Assignment 1
Left, Right, Center

Purpose

The purpose of this project is to develop a C program that replicates the dice game Left, Right, and Center (LRC), engaging multiple players. Each player rolls dice, and the results determine how chips are moved among players and into a pot in the center. The main goal of the game is for players to pass their chips based on the dice results. When all players have lost all of their chips, there will be only one winner. The game allows for three to ten players. The code consists of a game loop that controls the flow of rounds and chip distribution, as well as initializing game parameters (number of players, random seed for dice rolls) and checking the winner. One of the main features of this code is the implementation of the Pseudo-Random Number Generator (PRNG) seed. It guarantees that the dice rolls are random and unbiased, ensuring a fair game, and would allow us to reproduce specific game scenarios. The game is both unpredictable in its play and reliable in its execution.

Questions

<u>Randomness</u>: Describe what makes randomness. Is it possible for anything to be truly random? Why are we using pseudorandom numbers in this assignment?

Randomness is when there is no pattern or rules to follow; each outcome has the same probability of happening. We use pseudorandom because true randomness is hard to achieve because computers are based on predictable algorithms. Pseudorandom produces a sequence of numbers that appear random but are actually generated based on a deterministic process. If the initial seed is known, the numbers can be predicted. Pseudorandom is needed for this assignment to replicate the unpredictability and randomness of real dice rolls to ensure fairness in the game [1].

<u>What is Abstraction</u>: When writing code, programmers often use ”abstractions”. Define an abstraction in non computer science terms (Don't google it!)

Like an abstract painting, abstraction is like using colors or shapes to represent something more complex (real-life objects) by looking at a simpler version of it. It simplifies the artwork to its most essential elements. Ex. lists and arrays.

<u>Why?</u>: The last assignment was focussed on debugging. How can abstractions make debugging easier? What other uses are there for abstractions? Hint: Do you have to be the one to write the abstraction?

Abstractions make debugging easier since it helps break down complex problems into manageable parts. Abstractions can be created by someone else depending on the goal of the project. Not only do abstractions simplify debugging, but it also [2]:

- Isolates the problem
- Being able to test abstractions independently
- Being able to reuse code when similar function is needed somewhere else in the code

- Understand purpose of function better by using descriptive names, class, variables
- Easier to extend or modify code without affecting other parts

Function: When you write this assignment, you can choose to write functions. While functions might make the program longer, they can also make the program simpler to understand and debug. How can we write the code to use 2 functions along with the main? How can we use 8 functions? Contrast these two implementations along with using no functions. Which will be easier for you? When you write the Program design section, think about your response to this section.

Using 2 functions, I could initialize the game and game loop and would be called within the main function. Using 8 functions, I would break down the code I wrote using 2 functions into smaller parts with various functions like managing dice rolls, checking for winners, etc., and all called within the main function. Using no functions is suitable for a simple script. I would use more functions since it makes the program more organized and easier to understand each function, very helpful when isolating a problem and when it comes to debugging.

Testing: The last assignment was focused on testing. For this assignment, what sorts of things do you want to test?How can you make your tests comprehensive? Give a few examples of inputs that you will test.

To ensure my tests are comprehensive, I would test with different numbers of players, different initial seed values, and different edge cases (minimum, maximum).

- Accurate tracking of chips
- Correctly identify the winner of the game
- Make sure the rules of the game are followed correctly
- Valid input
- Make sure program is printing the correct output

Putting it all together: The questions above included things about randomness, abstractions and testing. How does using a pseudorandom number generator and abstractions make your code easier to test?

By being able to set a specific value for the seed, I would be able to predict the behavior of my code during testing. My tests would be more deterministic with a fixed seed. I would use abstraction to handle and create different and more focused test cases since my tests will produce the same results every time. If a test case fails, I could use PRNG with a fixed seed and abstractions to isolate the source of the problem.

How to Use the Program

- Enter the desired number of players.
    - The number of players must be an integer between 3 and 10.
    - If the input is not valid, the program defaults 3 players
- Enter a number of your choice to be used for the pseudo random number generator as a seed
    - Must be an unsigned integer
    - If the input is not valid, the program defaults 4823 as a seed
- The program should

Program design

- Data structures
    - Player Array: An integer array representing the number of chips each player has.
    - Enumerated Type (Position): Represents possible outcomes of a dice roll (DOT, LEFT, CENTER, RIGHT).
    - Player_names array: stores names of player from names.h file
- Main algorithms
    - input_num_players()
    - input_seed()
    - set_game()
    - dice_roll()
    - passChipLeft()
    - passChipRight()
    - addChipCenter()
    - game_loop()
    - check_winner()
    - print_winner()

