CS201 – Spring 2024-2025
Homework 4 – Adventure Game
Due May 14th, Wed, 22:00
(Late Deadline May 15th 22:00)

Introduction

This homework builds on your knowledge of vectors, matrices, file input/output, and structs by creating an Adventure Game Simulation. You will read player data from files, assign tasks, simulate dice rolls, compute scores, and display statistics via a menu interface. The RandGen class is used to simulate random rolls, and a matrix will be used to track all rolls for deeper analysis.

Objective

This assignment will help you practice:

- Read player names from a file.
- Read player data (number of tasks and base scores) from another file.
- Read a list of available tasks from a file.
- Shuffle tasks for each player using the provided RandGen class.
- Simulate dice rolls (1–6) using RandGen for each task.
- Compute player scores based on task-dice logic.
- Store dice rolls both in a vector<int> (per player) and in a vector<vector<int>> (matrix).
- Display requested statistics using a menu-driven system.

Inputs, Flow of the Program, and Outputs

Inputs:

players.txt:
Justin
Taylor
Katy

---

playerdata.txt:
5,10
7,5

3,0

---

tasks.txt:
Treasure
Trap
Battle
Rest
Jackpot
DoublePoints
Nothing

Struct Format

```
struct Player {
  string name;
  vector<string> tasks;
  vector<int> rolls;
  int baseScore;
  int score;
  int numTasks;
};
```

Task Logic

Task	Effect
Treasure	+20 points
Trap	-10 points
Rest	+5 points
Jackpot	50 points
Battle	+30 if roll > 3, otherwise -15
Double Points	+2 × roll
Nothing	+0 points

Menu Options

1. Show player stats
2. Show highest scorer
3. Show task statistics
4. Show total number of 6s rolled
5. Exit

Flow of the Program:

The program starts by asking the user to enter three file names (players file, player data file, tasks file).

The program loads player names, loads player base scores and task counts, and loads available tasks.

The tasks are randomly assigned to players.

Each task is processed with a random dice roll to update players' scores.

After loading and simulation, the program displays a menu with 5 options:

1. Show player stats
2. Show highest scorer
3. Show task statistics
4. Show total number of 6s rolled
5. Exit

The user selects a menu option:

If 1: the program asks for a player name and shows the player's stats (tasks, rolls, score).

If 2: the program shows the player with the highest total score.

If 3: the program shows how many times each task appeared.

If 4: the program counts and shows how many times the dice rolled a 6.

After completing an action, the menu is shown again until the user chooses Exit (5).

Outputs:

The program prints:

- The menu options.

The result of the selected action:

- For option 1:

Player name

Final score

List of tasks and corresponding dice rolls

- For option 2:

Name of the highest scorer and their score

- For option 3:

List of tasks with counts

- For option 4:

Total number of sixes rolled

- For option 5 :

“Goodbye!” when exiting.

Output Formatting:

Consistent and clean menu display after every action.

Player stats are listed clearly:

Tasks and their corresponding dice rolls aligned.

Scores are shown as integer values.

Task statistics are neatly listed.

Menu is always displayed again after completing any action.

Special Notes:

No welcome message at the start, the program directly asks for file names.

Tasks are randomly shuffled for players.

Dice rolls are generated randomly but seed is fixed for reproducibility (RandGen::SetSeed(0)).

Function Prototypes & Descriptions (as an example)

Your may choose use the following functions as defined like below as en example , not must, you can have a variety of coding options:

```
bool loadPlayers(const string& filename, vector<Player>& players);
```
Loads player names from a file into the players vector.
Returns true if loading is successful, otherwise false.

```
bool loadPlayerData(const string& filename, vector<Player>& players);
```
Loads each player's number of tasks and base score from a file.
Updates players' baseScore, score, and numTasks fields.
Returns true if successful, otherwise false.

```
bool loadTasks(const string& filename, vector<string>& baseTasks);
```
Loads available tasks from a file into a task list (vector).
Returns true if successful, otherwise false.

```
void shuffleTasks(vector<string>& tasks, RandGen& rng);
```
Randomly shuffles the list of tasks using a random number generator (RandGen).

```
void assignTasks(vector<Player>& players, const vector<string>&
baseTasks, RandGen& rng);
```
Assigns a shuffled list of tasks to each player based on the number of tasks they should receive.

```
void simulateGame(vector<Player>& players, RandGen& rng,
vector<vector<int>>& rollMatrix);
```
Simulates the game:
Rolls dice for each task, updates players' scores based on task logic.
Stores dice rolls in the rollMatrix for statistics.

```
void printMenu();
```
Displays the main menu options to the user.

```
void showPlayer(const vector<Player>& players, const string& name);
```
Shows detailed stats (score, tasks, rolls) for a specific player by name.

```
void showHighestScorer(const vector<Player>& players);
```
Finds and displays the player with the highest final score.

```
void swapStrings(string& a, string& b)
```

