Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2021

Homework 6 – Card Matching Game

Due: 7/05/2021, Friday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to implement *Card Matching* game to test human memory using *object sharing* and *templated class* concepts of C++. Main program, which includes the game implementation using classes, is given to you and you are expected to write two classes: *board* class and *player* class. We are going to explain these classes in more details in the following sections. Before that, we start with a general description of the game.

The Card Matching Game

The card matching game is a well-known multiplayer game that is played by a set of cards aligned on a board in matrix format. The cards of the board must be in pairs (i.e. a particular value must appear in exactly two cards). The game is usually played by 2 players (say, player1 and player2) and one of the players starts playing (let us say player1 starts). In each turn, the player chooses two cards (say, card1 and card2) one after the other. Player1 chooses card1, opens it and sees the value on the card. Afterwards, again player1 chooses card2 with the aim to find a match with card1 (two cards are said to be matched if they have the same value). If card1 and card2 match, these cards remain opened and the score of player1 is increased by 1; otherwise, the cards are turned over. Then, player2 starts playing and repeats the steps that player1 has done. The game finishes when all cards are opened. The player with the higher score becomes the winner of the game. If the scores of both players are equal, then there is a tie.

Inputs and Input Checks

First, the name of the input .txt file will be entered by the user in the main function. The first line of this file is the number of rows and number of columns of the board. These are read in main function, as you can see in main.cpp. The rest of the file contains values of the cards on the board organized as rows and columns as in the board. Since each card must have a match in the board, the total number of cards is an even number (no need to check this; it is an assumption). You may also assume that the input file is given to you in a correct format (i.e.

first line contains two positive integers and their multiplication is an even number; all elements exist in pairs; the number of rows and columns in the file match the row and column information on the first line; there is no type problem). Therefore, you do not need to make input check for the content of the input file, but you have to use proper files while trying the game (first game is with integer files, second game is with char files).

Two sample input files are given below.

```
3 6
1 4 89 12 -1 4
2 89 -4 1 -4 3
3 12 2 -1 5 5
```

```
4 7
a h j k l c s
c m t u q w s
p I a j u q p
I h l w t m k
```

As you see in the samples above, the type of board elements can be anything (integer, char, double, string or any other type). Your program should handle this without using different functions or classes for these different types. In other words, the classes that you will implement will be **templated**.

Players also provide row and column values of a card as inputs while playing. Here, you have to check these inputs considering the following constraints:

- The user cannot provide row and column values exceeding the indices of the board.
- The user cannot enter row and column values of a card which is currently open.

You can assume that the user enters integers for row and column values.

The row and column indexing starts with zero and upper-left element is at (0,0).

Actually, the necessary member function calls and related error messages for these controls have been incorporated in the `main` and `playGame` functions, which is provided to you in main.cpp. What you have to do is to implement the classes accordingly as detailed below.

Board Class

The board class will be used to create a board where the game will be played on. A board is represented by a matrix which is a private data member of the class. The matrix will be created dynamically with the row and column sizes given in the input file. Moreover, the matrix will be designed such that it will be able to keep any type of value on the cards (such as int, char, string, etc.). Since you also need to keep the open/close status of a card, in addition to the face value, you may consider using a struct as the element type of the matrix; but here notice that such a struct should also be templated.

Now, we will give constructor and some member function explanations of board class. You are allowed to add other member functions depending on your choice of implementation.

Parametric Constructor: Constructor of the board class takes two integer type parameters: number of rows and number of columns of the board. You are going to construct the board matrix here using dynamic memory allocation; however, you will not read a file or take keyboard input for matrix initialization in the constructor.

Destructor: By definition destructor deallocates all of the dynamically allocated memory of the class. It'd be safe to make the pointers NULL after deallocation.

readBoardFromFile: The readBoardFromFile function takes an input file stream as a parameter (hint: use reference parameter) and fills the board matrix by reading the input file. You can also initialize status of the cards of the board object within this function. Here remark that the filename is not an input or given to this function; the file has already been opened in main and the corresponding `ifstream` object is passed as parameter. Assume that file's current position is the beginning of card data (i.e. the beginning of second line).

displayBoard: The displayBoard function does not take any parameter. It only displays the current state of the board. This function should display **`x`** for the cards that are currently closed (you may assume that there is no X as the card value on the board).

closeCard: The closeCard function takes two integer parameters, row and column values of the card, respectively. This function should change the status of the card to *closed*.

getRow: This function returns the number of rows of the board.

getColumn: This function returns the number of columns of the board.

