Object Oriented Design and Data Structures

DESCRIPTION: The course combines a strong emphasis on Object-Oriented Design principles and design patterns with the study of data structures. Fundamental Abstract Data Types, their implementations and techniques for analyzing their efficiency will be covered. Students will design, build, test, debug and analyze medium-size software systems and learn to use relevant tools.

TEXTBOOKS TO CONSIDER:


TOPICS:

1. Object Oriented Software Development
   (a) Software Life Cycle
   (b) Problem Specification
   (c) Object Oriented Design
      • Abstraction
      • Encapsulation
      • Inheritance
      • Polymorphism
   (d) UML
   (e) Introduction to Design Patterns
      • Vectors and Iterator Pattern
   (f) Program Correctness
      • Exceptions
      • Testing and Debugging
      • Assertions
   (g) Program Efficiency
      • Analysis of Algorithms
      • Big-Oh
   (h) Event-Driven Programming, GUIs and MVC Pattern

2. Data Structures
(a) What is an ADT? Basic ADTs and implementations
  - Stacks, Queues, Lists
  - Array vs. Linked List

(b) Trees
  - Array vs. Linked Structures
  - Traversals
  - Binary Trees
  - Priority Queues and Heaps

(c) Searching
  - Dictionary ADT
  - Linear Search vs. Binary Search
  - Hashing
  - Binary Search Trees
  - Balanced Trees (AVL Trees)
  - Multi-dimensional Trees
    - kd-trees
    - quad-trees
    - range searching

(d) Sorting
  - Insertion Sort
  - MergeSort and Quicksort
  - Bucket Sort and Radix Sort

COURSE OBJECTIVES:

1. Learn professional responsibilities in program development, documentation and testing.
2. Justify the philosophy of Object-Oriented design and concepts of abstraction, encapsulation, inheritance and polymorphism.
3. Use Object-Oriented design methods to create effective and efficient problem solutions and learn to use tools like UML.
4. Demonstrate familiarity with some of the Design Patterns and be able to select and apply appropriate Design Patterns in their programs.
5. Learn basic algorithm analysis. Learn the tools for asymptotic analysis and compare asymptotic behaviour of different algorithms.
6. Improve abstraction skills. Understand the distinction between an Abstract Data Type (ADT) and its implementation.
7. Understand tradeoffs involved with the use of one implementation of an ADT over another.
8. Give the time and space needs of the chosen implementation.
9. Demonstrate familiarity with fundamental data structures.
10. Demonstrate familiarity with fundamental searching and sorting algorithms.
11. Design/implement/debug and test moderate size programs as an interaction between different ADTs.