Swaps the values of two strings.

```
void showTotalSixes(const vector<vector<int>>& rollMatrix);
```
Counts and displays how many times the dice roll resulted in a six across all players.

```
void showTaskStats(const vector<Player>& players);
```

Displays how many times each predefined task appeared across all players.
Tasks are always counted and printed in the following fixed order:
"Treasure", "Trap", "Battle", "Rest", "Jackpot", "DoublePoints", "Nothing".
This order must be hardcoded as:

```
    vector<string> tasks = {
```

```
        "Treasure", "Trap", "Battle", "Rest", "Jackpot",
"DoublePoints", "Nothing"
    };
```

Returns nothing.

Note – About the shuffleTasks Function

To prevent confusion and ensure consistency among student implementations, you can use the provided shuffleTasks function below. This function uses the Fisher–Yates shuffle algorithm to randomly shuffle the list of tasks.

Using different or custom shuffle implementations may lead to incorrect results and will be penalized.

Please use the following function as-is:

```
void shuffleTasks(vector<string>& tasks, RandGen& rng) {
   for (int i = tasks.size() - 1; i > 0; i--) {
      int j = rng.RandInt(0, i);
      swapStrings(tasks[i], tasks[j]);
   }
}
```

Important: To use this function, you must also implement the *swapStrings* function, which swaps two string values.

Cleaning File Input Lines

When reading lines from text files using getline, especially on Windows systems, it is possible for lines to include trailing carriage return characters (\r). These can affect string comparisons or formatting.

As an optional helper, you may use the following utility function to clean each line after reading:

```
void cleanString(string& s) {
```

```
    if (!s.empty() && s.back() == '\r') s.pop_back();
}
```

This is not required but can help avoid unexpected output mismatches during grading, especially when using files created or edited on different operating systems.

About Submission!

Homework will not be submitted automatically. Students are responsible for manually submitting their assignments on SUCourse before the deadline. Failure to submit will result in a zero grade

IMPORTANT!

If your code does not compile, then you will get **zero**. Please be careful about this and double check your code before submission.

Note: Please avoid using cin.get(), cin.ignore() in your code for your assignment. We are using CodeRunner, and these functions may cause unexpected behaviour.

VERY IMPORTANT!

Your programs will be compiled, executed and evaluated automatically; therefore you should definitely follow the rules for prompts, inputs and outputs. You can check the example test case outputs from SUCourse to get more information about the expected output.

Order of inputs and outputs must be in the mentioned format.

Following these rules is crucial for grading, otherwise, our software will not be able to process your outputs and you will lose some points in the best scenario.

Sample Run

Below, we provide only one sample run of the program that you will develop, for more sample runs please check the SUCourse example test cases.

The *italic* and **bold** phrases are inputs taken from the user.
NOTE THAT these inputs and the newlines after the inputs are missing at SUCourse in the outputs expected from you, so please ignore this as you copy/paste your C++ code from VS/XCode to SUCourse, the same will happen to your code too. You can see samples of SuCourse output in your assignment on SuCourse.

Visual Studio/XCode Outputs

Sample Run 1

```
Choose players file name: players.txt
Enter player data file name (tasks & base scores): playerdata.txt
Enter tasks file name: tasks.txt
Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 1
Enter player name: Justin
Name: Justin
Score: 75
Tasks and Rolls:
     Trap -> 1
     Treasure -> 4
     Nothing -> 1
     Jackpot -> 2
     Rest -> 1

Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 5
Goodbye!
```

Sample Run 2

```
Choose players file name: players.txt
Enter player data file name (tasks & base scores): playerdata.txt
Enter tasks file name: tasks.txt
Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 1
Enter player name: Katy
Name: Katy
Score: 10
Tasks and Rolls:
     Treasure -> 4
     Nothing -> 6
     Trap -> 4

Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 5
Goodbye!
```

