I-II-III concludes in CS III, where students are asked to step beyond the programmer role and take a broader view of software development; to consider its lifecycle from problem description to maintenance. Students first practice with analysis and design of medium-sized systems. Standard modeling tools are introduced and the students complete the phases of analysis, design, implementation and testing of a medium-sized team project that includes documents such as UML diagrams or CRC cards in addition to test plans. Students consider design patterns and write applications using data structures and templates. The software engineering topics are integrated with professionalism and ethics, as well as software and information assurance topics, such as security concerns and liabilities of computer-based systems.
The progression of the emphasis on professional and ethical conduct across CS I-II-III originates in CS I, where the curriculum is designed to consider the historical context of computing and programming as well as examining issues involving ethical conduct; plagiarism, intellectual property and software piracy issues are presented. Typically, student requirements for submitting original work as well as college policies regarding the use of computing resources and acceptable computing behavior on campus and the Brookings Institute "10 Commandments of Ethical Computing" can be used as relevant discussion starters.

The progression of the emphasis on professional and ethical conduct across CS I-II-III continues in CS II, which builds upon this foundation by examining societal issues, the Internet, and professionalism. Now that the students have gained some experience with developing programs, they can begin to see "what can go wrong" and the possible consequences to the user of their program, at a personal level, such as infinite loops and program crashes. Additionally, students are confronted with broader implications by design considerations regarding databases and data accessibility; ethical concerns regarding personal data, privacy and property rights should be explored. Integrating these topics with the software development process, security issues, and relevant cases of software errors will help students recognize that their work can have individual as well as societal consequences and encourage them to think carefully about the design and implementation of their programs.

The progression of the emphasis on professional and ethical conduct across CS I-II-III concludes in CS III, where a broader view is presented - encompassing computing sciences as a profession. Standards of professional behavior, organizations and publications are examined as well as a variety of occupational roles in the computing field. Course content materials presenting case studies of significant software failures amplify the topics of risks and liabilities. The students should start to recognize that invariably software production involves ethical choices. Incorporating these topics with the software lifecycle, engineering, human factors, and software assurance considerations will help students internalize the significance of professional and ethical behavior and subsequently demonstrate it through their individual and group projects.

**Minimum Contact Hours**

42

**Prerequisite(s)**

Computer Science II  
Discrete Structures

**Corequisite(s)**

Calculus I

**Topics**

<table>
<thead>
<tr>
<th>Title</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic strategies</td>
<td>2</td>
<td>brute-force; greedy; branch-and-bound; heuristics; pattern matching; string/text</td>
</tr>
<tr>
<td>Basic algorithmic analysis</td>
<td>3</td>
<td>asymptotic analysis of upper and average complexity bounds; best, average, and worst case behaviors; big O and little o notations; standard complexity classes; empirical measurements of performance; time and space tradeoffs; recurrence relations</td>
</tr>
<tr>
<td>Canonical data structures</td>
<td>7</td>
<td>stacks; queues; linked lists; hash tables; trees; graphs</td>
</tr>
</tbody>
</table>
Concurrency 2
- threads; scheduling, synchronization and timing; multi-threaded programs; race conditions
- efficiency of various sorting and searching algorithms; hashing; collision-avoidance strategies; binary search trees; depth- and breadth-first traversals; shortest-path algorithms; minimum spanning tree; transitive closure; topological sort

Formal computing algorithms 8
- universal principles; human-centered considerations; usability testing and verification; design trade-offs; secure user interfaces; standard API graphics
- recursive mathematical functions; divide-and-conquer, first-and-rest, and last-and-rest strategies; backtracking; recursion with linked lists, trees and graphs
- standards of professional behavior; professional computing societies and publications; professional responsibilities and liabilities; ACM Code of Conduct; career paths in computing
- conformance with assurance coding standards and practices, trustworthiness, and predictable execution testing; quality reviews; engineering and security tradeoffs; risks and liabilities of computer-based systems; fault prevention in software life-cycle stages; intentional and unintentional software security vulnerabilities
- standard approaches and implementation tools for analysis and design; software lifecycle stages, processes and documentation; Software Process Maturity Scale
- design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators

Intermediate Human-Computer Interaction 2
- best practices; human-centered considerations; usability testing and verification; design trade-offs; secure user interfaces; standard API graphics
- recursive mathematical functions; divide-and-conquer, first-and-rest, and last-and-rest strategies; backtracking; recursion with linked lists, trees and graphs
- standards of professional behavior; professional computing societies and publications; professional responsibilities and liabilities; ACM Code of Conduct; career paths in computing
- conformance with assurance coding standards and practices, trustworthiness, and predictable execution testing; quality reviews; engineering and security tradeoffs; risks and liabilities of computer-based systems; fault prevention in software life-cycle stages; intentional and unintentional software security vulnerabilities
- standard approaches and implementation tools for analysis and design; software lifecycle stages, processes and documentation; Software Process Maturity Scale
- design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators

Intermediate Recursion 7
- recursive mathematical functions; divide-and-conquer, first-and-rest, and last-and-rest strategies; backtracking; recursion with linked lists, trees and graphs

Professionalism 1
- standards of professional behavior; professional computing societies and publications; professional responsibilities and liabilities; ACM Code of Conduct; career paths in computing
- conformance with assurance coding standards and practices, trustworthiness, and predictable execution testing; quality reviews; engineering and security tradeoffs; risks and liabilities of computer-based systems; fault prevention in software life-cycle stages; intentional and unintentional software security vulnerabilities
- standard approaches and implementation tools for analysis and design; software lifecycle stages, processes and documentation; Software Process Maturity Scale
- design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators

Software and Information Assurance 3
- software lifecycle stages, processes and documentation; Software Process Maturity Scale
- design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators

Software engineering 4
- software lifecycle stages, processes and documentation; Software Process Maturity Scale
- design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators

Software reuse 3
- design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators

Course Objectives

Global Impact
An ability to analyze the global impact of computing on individuals, organizations, and society.

- Learning Outcomes
  CS. 3.
  Practice the tenets of ethics and professional behavior promoted by computing societies; accept the professional responsibilities and liabilities associated with software development.

Lifelong Learning
An ability to engage in continuous learning as well as research and assess new ideas and information to provide the capabilities for lifelong learning.

- Learning Outcomes
  CS. 4.
  Analyze the efficiency of recursive algorithms.
  CS. 11.
  Assess the appropriateness of using recursion to solve a given problem.
  CS. 12.
  Compare and contrast a range of searching and sorting algorithms and analyze time and space
efficiencies.
CS. 17.
Create effective, efficient and secure software, reflecting standard principles of software engineering and
software assurance.
CS. 21.
Design and construct programming solutions using a variety of recursive techniques.
CS. 22.
Design and develop reusable software using appropriate data structures and templates.
CS. 33.
Use standard analysis and design techniques to produce a team-developed, medium-sized, secure software
application that is fully implemented and formally tested.

Association for Computing Machinery

- Login
- Acceptable Use
- Privacy Policy

Copyright © 2021 ACM CCECC