The above functions are explicitly used in game implementation given to you in main.cpp. Other than these member functions, you may need to add and implement some other functions to manage the board and their elements. This is due to the fact that you will implement another class for Player and this Player class will need to access board. Since the use of friend functions and friend classes are **not** allowed in this homework, in the implementation of the Player class, you will need to access board via some accessors and mutators.

We do not enforce, and actually not implemented in our coding, a copy constructor for the board class. Normally, deep copy is needed since there is dynamic memory allocation. However, the provided `main` and `playGame` functions do not necessitate a copy constructor for the board class. If your internal class implementations do not require (explicitly or implicitly) to invoke the copy constructor as well, then you may not implement it. However, to be on the safe side, it'd be a good idea to implement a deep copy constructor for the board class.

Player Class

Player class will be used to create players of the game. There will be two players playing on the same board in a game. Thus, player objects must share a board object using *object sharing* concept of C++ as we have seen in class. We have seen two different methods for object sharing in class; due to our main function implementation, which is provided to you, you must use the reference variable method.

The player class should also keep the score (number of successful matches) of the player.

Now, we will give member function explanations of player class. You are allowed to add other member functions depending on your choice of implementation.

Parametric Constructor: Constructor of the player class takes only one parameter, which is the board object that will be played on. You should also initialize the score of the player to zero.

openCard: The openCard function takes two integer parameters, which are row and column values of the card on the board and opens the corresponding card. The function returns the value of the card (note that the return type is a templated one).

validMove: The validMove function takes two integer parameters which are row and column values of the card on the board. The function checks whether the row and column values are valid or not, and returns the validation result in an appropriate type. Row and column values are said to be invalid if they exceed limits of the board or the corresponding card is currently open.

increaseNumberOfSuccess: The increaseNumberOfSuccess function increments the private data member, which keeps the number of successful matches (i.e. the current score) of the player, by one.

getNumberOfSuccess: This function returns the private data member, which keeps the number of successful matches (i.e. the current score) of the player.

Important Rule about using `friend`

In this homework, you are **not** allowed to use **`friend`** keyword, which means the use of friend functions and friend classes are prohibited. Our aim by this restriction is not to make your life miserable, but to enforce you to proper object oriented design and implementation.

Provided files

Together with this homework package, we provide some extra files to you.

cardgame.exe: This is the executable file of our implementation for the card game. In this homework, we do not provide sample runs due to the complexity of the game. Instead we provide this executable so that you can try and understand the requirements.

main.cpp: This file contains the `main` and `playGame` functions, which contain the game implementation by using related class functions. In this homework, our aim is to reinforce object oriented design capabilities; thus, we did not want you to deal with the class usage, but focus on their design and implementation. Please examine these functions to understand how the classes are used. We will test your codes with this main.cpp with different inputs. You are not allowed to make any modifications in this file other than adding your #include lines; you have to use other files for class definitions and implementations.

Matrix files: We also provide some sample matrix input files. The names of the files explain the card types and the size of the matrix. While trying the exe file to understand the game, please open correctly typed files (first game is for integer, second game is char).

This homework is a bit Windows dependent. In main.cpp a Windows system call "cls" is used to clear the screen. If you do not have any chance other than using Mac, this line will not work. In such a case, you may comment it out to test your code without clearing the screen at each turn; however, put it back before submission. Another Windows dependency is due to the exe file provided; it works only on Windows OS.

Files to be Submitted and the Use of Templates

You will submit total of five (5) files. Two of them are the headers files; one for player class and the other is for board class (define card struct here). Three files are cpp files. One is the main.cpp that will remain intact other than the #includes. One of the other cpp files is for the player class and the other is for the board class.

Since the classes will be templated, you will need to solve the *fundamental dilemma* mentioned in the lecture notes (5-templated.ppt, pages 24-25). You have to apply "Solution 2" mentioned there.

A General Remark about Object Oriented Design

You need to analyze the requirements carefully and make a good object oriented design for the classes that you will develop in this homework. In this context, you have to determine the data members and member functions of each class correctly. You have to apply the object sharing principles correctly and use the templates properly. We will evaluate your object oriented design as well. Moreover, you are not allowed to use friend class or friend functions in your design.

Please see the previous homework specifications for the other important rules and the submission guidelines

Good Luck!
Albert Levi, Vedat Peran