Sample Run 3

```
Choose players file name: players.txt
Enter player data file name (tasks & base scores): playerdata.txt
Enter tasks file name: tasks.txt
Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
```

```
4.   Show total number of 6s
5.   Exit
Enter choice: 1
Enter player name: Taylor
Name: Taylor
 Score: 61
 Tasks and Rolls:
     Treasure -> 5
     Rest -> 1
     DoublePoints -> 3
     Nothing -> 1
     Trap -> 1
     Jackpot -> 6
     Battle -> 2

Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 5
Goodbye!
```

Sample Run 4

```
Choose players file name: players.txt
Enter player data file name (tasks & base scores): playerdata.txt
Enter tasks file name: tasks.txt
Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 2
Highest Scorer: Justin with 75 points

Menu:
1.   Show player stats
```

```
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 5
Goodbye!
```

Sample Run 5

```
Choose players file name: players.txt
Enter player data file name (tasks & base scores): playerdata.txt
Enter tasks file name: tasks.txt
Menu:
1.   Show player stats
2.   Show highest scorer
3.   Show task statistics
4.   Show total number of 6s
5.   Exit
Enter choice: 3
Task Statistics:
Treasure: 3
Trap: 3
Battle: 1
Rest: 2
Jackpot: 2
DoublePoints: 1
Nothing: 3

Menu:
Show player stats
1.   Show highest scorer
2.   Show task statistics
3.   Show total number of 6s
4.   Exit
Enter choice: 5
Goodbye!
```

Sample Run 6

```
Enter players file name: player.txt
Error loading players file.
```

Sample Run 7

```
Enter players file name: players.txt
Enter player data file name (tasks & base scores): playerdata
Error loading player data.
```

Sample Run 8

```
Enter players file name: players.txt
Enter player data file name (tasks & base scores): playerdata.txt
Enter tasks file name: task_file.txt
Error loading tasks file.
```

Sample Run 9

```
Enter players file name: players.txt
Enter player data file name (tasks & base scores): playerdata.txt
Enter tasks file name: tasks.txt
Menu:
1. Show player stats
2. Show highest scorer
3. Show task statistics
4. Show total number of 6s
5. Exit
Enter choice: 3
Total number of 6s: 2

Menu:
1. Show player stats
2. Show highest scorer
3. Show task statistics
4. Show total number of 6s
5. Exit
Enter choice: 5
Goodbye!
```

General Rules and Guidelines about Homework

The following rules and guidelines will be applicable to all homework unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) or LAs (Learning Assistants) of CS201. Office hours of TAs/LAs are at the SUCourse.

What and Where to Submit

You can prepare (or at least test) your program using MS Visual Studio 2022 C++ (Windows users) or using XCode (macOS users).

- Your code will be automatically graded using SUCourse. Therefore, it is essential that you ensure your output matches the exact same outputs given in the example test cases provided by SUCourse.
- After writing your code, use the "Check" button located under the code editor in SUCourse to see your grade based on the example test cases used. This grade will give you an idea of how well your code is performing.
- Note that the example test cases used for checking your code are not the same as the ones used for grading your homework. Your final grade will be based on different test cases. Therefore, it is important that you carefully follow the instructions and ensure that your code is working correctly to achieve the best possible grade on your homework assignment.
- To submit your homework, click on the "Finish attempt..." button and then the "Submit all and finish" button. If you wish to submit again before the due date, you can press the "Re-attempt quiz" button.
- Submit your work **through SUCourse only**! You will receive no credits if you submit by any other means (email, paper, etc.).

Grading, Review and Objections

Be careful about the automatic grading: Your programs will be graded using an automated system. Therefore, you should follow the guidelines on the input and output order. Moreover, It is important to use the exact same text as provided in the example test case outputs from SUCourse. Otherwise, the automated grading process will fail for your homework, and you may get a zero, or in the best scenario, you will lose points.

Grading:

- Late penalty is 10% of full grade (only 1 late day is allowed)
- Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.
- If your code does not work because of a syntax error, then we cannot grade it; and thus, your grade will be 0.
- Please submit your **own** work only. It is really easy to find "similar" programs!
- Plagiarism will not be tolerated. Please check our plagiarism policy given in the Syllabus.

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: It is your right to object to your grade if you think there is a problem, but before making an objection please try the steps below and if you still think there is a problem, contact the TA that graded your homework from the email address provided in the comment section of your announced homework grade or attend the specified objection hour in your grade announcement.

- Check the comment section in the homework tab to see the problem with your homework.
- Check the test cases in the announcement and try them with your code.
- Compare your results with the given results in the announcement.

Good Luck!
Filiz Yıldız & CS201 Instructors