AY 2020 Assignment 5 [9 Marks]

Date / Time	15 November 2020 – 1 December 2020 23:59
Course	**[M1522.600] Computer Programming**
Instructor	Youngki Lee

- You can refer to the Internet or other materials to solve the assignment, but you *__SHOULD NOT__* discuss the question with anyone else and need to code **ALONE**.
- We will use the automated copy detector to check for potential plagiarism in the codes between the students. The copy checker is made to be very reliable so that it is highly likely to mark a pair of code as a copy even though two students quickly discuss the idea without looking at each other's code. Of course, we will evaluate the similarity of a pair compared to the overall similarity for the entire class.
- We will also inspect the codes manually. In case we doubt that the code may be written by someone else (outside of the class), we reserve the right to request an explanation about the code. We will ask detailed questions that can only be answered when the codes were written by yourself.
- If one of the above cases happen, you will get 0 marks for the assignment and may get a further penalty. Please understand that we apply these methods for the fairness of the assignment.

- Download and unzip “HW5.zip” file from the ETL. “HW5.zip” file contains skeleton codes for Question 1 and Question 2 (in the “problem1”, “problem2” directory) .
- Do not modify the overall directory structure after unzipping the file, and fill in the code in appropriate files. It is okay to add new directories or files if needed.
- When you submit, compress the “HW5” directory which contains “problem1” and “problem2” directories in a single zip file named “20XX-XXXXX.zip” and upload it to ETL as you submit the solution for the lab tests. Contact the TA if you are not sure how to submit. Double-check if your final zip file is properly submitted. You will get 0 marks for the wrong submission format and a 5% deduction for the wrong submission file name.
- C / C++ Standard Library (including Standard Template Library) is allowed.
- Do not use any external libraries.

Contents

Submission Guidelines

1. You should submit your code on eTL.
2. After you extract the zip file, you must have a “HW5” directory. The submission directory structure should be as shown in the table below.
3. You can create additional directories or files in each problem directory. Make sure to update CMakeLists.txt to reflect your code structure changes.
4. You can add additional methods or classes, but do not remove or change signatures of existing methods.
5. Compress the “HW5” directory and name the file “20XX-XXXXX.zip” (your student ID).
6. The Autolab server for Assignment 5 will be available in a week.

Submission Directory Structure (Directories or Files can be added)

- Inside the “HW5” directory, there should be “problem1” and “problem2” directory.

Directory Structure of Question 1	Directory Structure of Question 2
problem1/ └── test/... └── CMakeLists.txt └── main.cpp └── CSI.cpp └── CSI.h └── TestHelper.cpp └── TestHelper.h	problem2/ └── test/... └── CMakeLists.txt └── main.cpp

Question 1: Wi-Fi Signal Decoder [4 Marks]

Objective: Develop a Wi-Fi signal decoder to convert “Wi-Fi Channel State Information (CSI)” into human-interpretable amplitude values. Do not worry about the terms like CSI or amplitude; we will explain necessary details later.

Description: You are a research intern at professor Youngki’s lab, currently working on analyzing Wi-Fi signal patterns to recognize a user’s activity (such as walking, standing, sitting). You are asked to implement a small sub-component, i.e., a Wi-Fi signal decoder, which converts low-level “Wi-Fi Channel State Information (CSI)” into human-interpretable amplitude values.

Note:

- For this question, you only need to modify CSI.cpp. The test codes are in main.cpp.
- We provide the basic test cases for the Autolab, and we will use a richer set of test cases for evaluation. We strongly encourage you to add more diverse test cases to make sure your application works as expected.

Background on CSI. CSI describes how Wi-Fi signals travel through space. We can determine which activities are performed by a person inside a room by analyzing the CSI values. A series of CSI values are recorded when two Wi-Fi devices in a room communicate with each other. In particular, when a data packet (a unit of data) is transferred, information is sent through multiple data transmission *channels* where a channel consists of multiple *subcarriers*. Here, a CSI value (represented as a complex number) is recorded per subcarrier to indicate the state of the corresponding subcarrier. Accordingly, for a single data packet, (*channels X subcarriers)* CSI values are recorded. In order to conduct useful analysis, we would like to convert these raw CSI values into “amplitude” values (meaning received strength of Wi-Fi signals). More details will come later.

