Fall 2023 COP 3502 Section 1 Final Exam - Part B (100 pts) 12/8/2023 Solution

1) (10 pts) Write a function that takes in a positive integer n, dynamically allocates a 2D integer array of size n by n, fills it with a design of integers in between 1 and n, where each cell stores how many items are on its "backward diagonal", and returns a pointer to the array. For example, for n = 5, the array should be filled as follows:

1	2	3	4	5
2	3	4	5	4
3	4	5	4	3
4	5	4	3	2
5	4	3	2	1

Fill in the function prototype below. You may use any functions from the math library as needed.

```
int** diagcount(int n) {

    int** sq = calloc(n, sizeof(int*));
    for (int i=0; i<n; i++)
        sq[i] = calloc(n, sizeof(int));

    for (int d=0; d<2*n-1; d++) {
        int start = d < n ? 0 : d-n+1;
        int draw = d < n ? d+1 : 2*n-1-d;
        for (int i=start; d-i>=0 && i<n; i++)
            sq[i][d-i] = draw;
    }

    return sq;
}


// Alternate solution using 2 loops.
int** diagcount_alt(int n) {

    int** sq = calloc(n, sizeof(int*));
    for (int i=0; i<n; i++)
        sq[i] = calloc(n, sizeof(int));

    for (int d=0; d<n; d++)
        for (int i=0; i<=d; i++)
            sq[i][d-i] = d+1;

    for (int d=n; d<2*n-1; d++)
        for (int i=d-n+1; i<n; i++)
            sq[i][d-i] = 2*n-1-d;

    return sq;
}
```

Grading: 3 pts for mem alloc, 2 pts for loop structure, 4 pts for logic to fill in grid, 1 pt for return, many other solutions possible.

2) (10 pts) Write a function that takes in a pointer to a linked list and uses it to create a new linked list of the same size storing the values in the input linked list in the **reverse order** of the original list, returning a pointer to the front of this newly created list. **Write your function iteratively, repeatedly creating a new node and inserting it into the front of the new list you are creating.** (For example, if the input list to the function stores 2, 6, 5, 7, then your function will create a new list, inserting 2 to the front, then 6 to the front, then 5 to the front and finally 7 to the front to generate a list in the order 7, 5, 6, 2, returning a pointer to the node storing 7.)

```
typedef struct node {
    int data;
    struct node* next;
} node;

node* newrevlist(node* front) {

    node* cur = NULL;
    while (front != NULL) {
        node* tmp = malloc(sizeof(node));
        tmp->data = front->data;
        tmp->next = cur;
        cur = tmp;
        front = front->next;
    }

    return cur;

}
```

Grading: works in null case – 1 pt
Works in one node case – 2 pts
Loops through input list – 2 pts
Mallocs new nodes in loop – 1 pt
Copies data from input list into - 1 pt
List building works – 2 pts
Return – 1 pt

3) (6 pts) Evaluate the value of the following postfix expressions. No need to show the stack, you'll only be graded on the final answer.

(a) 18 3 / 3 2 + 4 5 – + * **24**

(b) 6 5 4 3 2 1 + * + * + **71**

(c) 8 2 5 * + 9 3 – / 7 * **21**

Grading: 2 pts for each, all or nothing

4) (4 pts) What does the function f(13, 12) return, where f is defined below?

```
int f(int a, int b) {
    if (b == 0) return 0;
    int res = a * (b & 1);
    return res + (f(a, b>>1) << 1);
}
```

156

Note: What the code is doing is just multiplying a by b. To see this, notice that each time, if the least significant bit is 1, then a is added, otherwise it's not. What the recursive call does is the same operation as before, but chops off the least significant bit of b. Then, this result is left shifted and added to the original. This ensures that the "contribution" of each copy of a is multiplied by the appropriate power of 2, based on where it shows up in b's binary representation. Essentially, since 12 = 1100 in binary, this code will have the effect of adding 8 x 13 + 4 x 13 = 156.

