Assignment 4 (20% of Course Total)

Due date: 11:59pm, Apr 14, 2021

The assignment will be graded automatically. Make sure that your code compiles without warnings/errors and produces the required output. Also use the file names and structures indicated as requested. Deviation from that might result in 0 mark.

Your code MUST compile and run in the CSIL machines with the Makefile provided. It is possible that even with warnings your code would compile. But this still indicates there is something wrong with you code and you have to fix them, or marks will be deducted.

Your code MUST be readable and have reasonable documentation (comments) explaining what it does. Use your own judgement, for example, no need to explain i += 2 is increasing i by 2, but explain how variables are used to achieve something.

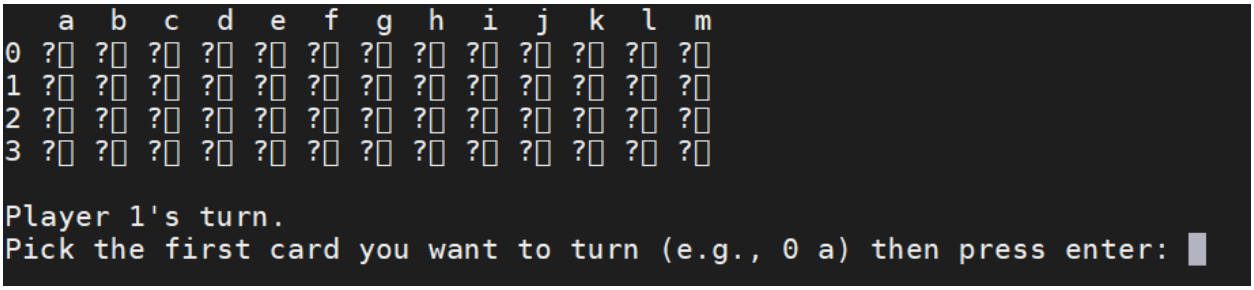
If you use any malloc in your answer, make sure you free the memory when you don't need it anymore.

Description

There is only 1 question in this assignment. However, it contains several parts that perform different functionalities. When combined, they become a **memory card game**. For each part, write your answer as required and include your student information. Unless otherwise specified, do not include any libraries. You can however write your own helper functions. Do not print anything unless the function asks you to do so. None of these files should contain the main function, except in main.c, which you need to complete.

The Memory Card Game

You are going to write a game that you might already have played before.



Here are the rules:

- There are 2 players and they take turns to play.
- During a player's turn, the player picks 2 cards. If the values of the cards are the same (ignoring the suits), the player wins those cards and gains an extra turn (this continues until the values don't match). Otherwise, it is the other player's turn.
- Continue taking turns until all cards are won. Then whoever has the most cards wins. Note that it is possible to have a tie (both players win 26 cards).

Part 1: gameObjects.h & gameObjects.c (submit gameObjects.c only)

These files implement functions that manages the entities of the game.

Here are the functions to be implemented (read the .h file for full details on how each should behave):

void initializeDeck(Deck theDeck, char* brandName);*

void shuffleDeck(Deck theDeck);*

void printDeck(const Deck theDeck, bool faceUp);*

void initializePlayer(Player thePlayer, char* theName);*

void clearPlayer(Player thePlayer);*

In particular, the Deck struct is the same as defined in the previous assignments and the Player struct represents the players in the game.

You'll notice that some of the functions are actually part of Assignment 3. However, you need to pay attention to how the functions are implemented here in this assignment:

- shuffleDeck – to allow us to quickly mark your submission, along with the only srand(0) call in main, this function must use the same Fisher-Yates shuffle algorithm as illustrated by John Leehey in https://stackoverflow.com/questions/6127503/shuffle-array-in-c. To verify that you have implemented it correctly, refer to the sequences of cards after the first few calls of shuffleDeck showed at the end of this document.
- printDeck – this function will not print the brand but to allow the players to indicate which card they want to pick, it instead prints the row (0-3) & column (a-m) indexes. It also takes an extra bool parameter to either print all the cards face up or down. In addition, when a card is won, its value becomes '0', which is used by this function to skip printing the card. Refer to the sample outputs in this document for details. Again, the symbols might not print properly in computers other than a CSIL workstation.

