**CS233 Data Structures and Systems Programming**

Homework 2 - Spring 2017 - 50 Points

DUE @ TAPS on Lesson 23, M-day: Tue, 7 March, T-day: Wed, 8 March.

***Documentation Policy:***

* You must document all help received from any source other than your instructor.
* The documentation statement must explicitly describe WHAT assistance was provided. WHERE on the homework the assistance was provided, and WHO provided the assistance.
* If no help was received on this assignment, the documentation statement must state “NONE.”
* If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.
* Vague documentation statements must be corrected before the assignment will be graded, and will result in a 5% deduction on the assignment.

***Help policy for Homework #1:***

**AUTHORIZED RESOURCES: Any, except another cadet’s work or programs.**

**NOTE:**

* Never copy another person’s work and submit it as your own.
* Do not jointly complete this assignment.
* You must document all help received from sources other than your instructor.
* **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.**

**Instructions:** Add your name to the header of this document. Please do not change the name of this file. Type your answers into this document and **Git** 🡪 **add** this document to the Homeworks folder of your *source code repository*. Make sure you *commit* and *push* your repository before the due date. The file name should be **black** (not red or green) after you have submitted the document.

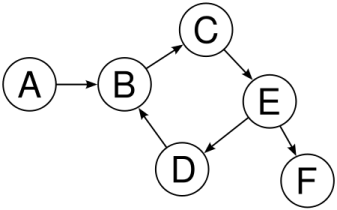
**Documentation Statement:** None.

1. (10 pts, 2 pts each) Read the description of a "tree data structure" in this Wikipedia page, <https://en.wikipedia.org/wiki/Tree_(data_structure)>. Then answer the following questions:
   1. Give a non-recursive definition of a tree data structure.

an ordered tree, can be represented with an array, globally looking at tree as whole

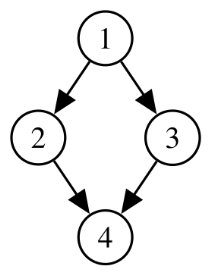
* 1. Give a recursive definition of a tree data structure.

a collection of nodes, node references children, none reference root



* 1. Explain why the example to the right (from the web site) is not a tree.

No concept of parent, children. Looks more like a haphazardly constructed link list



* 1. Explain why this example to the right (from the web site) is not a tree.

A node can only have one parent node

* 1. Can a *list* be considered a *tree*?

Yes, a degenerate tree is a list

1. (6 pts, 2 pts each) Given a **binary** tree with *n* nodes, what is the O() run-time behavior of the following tree functions? You must explain your answer. No explanation 🡪 no credit.
   1. void treeInorderTraversal(Tree \* tree);

(This function performs an inorder traversal of a specific tree.)

O(n), because traversing whole tree

* 1. void treeAddLeafNode(Tree \* tree, ElementType item);  
     (Assume that the tree is a *complete* tree and the new node added to the tree becomes a child of a node that has an empty child.)

O(log n), because tree is complete and doesn't have to navigate through whole tree

* 1. void treeGetHeight(Tree \* tree);  
     (This function returns the height of the tree, which is the maximum level of any node in the tree plus one.)

O(n), must look at whole tree

1. (15 pts) Suppose you have a C struct as defined below. Use "reverse engineering" techniques to discover how the struct is actually stored in memory. Then answer the following questions.

**struct** alpha {  
 **int** f1;  
 **double** f2;  
 **char** f3;  
};

* 1. (1 pt) In bytes, what is the size of this structure when the compiler creates an instance of this data type?

24

* 1. (6 pts) Draw a diagram of memory and show which bytes are used to represent each field of the structure. If a byte is not used by the structure, put an X in that byte.

|  |  |
| --- | --- |
| 4294953968 | F1 |
| 4294953969 | X |
| 4294953970 | X |
| 4294953971 | X |
| 4294953972 | X |
| 4294953973 | X |
| 4294953974 | X |
| 4294953975 | X |
| 4294953976 | F2 |
| 4294953977 | X |
| 4294953978 | X |
| 4294953979 | X |
| 4294953980 | X |
| 4294953981 | X |
| 4294953982 | X |
| 4294953983 | X |
| 4294953984 | F3 |
| 4294953985 | X |

* 1. (2 pts) Explain why the data is organized in memory like it is.

Each variable within the structure takes up space, can't store them right up next to each other

* 1. (6 pts) Paste the C program you used to "reverse engineer" your answers above. It is assumed that your program probably "morphed" over time as you experimented, so just paste the last version you used.

**int** main ()  
{  
 **struct** alpha thing;  
  
 thing.f1=5;  
 thing.f2=51.324;  
 thing.f3=**'g'**;  
  
 printf(**"Size of struct: %lu"**, **sizeof**(thing));  
  
 printf(**"\nThe address of f1: %p"**, &(thing.f1));  
 printf(**"\nThe address of f2: %p"**, &(thing.f2));  
 printf(**"\nThe address of f3: %p"**, &(thing.f3));  
  
}

1. (9 pts, 3 pts each) Given the binary tree in the diagram below, give the order that the nodes will be "processed" in each of the standard tree traversals.

Pre-order:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | d | g | h | c | e | j | k | f | m |

In-order:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| g | d | h | b | a | j | e | k | c | f | m |

Post-order:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| g | h | d | b | j | k | e | m | f | c | a |

1. (10 pts) *Tree* data structures are not as "generic" as *list* data structures because the rules for building a tree are often problem specific. Describe the rule(s) used to build each of the following types of trees.
   1. (2 pts) Binary search tree (<https://en.wikipedia.org/wiki/Binary_search_tree> )

Data is sorted, left child less than parent, right child greater than parent, height balanced

* 1. (3 pts) Quad trees for representing 2D space in a plane. (<https://en.wikipedia.org/wiki/Quadtree> )

each node has 4 children, subdivides space to clarify image, node data is spatial information

* 1. (3 pts) Octrees for representing 3D space. (<https://en.wikipedia.org/wiki/Octree> )

8 children for each node, subdivides space to better represent object

* 1. (2 pts) Game tree for representing the possible "moves" in a game. (<https://en.wikipedia.org/wiki/Game_tree> )

Children are all possible next moves, complete tree contains all possible outcomes of a game