Grading: 4 or 0, unfortunately it's all or nothing

5) (8 pts) Consider the problem of earning money until you get at least a target amount of dollars. You have a job where if on a particular day you earn d dollars, then the next day you will earn d + x dollars, where x is a constant. Write a **recursive function** that takes in integers target, d and x, where target is how much is left to earn, d is the pay for today, and x is the increase factor between days, and returns the minimum number of days (whole number) of work needed to earn the required amount of money.

```
int earnmoney(int target, int d, int x) {
    if (target <= 0) return 0;
    return 1 + earnmoney(target-d, d+x, x);
}
```

Grading: 2 pts if for base case, 1 pt return in base case
1 pt add 1, 1 pt each parameter (only get credit if rec call for these), 1 pt return

6) (10 pts) Please give **an exact solution** (a function that T(n) satisfies in terms of n) to the following recurrence relation using the iteration technique:

$$T(n) = 2T(n-1) + 2^n, for\ n > 0$$
$$T(0) = 4$$

$T(n) = 2T(n-1) + 2^n$

$T(n) = 2(2T(n-2) + 2^{n-1}) + 2^n$

$T(n) = 4T(n-2) + 2^n + 2^n$

$T(n) = 4T(n-2) + 2(2^n)$

$T(n) = 4(2T(n-3) + 2^{n-3}) + 2(2^n)$

$T(n) = 8T(n-3) + 2^n + 2(2^n)$

$T(n) = 8T(n-3) + 3(2^n)$

After k iterations, we have

$T(n) = 2^k T(n-k) + k(2^n)$

Plug in k = n:

$T(n) = 2^n T(0) + n(2^n) = 4 \times 2^n + n(2^n) = (n+4)2^n$

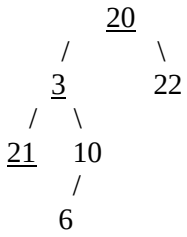
Grading: 1 pt first iteration, 2 pts second iteration, 2 pts third iteration, 2 pt general, 3 pts to finish

7) (6 pts) Show the contents of the following array right after a partition is performed on it using index 0 as the partition element. Please use the partition algorithm shown in class.

index	0	1	2	3	4	5	6	7	8	9	10	11
orig array	13	16	9	14	17	23	5	6	22	1	8	22
after part	5	8	9	1	6	13	23	17	22	14	16	22

Grading: floor(correct/2)

8) (10 pts) Consider the following problem: given a binary tree, find the maximum possible value that can be obtained by including the values stored in each node on a single path from the root node to any leaf node. For example, in the tree below, the nodes on the path that provides the maximum sum are underlined.

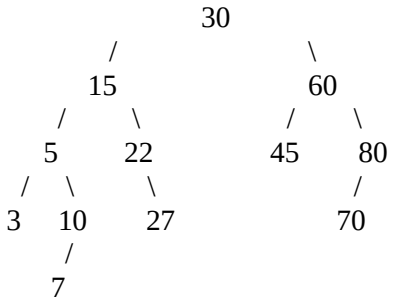


Write a function that takes in a pointer to the root of a binary tree and returns this maximal sum. Your function **must be recursive.**

```
typedef struct treenode {
    int data;
    struct treenode* left;
    struct treenode* right;
} treenode;

int summaxpath(treenode* root) {

    if (root == NULL) return 0;

    int left = summaxpath(root->left);
    int right = summaxpath(root->right);

    if (left > right)
        return left + root->data;
    return right + root->data;
}
```

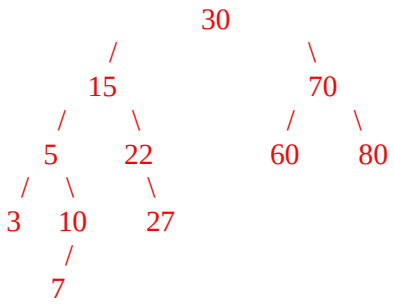
Grading: 2 pts root NULL case,
2 pts each rec call
2 pts if or ? for larger side
1 pt to add root->data to left call
1 pt to add root->data to right call

