CSC 101 Programming Assignment 3
Preparation for Card Game Programming

ISSUED: Monday, 23 April 2012
DUE: On or before 11:59:59PM Monday 7 May, via `handin` on unix1
POINTS POSSIBLE: 1
WEIGHT: 5% of total class grade

Overview

The purpose of this assignment is to write functions that will be used in programming assignments 4 and 5, where you will write a program to play a card game. Writing the program for assignment 3 requires that you understand the following concepts:

- arrays of strings
- functions with array parameters
- the `typedef` declaration
- use of the random number library function
- program organization with .c and .h files

Specification

The card game to be played will use a standard deck of 52 cards, where each card has a face value and suit. The face value is one of "Ace", "King", "Queen", "Jack", or a number between 2 and 10. The suit is one of "Spades", "Hearts", "Diamonds", or "Clubs".

The specification of this assignment covers two areas:

- the external representation of cards as input by and output to the user who is playing the card game;
- the specification of three functions that will be used in the game program;

External Representation of Cards

A human player is going to communicate with the computer when the card game is played. To do so, the human views and input card values on the terminal. The following external representation is used for card input and output:

The face value of the card is represented as follows:

Face Value	**External Representation**
Ace	A
King	K
Queen	Q
Jack	J
10 through 2	the number itself

The suit of the card is represented as follows:

Suit Name	External Representation
Spades	S
Hearts	H
Diamonds	D
Clubs	C

For example, the Ace of Spades is input and output as the string "AS". The two of clubs is "2C".

Card-Related Functions

For this assignment, you are implementing the following three functions:

1. A Boolean function that compares two cards and returns true if the first card "beats" the second card, in the normal rules of ace-high card ordering. Specifically,
    a. Ace beats king beats queen beats jack beats any numbered card.
    b. A numbered card beats another numbered card with a lower value. E.g., a 10 beats a 3.
    c. When two cards are of the same face value, the suit of the card is used to determine order. Specifically, spades beats hearts beats diamonds beats clubs.
2. A function to deal two hands from a deck of cards. The function takes as input the deck of cards and the number of cards to be dealt to each player. The function returns the hands of cards dealt to each player in two array parameters. As an integer return value, the function returns the number of cards dealt. The hands are dealt one card at a time, starting from the top of the deck. If there are not enough cards in the deck to deal both hands, the function deals as many as it can.
3. A function to shuffle a deck of cards. The function takes an unshuffled deck of 52 cards as input and returns the shuffled deck as an output array parameter. The meaning of shuffled is that the cards are in random order in the shuffled deck.

Program Structure

The program you are writing for this assignment consists of four separate files:

1. `cards.h` -- a header file that defines the types used in the program and the function prototypes
2. `cards.c` -- the program file that you implement, that has the definitions for the prototypes in `cards.h`, plus any additional functions you need to call to get the job done
3. `cards_test.h` -- a header file for the program testing functions
4. `card_test.c` -- implementation of the testing functions, including a the `main` function

Your primary job for this assignment is to implement all of the functions in `cards.c`. This file is 100% your responsibility.

You will use the `cards.h` file largely as it is provided. The only modifications you need to make to this file are if you define any new functions in `cards.c` that don't have a prototype in provided in the original `cards.h`. If you do this, all you need to do is add prototypes to `cards.h` for your new functions. Whether or not you define new functions is up to you, but it will likely help with your overall program design.

You use both `cards_test.h` and `cards_test.c` 100% as is. Note in particular that you do not write your own `main` function for this assignment. It's provided for you in `cards_test.c`.

You compile your finished program like this:

```
gcc -g -ansi -pedantic -Wall -Werror -o cards_test cards.c card_test.c
```

The `card_test.c` program will call the functions you write and print the results to `stdout`. The output of a successful test run is in the file named `test_output` . This is a "flow blown" test run of a properly implemented set of functions. As described below under "Testing Details" it is *strongly recommended* that you write your own simpler versions of `cards_test.c`, that you run before you do the full blown tests.

Design and Implementation Details

The Cards and Deck

One of the key design issues for a card game program is to determine an appropriate data representation for the deck of cards. The basic data representation needs to be an array of some form. There are a number of possibilities, the most straightforward of which is an array of strings, where each array element is the external representation of a card. This is the representation that is defined in the provided `cards.h`, and therefore the one you'll use for this programming assignment.

For both decks and hands of cards, you need to represent how cards are removed. Two approaches are marking and an integer position variable. Marking involves putting some empty value in the deck, in the place previously occupied by a card. The position-variable approach involves decrementing an integer variable each time a card is removed from the deck.

In this assignment, you will be using the the position-variable representation. This means that the functions above that refer to "a" deck actually refer to two parameters, one for the deck itself and the other for the integer position that's the top of the deck.

The Shuffling Algorithm

To shuffle a deck of cards you use the math library function named `rand`. The algorithm goes like this:

1. there are two decks of cards: an input deck that has all the cards in order, and an output deck that has the cards placed in random order
2. you loop through the ordered deck one card at a time
3. for each card in the ordered deck, you place it in a random position in the shuffled deck
4. the random position needs to be a number between 0 and 51, since these are the positions in the array that holds a deck of cards; you obtain this number between 0 and 51 by applying the modulus function to the return value of the `rand` function

To help you get started, there is an example of an algorithm similar to this in the file `rand_example.c`

Testing Details

You are given a testing program in the files named `cards_test.h` and `cards_test.c`. These files define a `main` function, which calls the testing functions, which in turn call your card functions.

Since you are not implementing an actual game-playing program for this assignment, the purpose of the `main` function is to perform testing. It has the following basic structure:

```
int main() {
    test_shuffle(...);            /* Test the deck shuffling function */
    test_deal_two_hands(...);     /* Test the dealing of hands */
    test_compare_two_cards(...);  /* Test the comparison of cards */
}
```

As noted above, the `cards_test.c` file is the "flow blown" testing program that you can run when you think you have implemented all of your functions correctly. Before you run the full blown tests, it is strongly recommended that you write your own simpler tests. An incremental development and testing strategy goes like this:

1. write just the `compare_two_cards` function in `cards.c`
2. test it using a simplified testing program, as shown in the example `compare_cards_mini_test.c`
3. compile your `cards.c` and the simplified testing program like this:

   ```
   gcc -g -ansi -pedantic -Wall -Werror cards.c compare_cards_mini_test.c -o mini_test
   ```

   and run `mini_test` to see how it works
4. add some more tests to confirm that `compare_to_cards` is working correctly
5. when you get the `compare_two_cards` function working, proceed in a similar fashion to implement and test the `deal_two_hands` function and the `shuffle` function
6. when you think all three functions are working, compile with the full blown `cards_test.c` and see how things go

Scoring

`compare_two_cards`: 30/100
`deal_two_hands`: 35/100
`shuffle`: 35/100

Collaboration

NO collaboration is allowed on this assignment. Everyone must do their own individual work.

Program Turnin Procedure

Hand in on unix1 with the command

```
handin gfisher 101_prog3 cards.c cards.h
```

If you do not modify `cards.h`, you don't need to hand it in.