Question 1-1: Printing Complex Values [0.5 Marks]

Objectives: Implement the following function in CSI.cpp to print the object of the custom struct `Complex` in a designated format.

- `std::ostream& operator<<(std::ostream &os, const Complex &c)`
    - This method overloads the insertion operator (<<) of this struct so that we can print out a Complex variable `z` using `std::cout << z`. The output of `z` should be: `a+bi`.
    - The purpose of this function is similar to overriding the `toString` method in Java!

Description: We will fill up a convenience struct called `Complex`, which represents a single raw CSI value. Remember, raw CSI values are expressed in complex numbers. As you are aware, a complex number `z = a + bi` has two parts: real and imaginary. The `Complex` struct stores the

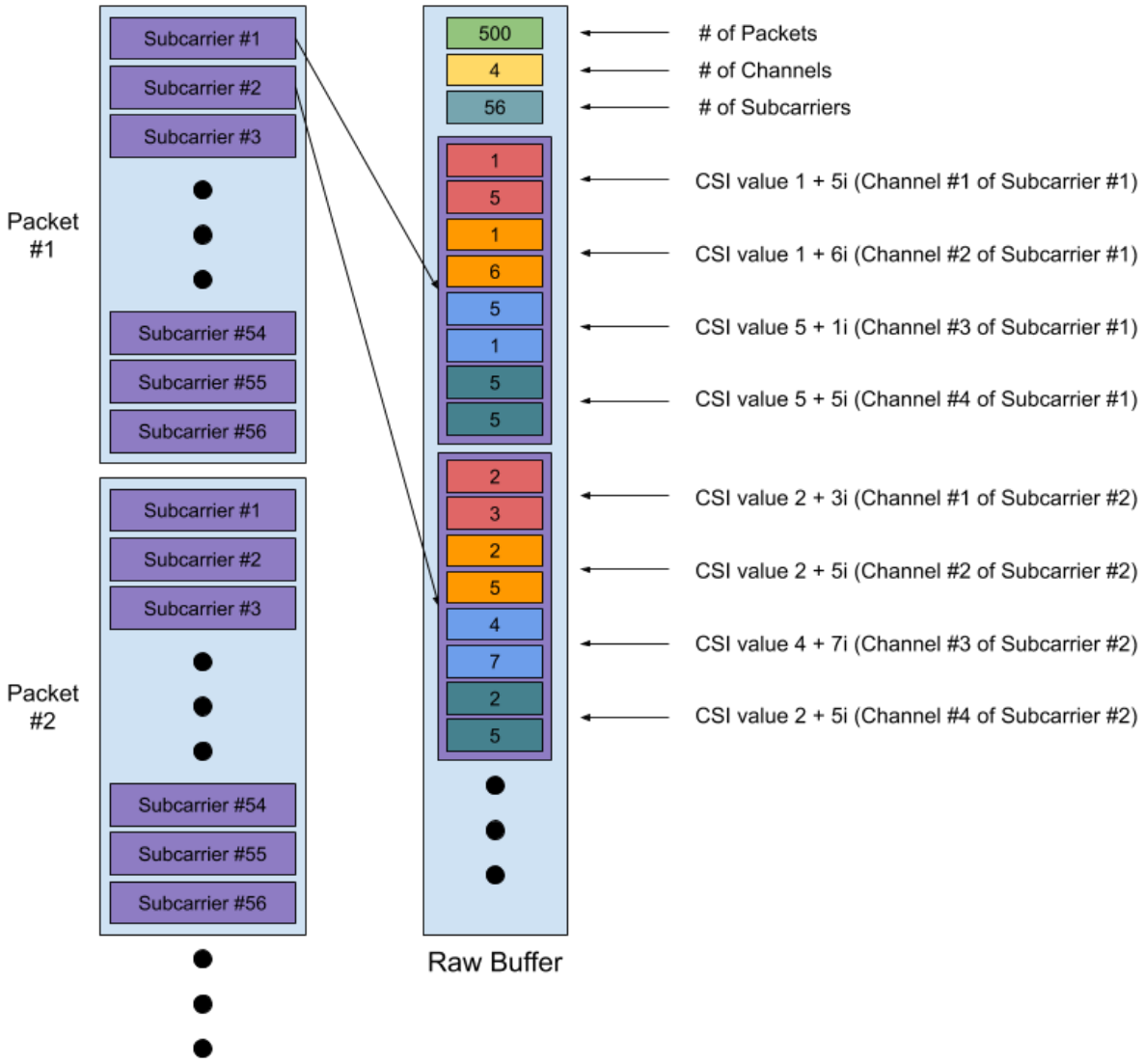
real and imaginary parts (a, b) as the public member variables real and imag. All the other parts of the Complex struct are given.

