CSci 235 Software Analysis and Design 2

Current Catalog Description Prerequisites: CSCI 135 and 15O, and MATH 150. Representation of information in computers, including process and data abstraction techniques. Topics covered include static and dynamic storage methods, lists, stacks, queues, binary trees, recursion, analysis of simple algorithms, and some searching and sorting algorithms. Fulfills GER 3/B requirement.

Remarks This course has conceptual and pragmatic components. The conceptual material is language independent. The pragmatic component is based upon C++. The course requires that the students do at least three significant programming assignments during the semester.

Data Structures

Topics

- Linked structures (i.e., structures using pointers)
- Implementation strategies for stacks, queues, binary trees
- Strategies for choosing the right data structure

Learning Objectives

1. Describe common applications for each data structure in the topic list.
2. Implement the user-defined data structures in a high-level language.
3. Compare alternative implementations of data structures with respect to performance.
4. Write programs that use each of the following data structures: linked lists, stacks, queues, binary trees.
5. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
6. Choose the appropriate data structure for modeling a given problem.

Recursion

Topics

- Recursive mathematical functions
- Recursion and mathematical induction
- Recursive backtracking

Learning Objectives

1. Compare iterative and recursive solutions for problems.
2. Implement, test, and debug simple recursive functions and procedures.
3. Determine when a recursive solution is appropriate for a problem.
4. Identify the base case and the general case of a recursively defined problem.

Basic Analysis

Topics

- Asymptotic analysis of upper and average complexity bounds
- Identifying differences among best, average, and worst case behaviors
- Big O and theta notation
- Standard complexity classes

Learning Objectives

1. Explain the use of big O and theta notation to describe the amount of work done by an algorithm.
2. Use big O and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of simple algorithms.
3. Determine the time and space complexity of simple algorithms.

Algorithmic Strategies

Topics

- Brute-force algorithms
- Divide-and-conquer
- Backtracking

Learning Objectives

1. Describe the shortcomings and benefits of brute-force algorithms (i.e., be able to state when it is appropriate to use them.)
2. For each of several kinds of algorithm (brute force, divide-and-conquer, backtracking), identify an example of everyday human behavior that exemplifies the basic concept.
3. Explain and implement a divide-and-conquer algorithm to solve an appropriate problem.
4. Use backtracking to solve a problem such as navigating a maze.

Fundamental Algorithms

Topics

- Sequential and binary search algorithms
- Quadratic sorting algorithms (selection, insertion)
- $O(N \log N)$ sorting algorithm (e.g., Quicksort, mergesort, or heapsort)
- Unbalanced binary search trees and tree traversals

Learning Objectives

1. Implement the most common quadratic and $O(N \log N)$ sorting algorithms.
2. Discuss the computational efficiency of the principal algorithms for sorting and searching and traversing binary trees.
3. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time and maintainability.
4. Demonstrate the following capabilities: to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in programming context.
5. Explain what it means for a sorting algorithm to be internal, external, stable, or natural.

Event Driven Programming

Topics

- Event-handling methods
- Event propagation
- Exception handling

Learning Objectives

1. Explain the difference between event-driven programming and command-line programming.
2. Design, code, test, and debug simple interactive programs that respond to user events.
3. Develop code that responds to exception conditions raised during execution.

Object Oriented Programming and Design

Topics

- Classes and subclasses
- Constructors and destructors
- Inheritance (overriding, dynamic dispatch)
- Polymorphism (subtype polymorphism vs. inheritance)

Learning Objectives

1. Justify the philosophy of object-oriented design and the concepts of inheritance and polymorphism.
2. Decompose a problem into appropriate classes.
3. Design, implement, and test the implementation of "is-a" relationships among objects using a class hierarchy and inheritance.
4. Compare and contrast the notions of overloading and overriding methods in an object-oriented language.

Foundations Information Security

Topics

- Security mindset
- Risk assessment and cost-benefit analyses
- Security versus usability, time, and/or money trade-offs

Learning Objectives

1. Analyze the trade offs inherent in security
2. Defend the need for protection and security, and the role of ethical considerations in computer use

Secure Programming

Topics

- How attackers use overflows to smash the run-time stack

Learning Objectives

1. Rewrite a simple program to remove a simple vulnerability
2. Explain why one or more language constructs may lead to security problems such as overflows.