Mini-project

A game with Implementation of dynamic data structures

Objectives of the work

- Use of dynamic data structures (lists and queues)
- Implementation of operations on chained data structures (in C)
- Evaluation of theoretical complexity
- Decomposing a program in sub-programs and design of the global solution
- Writing a report

Important – Sending Works

- The work should be done in **pairs (two students)**
- The report should be registered in **pdf** format
- The source code (a single file) well commented and structured should be registered in **wordPad**
- The files should be sent to the address **recupspace@gmail.com,** while specifying the object of the mail « MiniProject ALGO – Students names -groupes)
- Deadline : Thursday **15/01/2026**
- The display of results in screen should be well structured and readable
- Any copying (Fraud) will be penalized by **Zero.**

Many real-world problems rely on representing data as linear structures (lists, stacks, queues, etc.). In this work, we aim to implement a simple random game using linked lists and queues structures.

Game description

A group of players register for a game and are initially placed in a queue F (player number, player name, age). Queue F then contains **n** players.

Part I

A game round involves 2 players (taken from the head of the queue F). The 2 players ask the system to generate a random integer value, taking turns. If the sum of the digits of the generated value is a multiple of 5, the player receives a score of 1; otherwise, he receives 0.

For example, if the value is ***230541***, the sum of its digits is 15, which is a multiple of 5, so the player receives a score of 1.

The game round is stopped as soon as the *difference* between the scores of the 2 players reaches **3** points.

The round of game is then saved with a round number ("Game 1", "Game 2", etc.), the numbers and names of the two players, and their scores. For each player, the number of rounds won and the number of rounds lost must be recorded.

The **winning** player will play another round of the game, and the **losing** player is placed at *the tail of the queue F*. For another round, a player is selected from the head of the queue F, and the game is played again in the same way, with the winning player.

After *generation of **12 values*** in a round by the system, if the difference of **3** points is not reached, the round is stopped.

Il the *score is equal* between the two players, both are placed at the tail of queue F and, for another round of the game, two other players are selected from the head of the queue F.

Rules

A player who wins three (03) times consecutively is placed in a queue **F1** of priority 1

A player who loses three (03) times consecutively or separately is placed in a queue **F3** of priority 3

If a player loses five (5) times, he is eliminated from the game and is placed in a losing players list (**LP**).

If a player wins five (5) times, even if separated, he will be placed on a list of winning players **(LG),** *ordered in descending* order of points accumulated across the 5 rounds of the game.

For the different game rounds, players are selected according to *queue priority*:

- From queue **F1** if F1 is not empty
- Then from queue **F** if it is not empty
- Then from queue **F3** until it is empty.

If only **one** player remains in queue **F1**, a player from queue **F** will be selected.

If only one player remains in queue **F**, a player from queue **F3** will be selected.

If only one player remains in queue **F3** (at the end), he is declared a *loser* and will be placed on the **LP** list.

Part II : Change strategy

After **3n** rounds of the game (3 times the number of players in the initial queue), there are still players in the queues (the queues are not empty). The game strategy is changed as follows: The player asks the system to randomly generate 2 numbers (integers) successively. If the GCD (greatest common divisor) of the 2 numbers contains at least one digit that belongs to one of the numbers, the player obtains a score of 1; otherwise, he gets zero.

For example, if the system generates the two numbers 462 and 910, GCD (462, 910) = 14. The digit 4 is in the first number (and the digit 1 is also in 910; you only need one digit) --- therefore, the player receives a score of 1.

As soon as the *difference* in points between the players reaches **3 points,** the round of the game is stopped. The winning player will play another round**,** and the losing player moves to queue **F3**.

After *generation of **16 values*** by the system, if the difference of 3 points is not reached, the round is stopped, the player who has the score max is placed in **F** and the other player is placed in **F3**.

If the *score is equal* between the two players, both are placed at the tail of queue **F** and, for another part of the game, two players are selected (from F1, then from F, then from F3, see below).

If a player loses **twice**, even if the losses are separate, he is eliminated from the game and placed on a losing players list (**LP**).

If a player wins **twice in a row**, he is placed on a winning players list (**LG**), ordered by descending order of points accumulated over the two rounds of game.

For the different game rounds, players are selected according to *queue priority* (as in Part I):

- From queue **F1** if F1 is not empty
- Then from queue **F** if it is not empty
- Then from queue **F3** until it is empty.

If only **one** player remains in queue **F1**, a player from queue **F** will be selected.

If only one player remains in queue **F**, a player from queue **F3** will be selected.

If only one player remains in queue **F3** (at the end), he is declared a *loser* and will be placed on the **LP** list.

Stopping the game :

The game stops when all players are in one of the two lists LP or LG; the queues F, F1 and F3 files are empty.

If after **2n** parts of the game (with the new strategy), there are still players in the queues, the players of **F1** are put in the list **LG** and the players of **F and F3** are placed in the list **LP.**

Questions

I- Algorithmic conception

Describe all the data types used in this game.
Describe the necessary functions and procedures to implement this game (you should consider how to decompose the solution) and evaluate the complexity of each operation.
Write the main algorithm for the overall game.

II – Implementation in C Language

Implement all data structures in C.
Implement the various functions/procedures as well as the overall C program.
For each round, display the details of game progress and the end results.
Display the status of queues F, F1, and F3, and lists LP and LG at the end of each round.
At the end of the game, display the top 3 winners with their respective scores.

Other queries

Which players have not won any part of the game?

With the first strategy,
Show players who have won 1, 2, and 3 parts of the game
Show players who have lost 1, 2, and 3 parts of game

With the second strategy:
Show players who have won 1 or 2 parts of the game
Show players who have lost 1 or 2 parts of the game
PS: Timestamp the games (start time, end time)

PS : For testing, write an action that automatically builds the initial queue of players (we should not enter the data via the keyboard).

Indications

- The program's main interface must display the part game number, the players' numbers and names, the game's start and end dates, the game's progress, and the game's result.
- At the end of each part of game, display the states of queues F, F1, and F2, and the lists LP and LG.
- Show the change in strategy and the game's progress with the second strategy.

Report to write (10 to 15 pages max)

A cover page is required.
A report must be written following the plan below:

1. **Introduction**
2. **Objectives of the work**
3. **Part I** (see above)
4. **Part II** (Provide the well-commented C code in a separate WordPad file)

5. Conclusion (conclude regarding the use of data structures and the complexity of operations)
Can we conclude about the fairness of the game?

Appendix: show some screenshots illustrating the program's execution (choose some relevant screenshots without redundancy).