Question 1-2: Reading and Organizing CSI Values [1 Marks]

Objectives: Implement the following function in CSI.cpp to organize the input stream of CSI values in a file into a 2D array.

- void read_csi(const char* filename, CSI* csi)
    - It reads the input file (with the filename) containing raw CSI values and updates the csi object based on a description below.
    - Assume that the provided csi pointer is valid (i.e., the csi object has already been created before calling this function).

Description: The format of the input file is shown in the diagram below.

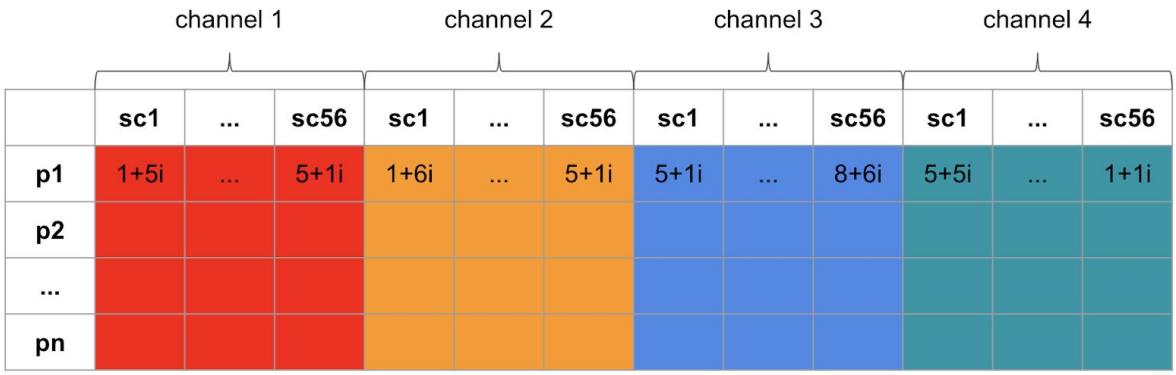


In the example above, the first three lines of the input file indicate the number of packets delivered (i.e., 500) along with the number of channels (i.e., 4) and subcarriers (i.e., 56). Note that the number of data packets, number of channels, and number of subcarriers is not a fixed number, and you should read the information from the input file.

From the line 4 below, each line holds a raw CSI value. In this example, the file has 224 raw CSI values per packet since a packet is delivered through 224 subcarriers (56 * 4); in total, the file has 112,000 (500 * 56 * 4) raw CSI values. For each data packet, you can see that CSI values are grouped by subcarriers (not by channels).

For further analysis of CSI values, we will organize them into a 2D array such that each row represents a packet (see the figure below). In each row, the CSI values are grouped by

channels starting from channel 1 and ending with channel 4. For each channel, the CSI values for 56 subcarriers are listed in order.



	sc1	...	sc56	sc1	...	sc56	sc1	...	sc56	sc1	...	sc56
p1	1+5i	...	5+1i	1+6i	...	5+1i	5+1i	...	8+6i	5+5i	...	1+1i
p2												
...												
pn												

The `csi` object (passed as the parameter) has a member attribute, `data` to store a 2D array of `Complex` objects.

`csi->data[num_packets][num_channels * num_subcarriers]`.

You need to store the reorganized CSI data into this array. Also, update the properties such as `num_channels`, `num_subcarriers`, and `num_packets`. You must update these member variables; otherwise, the member functions of the CSI struct (such as `packet_length` and `print`) will not work as expected.

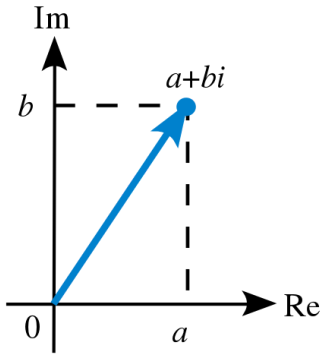
Hint: Use `std::stoi(std::string s)` to convert an integer string into an integer.

