COP 3530 – Data Structures and Algorithms I

Project 3

Due: Friday, March 30th 11:59 P.M. CST

Objective:

This project is meant to help you illustrate concepts of linked lists and sorting. In this project, you are asked to create the game of Go Fish. In this project, you will play against a computer opponent in a single player game. Again, you will be asked to create a makefile.

If the project fails to compile, the maximum grade that can be earned is a 15! These issues are caused by either incorrect syntax or not running on the SSH. Make sure you take care of both!

Problem Description:

Go Fish is an excellent and fun game for all ages. It is a card game that is based around the standard 52-card pack of playing cards that most people are familiar with.

The goal of Go Fish is to form the most "books" of cards. A book is any four of a kind (e.g. four kings, four aces, four fours, etc.). The suits of the cards are not important, only the card numbers are relevant.

At the beginning of a two player game of Go Fish, each player is given seven playing cards and the remainder of the deck is placed as a draw pile on the center of the table. In our game, the player will always go first. They begin by identifying a rank of a card that they want. They must have at least one card of the rank in their hand. They will then request any cards that the computer has in their hands. This is typically done by saying things such as "Give me all of your threes." The computer then must give all of their cards of that rank to the player. If the computer does not have any cards of the requested rank, then it will say "Go Fish!" and the player must draw the top card of the draw pile into their hand. As long as the player gets one or more cards of the rank that they asked for, either by guessing the computer's cards or by drawing it from the stack, then the player must reveal the card and the player will get another turn and this continues until the player fails to get the card they request. If the player gets the fourth card in a four of a kind, then they show all four cards and place them face up on the table and play continues.

If the player fails to get their requested card, either from requesting from the computer or drawing from the top of the draw pile, it is the computer's turn and the same rules apply. The game ends when all thirteen books have been formed. The winner is the player with the most books. If at any point during the game, the player is left without cards, he may, on his turn, draw five cards from the draw pile. If there are less than five cards, they draw all cards remaining. If no cards are remaining, then they are out of the game and are left with however many books they currently have.

Your task is to build a game of Go Fish where a single player can play against the computer.

Decks and Playing Cards

The focus of this project is on linked lists. As such, your deck of cards will be represented as a linked list. You will need each playing card then to be a node that will hold a suit and a rank. As part of your deck, you should implement the following functions:

- **Deck createEmptyDeck()** – will create an empty deck of cards that contains no cards. This is useful for using a deck of cards as a hand.
- **Deck createFullDeck()** – will create a full deck of 52 cards with all suits and ranks. This is useful for creating the initial deck that will be needed in the game.
- **void shuffle(Deck cards, int times)** – should perform a perfect riffle shuffle where a deck is cut into two halves and then the halves are interwoven. This should be repeated a number of times specified by the user.
- **void sort(Deck cards)** – should perform a merge sort on the deck. You can do this by calling a helper function within your file that does the actual sort. The merge sort should put all cards in order based on their rank (not their suit).
- **Card draw(Deck cards) –** returns the top of a deck of cards and removes it from the rest of the deck.
- **void insertIntoHand(Deck hand, Card newCard)** – inserts a card into an already sorted hand by searching for the sorted location of the new card.
- **void dealCards(Deck startingDeck, Deck hand, int numberOfCards) –** will deal the specified number of cards to the hand.
- **void freeDeck(Deck cards) -** frees everything necessary for the Deck.

You are allowed to have other functions in your deck, but these functions must be there. Furthermore, there is a caveat. You are not allowed to convert a deck of cards into an array to make any of the above steps "easier". You must perform the operations on linked lists.

As part of implementing the deck, you are required to create a merge sort. This merge sort should sort all cards in a deck by their rank with "two" being low and "ace" being high. Note that if we just use strings in our cards, this is complicated as sorting would not be something as simple as a numerical of alphabetical ordering. Instead, you will need to know how to compare a "three" to a "four" or a "queen" to a "jack". If you do this with conditionals that compare every possible pair of ranks to each other, then you are looking at a total of 132 different pairings (144 if you compare the ranks to themselves), far too many for this purpose.