Pseudocode

```
Position Enum with values DOT, LEFT, CENTER, RIGHT
Die array with values [DOT, DOT, DOT, LEFT, CENTER, RIGHT]
Player_names array with names [] in names.h file

main() {
    num_players()
    input_seed()
    game_loop()
}

input_num_players(){
    Initialize num_players to 3, default
    Prompt user to input number of players (between 3 and 10)
        If invalid (less than 3, more than 10)
            Use default value
    Return default number of players
}

input_seed(){
    Initialize seed to 4823, default
    Prompt user to input random-number seed
        If invalid
            Use default value
        Return default value
}

set_game() {
        Initialize each player’s chips to 3
                Loop through every player
}

dice_roll(int num_rolls, int *results){
        For each roll to num_rolls
                Generate random roll (0 ≤ roll < 6) using random()
                Result in results array
}

passChipLeft (playerChips, currentPlayer){
        Identify the left player (currentPlayer - 1)
        Decrease a chip from currentPlayer's chip count.
        Increase a chip in the left player's chip count.
}

passChipRight(playerChips, currentPlayer) {
        Identify the right player (currentPlayer + 1)
        Decrease a chip from currentPlayer's chip count.
        Increase a chip in the right player's chip count.
}

addChipCenter(playerChips, pot, currentPlayer) {
        Decrease a chip from currentPlayer's chip count.
        Increase the pot's chip count by one.
}

game_loop() {
        Initialize empty pot
        Start with player 0,
                While more than one player has chips
                        Print the current player's name
                        Iterate through each player in the list
                                If player has chips
                                        Create array to store die roll results
                                        Call dice_roll, number of rolls and results array
                                        For each die roll result
                                                Generate a random number between 1 and 6
                                                If the number is 1, 2, or 3 (DOT)
                                                        The player keeps the chip (nothing happens)
```

```
                    If the number is 4 (LEFT)
                        Call function passChipToLeft
                    If the number is 5 (CENTER)
                        Call function addChipToPot
                    If the number if 6 (RIGHT)
                        Call function passChipToRight
                Print remaining chips "ends her turn with %d\n"
}

check_winner() {
    Initialize winner to false
    Initialize a count of players with chips
    Iterate through each player
        If only one player has chips
            Set winner to true
            Return player index
        Else
            No winner yet
            Move to the next player to the left (clockwise)
}

print_winnner() {
    Print the name of the winner based on their index in the player_names array
}

End game
```

Function Description

- main():
- int input_num_players():
    - User input (number of players)
    - Returns number of players
    - Prompt user to enter number of players and make sure it's within the valid range (3 to 10)
    - If-statement make sure input is within range, if invalid default to 3.
- unsigned input_seed():
    - User input (random seed)
    - Returns seed value
    - Get a seed value for the random number generator, if invalid default 4823
    - Default value ensures there's always a valid seed
- void roll_dice(int num_rolls, Position *results):
    - Input number of dice to roll and store the outcome in results array
    - Results array with dice outcomes
    - Loop for rolling each die, modulus operator makes sure random outcomes within range

- void passChipLeft(int *chips, int currentPlayer, int num_players):
    - Updates chips array
    - Pass a chip from current player to the player on left
    - Player’s chip counts, current player, num players
    - Modulus operator, circular array
- void passChipRight(int *chips, int currentPlayer, int num_players):
    - Updated chips array
    - Pass chip from current player to the player on the right
    - Chips array, current player, number players
    - Similar to passChipLeft
- void addChipCenter(int *chips, int *pot, int currentPlayer):
    - Chips array, pot (center), current player
    - Updates chip array and pot
    - Move chip from current player to pot
    - Direct manipulation of chip counts
- void game_loop(int num_players, unsigned seed):
    - Input num_players, seed
    - Development/progress of game
    - Dice rolls, chip distribution, player turns
    - Nested loop that
- int check_winner(int *chips, int num_players):
    - Input chips array and num_players
    - Return index of winning player, return false if no player found
    - Check if there is a single winner in a game
    - Counting players with chips and simple way of ending game
- void print_winner(int winner):
    - Index of winning player
    - Prints name of winner
    - Access player_name array for name of winner

References

[1] WolfSSL. “True Random vs. Pseudorandom Number Generation - WolfSSL.” Woldssl.com, 13 July 2021, www.wolfssl.com/true-random-vs-pseudorandom-number-generation/. Accessed 21 Jan. 2024

[2] Hosk. “Understanding Levels of Abstraction Can Improve Your Code Design.” Hosk’s Dynamic Blog, 17 Sept. 2015, crmbussiness.wordpress.com/2015/09/17/understanding-levels-of-abstratction/.