Question 1-3: Decoding CSI Values [1 Marks]

Objectives: Implement the following function in CSI.cpp to decode raw CSI values, i.e., converting the raw complex CSI values into amplitude values.

- `float** decode_csi(CSI* csi)`
    - It returns amplitude values that are converted from the provided csi object.
    - You should allocate the proper amount of memory space to store amplitude values.
    - Assume that the provided `csi` pointer is valid.

Description: The amplitude, amp, for a complex number, a+bi, can be computed as $amp = \sqrt{a^2 + b^2}$ using the Pythagorean Theorem.



The below code snippet shows how the amplitude can be calculated using the `sqrt()` function in `<cmath>`.

```
int a = 1, b = 2;
cout << sqrt( lcpp_x: a*a + b*b) << endl;
```

The resulting amplitude array should be in the same format as the data array in the input `csi` object. Specifically, the output array should have the same number of rows and columns as the `csi->data` array. Also, each element in the output array should contain the amplitude (in the `float` type) of the corresponding `Complex` object of the `csi->data` array.

Question 1-4: Extracting Useful Information [1 Marks]

Objectives: Implement the following function in CSI.cpp to calculate the standard deviations of CSI values per packet.

- `float* get_std(float** decoded_csi, int num_packets, int packet_length)`
    - It calculates a standard deviation of the decoded amplitudes per packet and returns standard deviations for all packets in the 1D float array format.
    - You should allocate the proper amount of memory space (for `num_packets` float values) to store amplitude values.
    - Assume that the provided `decoded_csi` pointer is valid.

Description: Standard deviation is a useful feature to summarize decoded CSI data. When the Wi-Fi signal is stable, the standard deviation of signal amplitudes across all the channels and subcarriers is stable. When there is a disturbance in the environment (e.g., a person walking inside the room), the standard deviation fluctuates significantly.

Calculate the standard deviation of the decoded amplitudes across all channels and subcarriers in the same packet inside the `get_std()` function. Do this for all packets and return the output in the form of the following 1D array (as shown below). Consider using `standard_deviation()` function that is defined in CSI.h.

	p1	**p2**	**p3**	**...**	**pn**
std	10.38	3.535	5.268	...	0.254

Question 1-5: Saving Standard Deviation Values [0.5 Marks]

Objectives: Implement the following function in CSI.cpp to save standard deviation values in a file.

- `void save_std(float* std_arr, int num_packets, const char* filename)`
    - It prints the provided standard deviation values to a file.
    - Assume that the provided `std_arr` pointer is valid (i.e., it points to the array with a length of `num_packets`.

Description: Finally, we will save the extracted standard deviation values to a file. Write each standard deviation float value into the output file where the values are separated by a space.

Note that there should be 1 trailing space at the end of the file without any line break.

You can stick with a default precision (6) of floating-point of `printf` or `std::cout`.

Question 2: The Prime Tournament [5 Marks]

Objective: A tournament where players compete with each other in a game involving prime numbers.

Description: There is a game called “Prime Battle”, where two players hold number cards and compete with each other. It was named so because prime numbers are used to determine who is a winner for each match.

In this problem, we will first implement “Prime Battle” by completing logics involving prime numbers. Next, we will define several types of players regarding their risk-taking strategies. Finally, we will make a tournament consisting of many players and determine who wins the first prize.

Notes

1. We will only use standard input and output throughout this problem. main.cpp is also given that inputs from the user and outputs in a valid format. The remaining part from logic to design is up to you. Be sure not to change the behavior of main.cpp, and do not leave debugging messages printed so that your program is correctly graded.
2. The program you should implement will be quite complex. We strongly recommend you to modularize your program. Especially, please write your program in different files and leave main.cpp as a bridge interface that only deals with input and output.
3. Please make sure you understand the concept of prime numbers and integer factorization. We provide two important functions in main.cpp to solve this problem: one for the primality tests and another for integer factorization.
4. We provide the example test cases for Question 2-1 and Question 2-2 under the “test” directory. For instance, “1.in” and “1.out” under the “test/2-1” directory describes the first test case of Question 2-1. You need to create your own test cases from Question 2-3 to Question 2-6.
5. Note: we provide the basic test cases for the Autolab, and we will use a richer set of test cases for evaluation. We strongly encourage you to add more diverse test cases to make sure your application works as expected.
6. We will use the following STL classes `std::pair`, `std::vector`, and `std::multiset` in main.cpp. You can refer to the links below to grasp; these libraries provide necessary data structures and relevant logics as in Java Collections framework.
   https://en.cppreference.com/w/cpp/utility/pair
   https://en.cppreference.com/w/cpp/container/vector
   https://en.cppreference.com/w/cpp/container/multiset

