CS 235 Midterm
Version 0.6
Instructor: M. Clement

March 6 – 10, 2017 (Monday - Friday)
Due in the Lab on Friday no later than 8:00 p.m.
Penalty for submitting the midterm late:
20 points per day (**including weekend days**), advancing at 8:01 p.m. each day

Open Book (142 course text and your CS 235 course text only), Open Notes (personal notes including your own Lab solutions)
Open Course Website and the **reference section** of www.cplusplus.com, but no other Internet resources (no Google or StackOverflow)
Closed Neighbor (and everyone is thy neighbor)

Instructions
(Please read carefully)

1. This midterm consists of a C++ programming problem and an extra credit feature. Read and understand the statement of the problem completely before beginning to design, code, and test. As part of your design, prepare a UML diagram and consider the test cases that will establish the correctness of your solution. **Test your solution thoroughly before submitting it.**
2. Produce a solution, which consists of your C++ code, with a comment at the beginning of each file (both .h and.cpp) which includes your name, and "CS 235 Winter 2017 Midterm." When you are finished, go to the course website and follow the link labeled "Submit Exam" in the Grades menu. Upload your completed project by compressing the files and submitting through Learning Suite with TA assistance. **If a packet is not collected by a TA upon submission, your midterm will not be graded and you will receive no credit.** Attribute any code taken from or based on other sources (excluding the authorized sources). Attributed code copied from or based heavily on outside sources is worth half credit. Unattributed code copied from or based heavily on outside sources is worth no credit.
3. Understanding the problem correctly is part of the examination. If something seems unclear, ask a CS 235 TA (but no one else) for clarification. You may pose questions to the CS 235 TAs at any time. However, the TAs, generally, are not permitted to answer questions related to design, C++ implementation, debugging, or testing.
4. Prior to submitting your midterm, score it using the attached scoring sheet (this will help you maximize your points and will help us grade your exam accurately). If your score is within 5 points of the TA score (inclusive), you get a 3 point bonus. If your score is within 6 to 15 points of the TA score (inclusive), no bonus or penalty is imposed. If your score differs by more than 15 points from the TA score, you lose 3 points.
5. Your solution packet must be stapled together before it will be accepted by a TA, even if this results in a late submission penalty.
6. Submit your lab packet to a TA at the time that you submit your code on LearningSuite. Because no TAs will be available on Saturday and Sunday, you may submit your code on LearningSuite without TA help on Saturday or Sunday as long as you submit your lab packet to the TA office on the following Monday. Note that you will still receive the late penalty for submitting on Saturday or Sunday and free lab late days cannot be applied to the midterm.
7. Sign the grading sheet to request that your midterm be graded and to certify that no unfair information related to the midterm has been received by you, either directly or indirectly, and that none will be conveyed by you. If we discover that you cheated or assisted someone in cheating, intentionally or unintentionally (including accidentally), your score for this exam may (and probably will) be rand() % 1.

We're serious.

Go Fish

The purpose of this midterm is to simulate the game Go Fish by implementing a double-linked list. If a double-linked list is not used, you will be subject to a 50% penalty in any section of your solution where something other than a double-linked list is used. Your linked list will consist of *card nodes*. A card node consists of a string, number, and pointers to the nodes on each side, i.e. the predecessor and the successor. Do not use any predefined data structures in your program.

Here is how our Go Fish is played:

1. There are two players, you and the computer. Each has a hand of cards. There is also a draw deck with the remaining cards that are not in each player's hand.

2. The game starts by each player alternating drawing cards until the hand size is met or the deck runs out of cards.

3. Players alternate guessing cards that are in their opponent's hand from what cards they have in their own hand. This guess consists of a card title and number.

4. If the guess matches both the title and number associated with a card in the other player's hand, the associated card (node) is removed from both players' hands (linked-lists), and the guessing player earns a point.

5. If the guess matches neither title nor number in the other player's hand, then the guessing player draws a card from the deck.

6. Play continues until a player runs out of cards in his/her hand, i.e. his linked list is empty. The player who empties his/her hand receives an extra 3 points.

7. The game ends when any player's hand empties. The player with the most points wins.

No test driver will be provided. You will need to create your own double-linked list class and card class (You may adapt your linked list from Lab 2). You will also need to create a class that plays the Go Fish game and a main to be used for both your own testing and for the TAs to play the game for pass-off purposes. Use what you have learned in Lab 2 and Lab 4 to help you with this midterm. Both hands and the deck are to be created using your double-linked lists and card nodes.

