CS 201, Fall 2025

Homework Assignment 1

Due: 23:59, October 30, 2025

1 Introduction

In this assignment, you will design and implement a card game system called **MagicCard**. In this game, players compete using numbered and colored cards, where each move changes the state of the game. The system must keep track of multiple players, their individual hands of cards, as well as the draw pile and discard pile that determines the flow of the game. Players will draw cards, play them according to the rules, and try to beat each other. Your implementation should support creating and deleting players and piles dynamically. Therefore, you MUST use dynamically allocated arrays for storing players and piles as they grow and change over time. In this assignment, you are responsible for implementing the listed functionalities. The details of those functionalities are given below:

1.1 Players

Players are identified by a unique integer ID and a username. If unique IDs are not respected, the system must give a warning. The required functions are as follows:

- `addPlayer`: Adds a new player to the system with an ID and a username. The system should check whether or not the specified player already exists, and if the player exists, it should not allow the operation and display a warning message. Example log messages:
  - Added player Deniz with ID 5.
  - Cannot add player. ID 5 already exists.
- `removePlayer`: Removes a player from the system. If the player does not currently exist in the system, the system should not allow the operation and display a warning message. Removing a player also removes all cards held by that player from the game. Example log messages:
  - Removed player 2.
  - Cannot remove player. There is no player with ID 2.
- `printPlayers`: Prints the IDs and names of all players in the game. If there is no player in the game, gives a warning message. The entries should be shown in ASCENDING ORDER according to IDs. Example log messages:

– Cannot print players. There is no player in the game.

– Players :
  Player 25 : Emre Yilmaz
  Player 56 : Lucas Moreau
  Player 66 : Jane Doe
  Player 76 : Isabella Rossi
  Player 88 : Daniel Johnson

1.2 Cards and Piles

In this game, there are four different colors:

- Red (R)
- Yellow (Y)
- Green (G)
- Blue (B)

Each color contains cards numbered from 1 to 10. The notation used for representing a card is a concatenation of its color code and its number. For example:

- `R1` means "Red 1"
- `Y7` means "Yellow 7"
- `G10` means "Green 10"
- `B3` means "Blue 3"

The system manages two different piles of cards:

1. **Draw Pile**: This is the list of cards from which players draw new cards. Initially, it contains all cards that are not yet dealt to any player. Whenever a player needs a new card, the card is taken from the draw pile. The order of this pile is significant, since cards are drawn from specific positions (card in the n'th index). Initially, the pile is empty, and the system places the full deck into it at the start of the game. Note that the draw pile does not necessarily contain all 40 cards.

2. **Discard Pile**: This pile contains all cards that have been played by the players. The **top card** of the discard pile determines which cards can be played next: a newly played card must match either the color or the number of the top card. If it does not, the player must draw a card from the draw pile.

In summary, the draw pile is the source of new cards for players, while the discard pile is the history of played cards, with its top card enforcing the rules for valid moves. Note that in both piles, the first card is stored at index 0 of the array, and the last card is stored at the final index (number of cards - 1). The required functions are as follows:

- `setDrawPile`: Initializes the draw pile with a given array of cards. Assume that this function is called only once at the start of the game. Example log messages:
  - Draw pile has been initialized.
- `drawCardFromDeck`: It takes a user ID and an integer n. The specified player draws the card at the n'th index of the draw pile. Drawing a card also removes it from the draw pile. If n is not a valid index, a warning message should be displayed. Besides, the cards should be rearranged after one is drawn. For example, if a card is drawn from a pile of size 5, the pile should then become an array of size 4. Example log messages:
  - Pinar drew card R7 from the draw pile.
  - Cannot draw card. The input index is not valid.
- `switchHands`: It gets two player IDs as input and swaps the cards between them. The system should check whether these two players exist, and then print a proper message if one or both players are missing. Example log messages:
  - Switched hands between Basak and Ezgi.
  - Cannot switch hands. One or both players do not exist.
- `listDrawPile` : It prints the current draw pile starting from the first card (index 0). Example log messages:
  - Draw pile is empty.
  - Draw pile: [R1, G3, Y7, B10]
- `listDiscardPile` : It prints the current discard pile starting from the first card (index 0). Example log messages:
  - Discard pile is empty.
  - Discard pile: [G4, B5, G1, Y5]
- `listCardsOfPlayer` : It takes player ID and prints all cards currently held by the specified player. If the given ID does not exist, the system must give a warning. Example log messages:
  - Cards of Alper: [R2, G7, B1]
  - Cannot list cards. Player with ID 26 does not exist.

