COP 3502C Programming Assignment # 2

Recursion

Please read all the pages before starting to write your code

Please include the following commented lines at the beginning of your code to declare your authorship of the code:

```
/* COP 3502C PA2
This program is written by: Your Full Name */
```

Compliance with Rules: *The UCF Golden Rules apply to this assignment and its submission. In addition, all assignment policies outlined in the syllabus also apply here. The instructor or TA may ask any student to explain any part of their code to verify authorship and ensure clarity. Sharing this assignment description or your code, in whole or in part, with anyone or anywhere is a violation of course policy. Likewise, using code obtained from any outside source, including AI tools, will be considered cheating. Your primary resources should be class notes, lectures, labs, and TAs. These rules are in place to ensure fairness, maintain academic integrity, and give every student the opportunity to develop their own problem-solving and coding skills.*

Deadline:

Please check Webcourses for the official assignment deadline. Assignments submitted by email will not be graded, and such emails will not receive a response in accordance with course policy.

What to do if you need clarification on the problem?

If you need clarification on the problem, a discussion thread will be created on Webcourses, and you are strongly encouraged to post your questions there. You may also ask questions on Discord; if TAs are available, they can respond, and other students might also contribute, helping you get answers more quickly. For further clarification on the requirements, you are welcome to visit the TAs or the instructor during office hours.

How to get help if you're stuck?

As stated in the course policy, office hours are the primary place to seek help. While we may occasionally respond to emails, debugging requests are best addressed during office hours rather than through email.

PA2: Grand Meow-Sters Cat Team Challenge

Objective:

In this assignment, you will:

- Practice designing and implementing **recursive algorithms**
- Write recursive functions that act as **counters and checkers**
- Implement a **permutation algorithm using the used-array technique**
- Work with **structs, pointers, and pointer reassignment**

Background story:

The annual *Grand Meow-Sters Cat Team Challenge* is approaching, and you are responsible for assembling the strongest possible teams of cats. You will organize $n \times c$ cats into n teams, each containing exactly c cats. Each cat is evaluated across five traits (ABCDE) =>Agility, Beauty, Confidence, Determination, and Elegance. The **order of cats within a team matters**, as each position carries a different scoring multiplier.

Your program must consider **all possible arrangements** of cats, accounting for bonuses, penalties, and special interactions between cats. Some cats work exceptionally well together, while others do not. Your goal is to determine the team arrangement that produces the **maximum overall competition score**.

Problem Description:

You will read information about a set of cats and assign them to teams of a fixed size. Every team must contain **exactly** the specified number of cats (no more, no fewer). Your program must examine **all possible permutations** of the cats, evaluate each arrangement, and identify the one that yields the highest total score across all teams.

The final result represents the optimal grouping and ordering of cats that maximizes the combined team score.

Variables and Context:

Each cat is stored in a Cat struct containing dynamically allocated fields for the cat's name and breed, a statically declared array of five trait scores, and a base score equal to the sum of those traits.

Some cats are designated as rivals. Rival pairs are stored using a Rivals struct, which holds pointers to the two cats involved. Rivalry penalties apply only when rival cats appear **adjacent within the same team**.

Your program must generate permutations using the **used-array permutation technique**. A single integer array represents a permutation of cats, where contiguous segments of the array correspond to individual teams.

For example, with 12 cats divided into 4 teams of 3 cats each, a permutation array of size 12 is interpreted as:

- Team 0: indices 0–2
- Team 1: indices 3–5
- Team 2: indices 6–8
- Team 3: indices 9–11

You should **not** permute the Cat structs themselves. Instead, the permutation array stores indices used to access cat information from the global cats array.

A global POSITION_BONUS array provides percentage-based bonuses for cat positions within a team. Only the first c values are used, where c is the team size.

You are required to use **exactly four global variables** and no others:

- Cat *cats //for array of cats
- Rivals *rivals //for rivals array to store the which pair of cats are rivals
- float bestPermScore //store the best score so far
- int **tracker

The tracker array stores the best arrangement found so far, organized as a 2D array where rows represent teams and columns represent cat positions within a team. When a better permutation is discovered, you must copy values from the 1D permutation array into this 2D structure.