For this reason, we aren't going to store suits or ranks of our cards as strings. Instead we will use integer values. The struct that you use for your card should have the minimum of two integers, one for rank and one for suit, and a next pointer to the next card.

It is possible to just memorize what integers represent suits and ranks, however, this often makes your code harder to read and more prone to bugs. Instead, I want you to make use of the concept of enums. Enums are user defined types that can be used to assign integral values to names. For instance, in your program, you may have the following enum:

```
enum Suit {HEARTS, SPADES, CLUBS, DIAMONDS};
```

Once this enum has been defined, whenever you use HEARTS in our program, it will work the same as int HEARTS = 0. Similarly, SPADES would be 1, CLUBS would be 2, and DIAMONDS would be 3. This allows you to do things like:

```
if(getSuit(card) == HEARTS)
```

Note, that this is much more readable than:

```
if(getSuit(card) == 0)
```

By assigning ranks and suits to integers in the card, this means that you can sort on integers instead of trying to keep track of which ranks beat out which ranks. I would expect your program to have card.c and card.h files. Place your enums for ranks and suits into card.h. You can read more about C enums on Geeks for Geeks (https://www.geeksforgeeks.org/enumeration-enum-c/).

Playing the Game

When the program first starts, the player is asked if they would like to start a new game or exit the program.

```
Welcome to Go Fish!

     1. Play a new game
     2. Exit the program

Please choose an option to continue:
```

When a player chooses to play a new game, the computer will create a new deck of cards, shuffle it and deal seven cards to each player. The player always goes first, so the program will then print the player's hand and give the player options for viewing the player's books, the computer's books, or making a guess:

```
Your Hand:

     two clubs

     eight hearts, eight diamonds

     jack spades, jack diamonds, jack hearts

     king clubs

     1. Make a guess
     2. View your books
     3. View computer's books

Please choose an option to continue:
```

If the player chooses to make a guess, the computer should ask the player what rank they wish to guess. The player should enter a string, either "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "jack", "queen", "king", or "ace" (without the quotes). One technique then that you can use to determine which selection was made is to have an array of strings with all of

the ranks in it. The ranks should be in this array in the same order that they are in the enum for ranks. You would then run through the array to find the rand, and then use the index of where the rank appeared in the array to make your guess.

If the computer does not have any cards of the guessed rank in their hand, then the program will draw a new card into the player's hand. If the drawn card matched the player's guess, then they will get a new turn. If a book is formed, a message is printed to the player and the four of a kind is automatically added to a Deck of cards representing the player's books and the player's number of books should be updated.

After the player's turn, the computer will guess. During the computer's turn, the player will not be given any options. Instead messages will be printed that will update the player about any guesses or transitions of cards from the player's hand to the computer. Again, books will automatically be formed and the computer's number of books should be updated.

At the end of the game, when all books are formed, the number of books for the player and the computer is compared and the winner is announced.

The player will then be shown the starting screen again.

Sorting the Cards

You will need to sort the cards based on rank. This means that all twos are together, all threes are together, and so on in rank order. The best sorting algorithm for sorting linked lists is to sort using merge sort. This sort allows for a linked list to be broken down into the individual nodes and then for those nodes to be rejoined together into a linked list again. As part of implementing merge sort, you will need a way to find the middle of a linked list. This can be done using slow and fast pointers. The idea behind slow and fast pointers is simple:

1. At the beginning, set each pointer to point to the beginning of the linked list.
2. Pass through the linked list. With each pass, the slow pointer will move one node at a time while the fast pointer will move two nodes at a time.
3. Once the fast pointer has reached the end of the linked list, the slow pointer would have only moved half the distance that the fast pointer did. This means that the slow pointer will be pointing at the middle of the linked list.

You can then use this technique to split a starting linked list into two sub linked lists.

