Fall 2023 COP 3502 Section 4 Final Exam - Part A (25 pts) 12/7/2023 Solution

For this portion of the exam, you'll complete a solution to the following problem:

There are **n** points of interest on the Cartesian grid for this problem. You are an Uber driver starting at the first pointed listed amongst these **n** points of interest. You will make **n**-1 trips, visiting each of the other locations exactly once. The amount of money you make for a single trip is the square of the distance between the two points you are traveling. Since you want to maximize profit, the location you visit from your current location will always be the farthest location that has yet to be visited, from your current location. If there is a tie between multiple locations, go to the one with the minimum x coordinate. If there is still a tie, go to the one with the smallest y coordinate. Here is the input/output format:

First line will contain a single positive integer, **n** ($2 \leq \mathbf{n} \leq 1000$), the number of points of interest.

The following **n** lines will contain two space separate integers each: $\mathbf{x_i}$ ($0 \leq \mathbf{x_i} \leq 10^4$) and $\mathbf{y_i}$ ($0 \leq \mathbf{y_i} \leq 10^4$), respectively, representing the x and y coordinate of the i^{th} point ($1 \leq i \leq \mathbf{n}$). The output will be a single integer, the money you'll make following the directions stated.

Here is a skeleton of the solution, including function prototypes and some parts of main:

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

long long distsq(long long* pt1, long long* pt2);
int farthest(int curI, int* used, long long** pts, int n);
int beat(long long** pts, int curI, int i, int j);

int main() {

    int n;
    scanf("%d", &n);
    long long** pts = calloc(n, sizeof(long long*));
    for (int i=0; i<n; i++) pts[i] = calloc(2, sizeof (long long));

    for (int i=0; i<n; i++)
        scanf("%lld%lld", &pts[i][0], &pts[i][1]);

    /* Allocate and initialize used array. (Question 2) */
    int curI = 0;
    long long res = 0;

    for (int i=0; i<n-1; i++) {
        /* Fill in code for traveling each subsequent path. (Question 4) */
    }

    printf("%lld\n", res);
    /* Free dynamically allocated memory. (Question 5) */
    return 0;
}
```

Thus, the (x, y) points are stored in a 2 dimensional array of size n by 2. The coordinates are stored as long long to avoid overflow that might occur with int. The logic for the beat function is complicated, so that is given to you below:

```
int beat(long long** pts, int curI, int i, int j) {

    long long d1 = distsq(pts[curI], pts[i]);
    long long d2 = distsq(pts[curI], pts[j]);

    if (d1 > d2) return 1;
    if (d1 == d2 && pts[curI][0] < pts[i][0]) return 1;
    if (d1 == d2 && pts[curI][0] == pts[i][0] && pts[curI][1] < pts[i][1])
        return 1;

    return 0;
}
```

Basically, this function returns 1, if, when we are currently at point curI (the point stored in the 1D array pts[curI]), traveling to point i (pts[i]) is "better than" traveling to point j (pts[j]), according to the criteria we are using to select the next point to travel to.

For this question, you'll complete the distsq function, a portion of the farthest function, and the missing portions of main.

1) (5 pts) Complete the distsq function. This function should return the square of the distance between the two Cartesian points pointed to by pt1 and pt2. (Note: pt1[0] and pt2[0] store x coordinates for the 2 points and pt1[1] and pt2[1] store the y coordinates for the two points.) **No credit will be given if any floating point operations are used at all.**

```
long long distsq(long long* pt1, long long* pt2) {
    return (pt1[0]-pt2[0])*(pt1[0]-pt2[0]) +
           (pt1[1]-pt2[1])*(pt1[1]-pt2[1]);
}
```

Grading: 1 pt diff x, 1 pt square it without pow,
1 pt diff y, 1 pt square it without pow, 1 pt for add those and return

2) (5 pts) In main, the used array will keep track of which points out of the n will be visited. It should be an integer array. Dynamically allocate space for this array and set all elements to 0 except for the item at index 0, which should be set to 1, since this is the starting location. (This should be 2 lines of code.)

