CptS 122 – Data Structures

Exam 1 Review Guide

This document will serve as a guide to help you prepare for the first exam in CptS 122. You will find information about the exam format and topics you are expected to review within this guide.

What to Bring?

- Your WSU ID
- Two sharp pencils
- Calculators and notes may **not** be used during the exam!

Exam Timeframe

Please be aware that, because you will be taking the exam during a normal lecture period. You will have 75 minutes to complete the exam. *If you show up late to class, you will have less time to take the exam*. Note that, when you hand in your exam, you will be required to present your WSU ID.

Exam Format

Expect the exam to look like an hour version of quizzes, with a few more involved problems that are more in the spirit of a lab exercise. The exam will consist of some concepts questions and programming problems. For the concept section of the exam, expect true-false, fill-in-the-blank, multiple-choice, and/or short-answer questions similar to those found on the quizzes (~30% of exam). Some of these may be similar to the questions found at the end of chapters in your textbook! For the programming problem section of the exam, I will present you with programming problems, and you will be expected to write syntactically-correct C code solutions that exercise good design (~70% of exam). Note that your solutions do not have to be documented!

Exam Coverage

The exam covers the first three weeks of the session.

C Data Structures

- Compare and contrast the terms Abstract Data Types (ADTs) and data structures
  - For ADT think: “specification”; for data structure think: “implementation”
- Declare, define, and apply a self-referential structure
  - Nodes for linked lists and stacks require self-referential structures
- Define what is dynamic memory
- Define what is a dynamic data structure
- Apply malloc ( ) and free ( ) to dynamic data structures
- Compare and contrast linked lists and stacks implemented with linked lists

- Design and implement an ordered or non-ordered dynamic (singly and doubly) linked list including and variations of the following:
    - isEmpty ( ) - returns an integer or enumerated Boolean type; true for an empty list, false for non-empty list
    - insertAtFront ( ) – allocates a node dynamically; initializes it to the data passed in; inserts the node at the front of the list only; returns true or false for successful or unsuccessful insertion, respectively
    - insertAtBack ( ) - allocates a node dynamically; initializes it to the data passed in; inserts the node at the back or end of the list only; returns true or false for successful or unsuccessful insertion, respectively
    - insertInOrder ( ) - allocates a node dynamically; initializes it to the data passed in; inserts the node in the list in ascending or descending order only; returns true or false for successful or unsuccessful insertion, respectively
    - deleteAtFront ( ) – de-allocates the node at the front of the list; returns the data in the node
    - deleteNode ( ) – de-allocates a node; returns true if node was de-allocated, false otherwise
    - printList ( ) – prints out the data in each node of the list; may be printed iteratively or recursively
- Design and implement makeNode ( ) as a separate helper function for each of the above data structures
    - makeNode ( ) – allocates a node dynamically; initializes the node; returns a pointer to the dynamic node
- Design and implement functions that expand on the basic list operations. Example functions include, but are not limited to:
    - mergeLists ( ) – join two linked lists
    - insertAtPosN ( ) – insert data at position N in the list
    - compareLists ( ) – check to see if two lists have the same data
    - reverseList ( ) – reverse the links in a singly linked list
    - others…
- Given a problem, describe which data structure is most appropriate
    - For example:
        - Converting infix expression to postfix expressions (stack)
        - Storing personal contact information (list)
- Draw block/memory diagrams to illustrate how links are modified for any of the particular operations described above
- Design and implement a list and with arrays instead of dynamic “links”
- Define what is a memory leak
    - Think: lost pointer to dynamically allocated memory; can’t access the memory anymore, but memory is still allocated by system on the heap
- Define what is a dangling pointer
    - Think: have a pointer to memory that is no longer accessible
- What is defensive programming?
    - We check to see if lists are empty inside our corresponding removal functions
        - Some implementations add preconditions to deleteNode ( ) not empty

Other Topics

- Compare and contrast char *str vs. str[], and char *str[] vs. char str[][]
    1. A char * needs to point to some allocated memory; this memory may be allocated dynamically or via another automatic local variable
    2. A str[] has memory associated with it; str refers to the address of the 0^{th} element in the array
    3. A char *str[] is an array of pointers; where each pointer could refer to dynamically allocated memory or automatic local variable memory
    4. A str[][] may be considered an array of strings or a 2-D array of characters; all required memory is allocated upfront; str[] (one index specified) refers to the start of the corresponding row
- Compare and contrast strings vs. character arrays
- What is the three-file format?
- What is a model? How do they apply to software development?

Recommended Strategy for Preparing for the Exam

I recommend that you use the following activities and materials to prepare for the exam:


Review quizzes and lab exercises: These may well be your best resource. An excellent learning activity would be to retake the quizzes and review the lab exercises.


Lecture slides and example code: Study the lecture slides and example code.