Question 2-1: Battle of Two Numbers [0.5 Marks]

Objective: Implement `std::pair<int,int> number_fight(int a, int b)` in main.cpp

Description: Using integer factorization, we can make two positive integers fight with each other. The rule is as follows:

- Given two integers A and B, factorize them (denoted as FA, FB)
  - For convenience, let's define the factorization of 1 is { 1 }
- Make the list of **distinct** prime numbers that are in both FA and FB (denoted as FG)
- Calculate the product of numbers in FG (denoted as G)
  - If FG is empty, then set G as 1
- Divide A and B by G, respectively.

In other words, we will factorize two numbers and remove all intersecting prime factors. But if they share the same prime factor more than once, we will just take one, not all of them.

For example, let's consider two numbers 24 and 36

- A = 24, B = 36
- FA = {2, 2, 2, 3}, FB = {2, 2, 3, 3}
- FG = {2, 3}
- G = 6
- A ← 24 / 6 = 4, B ← 36 / 6 = 6

So, after one fight, 24 and 36 will become 4 and 6, respectively. Implement this algorithm in `std::pair<int,int> number_fight(int a, int b)`.

- Input
  - `a`, `b`: positive integers less than 10^6
- Output
  - `std::pair<int,int>` that contains the changed value of `a` and `b` after one fight.
  - The first element of the pair is for `a`, and the second for `b`.

Question 2-2: Fight or Flight? [0.5 Marks]

Objective: Implement `std::pair<int,int> number_vs_number(int a, int b)` in main.cpp

Description: Did you notice that fighting always makes numbers small, or the same at best? This is why we sometimes choose not to fight even when we are very offended. The same holds for the world of numbers. Now numbers itself has two options: to fight or not to fight.

So, when two numbers are matched to each other, there can be a total of 4(=2×2) cases.

- If both choose to fight, it is the same as the previous question.
- If both choose not to fight, nothing changes.
- If only one of them chooses to fight, we will call this *attack*. This is what happens when one attacks another.
    - Let's define A' and B' as follows: if both A and B choose to fight, then the values will change into A' and B'.
    - When A attacks B, we define **the *damage* D** of attack as (B - B')
        - If B has 7 as its prime factor, then both A and B get the half damage.
            - A ← A - floor(D / 2), B ← B - floor(D / 2)
            - If A or B becomes less than 1, set it as 1.
        - If B does not have 7 as its prime factor, then B gets all damage.
            - B ← B - D (= B')

According to the new rule, which option is good depends on the situation. In principle, both can choose not to fight for the sake of both. However, sometimes they may have to attack for self-defense. Maybe it reminds you of the Prisoner's Dilemma?

Now, a number A will choose to fight or not based on the following algorithm:

1. Assuming the opponent B will fight, the number compares two outcomes of choosing to fight or not.
    - Let's denote the outcome as A1(fight) and A2(not fight), respectively.
    - If A1 >= A2, it makes a *conditional decision* to fight with B. Otherwise, it makes a conditional decision not to fight with B.
        - The real decision is not made yet. Please read the following procedure 2 and 3.
2. Assuming the opponent B will not fight, it does the same to make another conditional decision.
3. If two conditional decisions in 1 and 2 agree, it will adopt that decision. Otherwise, it will choose to fight if and only if, before fighting, the opponent(=B) is **strictly bigger** than itself(=A).

The same decision-making procedure holds for B.

Consider the following example.

(A, B) = (14, 12)	12(=B) chooses to fight	12(=B) chooses not to fight
14(=A) chooses to fight	(A, B) ← (7, 6)	(A, B) ← (14, 6)
14(=A) chooses not to fight	(A, B) ← (11, 9)	(A, B) ← (14, 12)

In the perspective of A,

1) Assuming B will fight, A should choose not to fight to get the better result (=11).
2) Assuming B will not fight, A should choose to fight as two outcomes are the same (=14).
3) As the two decisions do not agree with each other, A will choose not to fight as B is not bigger than itself.

