

Important COP 3502 Final Exam Information

Date: 4/29/2022

Time: 10:00 AM (2.5 hours earlier than class time!!!)

Room: CB1-104

Exam Aids: Three sheets of regular 8.5"x11" paper, front and back, and the Foundation Exam Formula Sheet (provided)

Test Format:

Short Answer/Coding/Tracing/Problems

Areas of Focus: Binary Search, Backtracking, Base Conversion, Bitwise Operators, Recurrence Relations, Dynamic Memory Allocation, Recursion

Exam Archive:

www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/exam/

Archive shows types of questions I've asked in the past.

Outline of Topics for the Exam

I. Basics of C – if, loops, functions, array, strings

II. Problem Solving on Arrays

- a. Sorted List Matching**
- b. Binary Search**

III. Structs, Pointers and Dynamic Arrays

- a. how to allocate space dynamically
(array, 2d array, array of struct, array of ptr to struct,
linked list node, bin tree node, etc.)**
- b. how to free space**
- c. how to "resize" an existing array**
- d. how to declare structs**
- e. how to use pointers to structs**
- f. how to use arrays of structs**
- g. how to use arrays of pointers to structs**
- g. how to pass structs or pointers to structs into a function**

IV. Recursion

- a. Fibonacci**
- b. Factorial**
- c. Towers of Hanoi**
- d. Binomial Coefficients**
- e. Binary Search**
- f. Fast Modular Exponentiation**
- g. Floodfill**
- h Brute Force (odometer, combinations)**
- i. Generating Permutations**

V. Linked Lists

- a. Creating Nodes**
- b. Checking for NULL**
- c. Iterating through a list**
- d. Using recursion to visit all the nodes in a list.**
- e. Insertion, Searching**
- f. Deletion**
- g. difference between `ptr == NULL` and `ptr->next == NULL`**
- h. Circularly linked**
- i. Doubly linked**

VI. Stacks

- a. Array Implementation**
- b. Dynamically Sized Array Implementation**
- c. Linked List Implementation**
- d. Efficiency of push, pop**
- e. Determining the Value of Postfix Expressions**
- f. Converting Infix to Postfix**

VII. Queues

- a. Array Implementation**
- b. Dynamically Sized Array Implementation**
- c. Linked List Implementation**
- d. Efficiency of Enqueue and Dequeue**

VIII. Algorithm Analysis

- a. Average case vs. Worst case**
- b. Determining a Big-Oh bound via code segment**
- c. Use of sums**
- d. Big-Oh timing problems**
- e. Logs and exponents**
- f. Recurrence Relations**
- g. New problem analysis**

IX. Sorting

- a. Bubble Sort**
- b. Insertion Sort**
- c. Selection Sort**
- d. Merge Sort**
- e. Quick Sort**

X. Binary Search Trees

- a. Creating Nodes**
- b. Tree Traversals (preorder, inorder, postorder)**
- c. Insertion**
- d. Searching**
- e. Deletion**
- f. Code Tracing**
- g. Writing Code (recursive)**

XI. AVL Trees

- a. AVL Tree Property**
- b. Identifying nodes A, B and C for both insert and delete**
- c. Restructuring for both insert and delete**
- d. Delete may have multiple restructures**

XII. Tries

- a. Basic struct**
- b. Extra items to store in struct**
- c. Checking for NULL**
- d. Use of recursion on all 26 children**
- e. Coding problems**

XIII. Binary Heaps

- a. percolateUp**
- b. percolateDown**
- c. Insert**
- d. deleteMin**
- e. makeHeap**
- f. Heap Sort**

XIV. Hash Tables

- a. Properties of a good hash function**
- b. linear probing replacement technique**
- c. quadratic probing replacement technique**
- d. linear chaining hashing**

XV. Base Conversion and Bitwise Operators

- a. base conversion (2, 10, 16, other)**
- b. left shift, right shift, and, or, xor, complement**
- c. How to use a number to indicate a subset.**
- d. How to iterate through all possible subsets w/bitmask.**
- e. Use of operators for set tasks (intersection, union)**
- f. use of xor(^) in grading a T/F quiz.**

XVI. Binary Search

- a. Problem easy to do forwards, hard to do backwards**
- b. Search function is increasing or decreasing**
- c. Simplify original version of the problem**
- d. Crystal Etching (function inversion)**
- e. Airport Shuttle**

XVII. Backtracking

- a. Use in Eight Queens Problem**
- b. Digit Divisibility**
- c. Geppetto Problem (and use of bitwise ops)**
- c. Trying all possibilities and stopping if a path is doomed to fail.**