pa02 : Linked-list implementation of a card game

num	ready?	description	assigned	due
pa02	false	Linked-list implementation of a card game	Fri 02/09 09:00AM	Thu 02/22 11:59PM

Introduction

- Create a github repo following the correct naming convention.
- If you have a partner, add you partner as a collaborator to the repo. Partner should accept the invitations to complete this process
- Join a team on submit.cs
- Get the starter code for this lab by doing a git pull in your starter-code repo and copying the files from ~/cs24/cs24-w18-starter-code/pa02/

Goal of this assignment

- Practice using linked lists
- Learn to organize a project's code structure on your own (not just filling in a template)
- Learn to design classes using good OOP design principles discussed in class

Instructions

The game

Alice and Bob are playing a game a bit like Go Fish, although neither of them is very good at it. Rather than randomly guessing what card the other person has, they take turns to find a matching card. Each player goes through their list of cards in order and asks the other player to list all of their cards in order until a match is found. The game ends once they do not have any cards in common. Note that they do not draw any cards during this process.

You have pointed out to Alice and Bob that this game is completely deterministic. They're not sure what you mean, so you decide to demonstrate by showing that a computer can predict how a game will play out. To do this, you'll be using a linked list!

Your approach

At the start of the program, you will read in Alice and Bob's starting hands from two files. The names of these files are provided as command line arguments with Alice's file in `argv[1]` and Bob's in `argv[2]`. The starter code opens the files for you as `ifstream` objects, which you can treat much like `cin`. You should read Alice and Bob's cards into two linked lists, making sure to maintain the order in the file, that is the first card in the file should be at the head of the list.

Once you have the lists of cards, the game begins. Alice goes first and iterates over each card in her hand, checking whether Bob has that card. Once a matching card is found, you should print the line "Alice picked matching card <card value as a number/character>". The card should then be removed from both players hands. Make sure to delete any dynamically allocated memory when removing the cards from your lists!

The process then repeats, except this time, Bob looks through his cards for the first one that Alice also has. Each player should begin their search where they left off in a previous round. Once there are no matching cards, you should print out the final hands of both players with the matching cards removed.

You should write your own Makefile for this lab so that running `make` builds an executable called `game`.

Example

Contents of `alice_cards.txt`:

```
h 3
s 2
c a
c 3
h 9
s a
```

Contents of `bob_cards.txt`:

```
c 2
s a
d j
h 9
c 3
```

Correct output after running `make && ./game alice_cards.txt bob_cards.txt`:

```
Alice picked matching card c 3
Bob picked matching card s a
Alice picked matching card h 9

Alice's cards:
h 3
s 2
c a

Bob's cards:
c 2
d j
```

Note: a=ace, k=king, q=queen, j=jack

Requirements

For this lab, you will have a lot of flexibility on your implementation (which just means we won't be providing a code framework for you to fill in). However, there are a few requirements that your mentor will check for when they look at your code. Keep these in mind as you think about your solution:

- You must use a linked list you implemented yourself to solve the problem, not another data structure or a linked list in the standard library.
- Your solution must include at least one overloaded operator. This will probably be the operator on your Card class to check if two cards match i.e. they have the same suit and value.
- Your linked list must implement a constructor, a de-constructor and other other methods as needed
- You code should be readable
- Your classes should define clear interfaces and hide implementation details as much as possible.
- Your program must properly free all memory it allocates, including your linked list nodes and any dynamically allocated data stored inside them. We will also check this with valgrind when you turn in your code to submit.cs.
- Extra credit: Appropriate use of recursion to write concise and elegant code without compromising on performance.

Submission instructions

Files to submit

You must organize your program in three files: main.cpp cards.cpp and cards.h In addition you must create a Makefile that compiles your program to an executable called `game`

You must submit exactly four files with the names: Makefile, main.cpp, cards.cpp, cards.h

Checkpoint deadline: 02/16 at midnight (20 points)

Complete an initial design of your classes and push your code to github by the checkpoint deadline. For example if you decide to use two classes to create the card game you must provide the declarations of both classes in cards.h and describe the public interface of each class. You must also complete implementing the code that reads each player's cards from a file and stores it in a linked list as well as the code that prints each players hand. To get credit for the checkpoint your code must compile without error. This means that you must create stubs for any functions that you are yet to implement.

Include the commit message "CHECKPOINT SUBMISSION" when pushing your code to github.

Submit PA4 at https://submit.cs.ucsb.edu/, or use the following command from a CS terminal:

```
~submit/submit -p 951 Makefile cards.cpp cards.h main.cpp
```

You should see a score of 60/60

The remaining 40 points are broken down as:

- 20 pts for meeting the checkpoint requirements
- 20 points for style and overall design following the guidelines in the requirements (above)

Created by Jack Alexander and Diba Mirza