Similarly, we can conclude that B will choose to fight. This results in (A, B) becoming (11, 9) after one battle.

Implement the fight of two numbers in `std::pair<int,int> number_vs_number(int a, int b)`

- Input
  - `a`, `b`: positive integers less than 10^6
- Output
  - `std::pair<int, int>` that contains the changed value of `a` and `b`. Unlike Question 2-1, they will choose whether to fight or not according to the algorithm described above.
  - The first element of the pair is for `a`, and the second for `b`.

Question 2-3: Player Types [1 Marks]

Objective: Implement `std::pair<std::multiset<int>, std::multiset<int>> player_battle(std::string type_a, std::multiset<int> a, std::string type_b, std::multiset<int> b)` in `main.cpp`

Description: So far, we have discussed the battle between two numbers. Now we're going to think about a battle between two players. Each player has several numbers in their hands, and it will select one of the numbers and send it to the fight. After one battle, the two cards are sent back to their owners with their values (possibly) changed. The rule of the battle between two numbers is the same as Question 2-2.

Let's say a player has N numbers in its hands. It also knows all the M numbers of its opponent. Which number should the player select to fight?

As a player knows the consequences of all N x M cases, it can adopt several strategies to get a better result. Here, ***the better (or the best) result (outcome)*** is defined by the sum of all numbers in its hands. The bigger the sum, the better. Especially, it has nothing to do with how small the opponent's numbers become.

In the perspective of risk-managing strategy, there are 3 types of players:

1. **Maximize-Gain**: A player of this type always assumes and seeks the best case. It selects a number that can produce the best result if the opponent makes a choice the player wants.
2. **Minimize-Loss**: A player of this type always assumes the worst case among all possible outcomes of its choice and wants to be as safe as possible. It selects a number that can produce the best result under the assumption of the worst-case outcome.
3. **Minimize-Regret**: It takes into account both (1) the worst-case outcome of **its choice** and (2) one best-case outcome among **all other choices**. The regret of choice is defined as (2) - (1), and it selects the number with the minimum regret.

   For all three types of players, they will select the smallest number if there is a tie.

Consider the following example, where A and B have 3 numbers each.

(A, B)	B = 8	B = 14	B = 18
A = 2	(1, 4) → Δ = (-1, -4)	(1, 11) → Δ = (-1, -3)	(1, 9) → Δ = (-1, -9)
A = 12	(6, 4) → Δ = (-6, -4)	(9, 11) → Δ = (-3, -3)	(2, 3) → Δ =(-10, -15)
A = 14	(11, 5) → Δ = (-3, -3)	(14, 14) → Δ = (0, 0)	(14, 9) → Δ = (0, -9)

1) If A seeks “Maximize-Gain”, then it will select 14 because it can achieve the best result(Δ = 0) if B selects 14 or 18.
2) If A seeks “Minimize-Loss”, then it will select 2 because, in this way, it can guarantee the loss to be at most 1 regardless of B’s choice.
3) If A seeks “Minimize-Regret”, then it will select 2 because the regret in this case(= 1) is the smallest. (The regret of each choice is 1, 10, and 2, respectively)

Given two players, each having a strategy among the above three, determine the choices they will make and return the state after one battle. Implement this in `std::pair<std::multiset<int>, std::multiset<int>> player_battle(std::string type_a, std::multiset<int> a, std::string type_b, std::multiset<int> b)`

- Input
  - `type_a`, `type_b`: `std::string`, one of “Maximize-Gain”, “Minimize-Loss”, “Minimize-Regret”. Each specifies the strategy type of the corresponding player.
  - `a`, `b`: `std::multiset<int>` that shows the numbers each player has. The sizes of sets are between 1 and 10 (inclusive), respectively, and integers inside are positive and less than 10^6.
- Output
  - `std::pair<std::multiset<int>, std::multiset<int>>` that shows the numbers of each player after one battle.
  - The first element of the pair is for player a, the second one for b.

Question 2-4: Player vs. Player [1 Marks]

Objective: Implement `std::pair<std::multiset<int>, std::multiset<int>> player_vs_player(std::string type_a, std::multiset<int> a, std::string type_b, std::multiset<int> b)` in main.cpp

Description: One battle is not enough for players. They want to fight until there is nothing left to do. From now on, we will define a battle between two players as consecutive matches done until it converges.

