COP 3014: Spring 2022
Homework 5

Total Points: 200 (plus extra credit)
Due: Friday, 04/22/2022 11:59:00 PM EDT through Canvas

1 Objective

The purpose of this assignment is to

- You are familiar with dynamic memory allocation and two dimensional arrays.
- You are familiar with the concepts of bounds checking and maintaining state.
- You can work with pointers, structures, and files in C++.
- You can approach a complex problem, break it down into various parts, and put together a solution.

For this assignment, please make sure you conform to program specifications and Output requirements. You should now be familiar enough with cout statements and output formatting that you should be able to EXACTLY match the sample output (other than certain exceptions you will be informed about beforehand). You should also have had enough experience with functions to match the function specifications (name, argument list, return type, and functionality) EXACTLY. Failure to do either or both will result in a loss of points.

This assignment requires you to submit several files on Canvas. Please do so in a SINGLE submission. Canvas allows you to turn in multiple files in one submission. Once you have uploaded your first file, click "attach another file" to upload your second file and so on.

Turn in your files `snakeGame.cpp` and `roles.cpp` to Canvas.

2 Program 1 - Snake Game

You're at your Grandmother's over Thanksgiving, and your aunt is paying you quite handsomely to watch your 6 year old cousin. However, you have homework to do and video games to play, and like any self-respecting college student, you do not want to hand over your Switch to your cousin. Instead you decide to use one of the old brick cell phones (a family heirloom), as a distraction. Your cousin likes to keep playing a game until the game is well and truly beaten. In order to keep your cousin occupied and still make some money, you rig the game to be unbeatable.

Write a C++ program to simulate a game similar to the snake game. You have a rectangular "field", with a wall around it and obstacles at certain spots. There is one opening in the wall through which the snake enters the field. The snake is infinitely long. The snake makes turns according to input and the game ends when the snake runs into itself. Print an orthographic map (viewed from above) of the field when the game ends.

2.1 Specifications

- Write a function called `createField`.
  - This function takes 2 parameters - the length and the width of the field, and returns a 2 dimensional array of chars. (5 points)

- Create a dynamic 2 dimensional array of characters, with "`width`" number of rows, and "`length`" columns per row. (5 points)
- Mark the boundaries - the wall around the field with a 'W'. Mark all the area in the field with a '.' (10 points)
- Read a number 'N' from the user. Then, read in 'N' points marking the location of obstacles. Obstacles will only occupy one "spot" in the field, specified by a row and column number (0 indexed. Mark the obstacles with an 'O'. (7 points)
- Return this array. (3 points)

- Write a function called `playGame`.
  - This function takes the 2-dimensional array of characters, and the number of rows and columns as parameters, and returns nothing. (5 points)
  - Read in the location of the opening in the wall surrounding the field, specified as a 0-indexed row and column number. Mark the spot with a 'G', for Gate. (3 points)
  - Read in the number of turns 'T' from the user. (2 points)
  - Each turn will consist of a direction and a number of spots. The snake will turn in the given direction and travel the given number of spots.
  - The snake will begin from the location of the gate.
  - Directions are as follows: U - Up, D - Down, L - Left, R- Right. You may assume that the snake will never do a 180 degree turn in one go, that is, if the snake were travelling down, the next turn will only ever be left or right, never up. The directions are also according to your perspective on the map, not the snake's.
  - You may assume that the first turn will work with the location of the gate. That is, if the gate is on the Eastern wall, the first turn's direction will be 'L'. You may also assume the gate will never be at the corner, and the obstacles will not block entry into the field.
  - For every turn, read in the direction and number of spots, and move the snake in the direction, for the given number of spots, and mark the passage of the snake with 'S'. (15 points)
  - If the snake ever runs into itself, stop the game. (5 points)
  - You may assume that the snake will be able to move the given number of spots in the given direction without running into a wall. However, if it runs into an obstacle, move on to the next turn. (5 points)
  - If we finish out all the turns, and the snake is yet to run into itself, stop the game. (5 points)

- In the main function, call the `createField` function and store the address to the returned array in a pointer. (5 points)
- Call the `playGame` function. (5 points)
- When the game is done, print out the entire field. (10 points)
- Delete the dynamic array. (5 points)
- Please do not use `break` or `continue` statements or a dummy `return` in the `playGame` function.
- Make sure the program is well documented (comments). (5 points)