The main program should present the user with 4 options:

1. Create Deck – prompts for input of a text file with the deck info (see below)
2. Display Deck – displays the deck that was created (see below). If no deck has been created or the deck was deleted from a game finishing, it informs the player and returns to prompt.
3. Play – prompts the user for an initial hand size and then runs the game (see below). If no deck has been created or the deck was deleted as a consequence of a game finishing, it informs the player and returns to prompt.
4. Quit – quits the program

**All user interfaces need to be bullet-proofed so that the program does not break as a consequence of invalid inputs!

The following functions need to be implemented in order to pass off the midterm:

createDeck(string)

This function will read in a string representing a text file of cards to add to the deck. As a reminder, you can use "ifstream" to open a file and "getline" to read from the opened file one line at a time (include both <iostream> and <fstream>). For example:

ifstream myFile("filename") //opens a file called "filename"

getline(myFile, line) //reads a line from "myFile" and assigns it to the string variable "line"

The file will consist of lines of a string followed by a number. An example file is SimpleDeck.txt:

Fish 1
Fish 2
Fish 3
Cat 1
Cat 2
Cat 3

Two cards will need to be created from each line (each card needs to have something to match!), and the cards must be added to the front of the deck's double-linked list. The program then displays the deck created (see displayDeck below). You may assume that all files will be formatted correctly. If a deck has been created already, creating a new deck replaces the former deck.

shuffleDeck()

Shuffles the order of the cards in the deck. More than a few cards in the deck need to be moved around. When the TA looks at the deck, he/she should be able to tell that the deck has been sufficiently shuffled. You may use the random_shuffle function in the algorithm library or your own shuffling algorithm, whichever is easiest for you.

drawCard()

The current player draws a card from the deck. This draw should be performed by generating a non-negative random number less than the size of the deck's linked-list. If it is the player's turn to draw, traverse that number of card nodes beginning at the head of the deck's linked-list. The last card node encountered is added to the tail of the linked-list representing the player's hand. For the computer, do the same, but start traversing backwards from the tail of the deck's linked-list.

For example, if the computer were drawing from a 10-card deck and the random number generated was 4, this method would iterate from the back of the linked list and the computer would draw the 6^{th} card node in the linked-list.

If the deck is empty, no card is drawn.

dealCards()

Starting with the computer, both players alternate drawing a card from the deck until their hands reach the initial hand size or the deck runs out of cards. The same draw rules as stated in the drawCard() method apply here. Once hands have been dealt, look for matches within the same hand (not between the computer and player hands!). All matches created from the cards dealt to a player are discarded (removed from the linked-list) without any points being earned. Be careful to discard only the pairs and not all cards that match. No additional cards are drawn to refill to the initial hand size. For example, a player is dealt the hand:

Player: Fish 1, Fish 2, Cat 1, Cat 1, Fish 2, Cat 1, Fish 3

The discarded cards would be: Fish 2, Fish 2, Cat 1, and Cat 1. Even though there are three "Cat 1" cards, only a pair is discarded because there is still another match for the third "Cat 1" card somewhere! The player does not draw additional cards. Here would be the resulting hand:

Player: Fish 1, Cat 1, Fish 3

displayDeck(),displayPlayerHand(), displayComputerHand()

Displays the current contents of the deck or player/computer hand's linked-lists. Although viewing the computer's hand would be cheating in a normal game of Go Fish, this is needed both for your testing and for TA pass-off. The format for displaying the deck should be the same as the format of the file, with the card's title and number separated by a space and one card per line. The top of the displayed list should be the head of your deck's linked-list. If no deck has been created or the deck is empty, display "Empty". Hands should be displayed with all cards on the same line separated by commas starting from the head of the list. For example:

Player: Cat 2, Cat3, Cat 1

Computer: Fish 2, Fish 1, Cat 2, Cat 3

play(int)

Starts the game! The parameter is the hand size for the game. The deck should be shuffled and dealt to both the computer and the player. Check for end-game conditions (see below) before each player's turn, even on the first turn. The turn order starts from the user. Each human-player turn, his/her hand and both the player's and computer's points should be displayed for the player's reference. The player is then prompted with a set of options:

