1 Objectives

The objectives of this lab are for the student to get:

1. More practice with C programming in general.
2. Experience with an application that is coded with multiple .c and .h files.
3. Experience working with some additional compiler options.

2 Program Requirements

2.1 Big Picture

It is your job to complete the implementation of a very simple blackjack application.

2.2 Implement card.c (to deal cards)

The part of the card.c file that handles the dealing of cards is just "stubbed out" as it is called, meaning that the external function has been defined but not implemented. This function is called `card_get()`, which is supposed to return a card from a shuffled deck. Roughly, you need to do the following:

1. Figure out how to store a representation of a single deck of 52 playing cards within card.c.
2. Figure out how to produce a random sequence of cards from that deck. In other words, when you give out a sequence of 52 cards there should be no duplicate cards, and that sequence should be different from the last time the program was run.
3. When requested, give a card to the calling function.
4. After all 52 cards have been dealt, if `card_get` is called a 53rd time, then it will continue to give out cards, but in a new sequence.

There are some additional comments in card.c, just above `card_get()`, that provide the details of what needs to be returned when `card_get()` is called. Do not change the inputs and outputs of `card_get()`, and in general do not change card.h. View the contents of card.h to see the externally-published macros (i.e., those values published via a #define statement). You should also look at common.h.

Main.c runs the game by keeping track of the score for the player and the dealer, and determining the winner, etc. You should not need to make changes to main.c. There are no guarantees that it works correctly. It certainly does not perfectly capture the complete rules of blackjack.

Table.c is also provided as the module that manages the game board. It is used to display the dealt cards, and visually display the number of wins and losses for the player. You should not need to make changes to table.c. There is no guarantee that it works correctly.

Use the `random` function, not the `rand` function. You will not need to call `srandom` because it has already been done for you.

If you think you have found a bug in the supplied code, please inform the instructor.

2.3 Notes

1. To build the application, enter `make` at the command-line.
2. To run the application, enter `./blackjack` at the command-line.
3. To help you test your code, you have been provided some test code (instead of having to play the game and track the cards dealt).
    a. To build the test code, enter "`make test`" at the command-line.
    b. To run the test, enter `./test` at the command-line.

3 Compiler Options Exercise

a. Compile the program in different ways, as instructed below

   i. Compile the program using the typical compiler options:
      ```
      make clean
      make
      ```
      Record the size of the `blackjack` program.

   ii. Open the Makefile and change the LDFLAGS macro from "-o" to "`--static -o`". This will cause static linking to take place. Recompile the application:
      ```
      make clean
      make
      ```
      Record the size of the `blackjack` program.

   iii. Open the Makefile and change the LDFLAGS macro back to just "-o". **Also**, expand the CFLAGS macro from "`-Wall -c`" to "`-Wall -c -O1`". (Note that the `-O` is an upper-case 'o', **not** a zero). The `-O1` option tells the compiler to try to reduce code size and increase application speed, but without taking too much compilation time to figure it out. Recompile the application:
      ```
      make clean
      make
      ```
      Record the size of the `blackjack` program.

   iv. Open the Makefile and replace the "`-O1`" option with the "`-O2`" option. The `-O2` option tells the compiler to try to increase the application speed, even if it takes more time to compile. Recompile the application:
      ```
      make clean
      make
      ```
      Record the size of the `blackjack` program.

   v. Open the Makefile and replace the "`-O2`" option with the "`-O3`" option. The `-O3` option tells the compiler to optimize even more. Recompile the application:
      ```
      make clean
      make
      ```
      Record the size of the `blackjack` program.

   vi. Open the Makefile and replace the "`-O2`" option with the "`-Os`" option. The `-Os` option tells the compiler to try to reduce the size of the application. Recompile the application:
      ```
      make clean
      make
      ```

Record the size of the `blackjack` program.

b. Put the recorded results into a text file named “results.txt”.

4 Submission

Post the following **two** files to Sakai by the deadline:

a. **dist5.tar**, that contains the following 8 files:
   i. All .c files (main.c, card.c, table.c, test.c)
   ii. All .h files. (card.h, table.h, common.h)
   iii. Makefile

b. results.txt

5 Grading

Your grade shall be based on the following **guidelines**:

1	Compiles without errors:	30	
2	Compiles without warnings:	10	
3	Valgrind reports no errors (when run on test)	10	(and no seg. faults)
4	It deals 52 different cards in a row	10	
5	The next 52 cards are a different set than the first	10	
6	No noticeable delay when a card is requested	5	
7	Style guide followed properly	10	
8	Debugging code is either not present or commented out	5	
9	random was used instead of rand	5	
10	The results.txt file provides requested info	5	

I reserve the right to deduct points when it appears appropriate.