Shuffling the Deck

You will shuffle a deck of cards using what is known as a riffle shuffle. This is where you split a deck of cards into two halves and then merged them by interweaving the cards from the two halves. Anyone who has played with a deck of cards has normally seen someone use this type of shuffling technique at least a few times. A perfect riffle shuffle is one in which the cards from the two halves perfectly interweave with each other. In real life, a perfect riffle shuffle is very difficult to accomplish as cards like to stick together due to friction. However your shuffle function should implement perfect riffle shuffles. Again, you can use the slow and fast pointer technique to find the two halves of the linked list.

Dealing the Cards

Each player's hand is represented as a Deck of playing cards. When cards are dealt, you should be moving cards from one linked list which is the initial starting deck to the linked list that represents a player's hand. The number of cards that gets dealt is passed in as an argument to the function. After all cards are dealt to a player's hand, the cards in the hand should be shuffled for easy viewing and manipulation during play. When new cards are drawn from the draw pile into the player's hand, they should be inserted into their sorted position. Note that an alternative to dealing and then having a player sort is to use the insert into hand function each time a card is dealt. This would simulate an insertion sort. While this is an acceptable use of the deal function, you should still implement merge sort in your sort function.

Printing a Player's Hand and Books

At the beginning of a player's turn, their hand should be displayed. Each rank of card in a player's hand should appear on its own line, with each card of that rank separate by a comma. An example of this would be:

```
eight hearts, eight diamonds

jack spades, jack diamonds, jack hearts

king clubs
```

Note that they are being printed in sorted order because that is the way they should be held in the hand.

As a player creates four of a kinds, they get added to the books that the player has. Each player should have a Deck that holds the Cards that have formed books. Players may need to know what books they or the computer has on the table. In order to know this, they can choose to print their books or the computer's books.

Note, that it may be easiest to implement both tasks as a single function called printDeck that will print out any deck passed to it according to these guidelines.

Running Your Program

Your program should be started with one command line argument that indicates the player for your program. The argument is the letter A or the letter B, designating player A or player B. Here is an example of how the program should be ran:

$./gofish.x

Additional Guidance

You should create a struct that can capture the information about the player. This struct should contain at minimum, Decks for the player's hand and the player's books, and an integer that represents the number of books the player has formed. Note that you can have additional information if you find that you need it.

Develop the program incrementally. Break it down to the various tasks and use functional stubs when needed. This can help you ensure that you can submit a compiling program.

Try not to repeat code when it is not needed. Instead, identify tasks that are appropriate to break out into new functions and use those instead.

Extra Credit Opportunities

(10 points) Improve your sorting so that it no longer only sorts based on rank, but also sorts based on suit with the order being "hearts", "spades", "clubs", "diamonds". This means that if you have an "eight of hearts" and an "eight of clubs", the "eight of hearts" should appear before the "eight of clubs" in the deck after the sort.

(5 points) Printing strings is a rather unattractive way to display the suits of cards. Instead of printing the names of suits and ranks as strings; use character codes to display symbols representing suits in place of the suits themselves and use numbers and single characters for the ranks.

Breakdown of Graded Items:

The following is a breakdown of what will be graded in your project. I am not giving exact point values as it gives me the freedom to be lenient when necessary:

1. Your program should be able to successfully create a deck of cards as a linked list.
2. Your program should be able to successfully print a deck of cards as described.
3. Your program should be able to successfully deal cards.
4. Your program should be able to successfully sort cards.
5. Your program should be able to successfully shuffle cards.
6. Your program should be able to successfully form books.
7. Your program should be able to successfully perform the additional functions in the deck of cards.
8. Your program should successfully allow player's to make guesses.
9. Your program should successfully handle the game of Go Fish as described and announce a winner.
10. Your program should provide a makefile that will properly compile your program.

In addition, please add your name as a comment to the top of each file.

Submission Instructions:

Make sure that before submitting your program that you have tested it on the ssh server (new domain: cs-ssh.uwf.edu). There is information for how to do this on eLearning. For this project, submit a zip file containing all files required to run your project. *For this project, you have the freedom of choosing your file layout, however this means that your makefile is vital to getting your program to compile*. Submit your project using the eLearning dropbox. Remember that all students who turn in their projects by the due date that receive a B or better will receive an added 0.5% on their final overall average.