Only include the function definition(s) in your answer and name the source file containing it as **gameObjects.c**. Use the definitions of the structs and enum in various .h files for your answer.

Part 2: card_LList.h & card_LList.c (submit card_LList.c only)

These files implement a linked list structure where each node stores the address of a card from the Deck.

Here are the functions to be implemented (read the .h file for full details on how each should behave):

Card_LList createCard_LList();*

void clearCard_LList(Card_LList theList);*

bool isEmptyCard_LList(Card_LList theList);*

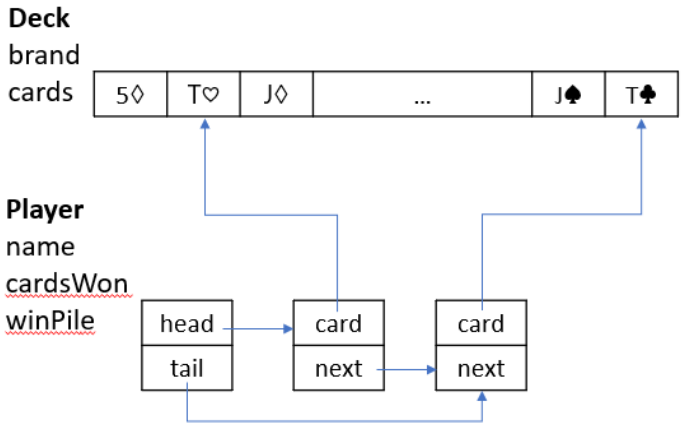
void insertFrontCard_LList(Card_LList theList, Card* theCard);*

void insertEndCard_LList(Card_LList theList, Card* theCard);*

Card removeFrontCard_LList(Card_LList* theList);*

Card removeEndCard_LList(Card_LList* theList);*

The figure below illustrates how the linked list is used to store the cards won by a player:



As the number of card addresses to be stored is unknown prior to the program, malloc is used to create nodes during the game. If malloc fails, the program terminates by calling the function: exit(0).

Only include the function definition(s) in your answer and name the source file containing it as **card_LList.c**. Use the definitions of the structs and enum in various .h files for your answer.

Part 3: gameFunctions.h & gameFunctions.c (submit gameFunctions.c only)

These functions implement the core functionalities to make the game work.

Here are the functions to be implemented (read the .h file for full details on how each should behave):

void addCardToPlayer(Player thePlayer, Card* theCard);*

bool checkPlayerInput(Deck theDeck, Player* thePlayer, char row, char col);*

bool checkForMatch(Deck theDeck, Player* thePlayer, char r1, char c1, char r2, char c2);*

bool checkForWinner(const Deck theDeck);*

Some of these functions will print a message to the player if an error occurs, including:

- Player picks a card with invalid indexes (out of bound)
- Player picks a card that is already won
- Player picks the same card in their round