1.3 The Game

- `play`: The player plays the specified card. The player can play any card in their hand. If the played card does not match either the color or the number of the last card on the discard pile (i.e., the card at the last index of the discard pile), the player must draw a new card from the top of the draw pile (i.e., the card at the first index of the draw pile). The new card is put to the end of the list of cards that player has. After one card is played or drawn, the card pile for the player should be rearranged as well. At the beginning of the game, any card may be played because the discard pile is empty.

  The game ends if the player has no cards left in their hand (in which case that player wins), or if the draw pile becomes empty at the end of a turn (in which case nobody wins).

  Example log messages:

  - There is no player with ID 23.
  - The player does not have R5 card.
  - Player 23 played R5.
  - Player 23 played Y2 then drew B3.
  - The game has ended. Player 23 won.
  - The game has ended. No cards remain in the draw pile.

Below is the required `public` part of the `MagicCard` class that you must write in this assignment. The name of the class must be `MagicCard`, and must include these public member functions.

The interface for the class must be written in the file called `MagicCard.h` and its implementation must be written in the file called `MagicCard.cpp`. You can define additional public and private member functions and data members in this class. You can also define additional classes in your solution and implement them in separate files.

```
class MagicCard {
public:
    MagicCard();
    ~MagicCard();

    void addPlayer( const int playerID, const string name );
    void removePlayer( const int playerID );
    void printPlayers() const;

    void setDrawPile( const string drawPile[], const int size );
    void drawCardFromDeck( const int playerID, const int n );
    void switchHands( const int player1ID, const int player2ID );
    void listDrawPile() const;
    void listDiscardPile() const;
```

```
    void listCardsOfPlayer( const int playerID ) const;

    void play( const int playerID, const string card );
};
```

Here is an example test program that uses this class and the corresponding output. We will use a similar program to test your solution so make sure that the name of the class is `MagicCard`, its interface is in the file called `MagicCard.h`, and the required functions are defined as shown above. Your implementation MUST use EXACTLY the same format given in the example output to display the messages expected as the result of the defined functions.

Example test code:

```
#include <iostream>
#include "MagicCard.h"
using namespace std;

int main() {

    MagicCard game;

    game.printPlayers();
    cout << endl;

    game.listDiscardPile();
    cout << endl;

    game.listDrawPile();
    cout << endl;

    game.addPlayer(5, "Deniz Akin");
    game.addPlayer(56, "Lucas Moreau");
    game.addPlayer(25, "Emre Yilmaz");
    game.addPlayer(5, "Derya Yildirim");
    cout << endl;

    game.printPlayers();
    cout << endl;

    game.removePlayer(56);
    game.removePlayer(99);
    cout << endl;

    game.printPlayers();
    cout << endl;

```

```
34      string cards[] = {"R1", "G3", "Y7", "B10", "B8", "G9"};
35      game.setDrawPile(cards, 6);
36
37      game.listDrawPile();
38      cout << endl;
39
40      game.drawCardFromDeck(5, 3);
41      game.drawCardFromDeck(25, 0);
42      cout << endl;
43
44      game.listCardsOfPlayer(5);
45      game.listCardsOfPlayer(25);
46      cout << endl;
47
48      game.drawCardFromDeck(5, 1);
49      game.drawCardFromDeck(25, 0);
50      cout << endl;
51
52      game.listCardsOfPlayer(5);
53      game.listCardsOfPlayer(25);
54      cout << endl;
55
56      game.switchHands(5, 25);
57      game.switchHands(5, 99);
58      cout << endl;
59
60      game.listCardsOfPlayer(5);
61      game.listCardsOfPlayer(25);
62      cout << endl;
63
64      game.drawCardFromDeck(25, 10);
65      cout << endl;
66
67      game.listCardsOfPlayer(49);
68      cout << endl;
69
70      game.listDiscardPile();
71      cout << endl;
72
73      game.play(5, "R1");
74      cout << endl;
75
76      game.listDiscardPile();
77      cout << endl;
78
79      game.play(123, "G3");
```

```
    game.play(25, "Y9");
    cout << endl;

    game.listCardsOfPlayer(5);
    cout << endl;

    game.play(25, "B10");
    cout << endl;

    game.listCardsOfPlayer(25);
    cout << endl;

    game.play(25, "B8");
    cout << endl;

    game.listCardsOfPlayer(25);
    cout << endl;

    game.play(5, "R1");
    cout << endl;

    game.listDiscardPile();
    cout << endl;

    game.play(5, "G3");
    cout << endl;

    return 0;
}
```