1. Make Guess
2. Shuffle Deck
3. Display Deck
4. Display Computer Hand
5. Give Up

When the player selects option 1, he/she is then prompted to type in a card title (string) and number (int) either as one string (e.g. "Foo 42") or being prompted first for the string then second for the int. Check the validity of the guess by first making sure the input is formatted correctly and then checking the player's hand for the card. If the player does not have the card, state so and ask for another guess. If valid, check the computer's hand. If the player's guess matches both title and number of a card in the computer's hand, the cards are removed from both hands and the player earns a point. If neither title nor number matches, the player is informed and draws a card from the deck. If the card drawn from the deck matches one already in hand, remove both from the player's hand and the player earns a point. Once this is finished, the computer's turn starts.

Option 2 shuffles the deck. This is to test the shuffle functionality to determine if it is sufficiently randomized.

Option 3 displays the contents of the deck as explained above (Cheater!). Options 3 and 4 are both for debugging purposes and TA pass-off.

Option 4 displays the computer's hand (Cheater!).

Option 5 brings the player back to the main menu. All cards remaining in both hands and the deck are deleted.

The computer's turn is similar to the player's turn. A card from the computer's hand is chosen at random as the guess and is displayed to the user. If it is a match, both cards are removed and the computer gets a point. No match causes the computer to draw a card from the deck, removing any match created with the new card and giving a point for the match. Once this is finished, the play reverts back to the player.

<u>End Game Conditions</u>:

If at any time the computer or player has no cards left, whoever emptied his/her hand receives 3 bonus points and the game ends. It is possible for both players to empty their hands at the same time. In this case, both players would receive 3 points. The player with the most points is declared the winner. All cards left in the deck and player hands are deleted and the program then returns to the main menu to create a new deck and play again, or quit.

Extra Credit

Create a way for intelligent computer guessing instead of simple randomization and one that does not involve the computer knowing the cards in the player's hand, such as remembering both its own and the player's guesses to make better guesses in the future. Implement a way for this intelligence to be visible by adding another prompt on the player's turn to display what the computer "knows" or is "thinking." Remember that you will not be present when this is graded, so be descriptive.

UML Diagram

You must also provide a UML diagram in order to receive full credit for the midterm. Your diagram should include every class that you use and should contain all the elements of a standard UML diagram (connecting lines between classes, data members and functions of each class, etc.). You may either include a PDF of your UML with your lab code or print it and staple it to your grade sheet. If you need to print your UML before submission, printers are available on the first floor of the TMCB and in the Library.

Valgrind

To receive full credit, your program must pass Valgrind. You may run this through the terminal on the Linux machines. You must first navigate to your project folder and find your project's executable file. If you do not have an executable file, you can make one through the terminal command:

```
g++ -std=c++11 -o <name of executable file you want to create> *.h *.cpp
        e.g.
        g++ -std=c++11 -o goFish *.h *.cpp
```

This will create an executable file with the name you entered within your project code folder. Now to run the program with Valgrind, the command is:

```
valgrind --tool=memcheck --leak-check=yes ./<name of executable>
        e.g.
        valgrind --tool=memcheck --leak-check=yes ./goFish
```

If the result of running this command says:

```
All heap blocks were freed -- no leaks are possible
```

and there were no other Valgrind-related errors (such as invalid reads/writes, conditional jumps, etc.) then you passed Valgrind :)

Note that there is a bug with the current version of the g++ compiler that will cause a memory leak of exactly 72,704 bytes in 1 block. If you have exactly 72,704 bytes leaked in 1 block and no other Valgrind errors, you will still receive full credit.

If you have any questions about how to use the terminal to create an executable file and run Valgrind, you may ask the TAs to demonstrate how to run a program through the terminal.

Midterm Submission

The midterm is due on Friday, March 10 at 8:00p.m. When you are finished with your solution and UML diagram, have filled out the grading sheet, and are ready to submit your midterm, prepare your files for submission and find a TA. Then, with a TA present, go to the course website and follow the link labeled "Submit Exam" in the Grades menu. With a TA present, compress your files, including your UML (if you choose not to print and attach it to your grade sheet), and submit the compressed folder to LearningSuite. The TA will then collect your packet and verify that you have signed the grading sheet.

Note: Please double and triple-check that you are zipping and submitting the correct code when you submit your midterm and that your UML is included. Any re-submission of midterm code will incur the full late day penalties for when the code is re-submitted after the deadline.