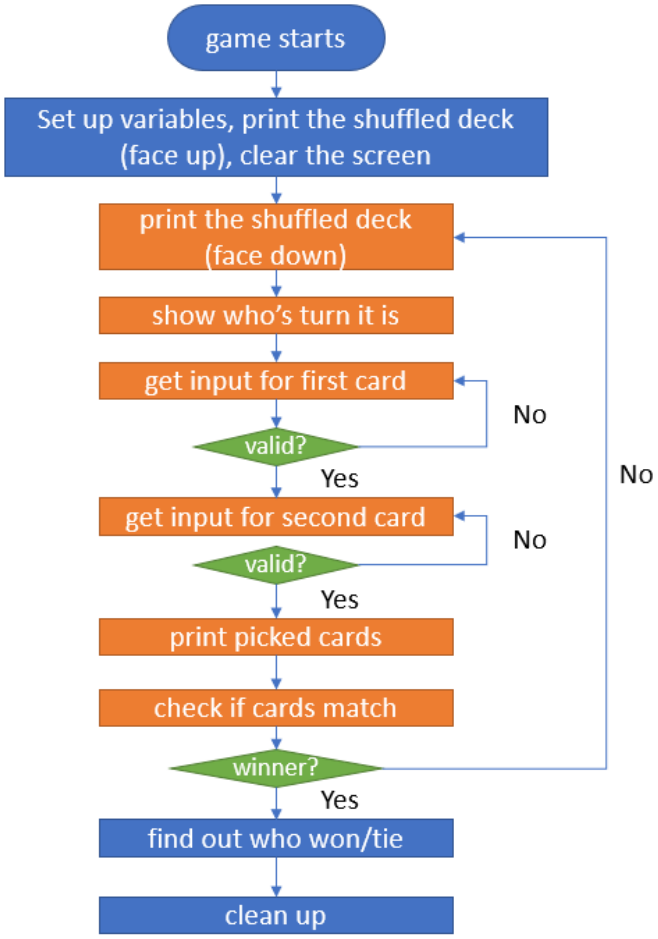
Only include the function definition(s) in your answer and name the source file containing it as **gameFunctions.c**. Use the definitions of the structs and enum in various .h files for your answer.

Part 4: main.c (submit main.c)

This file is the main driver of the game program.

It begins with a definition of a function called “clear”, which allows the game to “cheat” by printing the shuffled deck and then pushing the print out of the display area (still accessible by scrolling up). This may or may not work outside of a CSIL workstation. Do not modify this function.

Next is the main function, which implements the logic of the game. The first part in the function is written so that you can use the variables to complete the function. You can add other variables if you want to. The do-while-loop is where each round should work and at the end of the loop the function *checkForWinner* will determine who has won (if no one wins the loop repeats). This game will not repeat once there is a winner. Follow the steps in the comments to complete it.



The figure above shows how the logic of the game is related to the steps in the main function.

Modify the main function as needed and keep the file name as **main.c**. Use the definitions of the structs and enum in various .h files for your answer.

Coding Style [4 marks]

Your program should be correctly indented, have clear variable names and enough white space and comments to make it easy to read. Named constants should be used where appropriate. Each line of code should not exceed 80 characters. White space should be used in a consistent manner.

To help you to get into the habit of good coding style, we will read your code and marks will be deducted if your code is not styled properly.

Using the Makefile and Other Supplied Files

The Makefile provided in this assignment is used by a command in the CSIL machines called "make" to quickly compile your code. It is especially useful if you have multiple source files. To use it, type the following command in the prompt (make sure you are in the directory with all the files of this assignment):

```
$ make game
```

The example above illustrates how the game is compiled into an executable called "game" when using the Makefile. You can then run the executable by typing "./game" to test your code. If you make changes to your code, use the make command again. In this assignment, "make", "make all" will do the same thing.

There is no particular order of parts in which you need to implement. It is not uncommon to implement a few functions in one part and work on another, then get back to this part to modify or complete the rest.

The header files are there to make the compilation work and inform you about the functions you need to implement. Pay attention to the comments there but do not modify them, and do not submit them.

Submission

Submit **only the files indicated for each part** by compressing them into a zip file (**do not** put them into a folder and zip them) and upload it to Canvas **by 11:59p Apr 14**. Name the zip file in this format: **<firstname_lastname>_<studentID>_Assignment4.zip**.

For example, John_Smith_012345678_Assignment4.zip

Assignment late penalty: 10% per calendar day (each 0 to 24 hour period past due), max 2 days late.