```
    int* used = calloc(n, sizeof(int));
    used[0] = 1;
```

Grading: array name can be anything as it's not referenced at all in given code.
4 pts first line - 1 pt dec, 1 pt malloc/calloc, 1 pt params, 1 pt fill all 0
1 pt second line (1 pt to set used 0 to 1.)

3) (6 pts) Now you will complete the farthest function. This takes in the current index of the point you are at, the used array indicating which points have already been visited, the points array itself, and the length of the points array. The goal of this function is to return the **index** of the farthest point from pts[curI] that hasn't yet been used/visited, applying the previously stated tie-breakers. Your code **must call** the beat function to earn any credit. Initially, we will set the result to -1 and update it as necessary. Please place all of your code in the for loop.

```
int farthest(int curI, int* used, long long** pts, int n) {

    int res = -1;
    for (int i=0; i<n; i++) {

        if (used[i]) continue;

        if (res == -1) res = i;
        else if (beat(pts, curI, i, res))
            res = i;
    }

    return res;
}
```

Grading: 1 pt skipping used, 2 pts setting res to I if res was -1,
3 pts for last case (note cases 2, 3 can be combined via short circuiting)

4) (5 pts) In main, fill in the body of the for loop that is missing. Four things need to be done: (1) calculate the index of the next point to travel to via function call. (2) Mark the appropriate point as used, (3) Update the total result of money made, (4) Update the current index location. Please write these four lines of code below:

```
    int nextI = farthest(curI, used, pts, n);
    used[nextI] = 1;
    res += distsq(pts[curI], pts[nextI]);
    curI = nextI;
```

Grading: 1 pt for lines 1, 2, 4, 2 pts for line 3, several orders of these lines work.

5) (4 pts) Please free the memory dynamically allocated for the array pts and the used array which you allocated memory for in question 2. You should have 4 lines of code.

```
    for (int i=0; i<n; i++)
        free(pts[i]);
    free(pts);
    free(used);
```

Grading: 1 pt for each line, pts must be freed after pts[i]…

Fall 2023 COP 3502 Section 4 Final Exam - Part B (100 pts) 12/7/2023 Solution

1) (10 pts) Write a function that takes in a positive integer n, dynamically allocates a 2D integer array of size (2n+1) by (2n+1), fills it with a design of integers in between 0 and n, labeling which "level" a cell is from the center, and returns a pointer to the array. For example, for n = 2, the array should be filled as follows:

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

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

```
int** levelsquare(int n) {

    int** grid = calloc(2*n+1, sizeof(int*));     // 1 pt
    for (int i=0; i<2*n+1; i++)                   // 1 pt
        grid[i] = calloc(2*n+1, sizeof(int));     // 1 pt

    for (int i=0; i<2*n+1; i++) {                 // 1 pt
        for (int j=0; j<2*n+1; j++) {             // 1 pt
            int left = abs(i-n);                  // 4 pts total
            int right = abs(j-n);
            grid[i][j] = left > right ? left : right;
        }
    }

    return grid;                                  // 1 pt
}
```

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 that stores the first, third, fifth, etc. values in the input linked list in the original order, returning a pointer to the front of this newly created list. **Write your function recursively.** (For example, if the input list to the function stores 2, 6, 5, 7, 3, 9 then your function will create a new list storing 2, 5, 3, returning a pointer to the node storing 2. If the input list stored 12, 10, 13, then your function will create a new list storing 12, 13, returning a pointer to the node storing 12.) Watch out for NULL pointers!!!

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

node* oddrankedterms(node* front) {

    if (front == NULL) return NULL;

    node* tmp = malloc(sizeof(node));
    tmp->data = front->data;
    tmp->next = NULL;

    if (front->next != NULL)
        tmp->next = oddrankedterms(front->next->next);

    return tmp;

}
```

Grading: null case – 2 pts
Create node storing first item – 3 pts
Recursive call – 3 pts
Avoiding NULL ptr error with if – 1 pt
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) 12 3 / 2 5 + 9 3 – + * **52**

(b) 1 2 3 4 5 6 + * + * + **95**

(c) 2 3 4 * + 6 4 – / 5 * **35**

4) (4 pts) What does the function g(12, 123) return, where g is defined below?