Due to the large number of students who may want to submit on Friday evening, the TAs will be working with students on a first-come first-served basis on the final evening of submission. In order to be fair to students who have finished the lab and wish to submit, there will be a line outside of the TA office for students who are ready to submit their midterm. **You must be in this line before 7:45 on Friday evening in order to submit your midterm on time. At 7:45 the line will be closed and students who are in the line will be helped. All further submissions will be counted as late.** The TAs will make announcements in the computer labs at 7:35 and 7:40 so that students are not caught off guard by the deadline.

A Few Good Hints:

You will find in your coding career that you can save a ton of time and heartache by *designing* a solution before *implementing* it. Remember the wisdom of the ancients who observed that "8 hours of programming can sometimes save you up to 10 minutes with a pen and paper."

You've already coded a linked list; consider using your code as a basic outline for how to implement your double-linked list. Think about what functions you will need in order to insert into and remove from a double-linked list, and consider how you might use those functions to implement your Go Fish solution.

You should have coded a user interface in Lab 4. Use what you learned there to make coding the user interface for this lab easy. Remember that there are two separate interfaces, one before the game starts, where you create a deck, and one when the player's turn starts.

Draw pictures of your linked list and of decks/hands to make sure that you have considered all the relevant pointers, etc. that need to change with each insert/delete.

Test your code as you go: bugs become much more complicated and harder to find as you write more lines of code. Test as you go to make sure that you don't have any sneaky bugs that would have been easy to remove if you had found them earlier. For example, you might first write and test your double-linked list insert and remove functions, then take one function at a time in your Go Fish class and get it working before moving on to the next part of the midterm.

Design your test cases before beginning to code. You don't need to spend a lot of time doing it, but your test cases are much more likely to be complete if you spend 30 seconds thinking of all the border cases that you can and then writing up the code to test whether or not you have a working solution.

Grading Sheet

Student Name_____ TA Initials____

(print) Days Late____

Student:	TA:	Main Interface:
__ / 5pts	__ / 5pts	1. User interfaces follow format explained in midterm specifications.
__ / 5pts	__ / 5pts	2. Invalid input is caught; user is re-prompted for valid input.
__ / 5pts	__ / 5pts	3. When quitting, program does not crash, but closes gracefully.
__ / 10pts	__ / 10pts	4. createDeck() properly reads from inputted file and generates correct deck with two cards per line in file. New decks replace old.
		Game Initialization:
__ / 5pts	__ / 5pts	1. shuffleDeck() results in a visibly shuffled deck.
__ / 10pts	__ / 10pts	2. drawCard() correctly makes player draw counting from head of the deck's linked list and computer counting from tail.
__ / 5pts	__ / 5pts	3. dealCards() correctly fills hands to hand size or until deck is exhausted. Initial matches created are discarded without giving points.
__ / 5pts	__ / 5pts	4. Dealt cards are removed from deck and added to player hands.
		Gameplay:
__ / 5pts	__ / 5pts	1. Incorrect guess format caught and user re-prompted for valid input.
__ / 5pts	__ / 5pts	2. displayDeck(),displayPlayerHand(), and displayComputerHand() are formatted and displayed in correct order.
__ / 5pts	__ / 5pts	3. Giving up returns to main menu; both hands and deck are deleted.
__ / 5pts	__ / 5pts	4. Matched cards are removed from hands.
__ / 10pts	__ / 10pts	5.Program is able to play game from start to finish, and finished games delete remaining cards and return to main menu.
		Intelligent Guessing: (extra credit)
__ / 15pts	__ / 15pts	1. Successfully created an intelligent guessing system for computer. Included menu item on player turn runs correctly and gives sufficient information to demonstrate system works.
		Other:
__ / 10pts	__ / 10pts	1. Program passes Valgrind with no memory leaks and does not Crash (no partial credit).
__ / 5pts	__ / 5pts	2. UML diagram is neat, correctly formatted, and complete.
__ / 5pts	__ / 5pts	3. Neat and consistent code with comments where appropriate.
__ / Total	__ / Subtotal	

__ / Late Penalty (20 points per day)

__ / Accurate Grading Bonus (+3/0/-3)

___ / **Total**

Hours spent on Double-Linked List:_____ Hours spent on Go Fish: _____ Total time spent: _____

_____ _____

Student Signature TA Grader (print name)