2.2 Sample Run

```
Enter the number of rows: 10
Enter the number of columns: 12
Enter the number of obstacles: 5
Enter the locations of the obstacles:
3,3
5,6
7,7
2,6
7,3
Enter the location of the gate: 0,5
Enter the number of turns: 7
Enter the turns:
D 6
R 5
U 1
L 2
U 2
L 4
D 3
The field is:
WWWWWGWWWWWW
W....S.....W
W....SO....W
W..O.SSSS..W
W....S..S..W
W....SO.SSSW
W....SSSSSSW
W..O...O...W
W..........W
WWWWWWWWWWWW
```

3 Program 2 - Dungeons and Dragons

You are the Dungeon Master(DM) for a great DnD campaign. However, you've had a very busy week, and you don't have a story prepared for this weekend's session. You decide to wing it with a Free for All combat, where the party decides to fight each other to see who comes out on top. To make things easier, you decide to write a program to basically let the session run itself. Please write a program that conforms to the following requirements:

1. Create a structure called `Move` that contains the following data members (5 points)
    - name = cstring of 50 characters
    - damage - integer
2. Create a structure called `Character` that contains the following data members (10 points):
    - name - cstring of 50 characters
    - char_class - cstring of 50 characters
    - hitPoints - integer
    - armorClass - integer
    - isAlive - boolean

- moveName - cstring of 50 characters

3. Open the file `characters.txt`. The file will be in the following format (5 points):

```
numMoves
move1 damage1
move2 damage2
.
.
.
moveM damageM
numChar
name1 class1 HP1 AC1 move1
name2 class2 HP2 AC2 move2
.
.
.
nameN classN HPN ACN moveN
```

4. The first line of the file is an integer M, representing the number of Moves. The next 'M' lines of the file will consist of 2 pieces of data - the name of the move, followed by the damage the move can do. These are separated by a tab. The next line is an integer N, representing the number of characters. The next N lines of the file will consist of 5 pieces f data - the name of the character, the character class, the hitpoints for that character, the armor class of that character and finally the signature move of that character. This data is tab separated as well.

5. Open the file `session.txt`. This file will be in the following format (5 points):

```
numAttacks
character1 move1 target1 roll1
character2 move2 target2 roll2
.
.
.
characterK moveK targetK rollK
```

6. The first line of the file is an integer K, representing the number of attacks. The next K lines, will each consist of the name of a character, the name of a move, the name of a second character who is being attacked, and the dice roll for the success of the attack.

7. Read in the number of Moves M from `character.txt`. Create a dynamic array of M Moves, and populate the structure variables with the data from the file. (10 points)

8. Read the number of characters N. Create a dynamic array of N structure variables, and populate the structure variables with the data from the file. In the beginning, all the characters are alive (10 points).

9. Read in the attacks K from the file `session.txt`. (2 points)

10. For each attack, check if the move used is the signature move of the character using the move. Also check if the roll for the attack is at least equal to the armor class of the character being attacked. If both conditions are met, subtract the move's damage from the hitpoints of the character being attacked. (25 points)

11. If ever the hitpoints of the character drops BELOW 0, that character is dead (5 points).

12. Open the output file `alive.txt`. Print the names of the characters that are still alive and the hitpoints they have left, one per line, in the order they appear in the input file, (10 points).

13. Close the files and delete the dynamic arrays (8 points).

14. Make sure your program is commented properly (5 points)

4 Sample Files

4.1 character.txt

```
9
Eldritch Blast   6
Strike with Longsword   7
Magic Missile   10
Shot with Crossbow  5
Unarmed Strike  4
Whip of Fire    3
Slingshot   3
Poison Dart 4
Bewitching Song 5
6
Bethryn Esvele  Oracle  17  10  Bewitching Song
PatPat  Barbarian   25  16  Strike with Longsword
Kold Steele Fighter   27  14  Unarmed Strike
Grendain    Sorcerer    14  11  Whip of Fire
Fran Solo   Rouge   18  12  Shot with Crossbow
Thruchet    Druid   15  10  Poison Dart
```

4.2 session.txt

```
15
Bethryn Esvele  Bewitching Song Kold Steele 16
Thruchet    Flaming Strike  Grendain    4
Fran Solo   Shot with Crossbow  Patpat  8
Patpat  Strike with Longsword   Thruchet    12
Kold Steele Unarmed Strike  Grendain     13
Grendain    Whip of Fire    Bethryn Esvele  10
Fran Solo   Shot with Crossbow  Patpat  17
Thruchet    Poison Dart Bethryn Esvele  11
Kold Steele Unarmed Strike  Grendain    11
Bethryn Esvele  Bewitching Song Thruchet    16
Patpat  Strike with Longsword   Fran Solo   12
Grendain    Whip of Fire    Thruchet    11
Fran Solo   Shot with Crossbow  Kold Steele 15
Kold Steele Unarmed Strike  Patpat  18
Patpat  Strike with Longsword   Grendain    13
```

