Assignment 2 - LRC Report Template

Your Name
CSE 13S - Spring 24

1. Purpose

This program simulates a simplified version of the dice game Left, Center, Right. It allows users to specify the number of players, input a random seed, and then proceeds to simulate the game according to the specified rules. The game involves players rolling dice and redistributing chips based on the outcome of their rolls until only one player remains with chips.

1.1 Randomness

Randomness refers to the concept of unpredictability or lack of pattern in outcomes. While true randomness is theoretically difficult to achieve, pseudorandom numbers are used in this assignment to simulate randomness. Pseudorandom numbers are generated by algorithms and provide a sequence of numbers that appear random, though they are deterministic based on a seed value.

1.2 What is an abstraction

An abstraction is a simplified representation or model of a complex system or concept. In non-computer science terms, it's like using a map to navigate a city instead of memorizing every street and building. Abstractions allow programmers to work with high-level concepts and hide the underlying complexity, making it easier to understand and manage code.

1.3 Why?

Abstractions make debugging easier by isolating specific functionalities, allowing developers to focus on one aspect of the code at a time. Additionally, they enhance code reusability and modularity by breaking down complex systems into manageable components. Abstractions can be written by anyone and serve to encapsulate complex logic or behavior for reuse across different parts of the program.

1.4 Functions

Using functions in the program can enhance readability and maintainability. By breaking down the code into smaller, self-contained functions, it becomes easier to understand and debug. Implementing two functions along with the main function can simplify the code structure while providing clear separation of concerns. However, using too many functions (e.g., 8) might lead to unnecessary complexity if not carefully managed. It's important to strike a balance between modularity and simplicity.

1.5 Testing

For this assignment, testing should cover various aspects of the program's functionality, including input validation, chip redistribution, and winner determination. Comprehensive testing involves considering different scenarios and edge cases to ensure robustness. Examples of inputs to test could include different numbers of players, edge cases for dice rolls, and extreme values for random seeds.

1.6 Putting it all together

Using a pseudorandom number generator and abstractions helps in creating a more organized and modular codebase, which in turn simplifies testing. Abstractions allow developers to isolate specific functionalities, making it easier to verify each component independently. Pseudorandom number generation ensures that the simulation closely resembles real-world randomness, enhancing the accuracy of the test results.

2. How to Use the Program

The user can compile and run the program by following these steps:

- Compile the program using the provided Makefile: make lcr
- Run the program: ./lcr
- Input the number of players (between 3 and 11) when prompted.
- Input the random seed for the simulation when prompted.
- Follow the instructions displayed on the screen to proceed with the game.

3. Program Design
Audience: Future maintainers of the program
Main Data Structures:

1. Player: A struct representing each player in the game. It contains information such as the player's name and the number of chips they currently hold.
2. Die: An array representing the possible outcomes of rolling a die. It maps roll values (0-5) to die positions (DOT, LEFT, CENTER, RIGHT).

Main Algorithms:

1. Prompting and Input Validation: The program prompts the user to input the number of players and the random seed. It then validates these inputs to ensure they are within the specified range and format.
2. Game Initialization: The program initializes the players with chips and sets up the game environment.
3. Simulating Game Rounds: The core algorithm simulates the game rounds, where each player takes turns rolling the dice and redistributing chips based on the outcomes. This algorithm determines the actions of each player and updates the game state accordingly.
4. Determining the Winner: At the end of each round, the program checks if there is only one player remaining with chips. If so, that player is declared the winner.
5. Input/Output Handling: The program handles input/output operations such as displaying prompts, reading user inputs, and printing game results.

3.1 Overall Pseudocode
The program will:

```
Prompt the user for the number of players and random seed.
Initialize players with chips.
while(there is no winner) {
    Loop through number of players{
        Loop through number of times player should roll
            Return a random die number between 0 and 5
            if left
                give to next highest number
            else if right
                give to next lowest number
            else if center
                give to pot
        Loops through number of players //check to see if there is winner
            If num of chips == 0
                Count++
        If count == num players - 1
            Loop through array
                If player_chips > 0 for given player {
                    Winner = given player
                    Add 1 to winner variable
        Print name of player and how many chips they ended with
}
Print player_winner
        }
```

3.2 Descriptions of Functions

Inputs:

1. **Number of Players (stdin)**: The number of players in the game. Input is read from the standard input.
2. **Random-number Seed (stdin)**: The seed used to initialize the pseudo-random number generator. Input is read from the standard input.

Outputs:

1. **Winner (stdout)**: The name of the player who won the game. Output is printed to the standard output.

Functions:

1. **Main Function**:
   - **Inputs**: None (Implicitly reads from stdin)
   - **Outputs**: Winner (Implicitly prints to stdout)
   - **Purpose**: The main function orchestrates the entire game. It prompts the user for the number of players and the random seed, validates the inputs, simulates the game rounds, and determines the winner.
2. **Decision Making**:
   - The program first prompts the user for the number of players and the random seed.
   - It then initializes the player chip counts and validates the inputs.
   - The game loop iterates over each player, simulating their turns and updating chip counts until a winner is found.
   - The winner is determined based on the chip counts of the players.
3. **Game Simulation**:
   - Inside the game loop, each player's turn is simulated.
   - The number of dice rolls is determined by the number of chips the player has, with a maximum of 3 rolls per turn.
   - For each roll, a random number is generated to determine the action (DOT, LEFT, CENTER, RIGHT).
   - Chip counts are updated based on the roll outcomes (keeping chips, passing chips left/right, placing chips in the center).
   - The game continues until only one player has chips left.
4. **Winner Determination**:
   - After each player's turn, the program checks if there is only one player left with chips.
   - If so, that player is declared the winner, and the game loop is terminated.
5. **Output**:
   - Throughout the simulation, the program prints the player name and their chip count at the end of each turn.

- Once a winner is found, their name is printed as the winner of the game.