Assignment1 – LRC Report

Airi Kokuryo

CSE 13S – Winter 24

Questions

1. Randomess

It is not possible to create a true randomness as long as it is being calculated on computers. We need to use a human interacted number, such as the seed inputs by human to create a non-predicted randomness. In this assignmetn 1, psudorandom numbers are calculated through a definite mathematical procedure, so it may be predicted. However, the numbers used for this calculation is decided randomly by human inputs, therefore can be said is fairly random.

2. Abstractions

Abstractions are something that takes out the important/main concepts of something more complicated and long.

3. Why?

Abstractions can help programmers organize and look at the whole program in a more simple style. This can help make the debugging easier, because it can also abstract places where bugs may occur. Abstractions can be used for finding unnecessary functions and overheads, making the whole program efficient and simple as possible.

4. Functions

To use several functions with the main, you can call out the other functions in the main and run the main function. When a program is complicated and long, using functions will make the whole program understandable and easier to debug without using no functions, because you can visualize better which part of the program is running when. Also, if the functions are run on main, it is easy to comment out by functions to find out which place is causing the error. On the other hand, if the program is very simple and short, it may be better to write without functions because it can be written quickly. I believe having functions are much easier, because especially in C, we will be facing more problems debugging, and finding out the bugs will take more time than taking time to organize the program so we can easily find out the bugs.

5. Testing

I would like to test for the inputs and for mistakes I may make when programming the systesms. For example, I would like to implement tests for the user inputs, whether it is the valid number or not, as well as check wheher it is a int and not a string or float. Also, I would like to test for errors such as when all player has no chips, the game ends while more than one player has a chip, when there is trouble reading the file “names.h”, and array errors such as there are more input than the array length. The tests may be made comprehensive by

thinking about what kind of problems may occur for wherethere is an output.

6. How to use the program

I would use the pseudorandom number generator to make the output of the die numbers closer to the real random numbers, so the output will be different every time randomly. Also, the idea of abstractions will help me organize the whole system of the program, so I can start thinking from the main functions rather than from inside those functions, making it easier to program and test.

Program Design

Pseudocode and function descriptions:

Function player_num_init:

Inputs: player number by user input Output: num_players Purpose: To output num_players to use on player_name function

- User input of player number
  - if player number is between 3-10, num_players equals user input
  - else, num_players equals 3

Function random_init:

Inputs: random number by user input Output: seed Purpose: To get a seed to use on getting a random number

- User input of random number seed
  - if 0 <= input, seed eqeuals user input
  - else, seed = 4823

Function player_name:

Inputs: num_players Output: list of player_names Purpose: To output the list of player names to use on the function game_run

- create a list to hold player names
  - make exception for reading files

- make exception for input longer or shorter than the list length

Function game_run:

Inputs: num_players, seed, Output: winner_name Purpose: run the actual game and return the winner Description: This is constructed by a loop that goes through each players starting from player0. It includes a counter called no_chip, and this is added for every player that has no chip. If no_chip becomes num_players - 1(this is because it shouldn't include the winner), the player with the chip number != 0 is returned as a winner. I chose the loop structure, because it keeps running the same process, which I thought would be good for saving unnecessary overlay.

- create a list to hold chips for each player
- enum for the die results
- run function player_name

loop: go through all players

- if player has no chip
  - do nothing, no_chip++
- if player has a chip: roll the die
  - if the die is 1,2,3: (DOT) do nothing
  - if the die is 4: (LEFT) add 1 chip to the next player, minus one chip from current player
    - if the current player's chip number becomes 0, no_chip--
  - if the die is 5: (CENTER) minus one chip from current player
    - if the current player's chip number becomes 0, no_chip--
  - if the die is 6: (RIGHT) add 1 chip to the previous player, minus one chip from current player
    - if the previous player's chip number was 0, no_chip++
    - if the current player's chip number becomes 0, no_chip
- if no_chip does not equal num_players - 1
  - reset no_chip to 0
  - start from the first player again
- else if no_chip equals num_players – 1
  - winner equals player with chip number other than 0
  - (error if there is more than one winner)
  - break from the loop
- return winner name

Function Main:

Purpose: This is the main function to run the whole program. It first runs the user input functions, and use the output as the input for game_run function, which gives the output of the game winner.

- Function player_num_init
- Function player_random_init
- Function game_run
- From the return of game_run, print out “winner is: winner_name”
  - Try and exception for both functions

Program Design- revised

Pseudocode and function descriptions:

Function get_player_num

Output: num_players

- User input of player number
  - if player number is between 3-10, num_players equals user input
  - else, num_players equals 3

Function get_seed:

Output: seed

- User input of random number seed
  - if 0 <= input, seed eqeuals user input
  - else, seed = 4823

Function get_random:

Input: seed, count Output: num Purpose: To return a random number(num) according to the count

- decide the srandom using the seed from user input
- take out only the last random number from the for loop

Function *get_player_name:

Input: num_player

Output: player_name[num_player – 1]

Purpose: to return the name of the player using the array from names.h

Function start_game:

Inputs: seed, num_players, chip[], *player[]

Output: winner

Purpose: run the actual game and return the winner

Description: This is constructed by a loop that goes through each players starting from player0. It includes a counter called no_chip, and this is added for every player that has no chip. If no_chip becomes num_players - 1(this is because it shouldn't include the winner), the player with the chip number != 0 is returned as a winner. I chose the loop structure, because it keeps running the same process, which I thought would be good for saving unnecessary overlay.

loop: go through all players

int i=0; //this is used for returning the random dice number, so it will be added every time the dice is thrown

Int no_chip; //this resets every loop where all characters had a turn, until there is only one left without no_chip

Int winner; //this is the array number of the winner, so this will be returned as an integer

Int play=0; //this is used to decide whether the player was able to play the game, and if so, will print out how its turn ended

- do the loop while there is no winner

```
do{
for loop1(){}

for loop2(){}

}while();
```

- for loop1
- if player has no chip
  - do nothing, no_chip++ and i++
- if player has a chip: roll the die
  - if the die is 1,2,3: (DOT) only i++
  - if the die is 4: (LEFT) add 1 chip to the next player, minus one chip from current player
  - if the die is 5: (CENTER) minus one chip from current player
  - if the die is 6: (RIGHT) add 1 chip to the previous player, minus one chip from current player

- for loop2
- go through each character, and if they have no chip, do no_chip++. If they do, write their array number in the winner integer
- if no_chip does not equal num_players - 1
  - reset no_chip to 0
  - start from the first player again
- else if no_chip equals num_players – 1
  - winner equals player with chip number other than 0
  - (error if there is more than one winner)
  - break from the loop
- return winner using the number

Function Main:

Purpose: This is the main function to run the whole program. It first runs the user input functions, and use the output as the input for game_run function, which gives the output of the game winner.

- run get_player_num
- run get_seed
- create a list to hold names of each player
- create a list to hold chips for each player
- enum for the die results
- run function start_game and return winner
- print out the winner's name using the return from above