Data Structures

2023-2024 Catalog

[ARCHIVED CATALOG]

CSCI 202 - Data Structures

PREREQUISITES: (CSCI 201 - Computer Science II or ((SDEV 200, SDEV 220, SDEV 230, or SDEV 240) AND CSCI 179))
PREREQUISITES/COREQUISITE: MATH 211 or Higher

PROGRAM: Computer Science
CREDIT HOURS MIN: 3
LECTURE HOURS MIN: 2
LAB HOURS MIN: 2
DATE OF LAST REVISION: Fall 2020

Builds on the foundation of Computer Science I and II. The course provides a working understanding of the fundamentals of data structures and algorithms used in modern computer programming. Introduces a variety of data storage alternatives, including stacks, queues, linked lists, hash tables, trees and graphs. Employs the basics of algorithmic analysis, recursion, language translation and software engineering. Discusses the overview, history and comparison of programming languages, as well as virtual machines and language translation.

MAJOR COURSE LEARNING OBJECTIVES: Upon successful completion of the course the student will be expected to:

1. Demonstrate with proficiency basic procedural and object-oriented concepts of computer programming.
2. Demonstrate use of a variety of data structures, including:
   a. Stacks and queues
   b. Sets and maps
   c. Linked lists,
   d. Binary trees, AVL trees, and B-Trees,
   e. Hash tables and graphs
3. Demonstrate fundamental computing algorithms, including:
   a. Sorting
   b. Sequential and binary searches
   c. Insertion, deletion, and balancing trees
   d. Hash tables and collision-avoidance
   e. Directed, undirected and weighted graphs
   f. In-order, depth- and breadth-first graph traversals.
   g. Recursive traversal of various data structures.
   h. Regular expressions and Regex
   i. Demonstrate basic algorithmic analysis:
   j. Analyze the performances of an algorithm with various input sizes
   k. Compare upper and average complexity bounds
   l. Differentiate best, average and worst-case behaviors
   m. Describe algorithm complexity in notation (big O, little O, omega and theta)
   n. Contrast standard complexity classes
   o. Determine empirical measurements of performance
   p. Evaluate time and space tradeoffs in algorithms
   q. Determine whether a recursive or iterative solution is most appropriate for a problem
   r. Identify a variety of real-world problems in computer science solvable using various data structures

4. Demonstrate the effective use of software engineering principles and practices:
    a. Compare various data structures for a given problem
    b. Investigate factors other than computational efficiency that influence the choice of algorithms.
    c. Apply consistent documentation and program style standards that contribute to the readability and maintainability of software.
    d. Conduct peer code reviews (focused on common coding errors) on program components.
    e. Differentiate between program validation and verification.
    f. Investigate potential vulnerabilities in provided programming code.
    g. Use software tools to evaluate, test, and debug programs.
5. Demonstrate the principles of secure programming and design:
    a. Investigate security vulnerabilities in various data structures
    b. Implement programs that properly handle exceptions and error conditions.
    c. Describe key terms in cryptology, including cryptography, cryptanalysis, cipher, cryptographic algorithm, and public key infrastructure.
    d. Use a cipher to encrypt plaintext into ciphertext.
    e. Describe the use of cryptographic hash functions for authentication and data integrity.
    f. Contrast symmetric and asymmetric encryption in relation to securing electronic communications and transactions.
    g. Describe security concerns in designing applications for use over networks.
    h. Illustrate secure connectivity among networked applications.
6. Analyze the goals of parallelism and concurrency.
    a. Describe various programming constructs for synchronization.
    b. Contrast low-level data races with higher level races.
    c. Investigate methods, such as mutual exclusion, used to avoid race conditions.
    d. Discuss security vulnerabilities related to parallelism and concurrency.
7. Discuss the history of programming languages, and be able to compare languages for features and applicability.
8. Discuss the concept of a virtual machine and the use of intermediate languages, and/or linked libraries.
9. Discuss interpreters, compilers, and the machine-dependent and -independent aspects of translation.

COURSE CONTENT: Topical areas of study include -

- Algorithmic analysis
- Interpreters
- Artificial Intelligence
- Linked libraries
- Basic procedural concepts
- Linked Lists
- Compliers
- Object-oriented programming
- Cryptography
- Recursion
- Data Structures
- Regular Expressions
- Expert Systems
- Security
- Fundamental computing algorithms
- Software engineering principles
- Graphs
- Trees
- History of programming languages
- Virtual machine
- Intermediate languages

Course Addendum - Syllabus (Click to expand)