4.3 alive.txt

```
Bethryn Esvele  10
Patpat  16
Kold Steele 17
Fran Solo   11
Thruchet    0
```

4.4 Extra Credit - 50 points

Write a function called `sort` according to the following specifications:

- The function should take in 3 parameters - a dynamic array of Character structures, the number of elements in the array, and a character `sortBy` (5 points)
- If the `sortBy` character is 'n', sort the characters by their name, in ascending order. (20 points)
- If the `sortBy` character is 'h', sort by hit points, in descending order. (20 points)
- if the `sortBy` character is anything else, leave the array untouched.
- Call the sort function before printing to the `alive.txt` file. Set `sortBy` to 'n'. We will change it while grading, if necessary. (5 points)
- If you chose to implement this, the output will still be the same characters in `alive.txt`, just arranged differently.

5 Generic Guidelines

1. Include the header comment with your name and other information on the top of your files.
2. **This is individual work. You may NOT collaborate with other students in the course, former students, hire tutors to "help", copy solutions off the internet, or use pay-for solution websites, including but not limited to Chegg, CourseHero, WiseAnt, Bartelby, tutor.com, assorted Social Media groups, whatever GroupMe students might have created, etc.) This includes posting the problem statements on these websites even if you do not use the answer obtained. Doing so is a violation of the Academic Honor Code. Violation of the Honor Code will result in a 0 grade on the program, a reduced letter grade in the course and potentially more serious consequences.**
3. Please make sure you're only using the concepts already discussed in class. That is, please try and restrict yourself to input/output statements, variables, selection statements, loops, functions, arrays, strings, pointers and structures. Using any other concept (like classes, C++ STL data structures, or C++ 11 style iterators) will result in loss of points.
4. If we have listed a specification and allocated point for it, you will lose points if that particular item is missing from your code, even if it is trivial.
5. No global variables (variables outside of main()). constants and symbolic constants are allowed.
6. No `goto` statements or the `auto` keyword.
7. No `break` or `continue` statements. The only exception is the use of the `break` statement to prevent the fall-though in a `switch` statement.
8. Functions have to be declared above main and defined below main. Not doing so will result in a loss of 10 points.
9. This assignment requires the use of dynamic memory allocation. If a program crashes with a segmentation fault, or memory leak, it will result in a loss of 10 points per issue. Please test extensively.
10. Issues with dynamic memory allocation could result in PERMANENT issues on your computer. Please make sure you develop and test using CLion, which is protected against those issues.
11. All input and output must be done with streams, using the library `iostream` or `fstream`

12. You may only use the `iostream, cstring` and `fstream` libraries for programs (you do not need any others for these tasks). You are also not allowed to use C++ string objects. Using other libraries will result in a loss of 10 points.

13. Please make sure that you're conforming to specifications (program name, function names, print statements, expected inputs and outputs etc.). Not doing so will result in a loss of points. Please note that names are case sensitive.

14. NO C style printing is permitted. (Aka, don't use printf). Use cout if you need to print to the screen.

15. When you write source code, it should be readable and well-documented (comments).

16. Make sure you either develop with or test with CLion (to be sure it reports no compile errors or warnings) before you submit the program.

17. Testing your program thoroughly is a part of writing good code. We give you sample runs to make sure you match our output requirements and to get a general idea of how we would test your code. Matching your outputs for JUST the sample runs is not a guarantee of a 100. We have several extensive test cases.

18. Please make sure you've compiled and run your program before you turn it in. Compilation errors can be quite costly. We take 5 points off per compiler error for the first 9 errors. The 10th compiler error will result in a grade of 0.

19. Only a file turned in through Canvas counts as a submission. A file on your computer, even if it hasn't been edited after the deadline, does not count.

20. The student is responsible for making sure they have turned in the right file(s). We will not accept any excuses about inadvertently modifying or deleting files, or turning in the wrong files.

21. **Program submissions** should be done through the Canvas class page, under the assignments tab (if it's not there yet I'll create it soon.) Do not send program submissions through e-mail – e-mail attachments will not be accepted as valid submissions.

22. The ONLY files you will submit via Canvas are `snakeGame.cpp` and `roles.cpp`.

23. **General Advice** - always keep an untouched copy of your finished homework files in your email. These files will have a time-stamp which will show when they were last worked on and will serve as a backup in case you ever have legitimate problems with submitting files through Canvas. Do this for ALL programs.