Programming Assignment 1

COP3502 Computer Science I – Spring 2021

Overview

This assignment is intended to make you do a lot of work with dynamic memory allocation, pointers, and pointer arrays.

This assignment is challenging. Read this *entire* document to be sure you understand the assignment. If you wait until the last minute to begin, you will not succeed.

Details

Several small monster trainers have come to you for advice regarding expeditions they're planning into various regions. **You are writing a program to estimate how many monsters of each type they can expect to capture in each region.**

- You've got a Small Monster Index that tells you the name, type, and relative commonality of all the small monsters in question.
  - (A monster's absolute commonality is the same in each region. A monster's *relative* commonality will change region by region as calculations are performed – we'll show you how that works shortly.)
- You've also got an atlas that tells you about the relevant regions and which small monsters are present in them.
- Each trainer tells you which regions they're visiting, and how many monsters they intend to capture per region.
- To estimate the number of a given monster M a trainer will capture in a region R:
  - Divide the relative population of M in R by R's total relative population.
  - Multiply the result by the total number of captures the trainer intends per region.
  - Round this result to the nearest integer. .5 rounds up, so you can use `round()` and its friends. Note that this can result in a total slightly different than the trainer intended!

Data Structures

The structures you'll use for the monsters, regions, itineraries and trainers are shown in the sidebar, and are also provided in **sp21_cop3502_as1.h**. ***You must include this file and use these structures.***

You'll need to allocate, read, compute upon, output from, and subsequently free:

- The monster index.
  - The names and elements of each monster in the monster index.
- The region atlas.
  - The names and monster lists of each region.
- A list of trainers.
  - The names and itineraries of each trainer.
  - The region list of each itinerary.

```
typedef struct monster {
        int id;
        char *name;
        char *element;
        int population;
} monster;

typedef struct region {
        char *name;
        int nmonsters;
        int total_population;
        monster **monsters;
} region;

typedef struct itinerary {
        int nregions;
        region **regions;
        int captures;
} itinerary;

typedef struct trainer {
        char *name;
        itinerary *visits;
} trainer;
```

Example Input and Output

This is one of two example input and output sets we've made available to you.

Example Input

```
8 monsters
StAugustine Grass 12
Zoysia Grass 8
WholeWheat Bread 6
MultiGrain Bread 10
Rye Bread 10
Cinnamon Spice 5
Pepper Spice 10
Pumpkin Spice 30

3 regions

Rome
4 monsters
StAugustine
Zoysia
WholeWheat
Pepper

Helve
5 monsters
StAugustine
WholeWheat
MultiGrain
Rye
Cinnamon

Aria
5 monsters
Zoysia
MultiGrain
Cinnamon
Pepper
Pumpkin

3 Trainers

Alice
5 captures
2 regions
Rome
Aria

Bob
4 captures
3 regions
Rome
Helve
Aria

Carol
10 captures
1 region
Aria
```

Example Output

```
Alice
Rome
2 StAugustine
1 Zoysia
1 WholeWheat
1 Pepper
Aria
1 Zoysia
1 MultiGrain
1 Pepper
2 Pumpkin

Bob
Rome
1 StAugustine
1 Zoysia
1 WholeWheat
1 Pepper
Helve
1 StAugustine
1 WholeWheat
1 MultiGrain
1 Rye
Aria
1 Zoysia
1 MultiGrain
1 Pepper
2 Pumpkin

Carol
Aria
1 Zoysia
2 MultiGrain
1 Cinnamon
2 Pepper
5 Pumpkin
```

Mapping Example

Here’s the table of how each individual trainer’s results are computed. It also shows how rounding issues can lead to trainers capturing more monsters than they intend!

Your program does not need to produce this table. It’s here to show you how the commonality computations work.

Rome	*Raw*	*Divided*	*Alice*	*Round*	*Bob*	*Round*
Coefficient	1.00	36.00	5.00		4.00	
StAugustine	12.00	0.33	1.67	2.00	1.33	1.00
Zoysia	8.00	0.22	1.11	1.00	0.89	1.00
WholeWheat	6.00	0.17	0.83	1.00	0.67	1.00
Pepper	10.00	0.28	1.39	1.00	1.11	1.00
Total	36.00	1.00	5.00	5.00	4.00	4.00

Helve	*Raw*	*Divided*			Bob	*Round*
Coefficient	1.00	43.00			4.00	
StAugustine	12.00	0.28			1.12	1.00
WholeWheat	6.00	0.14			0.56	1.00
MultiGrain	10.00	0.23			0.93	1.00
Rye	10.00	0.23			0.93	1.00
Cinnamon	5.00	0.12			0.47	0.00
Total	43.00	1.00			4.00	4.00

Aria	*Raw*	*Divided*	*Alice*	*Round*	*Bob*	*Round*	*Carol*	*Round*
Coefficient	1.00	63.00	5.00		4.00		10.00	
Zoysia	8.00	0.13	0.63	1.00	0.51	1.00	1.27	1.00
MultiGrain	10.00	0.16	0.79	1.00	0.63	1.00	1.59	2.00
Cinnamon	5.00	0.08	0.40	0.00	0.32	0.00	0.79	1.00
Pepper	10.00	0.16	0.79	1.00	0.63	1.00	1.59	2.00
Pumpkin	30.00	0.48	2.38	2.00	1.90	2.00	4.76	5.00
Total	63.00	1.00	5.00	5.00	4.00	5.00	10.00	11.00

Input and Output in General

Read input from **cop3502-as1-input.txt**. Write output to **cop3502-as1-output-<yourlname>-<yourfname>.txt**. For example, my output file will be named **cop3502-as1-output-gerber-matthew.txt**.

There are blank lines in the sample inputs to make them more readable. They may or may not be present in the actual inputs; you should completely ignore blank lines.

You’ll always get monsters, then regions, then trainers. Print order should be consistent with input:

- Print the trainers in the order you got them.
- Within the trainers, print the regions in the order you got the visits.
- Within the regions, print the monster counts in the order they show up in the atlas for that region.
- Print blank lines between each trainer.

Code Conventions and Specific Requirements

- You need to free *everything* before closing the program. To help you make sure you have:
  - You must `#include "leak_detector_c.h"` in your code, and
  - You must call `atexit(report_mem_leak)` as the first line of your `main()`.
  - (`leak_detector_c.h` and `leak_detector_c.c` are provided. Keep them in your project directory while you're working.)
- I expect to see constructors and destructors for each of the structure types, with appropriate parameters.
- You do not need to comment line by line, but comment every function and every "paragraph" of code.
- You don't have to hold to any particular indentation standard, but you must indent and you must do so consistently within your own code.
- ***You may not use global variables.*** Part of the assignment is to make sure you understand how to pass pointers to complex structures up and down the function stack.
- ***You may not use the return values of assignment statements.*** This makes code far harder to read. Don't do it!
- ***You may not use goto under any circumstances.***
- You may use i/j/k, s/t/u, and x/y/z as integer, string and float/double names for loop control variables, temporary variables and contexts where there is only a single important variable (like a function that takes only one parameter). In similar situations, you may use obvious-from-context single-letter names for structures you have created. ***All other variable and function names must be meaningful!*** This means:
  - If you have more than three variables of a type for any reason, they need names.
  - If you have more than one variable of a type that isn't a loop control or temporary variable, they need names.
  - If you ever find yourself asking "wait, what did that variable do again", it needs a name.