Academic Honesty

It is expected that within this course, the highest standards of academic integrity will be maintained, in keeping with SFU's Policy S10.01, "Code of Academic Integrity and Good Conduct." In this class, collaboration is encouraged for in-class exercises and the team components of the assignments, as well as task preparation for group discussions. However, individual work should be completed by the person who submits it. Any work that is independent work of the submitter should be clearly cited to make its source clear. All referenced work in reports and presentations must be appropriately cited, to include websites, as well as figures and graphs in presentations. If there are any questions whatsoever, feel free to contact the course instructor about any possible grey areas.

Some examples of unacceptable behavior:

- Handing in assignments/exercises that are not 100% your own work (in design, implementation, wording, etc.), without a clear/visible citation of the source.
- Using another student's work as a template or reference for completing your own work.
- Using any unpermitted resources during an exam.
- Looking at, or attempting to look at, another student's answer during an exam.
- Submitting work that has been submitted before, for any course at any institution.

All instances of academic dishonesty will be dealt with severely and according to SFU policy. This means that Student Services will be notified, and they will record the dishonesty in the student's file. Students are strongly encouraged to review SFU's Code of Academic Integrity and Good Conduct (S10.01) available online at: http://www.sfu.ca/policies/gazette/student/s10-01.html.

Sample output after each of the first 3 consecutive shuffles

Refer to the ordering of the cards to verify that you have implemented the shuffle function correctly. We will use different numbers of shuffles to test your code to prevent hard-coding. Ignore the brand name.

```
Brand of Deck: Bicycle
5♢ 9♡ 3♢ 4♢ 9♢ 2♡ 2♠ 3♠ 8♡ 7♣ 5♣ J♣ A♣
8♣ Q♢ T♢ K♣ 4♠ T♡ A♢ 9♠ 3♣ K♠ 8♢ 7♠ T♣
4♣ 6♡ K♢ 9♣ 8♠ 3♡ 6♢ 4♡ 7♢ A♠ K♡ Q♡ 5♡
J♢ J♡ 7♡ 2♢ Q♣ A♡ 5♠ 6♠ Q♠ 6♣ 2♣ J♠ T♠

Brand of Deck: Bicycle
9♢ 5♣ A♠ 6♠ 9♠ 8♡ 2♠ 6♡ 9♡ A♢ J♠ Q♠ 4♢
8♢ 7♢ K♢ Q♣ Q♢ J♡ K♡ 3♣ 4♠ 2♣ K♠ A♡ 6♢
5♠ 2♢ 7♣ 7♠ 4♡ T♡ 8♣ 6♣ J♢ 5♡ 2♡ 3♠ J♣
T♠ 3♢ 8♠ T♢ A♣ 5♢ 3♡ Q♡ K♣ 9♣ 4♣ T♣ 7♡

Brand of Deck: Bicycle
4♣ K♣ A♢ Q♡ J♢ 6♢ 9♠ K♡ T♢ 2♣ 7♣ 3♣ 6♡
K♠ 5♡ 4♡ A♠ 9♣ 4♠ 8♢ 7♢ 5♢ 7♡ T♣ A♣ Q♠
3♡ 2♡ 8♡ 2♠ 9♡ 2♢ J♠ 6♠ T♡ Q♢ J♣ 8♣ T♠
Q♣ 3♢ 6♣ A♡ 5♠ 4♢ J♡ 9♢ 7♠ K♢ 5♣ 3♠ 8♠
```

Sample outputs of the game

Try your best to reproduce the messages and format of the output. Refer to the demo video (available shortly, stay tuned to the announcement in the class).

```
   a  b  c  d  e  f  g  h  i  j  k  l  m
0 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠
1 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠
2 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠
3 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠

Player 1's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter:
```

Figure 1: the game begins with all cards printed face down (if scrolled up you'll see all cards printed face up once), then asks the player to pick the first card (name of the player is Player 1).

```
   a  b  c  d  e  f  g  h  i  j  k  l  m
0 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠
1 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠
2 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠
3 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠 ?🂠

Player 1's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter: 0 a
Pick the second card you want to turn (e.g., 1 b) then press enter:
```