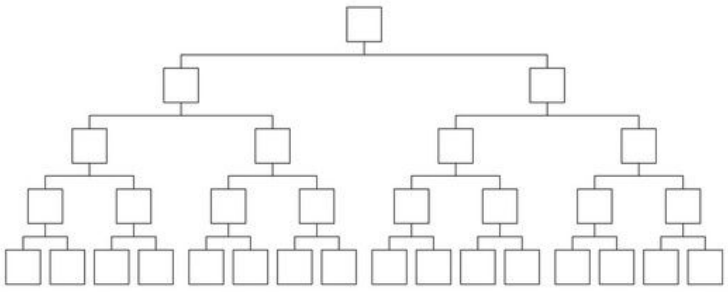
- In each match, both players select the card and have one battle by the same rule as Question 2-3.
- They repeat matches until the state (numbers of both players) does not change anymore.

Implement this in `std::pair<std::multiset<int>, std::multiset<int>> player_vs_player(std::string type_a, std::multiset<int> a, std::string type_b, std::multiset<int> b)`. Input and output format are the same as Question 2-3.

Question 2-5: A Tournament [1 Marks]

Objective: Implement `int tournament(std::vector<std::pair<std::string, std::multiset<int>>> players)` in main.cpp

Description: Why only two players? In this problem, we will make a tournament that up to 16 players can participate in. Here, the definition of the battle between two players is the same as Question 2-4.



- The number of players is between 1 and 16, inclusive. Each player has an integer ID.
- There are several rounds. Players are eliminated in every round, and it continues until there is only one player left in the tournament.
- In each round, players stand side by side in a line. In other words, they are arranged in a 1D list. This should be in the ascending order of players' IDs.
- Then we make a typical tournament:
    - The 1st and 2nd players compete with each other, and the winner goes to the next round. The 3rd with 4th, and so on.

    - If the number of players is odd, the last player will proceed to the next round without having a battle.
    - A match result of a player is the sum of all the numbers it has after the battle. A player with a bigger match result is considered a winner.
    - If there is a tie, the one with a smaller index is considered a winner. (e.g. If 1st and 2nd ties, 1st proceeds to the next round)
- In every battle of two players, they start with their initial state (=numbers each player had at the start of the tournament). In other words, battles the players had before do not affect the starting condition.

For example, if there are 15 players in a tournament, it should be as follows.

1) The first round (15 players)
    a) Battles: (Player 0 with Player 1), …, (Player 12 with Player 13)
    b) Player 14 proceeds to the next round
2) The second round (8 players)
3) ...

Implement this tournament algorithm in `int tournament(std::vector<std::pair<std::string, std::multiset<int>>> players)`

- Input
    - `std::vector<std::pair<std::string, std::multiset<int>>>` that contains information of players in a tournament. The size of the vector is between 1 and 16 (inclusive).
    - One `std::pair<std::string, std::multiset<int>>` describes one player. The first element of the pair is the strategy type of the player. The second one contains numbers the player has. The size of the set is between 1 and 10 (inclusive).
    - The player's ID is defined as the 0-based index in a vector. (e.g., the first player has an ID of 0)
- Output
    - The ID of the player

Question 2-6: Steady Winner [1 Marks]

Objective: Implement `int steady_winner(std::vector<std::pair<std::string, std::multiset<int>>> players)` in main.cpp

Description: Now, we will repeat tournaments to check who wins the most times. Let's say we have n players in a tournament. Then we will make a total of n tournaments as follows.

- A tournament of [Player 0, Player 1, Player 2, …, Player (n-1)]
- A tournament of [Player 1, Player 2, …, Player (n-1), Player 0]
- A tournament of [Player 2, …, Player (n-1), Player 0, Player 1]
- …
- A tournament of [Player (n-1), Player 0, Player 1, Player 2, …, Player (n-2)]

That is, we will make all tournaments from all *rotated* sequences of players. Return the player who won (the 1st prize) the most. If there is a tie, return the player with a smaller ID.

Note: Player IDs of this function have nothing to do with those of each tournament. Each tournament should be done by considering the ID of each player as the index of the player in the given list. It is this function that should track players' identities in each tournament and count who won the most.

Implement this in `int steady_winner(std::vector<std::pair<std::string, std::multiset<int>>> players)`. Input and output format are the same as Question 2-5.