For example, if we are working with 12 cats divided into 4 teams, 3 cats in each, and the permutation array currently holds this arrangement:

```
perm = {2, 4, 1, 5, 6, 7, 3, 0, 10, 11, 9, 8}
```

Then, after updating the tracker, which is of size **4 x 3**, it will hold these values

```
tracker = {2, 4, 1,
           5, 6, 7,
           3, 0, 10,
           11, 9, 8}
```

Where **`tracker[0] = {2,4,1}`** represents the cats placed in **team 0,**
tracker[1]={5,6,7} represents the cats placed in team1, and so on.

Required Structures:

```
typedef struct Cat{
        char *name; // dynamically allocated cat name
        char *breed; // dynamically allocated breed
        int scores[MAX_SCORES]; // array of size 5 representing trait scores
        int baseScore; // sum of scores array
} Cat;

typedef struct Rivals{
        Cat *cat1; // pointer to cat1
        Cat *cat2; // pointer to cat2
} Rivals;
```

Must maintain global constants and variable

```
#define MAX_SCORES 5 // size of a cat scores array representing 5 traits
#define MAX_STR 25 // maximum name length of a cat

const float POSITION_BONUS[10] = {3.0, 5.0, 4.0, 6.0, 7.0, 2.0, 8.0, 1.0, 9.0, 1.5};
 // bonus percentages that apply to each position

Cat *cats;
Rivals *rivals;
float bestPermScore;
int **tracker;
```

Team Scoring system:

To compute the score of a current team, you will go through three main steps in order:

1. Find the base score
2. Add bonus points, and
3. Subtract penalty points.

Base Score:

The base score of a team is the average of the base scores of all cats on the team and must preserve decimal precision as you need it while printing up to two decimal places.

$$\boldsymbol{Team\ Base\ Score} = \frac{1}{m} \sum_{i=1}^{m} B_i$$

Where, B_i *is the base score of* i_{th} *cat in the team and* m *is the number of cats on that team.*

Bonus Points:

1. High Performer Bonus
   - For each cat in the team, add **+5 points for each trait score that is 90 or higher**.
2. Synergy Bonus
   - If each cat on the team (e.g., all 3 cats in a 3-cat team) has at least one trait score of 85 or higher, add +30 bonus points to the team's score.
3. Breed Diversity Bonus
   - If all cats on a team are of different breeds, add +10 points to the team score.
4. Position Bonus
   - For each cat on a team, add a position bonus equal to the given percentage of that cat's base score. The percentage is determined by the cat's position within the team using the POSITION_BONUS array. For example, if a cat with a base score of 80 is placed at position 0, add a bonus of 80 × 0.03 = 2.4 to the team score, since POSITION_BONUS[0] = 3.0.

Penalty Points:

1. Rival Penalty
   - For each pair of cats on the team that are listed as "rivals" subtract 25 points from the total team score **if the rivals are next to each other** in this team arrangement**.**
   - Rival penalties are **only applied** when two cats are adjacent to each other on the team.
     i. If the size of a team is 3 and if a pair of rivals exists, and the rivals are placed at positions 0 & 2, the team avoids the penalty altogether
     ii. If the size of a team is 2, this penalty is completely unavoidable
     iii. This is generalized to work on any team size. If rivals are adjacent, apply penalty, otherwise, they're safe.
2. Duplicate Breed Penalty
   - If at least two cats share the same breed in a team, subtract 15 points in total from the team score
     i. Hint: This condition is the logical complement of the diversity bonus criteria. That is, each team will either get the diversity bonus or the duplicate breed penalty. This means that you can perform this check only once, to decide whether the team gets the diversity bonus or the duplication.

Scoring Order: ***Calculate base score → add bonuses → subtract penalties***

Required recursive functions:

In addition to the **required recursive used-array permutation function**, you are required to implement **three recursive** functions to use during the team scoring process. These functions are as follows:

```
countHighPerformersTraits(...)//to count high performer bonus traits in a team

synergyBonusApplies(...)//to check if a team gets a synergy bonus or not

rivalPenaltyApplies(...)//to check if a team gets a rival's penalty or not
```

You are responsible for designing appropriate parameters, return types, base cases, and recursive logic.

