**Honors Computer Science I (COP 3502) Exam #2 Solutions**
**Date: 3/14/2024**

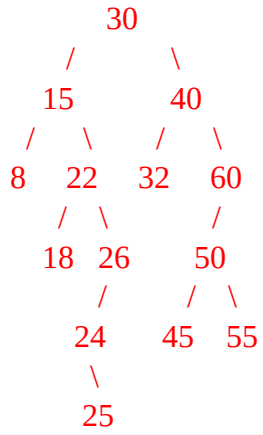
1) (6 pts) What is the output of the following C program?

```
#include <stdio.h>

void f(int a, int b) {
    if (b == 0) return;
    printf("%d ", a);
    f(2*a, b-1);
}

int main() {
    f(3,6);
    return 0;
}
```

**3 6 12 24 48 96**
**Grading: 1 pt per item**

2) (7 pts) The following is the preorder traversal of a **binary search tree**. Draw the tree below.

30, 15, 8, 22, 18, 26, 24, 25, 40, 32, 60, 50, 45, 55




**Grading: ½ per each item, round down, item has to be in 100% correct place to get credit.**

3) (12 pts) In class we wanted to try subsets of 7 unique letters to see how many scrabble words they could form. Complete the code below so that it prints out all combinations of 7 letters out of the 26 letters (each in alphabetical order).

```
#include <stdio.h>
#include <string.h>

void go(char* cur, int k, int n);

int main() {
    char word[8];
    word[0] = '\0';
    go(word, 0, 7);
    return 0;
}

void go(char* cur, int k, int n) {

    if (strlen(cur) == n) {

        printf("%s\n", cur);        // 2 pts

        return;                     // 1 pt

    }

    int start = 0;
    if (k > 0) start = cur[k-1]-'a'+1;  // 3 pts

    for (int i=start; i<26; i++) {

        cur[k] = (char)('a'+i) ;        // 2 pts

        cur[k+1] = '\0' ;               // 2 pts

        go(cur, k+1, n) ;               // 2 pts
    }
}
```

4) (15 pts) Imagine using a doubly linked list to store subway stops on a line. Write a function, printLoop, that takes in a pointer to a node in a doubly linked list representing a subway line, and prints out each stop visited on a round trip that starts at that node, goes all the way "to the right" following the next pointers until reaching the last node/subway stop, then goes all the way "to the left" following the previous pointers, and finally comes back moving right, ending at the original starting location. For example, if the full double linked list stored "first" ↔ "second" ↔ "third", ↔ "last", and the function was given a pointer to the third node, then the function should print

```
third last third second first second third
```

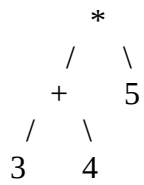
Notice that at both turns, the stops at the end are only printed once, not twice. (Hint: writing three separate for loops, instead of while loops, shortens the code.)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct dll {
     char name[100];
     struct dll* prev;
     struct dll* next;
} dll;

void printLoop(dll* current) {

     dll* tmp;
     for (tmp=current; tmp->next!=NULL; tmp=tmp->next)
          printf("%s ", tmp->name);
     for (;tmp->prev!= NULL; tmp=tmp->prev)
          printf("%s ", tmp->name);
     for (;tmp!=current; tmp=tmp->next)
          printf("%s ", tmp->name);
     printf("%s ", tmp->name);


}
```

**Grading: This can be written so many ways that the points are for tasks:**

**Going from left to right 1st time: 4 pts**
**Going all the way from right to left: 5 pts**
**Going from left to right 2nd time: 4 pts**
**Making sure ends print exactly once: 2 pts**

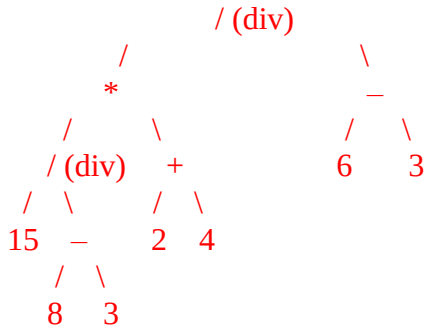
5) (10 pts) Evaluate the following postfix expression shown below (just figure out its final value) and represent this expression in a binary expression tree. To form a binary expression tree, put the operator in the root node, the first operand in the left child and the second operand in the right child. For example, the expression tree for (3 + 4) * 5 is




Postfix Expression: 15 8 3 – / 2 4 + * 6 3 – /

Value of Expression: **6**

Draw Expression Tree Below:




**Grading: 2 pts root**
**2 pts right subtree**
**6 pts left subtree (3 pts its left, 2 pts its right, 1 pt operator)**

6) (5 pts) What is the fewest number of nodes in an AVL tree with height 6? Briefly explain how you got your answer.

Answer: **33**

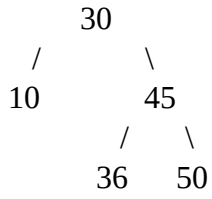
Reasoning/Work:

In class we showed that to build AVL tree of height h with the minimal number of nodes, you use one AVL tree of height h-1 another of h-2 and make them subtrees with a new root. Let f(h) = the minimal number of nodes in an AVL tree of height h. Then f(h) = f(h-1) + f(h-2) + 1, with f(0) = 1, and f(1) = 2. Now, just plug in repeatedly: f(2) = 2 + 1 + 1 = 4, f(3) = 4 + 2 + 1 = 7, f(4) = 7 + 4 + 1 = 12, f(5) = 12 + 7 + 1 = 20, and f(6) = 20 + 12 + 1 = 33. **Grading: 2 pts all or nothing for answer, 3 pts for reason. Can state $F_{h+3}$ result if they remember it.**

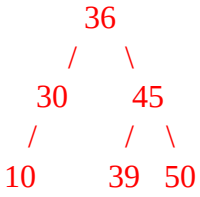
7) (15 pts) Question 5 referred to the fact that we can store an arithmetic expression in an expression tree. Note that all operands are stored in leaf nodes and all operators are stored in internal nodes with two children in such a tree. Write a function that takes in a pointer to the root of an expression tree and returns the value of the expression. It's guaranteed that the pointer is pointing to a valid expression tree (so no NULL check is necessary for the root). The struct storing a node is stored below, but to allow to store both numbers and operators, both fields will exist in every node, with only one of the fields being meaningful. The struct is given below, The valid operators are '+', '-', '*' and '/'. Numbers are stored as doubles and you may assume that you will be given well-formed expressions that do not contain any sub-operation that is divide by 0.

