Course Outline for Computer Science 20
Introduction to Data Structures

Catalog Description:

20 – Introduction to Data Structures 4 Units

Design and implementation of larger projects using object-oriented software engineering principles. Emphasis on definition and use of data structures. Includes specification of Abstract Data Types, recursion, dynamic memory allocation, stacks, linked lists, priority queues, graphs, binary trees, heaps, sorting and searching, algorithm analysis, hashing techniques, random access files. Prerequisite: Computer Science 15 (completed with a grade of “C” or higher). 3 hours lecture, 3 hours laboratory.

(Typical contact hours: lecture 52.5, laboratory 52.5)

Prerequisite Skills:

Before entering the course the student should be able to:

1. explain the steps in the software engineering lifecycle;
2. use accepted design methods (structure charts, pseudocode, simple class diagrams) to develop and code several (six-ten) programs of intermediate difficulty and length in an object–oriented language, including programs broken into several classes.
3. adhere to style and documentation standards in writing programs;
4. use system debuggers to step into and over code, set breakpoints and watch variables;
5. identify what makes a good test data suite and use it to thoroughly test a program under development;
6. write overloaded functions and simple recursive functions, function templates (C++) or generic methods (Java);
7. define, initialize, dereference and manipulate pointers;
8. manipulate arrays using pointer notation;
9. create multi-dimension arrays dynamically using pointers (C++) or references (Java).
10. define, design and use simple classes, including at least one project that uses a class inheritance hierarchy;
11. manipulate objects of the standard class libraries such as strings, vectors and streams,
12. overload operators (C++) or override clone, equals and toString (Java), define and use virtual member functions to implement polymorphic behavior;
13. identify bitwise, unary and binary and conditional operators;
14. use the new operator to implement a singly linked list.

Expected Outcome for Students:

Upon completion of the course the student should be able to:

1. define the term abstract data type (ADT);
2. define a list ADT and implement list ADT as array and vector class;
3. implement list ADT as singly and doubly linked list class;
4. define a stack ADT and implement stack ADT as array (or vector) and as a linked list;
5. define a queue ADT and implement queue ADT as array (or vector) and as linked list;
6. define a binary tree ADT and implement binary tree class;
7. define a binary search tree ADT and derive a binary search tree class from binary tree class;
8. write recursive methods and explain how recursion is implemented;
9. implement selected iterative and recursive searching and sorting algorithms;
10. design and implement a search that uses a hashing function;
11. explain the concept of time efficiency for algorithms using Big-O notation;
12. perform a runtime analysis of InsertionSort, QuickSort and other sorting algorithms;
13. define a graph ADT and implement a graph using an adjacency list or an adjacency matrix;
14. design and code a complete program of 500 lines or more.
Course Content (Lecture):

1. Data abstraction
   a. Concept of ADT
   b. Typical operations on an ADT
   c. Implementation issues
2. Elementary search algorithms
   a. Sequential search
   b. Binary search
3. Time efficiency of algorithms
   a. Big-O, theta and omega notation and time-behavior
   b. Examples: Time order of sequential and binary search, sorting algorithms
   c. Notions of intractable and unsolvable problems
4. Implementing linear data structure ADTs
   a. List as array or vector
   b. List as singly and doubly linked list
   c. Queue and priority queue as array, vector or linked list
   d. Stack as array, vector or linked list
   e. Big-O time-order of traversal, insertion, and deletion for above data structures
   f. Applications such as evaluation of postfix (RPN) and infix expressions
5. Recursion
   a. Base and general case
   b. Applications such as evaluation of the binomial coefficient, reverse traversal of a linked list, binary search, sorting, tree traversal
   c. Stack activation record and its role in implementing recursive calls
   d. Comparison of recursive and iterative algorithms
6. Implementing binary tree ADTs
   a. Basic terminology: leaf, root, level, depth, balanced, full, complete
   b. Building a binary tree: inserting and deleting nodes
   c. Binary tree traversals: preorder, inorder, postorder
   d. Binary search tree
   e. Building a binary search tree: inserting nodes. Deleting nodes is optional.
   f. Use of binary search tree to search and sort data
   g. Complete binary trees: building a heap
   h. Time order of various binary tree operations
7. Sort algorithms
   a. Elementary O(n-squared) sorts: InsertionSort, SelectionSort, BubbleSort
   b. Recursive O(n log n) sorts: QuickSort, MergeSort, HeapSort
8. Hashing algorithms
   a. Concept of searching by direct access: Hash tables
   b. Construction of an elementary hashcode function
   c. Resolving collision by linear probing and chaining
   d. Application of hash codes to storage of records in an direct access file or object
9. Graphs
   a. Basic terminology: vertex or node, edge or arc or link, weighted graphs, connected or complete graphs, cyclic or acyclic graphs, sparse and dense graphs, directed and undirected graphs
   b. Basic graph algorithms such as: depth-first and breadth-first search, graph partitioning, Hamiltonian Path and Hamiltonian Cycle, Shortest Path, Traveling Salesman, Minimal Spanning Tree

Course Content (Laboratory):

1. Designing larger projects (500 lines of code or more)
   a. Class design diagrams
   b. Industry documentation standards
   c. Use of inheritance for code reuse
d. Top-down and bottom-up design
e. Testing issues and techniques

Methods of Presentation:

1. Lecture, discussion, and classroom demonstrations
2. Student use of appropriate computer laboratory
3. Hands-on exercises in the laboratory

Assignments and Methods of Evaluating Student Progress:

1. Typical Assignments
   a. Write a program that uses a string hash function to create a hash table of size 499 using the words in a given file. For each word in the file, output: 1) the word 2) the hash index obtained for that word using linear probe addressing, and, 3) the number of probes needed to insert the word in the table. After all the words have been processed, output the load factor and the average number of probes required to locate a word in the table or determine that a word is not in the table.

2. Methods of Evaluating Students Progress:
   a. A minimum of two midterms and a final examination
   b. Writing and implementation of various assigned programming projects that demonstrate competent knowledge of a range of the topics in the course outline. Design, coding, testing and documentation of at least one project of 500 lines of code or longer.

Textbook(s) (Typical):


Data Structures Using C++, D.S. Malik, Course Technolgy, 2007

Special Student Materials:

USB portable storage device