Figure 2: the player inputs a digit, a space, a letter, and presses enter (you can assume the format is always correct, but the input can be invalid, see examples below). If valid, the game asks for the second card.

```
   a  b  c  d  e  f  g  h  i  j  k  l  m
0 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
1 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
2 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
3 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□

Player 1's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter: 0 a
Pick the second card you want to turn (e.g., 1 b) then press enter: 0 b

First card picked: 5♢
Second card picked: 9♡

Player 1 has not found a match.
   a  b  c  d  e  f  g  h  i  j  k  l  m
0 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
1 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
2 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
3 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□

Player 2's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter:
```

Figure 3: if both inputs are valid, the game shows the 2 picked cards, then displays the result (in this case, the 2 cards do not match). The game continues by printing all cards printed face down again and asking the other player to pick the first card (name of the other player is Player 2).

```
Player 2's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter: 2 a
Pick the second card you want to turn (e.g., 1 b) then press enter: 0 d

First card picked: 4♣
Second card picked: 4♢

Player 2 has found a match! Earns an extra turn.
   a  b  c  d  e  f  g  h  i  j  k  l  m
0 ?□ ?□ ?□    ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
1 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
2    ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
3 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□

Player 2's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter:
```

Figure 4: Player 2 has picked 2 cards that match (these cards will be added to Player 2's winPile in the code). Since Player 2 has found a match, it will still be Player 2's turn. Note that this time when the game prints all the cards, the cards won by Player 2 will not be printed.

```
Player 2's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter: 0 d
Error: The card you picked is already taken.
Pick the first card you want to turn (e.g., 0 a) then press enter: 4 d
Error: The card you picked has invalid index(es).
Pick the first card you want to turn (e.g., 0 a) then press enter: 0 p
Error: The card you picked has invalid index(es).
Pick the first card you want to turn (e.g., 0 a) then press enter:
```

Figure 5: Here Player 2 inputs a number of invalid choices, including a card that is already won and indexes that are invalid (out of bound). The game will display the corresponding error message and continue to ask Player 2 for a valid input.

```
Pick the first card you want to turn (e.g., 0 a) then press enter: 3 m
Pick the second card you want to turn (e.g., 1 b) then press enter: 3 m

First card picked: T♠
Second card picked: T♠
Error: Both cards are the same.

Player 2 has not found a match.
   a  b  c  d  e  f  g  h  i  j  k  l  m
0 ?□ ?□ ?□    ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
1 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
2    ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□
3 ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□ ?□

Player 1's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter:
```

Figure 6: Here Player 2 picks the same card twice. Since those are valid index the game will proceed to check if the picked cards are a match. But it should detect that those are actually the same card and print the corresponding error message. As Player 2 does not find a match it's now Player 1's turn.

```
First card picked: 6♠
Second card picked: 6♣

Player 1 has found a match! Earns an extra turn.

Player 1 has won 12 cards.
Player 2 has won 40 cards.
Player 2 has won!
```

Figure 7: Eventually all the cards will be won. When this happens, the game will announce the number of cards won by the 2 players. Whoever has more cards wins the game (it's also possible for a tie, if that's the case, the last line will be replaced by: It's a tie!).

To help you to test your code quickly, there is a file called **firstShuffle_Seq.txt** in the supporting zip file. This file contains all the inputs needed to reach the end of the game as shown in Figure 7. To use it, make sure you have correctly implemented the *shuffleDeck* function so it produces the exact ordering. Then call it once and use the stream redirection tool that you learned in CMPT 127 to stream the content of the file into the standard input of the game:

```
$ ./game < firstShuffle_Seq.txt
```

If you want you can rearrange the content in this file to let the other player win or create a tie. We will use our own version to test your code so you don't have to submit this file.

```
Player 2's turn.
Pick the first card you want to turn (e.g., 0 a) then press enter: Pick the second card
 you want to turn (e.g., 1 b) then press enter:
```

Since the input is now streamed from a file, you will not see them in the stdout and thus the prompts from the game will be grouped together. This is normal.