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

72

Note: What the code is doing is looking at each bit in b one at a time. For each bit that is 1, it adds the value a to the total of what gets returned. Here a = 12 and b = 123. The binary representation of 123 is 1111011, which has 6 ones. It follows that the function returns 12 x 6 = 72.

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

5) (8 pts) Consider the problem of determining in how many whole seconds an object dropped from a given height hits the floor. Any such object has a current height, h, a current downward velocity, v, and a current downward acceleration, a, where a is assumed to be constant. In one second from this state, the new height will be h – v and the new velocity will be v + a. This continues, of course, until the height calculation becomes 0 or negative, indicating that the object hit the floor at that instant. Write a **recursive function**, hitfloor, which will take in the parameters described above, h, v and a, as ints (assume a is positive), and return the number of whole seconds until the object hits the floor. (Note: Your base case should return 0 for any input indicating that an object is already on the ground or below it.)

```
int hitfloor(int h, int v, int a) {

    if (h <= 0) return 0;
    return 1 + hitfloor(h-v, v+a, a);
}
```

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:

$$T(n) = T(n-1) + n2^n, for\ n > 1$$
$$T(1) = 0$$

$T(n) = T(n-1) + n2^n$
$T(n) = T(n-2) + (n-1)2^{n-1} + n2^n$
$T(n) = T(n-3) + (n-2)2^{n-2} + (n-1)2^{n-1} + n2^n$

After k iterations, we have:

$T(n) = T(n-k) + \sum_{i=n-k\_1}^{n} i2^i$

Since we know T(1), plug in n – k = 1, so n – k + 1 = 2:

$T(n) = T(1) + \sum_{i=2}^{n} i2^i = \sum_{i=2}^{n} i2^i$

Thus, solving for T(n) boils down to solving the given summation:

$S = 2(2^2) + 3(2^3) + 4(2^4) + \ldots + n(2^n)$

Multiply S through by 2 and subtract, shifting terms:

$S = 2(2^2) + 3(2^3) + 4(2^4) + \ldots + n(2^n)$

$-2S = \quad 2(2^3) + 3(2^4) + \ldots + (n-1)(2^n) + n(2^{n+1})$

$-S = (2)2^2 + 2^3 + 2^4 + \ldots + 2^n - n(2^{n+1})$

$S = n(2^{n+1}) - (2^2 + 2^2 + 2^3 + 2^4 + \ldots + 2^n)$

$S = n(2^{n+1}) - (4 + (2^{n+1} - 1 - 1 - 2))$, using sum of $1,2,4...,2^n$ then subtracting out missing terms

$S = n(2^{n+1}) - (4 + 2^{n+1} - 4)$

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

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

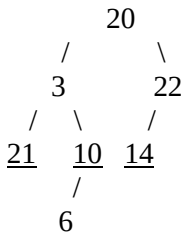
Grading: 1 pt each iteration of rec rel, 1 pt after k iterations, 1 pt written as sum
5 pts determining sum (give partial as needed)

7) (6 pts) Show the contents of the following array after each merge in Merge Sort.

Original	17	13	7	19	22	2	15	14
After 1st	13	17	7	19	22	2	15	14
After 2nd	13	17	7	19	22	2	15	14
After 3rd	7	13	17	19	22	2	15	14
After 4th	7	13	17	19	2	22	15	14
After 5th	7	13	17	19	2	22	14	15
After 6th	7	13	17	19	2	14	15	22
Last	2	7	13	14	15	17	19	22

Grading: 1 pt per row, must get row 100% to earn the point.

8) (10 pts) Consider the following problem: given a pointer to the root of a binary tree, and a non-negative integer, k, find the sum of all the values in the nodes that are exactly k levels below the root node. For example, in the tree below, the sum of the values 2 levels below the root is 21 + 10 + 14 = 45.

