Electrical & Computer Engineering Department, University of Maryland, College Park

Spring 2022 **ENEE150** Dr. Gang Qu

Project 1: A Random Number Game

Announced: Thursday, 2/10/2022
Posted: Monday, 2/14/2020
Due date: 11:59 pm Mondays, 2/21/2022, 3/7/2022, 3/14/2022

Project Objectives

1. Learn how to design data structure (variables of known or user-defined data type) to complete a project
2. Master the use of `rand()` and `srand()` to generate random numbers
3. Master basic programming concepts: array, file I/O, and command line argument
4. Get familiar with UNIX programming environment
5. Gain experience in data analysis

Project Description

In this project, you will implement a simple game played by four friends, A, B, C, and D, in short. Each of the four players will start with some initial money. In each round, each player will generate a random number between 0 and 99, calculate the sum of the two digits and denote these values by *sum_A, sum_B, sum_C,* and *sum_D* respectively. The person who has the largest value, say player A with *sum_A* for example, is considered as the "winner" of the round. The "winner" will then contribute *sum_A* from his/her current balance (initial money for the first round) for four players to split. Each of the four players will take a random amount of money from *sum_A*. Each player then update his/her balance (note, player A will pay *sum_A*, but will also get a fraction of it back). The game continues until there is one player goes bankrupt (i.e. current balance is less than the amount he/she has to contribute) or a maximal number of rounds have been played. At the end of the game, your program should report some statistics about the game play. Details can be found in the section of "**Rule of the game**".

General Requirements

For this project, your program should run with the following command line arguments:

```
a.out mode seed round names.txt numbers.txt outputFile.txt
```

where `a.out` is the compiled executable, `mode` has a value of 1 or 2 indicating the mode of the game, `seed` is a positive integer representing the seed value for random number generation, `round` is a positive integer representing the maximal number of the rounds the game will be played, `names.txt` is the input file with the names of the four (4) players and their initial balance, `numbers.txt` is the input file of the random numbers the players generate in each round (this file will not be provided in option 2, see below in the section of *Mode of the game*), and `outputFile.txt` is the output file that keeps all the required information of the game. You should do a safety check on the command line arguments. Each of the command line arguments (except the output file, which will be discussed in a later section) are elaborated below.

Mode of the game

There are two (2) options to play the game based on how the random numbers in each round are obtained. Option 1 (`mode = 1`): read from the input file `numbers.txt` (see below for the file format); Option 2 (`mode = 2`): players generate their random numbers during the play.

seed
This is the seed value that will be used to generate all the randomness you would need in the play. The use of a seed value allows you to repeat the execution of your program such that the same results could be created. Note that seed value is needed for both modes of the game.

round
This value indicates the maximal number of rounds a game can play. It ensures that the game will not last forever. It is needed for both modes of the game. More specifically, the game will end if

Option 1: one player bankrupts, or all data from `numbers.txt` have been played, or the maximal number of rounds have been played.
Option 2: one player bankrupts, or the maximal number of rounds have been played

names.txt
This input file contains exactly four (4) lines, each line will have one name for a player and the initial balance of that player. The player's name consists of English alphabets (a-z and A-Z) only and has no more than 15 characters. The initial balance is a positive real number with 2 digits after the decimal point. Below is one sample file.

```
Alice 40.00
Bob 35.85
Carol 50.25
David 45.00
```

numbers.txt
This input file contains multiple lines. The number of lines is not given and you should not assume there is a maximal number of lines for this file. Each line has exactly four (4) positive integers between 0 and 99. They represent the random numbers generated by each of the four players in that round. Below is the first 8 lines of one such file.

```
15 77 86 83
92 86 35 93
27 62 21 49
26 63 59 90
36 72 26 40
29 67 68 11
23 62 30 82
02 29 35 67
```

For example, in the first round, the numbers for Alice, Bob, Carol, and David are 15, 77, 86, and 83 respectively. For single digit numbers, there will be a leading 0. See 02 in the last line.

If any of the input files does not exist, your program should print an error message and exit. If they exist, you can assume that the files follow the above format and requirements.

Output file
This will be discussed after we describe how the game will be played.

Rule of the game

1. Four players will start with their initial balances (provided in the input file *names.txt*).
2. In each round, each player will have a random number between 0 and 99. These numbers are either generated randomly by the rand() function or obtained from input file (*numbers.txt*).
3. Each player adds the two digits in his/her random number to get a sum.
4. The player with the largest sum is the "winner" of this round.
5. The "winner" will contribute to the prize pool from his/her balance in the amount of "sum". If the "winner" does not have enough in his/her balance, the game terminates.
6. Each player will take a random amount from the prize pool such that (1) each player will get at least \$0.01, and (2) there will be no money left in the pool.
7. Update the balance for each player.
8. In option 1, if the maximal number of rounds are played, the game terminates.
9. Go back to step 2 for next round.

The tie break rule is explained in a later section.

A: \$40.00	B: \$35.85
C: \$50.25	D: \$45.00
A: 27	B: 62
C: 21	D: 49
A: 9 (2+7)	B: 8 (6+2)
C: 3	D: 13
D is the winner.	
D contributes \$13.00	
D's balance: \$32.00 (45.00-13.00)	
A takes \$5.08	B takes \$2.64
C takes \$1.37	D takes \$3.91
A: 45.08	B: 38.49
C: 51.62	D: 35.91

This table shows the step-by-step example of one round of play.

