Cape Cod Community College

Course Syllabus

Prepared by the Department of Mathematics
Date of Departmental Approval: April 11, 2017
Date approved by Curriculum and Programs: April 12, 2017 **Effective: Fall 2017**

1. **Course Number: CSC230**
   Course Title: Data Structures

2. **Description:** Students use data structures and recursion in Java to solve complex problems. Abstract Data Types (ADTs) including lists, stacks, queues, tables, sets, maps, heaps, and trees are examined and implemented. Students analyze the theoretical and actual running times of the alternate ADT implementations as well as internal/external searching and sorting algorithms, graph algorithms, and hashing.

3. **Student Learning Outcomes (instructional objectives, intellectual skills):** Upon successful completion of the course, the student are able to do the following:
   - Identify the data components and behaviors of multiple abstract data types.
   - Create alternative representations of ADTs either from implementation or the standard libraries.
   - Apply object-oriented principles of polymorphism, inheritance, and generic programming when implementing ADTs for data structures.
   - Compare alternative implementations of data structures with respect to performance; in particular, compare and contrast the costs and benefits of dynamic and static data structure implementations.
   - Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data.
   - Contribute to a small-team code review focused on component correctness.
   - Choose appropriate data structures and abstract data types (ADT) including lists, stacks, queues, trees, tables, sets, maps, and graphs for modeling a given problem.
   - Identify the relative strengths and weaknesses among multiple designs or implementations for a problem.
   - Explain how tree balance affects the efficiency of binary search tree operations.
   - Identify an appropriate algorithmic strategy (brute-force, greedy, divide-and-conquer, recursive backtracking, and dynamic programming) for a given problem; implement each strategy for an appropriate problem.
   - explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm in the context of specific algorithms, identify the characteristics of data and/or other conditions or assumptions that lead to different behaviors.
   - Determine informally the time and space complexity of simple algorithms; compare algorithms to search and sort algorithms studied in previous course.
   - Describe the heap property and the use of heaps as an implementation of priority queues.
   - Model *a variety of* real-world problems in computer science using appropriate forms of graphs and trees, including solving problems using graph algorithms such as searches, single-source and all-pairs shortest paths, and at least one minimum spanning tree algorithm.
   - Describe the implementation of hash tables, including collision avoidance and resolution.
   - Trace and implement a string-matching algorithm.

- Apply recursion as a problem solving technique; Determine whether a recursive or iterative solution is most appropriate for a problem.
- Determine appropriate ADTs and data structures for various sorting and searching algorithms.
- apply consistent documentation and program style standards that contribute to the readability and maintainability of software

4. **Credits(s):** 4 credits

5. **Satisfies General Education Requirement:** No

6. **Prerequisite(s):** CSC130 (Computer Programming II: JAVA)

7. **Semester(s) Offered:** Fall

8. **Suggested General Guidelines for Evaluation:** Comprehensive final examination, group projects, programs, quiz papers, and homework papers.

9. **General Topical Outline:**

1. Java Concepts
    a. Java Types and Classes
    b. Java Interfaces;
    c. Generics
    d. Iterators
2. Algorithm Analysis
    a. Formal Notation
    b. Searching and Sorting Review
3. Stack ADT
    a. Stack: Array Implementation
    b. Stack: Linked Implementation
    c. Comparison and analysis of implementations
4. Recursion
    a. Applications
    b. Recursive algorithm conversion using Stack
    c. Analysis of recursive algorithms
5. Queue ADT
    a. Queue: Array Implementation
    b. Queue: Linked Implementation
    c. Comparison and analysis of implementations
6. List ADT
    a. List: Array Implementation
    b. List: Linked Implementation
    c. Comparison and analysis of implementations
7. Trees
    a. Binary Trees
    b. Binary Search Trees (BSTs)
    c. Balancing BSTs
8. Heaps
    a. Implementations
    b. Priority Queues
    c. Heap Sort
9. Sets
10. Graphs
    a. Adjacency Matrix, List, and Map Implementation
    b. Graph Searches
    c. Minimum Spanning Trees
11. Hashing
    a. Implementations and collision resolution
    b. Comparison of implementations