Output of the example test code:

```
Cannot print players. There is no player in the game.

Discard pile is empty.

Draw pile is empty.

Added player Deniz Akin with ID 5.
Added player Lucas Moreau with ID 56.
Added player Emre Yilmaz with ID 25.
Cannot add player. ID 5 already exists.

Players :
Player 5 : Deniz Akin
Player 25 : Emre Yilmaz
```

```
Player 56 : Lucas Moreau

Removed player 56.
Cannot remove player. There is no player with ID 99.

Players :
Player 5 : Deniz Akin
Player 25 : Emre Yilmaz

Draw pile has been initialized.
Draw pile: [R1, G3, Y7, B10, B8, G9]

Deniz Akin drew card B10 from the draw pile.
Emre Yilmaz drew card R1 from the draw pile.

Cards of Deniz Akin: [B10]
Cards of Emre Yilmaz: [R1]

Deniz Akin drew card Y7 from the draw pile.
Emre Yilmaz drew card G3 from the draw pile.

Cards of Deniz Akin: [B10, Y7]
Cards of Emre Yilmaz: [R1, G3]

Switched hands between Deniz Akin and Emre Yilmaz.
Cannot switch hands. One or both players do not exist.

Cards of Deniz Akin: [R1, G3]
Cards of Emre Yilmaz: [B10, Y7]

Cannot draw card. The input index is not valid.

Cannot list cards. Player with ID 49 does not exist.

Discard pile is empty.

Player 5 played R1.

Discard pile: [R1]

There is no player with ID 123.
The player does not have Y9 card.

Cards of Deniz Akin: [G3]

Player 25 played B10 then drew B8.
```

```
61
62 Cards of Emre Yilmaz: [Y7, B8]
63
64 Player 25 played B8.
65
66 Cards of Emre Yilmaz: [Y7]
67
68 The player does not have R1 card.
69
70 Discard pile: [R1, B10, B8]
71
72 Player 5 played G3 then drew G9.
73 The game has ended. No cards remain in the draw pile.
```

IMPORTANT NOTES:

Do not start working on your homework before reading these notes!!!

2 Specifications

1. You ARE NOT ALLOWED to modify the given parts of the header file. **You MUST use dynamically allocated arrays with only the necessary amount of memory in your implementation.** That is, if there are 10 players in the system, it should use memory only for these 10 players. In other words, you cannot initially allocate a large array for players and expect it to get filled later. The same argument applies to space used to store piles. **You will get no points if you use fixed-sized arrays, linked lists or any other data structures such as vectors/arrays/etc. from the standard library.** However, if necessary, you may define additional data members and member functions.

2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.

3. Output message for each operation MUST match the format shown in the output of the example code. Your output will be compared verbatim with the expected output during evaluation.

4. Your code MUST NOT have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To detect memory leaks, you may want to use Valgrind which is available at http://valgrind.org.

5. You may assume that the inputs for the functions are in correct format so that you do not need to make any input checks for format.

3 Submission

1. In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files) for your class. Your class name MUST BE `MagicCard` and your file names MUST BE `MagicCard.h` and `MagicCard.cpp` for this class. Note that you may write additional class(es) in your solution.

2. The code (`main` function) given above is just an example. We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using this example code. We recommend you to write your own driver files to make extra tests. However, you MUST NOT submit these test codes (we will use our own test code). In other words, do not submit a file that contains a function called `main`.

3. You should put all of your `.h` and `.cpp` files into a folder and zip the folder (in this zip file, there should not be any file containing a `main` function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. The submissions that do not obey these rules will not be graded. Please do not use Turkish letters in your file and folder names.

4. Make sure that each file that you submit (each and every file in the archive) contains your name, section, and student number at the top as comments.

5. You are free to write your programs in any environment (you may use Linux, Windows, MacOS, etc.). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the dijkstra machine. Your code will be tested by using an automated test suite that includes multiple test cases where each case corresponds to a specific number of points in the overall grade. We will provide you with example test cases by email. Thus, we strongly recommend you to make sure that your program successfully compiles and correctly works on dijkstra.ug.bcc.bilkent.edu.tr before submitting your assignment. If your current code does not fully compile on dijkstra before submission, you can try to comment out the faulty parts so that the remaining code can be compiled and tested during evaluation.

6. This assignment is due by 23:59 on Thursday, October 30, 2025. You should upload your work to Moodle before the deadline. No hardcopy submission is needed. Late submissions will not be accepted (if you can upload to Moodle, then you are fine). There will be no extension to this deadline.

7. We use an automated tool as well as manual inspection to check your submissions against plagiarism. For questions regarding academic integrity and use of external tools (including generative AI tools), please refer to the course home page and the Honor Code for Introductory Programming Courses (CS 101/102/201/202) at `https://docs.google.com/document/d/1v_3ltpV_lClLsROXrMbojyuv4KrFQAm1uoz3SdC-7es/edit?usp=sharing`.

8. This homework will be graded by your TAs **Pınar Gül** (pinar.gul@bilkent.edu.tr), and **Mehmet Alper Yılmaz** (mehmet.yilmaz@bilkent.edu.tr). Thus, you may ask your homework related questions directly to them. There will also be a forum on Moodle for questions.