9) (8 pts) Show the result of deleting the value 45 from the AVL tree shown below:

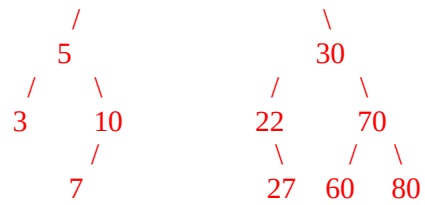


Since there are two tree rebalances, please draw a box around the state of the tree after the first rebalance, and then again after the second/final rebalance. **So you should have two boxes around two trees.**

Tree After First Rebalance



Tree After Second Rebalance



Grading: First Pic →1 pt remove 45, 3 pts for positions of 60, 70 and 80. (0 if not BST)
Second Pic → 1 pts 15 at root, 1 pt 5 left, 1 pt 30 right, 1 pts attach rest (0 if not BST)

10) (10 pts) Consider the task of printing out (in lowercase letters) the **last word** alphabetically stored in a trie. (You may assume if a link exists in a trie, that some word exists further down that link.) Write a **void recursive** function to accomplish this task. (Note: Different recursive calls will print different letters in this word.)

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

void printlast(trienode* root) {

    if (root == NULL) return;

    for (int i=25; i>=0; i--) {
        if (root->next[i] != NULL) {
            printf("%c", (char)('a'+i));
            printlast(root->next[i]);
            break;
        }
    }

}
```

Grading: 2 pts base case
3 pts loop backwards 25 to 0
1 pts check for NULL
2 pts print letter
1 pt rec call
1 pt break or return here or some mechanism to not print too much

11) (5 pts) Convert 917 in base 10 to base 5.

5 | 917
5 | 183 R 2
5 | 36 R 3
5 | 7 R 1
5 | 1 R 2
5 | 0 R 1

12132

Grading: 1 pt each "digit", if correct digits but wrong order do 4/5. Must do division, remainder process to get points.

12) (10 pts) In the game of NIM, a player is given several piles of stones and must select one of the piles to take 1 or more stones from. A player (currently moving) can win if the bitwise XOR of the number of stones in all the piles is any non-zero number. Any move the player makes to change this bitwise XOR to 0 (after her move) is an optimal move. In particular, let x = the bitwise XOR of the number of stones in all the piles and let piles[i] be the number of stones in piles[i]. As long as the bitwise XOR between x and piles[i] (call this xor computation y) is less than piles[i], then an optimal move would be take y stones from pile i. Write a function that takes in an array, piles, and integer n, representing the number of piles, and returns an integer array of size 2, where index 0 stores which pile (0 based) to remove stones from, and index 1 stores how many stones to take for an optimal move. If no move exists to return the state of the piles to have a bitwise XOR of 0, then return NULL. Some of the code has been given.

```
int* getwinmove(int* piles, int n) {

    int xor = 0;
    /* Fill in code here. */
    for (int i=0; i<n; i++)            // Grading: 1 pt
        xor ^= piles[i];               // Grading: 2 pts

   if (xor == 0) return NULL;

   int* res = calloc(2, sizeof(int));

    /* Fill in code here */
    for (int i=0; i<n; i++) {          // Grading: 1 pt
        int leave = piles[i]^xor;      // Grading: 2 pts
        if (leave < piles[i]) {        // Grading: 2 pts
            res[0] = i;                // Grading: 1 pt
            res[1] = piles[i]-leave;   // Grading: 1 pt
        }
    }



    return res;
}
```

13) (3 pts) George Boole passed away on December 8th, 1864, exactly 159 years ago. What type of expression, found in if statements, is named after him?

Boolean (Give to All)