Project Requirements

In this project, you need to implement a C program that can be compiled and executed under GLUE UNIX system to simulate this 4-player random number game. The project is designed in three phases with separate submission deadlines so you can complete the whole project on time.

Phase 1. Read this project description and sample input files carefully to understand the project requirements. Make a plan and design the data structure that you would need to complete the project. This includes, but not limited to (1) declaration of the variables, (2) definition of the functions, and (3) definition of the new data type (i.e. structure). For each of these, provide a clear documentation. No real coding is required for this phase.

Here are some examples of variable declarations:

```
FILE *IN1, *IN2, *OUT;    // file names for the input and output
int cur_number[4];        // temporary variables for the random
                          // numbers of the 4 players for the current
                          // round
float balance[4];         // the balance of the 4 players, initial
                          // values are given by input file IN1, will
                          // be updated during each round
```

Phase 2. Complete a C program that implements this random number game, collects information to determine and report the following three categories of outputs:

(1) detailed information of each round in the following format

```
Round 1:
Alice: 27-->09   Bob: 62-->08    Carol:21-->03   David:49-->13
David contributes $13.00 to the pool. New balance: $32.00
Alice takes $5.08, new balance is $45.08
Bob takes $2.64, new balance is $38.49
Carol takes $1.37, new balance is $51.62
David takes $3.91, new balance is $35.91
<leave an empty line here before the next round>
```

(2) individual player information at the end of the game

```
Alice:
initial balance: $40.00
ending balance: $20.14
lost $19.86
average random numbers: 49.97
average sum: 8.67
number of rounds won: 7
<leave an empty line here before the next player>
```

(3) game information at the end of the game

```
largest winning sum: 18 in round 21 (15 18 9 17)
smallest winning sum: 8 in round 15 (8 5 3 7)
largest non-winning sum: 17 in round 21 (15 18 9 17)
average winning sum: 12.33
largest net win: $15.09 Alice in round 21 ($15.09 -$17.50 $1.11
$1.30)
largest net loss: $17.50 Bob in round 21 ($15.09 -$17.50 $1.11
$1.30)
number of rounds with 2-way tied winning sum: 5
number of rounds with 3-way tied winning sum: 0
```

Phase 3. design experiments to run the program from Phase 2, either you own code or the sample executable, to answer the following questions:

(1) if all players start with the same amount of initial balance, what is the relationship between this balance and the number of rounds that a game lasts?

(2) if the player's initial balances are different, how this will impact the number of rounds that a game lasts?

(3) the random numbers generated for all the players are random between 0 and 99 with average (roughly) 49.50. What can you say about the average of sums and the average of winning sums?

Your answer should be concise with supports of experimental data. Take the question "what is the average of random numbers generated by all players?" for example, an answer could be

```
The average of random numbers generated by all players is 49.50. I
ran the sample executable 10 times with the following command

    a.out 2 seed round names.txt outputFile.txt

the seed values for these 10 runs are: xx, xx, …
the values for round are: yy, yy, …
I used the sample names.txt file.
```

```
In the output files, the average of random numbers for these 10 runs
are: zz, zz, …
These 10 averages and their average xyz are all very close to 49.50.
```

Tie break and other special cases

Whenever there is a tie (e.g. when there are two or more players have the same largest sum in a round, when the largest net win/loss occurs on multiple players and/or multiple rounds, when the largest/smallest winning sum occurs in multiple rounds, when the largest non-winning sum occurs on multiple players and/or multiple rounds), you only need to report one instance. Reporting all the instances is a little challenging and not required for this project. You can break the tie anyway you want, but you need to document it clearly in the header part of the your program `p1.c`.

When there is a tie in the largest sum in a round, this largest sum will also be considered as a non-winning sum (because only one player can be the "winner").

Other Information

1. You will be provided a sample executable, some sample input and output files which you can use to test your program. Run the sample executable or check the output files for the output file format requirements.
2. More input files will be used to test your program and they will be posted later.
3. A question and answer file will be posted in the `/public/Project/proj1` directory

Project Submission Instructions:

1. The project must be implemented using C under the GLUE UNIX system in a file named `p1.c` which should be submitted with the following command
   ```
   submit 2020 spring enee 150 010x 3 p1.c
   ```
2. Phase 1: in the `p1.c` file, document the project and define the data/storage structures that you plan to use to complete the project. It is ok to modify this during Phase 2. Only this portion will be graded even if you submit a complete functional code.
3. Phase 2: complete the code `p1.c` to report the required information to the output file.
4. Phase 3: in a text file `p1.txt`, provide your design of the experiments and your answers to the three questions listed in Project Requirements section. Submit `p1.txt` with the submit command for `p1.c` in Phase 1 and Phase 2.

Grading Criteria:

15% Phase 1

75% Phase 2

5%	Correct handing of command line and input/output files
65%	Correct implementation and generation of the output file
5%	Coding style and documentation
-70%	Code that does not compile (will not be graded):
-50%	Code that compiles but cannot execute (will not be graded)

10% Phase 3

Submissions after their respective deadlines will not be accepted for grading for all three phases.