```
typedef stuct exprnode {
    double x; // value if value is stored.
    char op; // operator if operator is stored.
    struct exprnode* left;
    struct exprnode* right;
} exprnode;

double eval(exprnode* root) {

    // 3 pts.
    if (root->left == NULL && root->right == NULL)
        return root->x;

    // 2 pts for each recursive call.
    double leftVal = eval(root->left);
    double rightVal = eval(root->right);

    // 3 pts each for first three choices.
    if (root->op == '+') return leftVal+rightVal;
    if (root->op == '-') return leftVal-rightVal;
    if (root->op == '*') return leftVal*rightVal;

    // 1 pt for last default choice.
    return leftVal/rightVal;
}
```

8) (5 pts) Show the result of inserting 39 into the AVL Tree shown below. (Draw a box around your final answer.)




Inserting the 39 regularly, we get an imbalance at 30. Label A = 30, C = 45, B = 36. The final result is:




**Grading: 1 pt root, 1 pt left subtree, 3 pts right subtree, 3/5 if 39 is attached as 30's right.**

9) (9 pts) A calculator has a single positive integer on its display and has three operations: (a) add a digit to the number (1 through 9), (b) multiply the number by a digit greater than 1 (2, 3, …,9) or (c) concatenate a digit (0 through 9) to the end of the number. Given a starting number on the calculator display and a desired finish number on the calculator display, a breadth first search (discussed in class) can be used to figure out the fewest number of operations necessary to transform the starting number to the ending number. In the breadth first search, given a current number, all numbers that can be reached with a single operation must be calculated. Write a function that takes in a positive integer $n < 10^5$ and prints out each number that can be obtained from n in a single operation described above. It's okay if a particular number is printed more than once because it can be obtained in a single move in more than one way. Values can be printed in any order.

```
void printnext(int n) {

    for (int i=1; i<10; i++) printf("%d ", n+i);
    for (int i=2; i<10; i++) printf("%d ", n*i);
    for (int i=0; i<10; i++) printf("%d ", 10*n+i);
    printf("\n"); //optional;
}
```

**Grading: 3 pts for each option.**

10) (15 pts) In games such as Wheel of Fortune, we often know some of the letters in a word but not all of them. For example, if the word we were trying to guess was "vacation", then it's possible we might only know the location of the a's and n. In this case, we could express our knowledge of the word as "*a*a***n", where a '*' character stands in for each unknown character. For this problem, you'll use a trie to determine the number of words that a given string in this format matches in the dictionary of words stored in the trie. Complete the code below (wrapper function is complete and has initial call to the recursive function) to solve the problem.

```
typedef struct trie {
    int isWord;
    struct trie* next[26];
} trie;

int numfit(trie* root, char* mold) {
    return numfitrec(root, mold, 0, strlen(mold));
}

int numfitrec(trie* root, char* mold, int k, int n) {

    // 1 pt
    if (root == NULL) return 0;

    // 2 pts
    if (k == n) return root->isWord;

    // 1 pt
    int res = 0;

    // 2 pts check for case.
    if (mold[k] == '*') {

        // 5 pts total
        for (int i=0; i<26; i++)
            res += numfitrec(root->next[i], mold, k+1, n);
    }

    // 3 pts total
    else
        res += numfitrec(root->next[mold[k]-'a'], mold, k+1, n);

    // 1 pt
    return res;
}
```

Note: The Wheel of Fortune analogy was misleading. In that game, when something is covered, you know it can't equal one of the letters showing, but my intention here was for the '*' to be a wildcard match. I'll give full credit if someone disallowed all letters showing in the rest of the mold (but coding this is very annoying.)

11) (1 pt) What's the first name of the owner of Beth's Burger Bar? **Beth (Give to All)**