Input:

The first line of input will contain two integers, **n (1 <= n <= 6)** and **c (1 <= c <= 10),** where *n* is the number of teams to divide cats into, and *c* is the number of cats per team. It is guaranteed that n+c will be less than or equal to 10. The next **n*c** lines of input will each contain information about a single cat in the following format:

```
<cat_name> <cat_breed> <s0> <s1> <s2> <s3> <s4>
```

where **cat_name and cat_breed are strings (of maximum length of 25)** representing the cat's name and breed, and are to be dynamically allocated, and s0 through s4 are five trait scores for the cat, and their sum is used to compute the cat's base score. Cat names are guaranteed to be unique.

Cats' information is followed by a line with a single integer **r (0 <= r <= n*c)** representing the number of rival pairs in the program. The next **r** lines will each hold two strings (of maximum length of 25) in the following format:

```
<cat1_name> <cat2_name>
```

representing two cats that are rivals. These two names are guaranteed to be on the list of cats previously given in the previous part of the input about cat information.

Output:

The program should output the best permutation arrangement's overall score (to two decimal places), followed by a breakdown of the cats assigned to each team and the teams' scores (to two decimal places). The order of cats in each team matters due to the penalties and bonuses previously explained. Therefore, the members of each team should be printed in the order that maximizes the arrangement score. This is simply printing them in the order they are stored in the array holding the best permutation (the tracker array). Last line of output will indicate which of your cats' teams you are entering the competition with is the best team candidate across all your teams. **If multiple teams have the same score, choose the team that appears first.** Team numbers are displayed using 1-based indexing to align with standard human numbering conventions

Output should be printed in the following format:

```
Best Teams Grouping score: <permutation_score>
Team 1: <cat_1> <cat_2> ... <cat_n> <team_score>
Team 2: <cat_1> <cat_2> ... <cat_n> <team_score>
..
..
Team <t>: <cat_1> <cat_2> ... <cat_n> <team_score>
Best Candidate: <cat_1> <cat_2> ... <cat_n>
```

Sample Program Run:

Example Input (standard input)	Example Output (standard output)

`3 3`
`Coco Ragdoll 85 90 75 80 88`
`Polter Ragdoll 70 85 92 78 81`
`Hana DSH 95 72 80 91 77`
`Leo Tux 88 12 71 83 79`
`Fudge DSH 76 81 89 92 84`
`Pieces DSH 82 77 86 79 93`
`Caesar Siamese 47 82 15 63 91`
`Percy DSH 28 54 76 39 12`
`Pikachu DSH 58 23 87 41 69`
`1`
`Coco Polter`
	`Best Teams Grouping score: 1297.89`
	`Team 1: Caesar Fudge Polter 476.61`
	`Team 2: Percy Coco Leo 375.49`
	`Team 3: Pikachu Pieces Hana 445.79`
	`Best Candidate: Caesar Fudge Polter`

Implementation Requirements:

- You must use the permutation used-array technique in your solution.
- Your permutation algorithm **must** be recursive and must follow the approach shown in class, or a penalty will be applied.
- Your code must include the three required recursive functions
- Your solution **may not include** any global variables other than the four given ones
- You **must free all dynamically allocated memory**. Your code must have zero leaks for full credit.
- Use standard input (scanf/printf). No file I/O is allowed or accepted.
- **For strings, you must first read the input into a statically declared char array, then allocate the exact amount of memory** needed based on its length and copy the string using strcpy function as demonstrated in class. Any other approach will result in a **-100 penalty**.
- Your code must compile and execute on the Codegrade to receive any credit
- Make sure to submit well-structured and well-commented code to avoid penalties.

Hints and Recommended Approach (Important):

- Start by collecting input and ensuring all inputs match properly and are stored in the cats and rivals arrays.
- Consider writing helper functions to calculate a single team's total score given the team indexes as a part of the permutation array
- Write separate functions to aid your recursive solution

- Use the permutation function to generate all permutations of cats, for every permutation, score individual teams by calling a helper scorer function on the corresponding indexes of the team, sum all teams' scores, and store the current permutation evaluation. If this current permutation beats the best perm score so far, update it and update the arrangement stored in the tracker.
- Test with small inputs before trying larger cases
  - **Note the following:** Waiting till the last minute may not be the best idea, as some of the larger test cases will take a while to fully run.

Deliverables:

- Please submit the source file, `main.c`, via Webcourses->Codegrade.