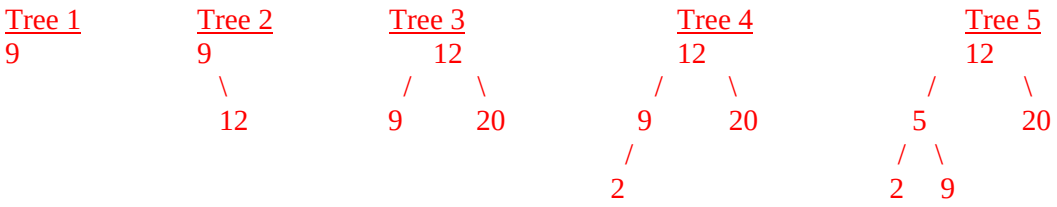


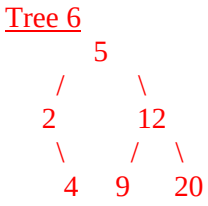
Write a function that takes in a pointer to the root of a binary tree and the value of k and returns this sum. Your function **must be recursive.**

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

int sumdepthk(treenode* root, int k) {

    if (root == NULL) return 0;
    if (k == 0) return root->data;

    return sumdepthk(root->left, k-1) + sumdepthk(root->right, k-1);
}
```

Grading: 2 pts root NULL case,
3 pts k == 0 case
2 pts each rec call, 1 pt add and return

9) (8 pts) Show the result of inserting each of the following items into an initially empty AVL tree in this order: 9, 12, 20, 2, 5 and 4. **Draw a box around the state of the tree after EACH insertion completes. So you should have six boxes around 6 trees.**





Grading: 1 pt for trees 1, 2, 3 and 4
2 pts for trees 5 and 6 (no partial award 0 or 2 for each)

10) (10 pts) Consider the task of printing out the (in lowercase letters) the **first 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 printfirst(trienode* root) {

    if (root == NULL) return; // Only necessary for init. call
    if (root->isWord) return;

    for (int i=0; i<26; i++) {
        if (root->next[i] != NULL) {
            printf("%c", (char)('a'+i));
            printfirst(root->next[i]);
            break;
        }
    }

}
```

Grading: 0 pt first base case
2 pts second base case
2 pts loop forwards 0 to 26
2 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 782 in base 10 to base 4.

4 | 782
4 | 195 R 2
4 | 48 R 3
4 | 12 R 0
4 | 3 R 0
4 | 0 R 3

30032

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) There are n teams in a software competition. Each team has m students. Each student knows some subset of programming languages. There are 20 recognized languages for the competition, numbered 0 to 19, so the subset of languages a single student knows can be stored in a single integer bitmask, which indicates that a student knows language i, if and only if bit i is set to 1. A team can complete a task that requires language i as long as at least one of its team members knows language i. Write a function that takes in an array, languages, where languages[i][j] stores an integer bitmask representing the set of languages that student j on team i knows, as well as the previously mentioned values of n and m, and returns a single integer bitmask storing the set of languages such that all teams could complete a task with that particular language. (Thus, if all teams have at least one individual who knows languages 0, 1, 3 and 5, and the same statement can't be made about any other language, then the function should return $101011_2 = 44$.

```
int getcommonlang(int** languages, int n, int m) {

    // Initialize to all languages.
    int res = (1<<20)-1;

    for (int i=0; i<n; i++) {

        int cover = 0;
        for (int j=0; j<m; j++)
            cover |= languages[i][j];

        res &= cover;
    }

    return res;

}
```

Grading: 2 pts outer loop to n (do 1 pt if to m or something else)
1 pt init var to 0
2 pts inner loop to m (do 1 pt if to n or something else)
3 pts for or equals line
2 pts for correct & equals line

13) (3 pts) After which American linguist/computer scientist/philosopher is Chomsky Normal Form named after? (Note: He was born on December 7, 1928 and is still alive!)

Noam Chomsky (Give to All)