Lab 0 : C Programming Familiarization

The purpose of this assignment is to (re)familiarize you with some aspects of C that may not be intuitive to students who are new to C. Even if you know C, work this assignment to re-familiarize yourself.

If you work faithfully at understanding the portions of code that you're instructed to study, and if you work faithfully at writing the code you're instructed to write you will receive credit for this assignment. The instructions are written assuming you will edit and run the code on the *csce.unl.edu* Linux server. If you wish, you may edit and run the code in a different environment; be sure that your compiler suppresses no warnings, and that if you are using an IDE that it is configured for C and not C++.

1 Terminology

The standard 52-card deck of "French" playing cards[1] consists of 52 cards. The cards are divided into 4 "suits," clubs (♣), diamonds (♢), hearts (♡), and spades (♠). Each suit consists of 13 cards: the number cards 2-10, the "face cards" (Jack, Queen, King), and the Ace. In most card games (including Poker), the Jack is greater in value than the 10, the Queen is greater in value than the Queen, and the King is greater in value than the Queen. In some games, the Ace is lesser in value than the 2, in other games it is greater in value than the King, and in some games, it can be either.

Poker[2] is a game of chance and skill played with a standard deck of 52 playing cards, in which players attempt to construct the best "hand" they can. While there are many variations of the game, they all have this in common. A hand is a set of five cards, and it can be categorized into types of hands (described on the third page), which are ranked according to the statistical likelihood of being able to construct such a hand. When completed, the code in this assignment will generate a random hand and evaluate what type of hand it is.

2 Getting Started

Download *lab0.zip* or *lab0.tar* from Canvas and copy it to a directory on the *csce.unl.edu* server. Once copied, unzip the file. The three files (*card.h*, *card.c*, *poker.c*) contain the starter

[1] https://en.wikipedia.org/wiki/Standard_52-card_deck
[2] https://en.wikipedia.org/wiki/Poker

code for this assignment. The header file *card.h* defines a "card" structure and specifies two functions that operate on cards. The source file **card.c** has the bodies for the specified functions, but some code is missing. Finally, the source file *poker.c* is supposed to generate a poker hand of five cards, print those five cards, and then print what kind of hand it is – but much of its code is missing. To compile the program, type:

```
gcc -std=c99 -Wall -o poker poker.c card.c
```

If you compile the starter code, it may generate a warning:

```
card.c:59:30: warning: format string is empty [-Wformat-zero-length]
        sprintf(valueString, "", value);
                             ^~
```

Before you make any other changes, you should edit *card.c* so that the program compiles without generating any warnings or errors. If you look at the source code, you'll see a comment with instructions "`PLACE THE CONTROL STRING IN THE SECOND ARGUMENT THAT YOU WOULD USE TO PRINT AN INTEGER`." The command **sprintf()** is like **printf()** and **fprintf()** except that it "prints" to a string. See §7.2 of *The C Programming Language* on pages 153-155 for a description of **printf()** and **sprintf()**, including some of the control sequences you can put in the control string.

If you also get a warning for an unused variable

```
poker.c:154:9: warning: unused variable ‘sizeOfHand’ [-Wunused-variable]
     int sizeOfHand = 5;
         ^
```

then you can temporarily fix this warning by commenting-out the line `int sizeOfHand = 5` in *poker.c*'s **main()** function.

3 Completing *card.c*

Look over the rest of the code in *card.c* and work at understanding anything that you don't initially understand. When you have done so, add the missing code to **createCard()** to populate a card's fields. Finally, change the first two lines of **displayCard()** so that this function uses the fields from the card argument that is passed to the function.

You may want to add a **main()** function to *card.c* and compile only *card.c* to check that you made the correct changes. Catching errors now will be easier than trying to catch them after you've started the next task.

Examine the remaining starter code in *poker.c* to make sure you understand it.

4 Completing *poker.c*

If you added a **main()** function to *card.c*, remove it so that there is only one **main()** function when you compile the full program.

In *poker.c*, the first thing you'll want to do is write the code for **populateDeck()**. Using **createCard()** from *card.c*, create cards corresponding to the 52 standard playing cards and add them to the **deck**[] array. You might put code in **main()** to print out all 52 cards in **deck**[] using **displayCard()**, to confirm that you wrote **populateDeck()** correctly.

4.1 Types of Poker Hands

In the game of poker, hands are characterized by the similarities of the cards within. Traditionally, you characterize the hand by the "best" characterization (that is, the one that is least likely to occur); for example, a hand that is a three of a kind also contains a pair, but you would only characterize the hand as a three of a kind. The types of hands (from most desirable to least desirable) are:

Royal Flush This is an Ace, a King, a Queen, a Jack, and a 10, all of the same suit. There is no function in the starter code for a royal flush, nor do you need to write one, since a royal flush is essentially the best- possible straight flush.

Straight Flush This is five cards in a sequence, all of the same suit; that is, five cards that are both a straight and a flush. This characterization is checked by the function **isStraightFlush()**.

Four of a Kind Four cards all have the same value. This characterization is checked by the function **isFourOfKind()**.

Full House The hand contains a three of a kind and also contains a pair with a different value than that of the first three cards. This characterization is checked by the function **isFullHouse()**.

Flush Five cards all of the same suit. This characterization is checked by the function **isFlush()**. In the interest of simplicity, for this assignment we changed the definition of a flush to "all cards are of the same suit" (this distinction only matters if the number of cards in the hand is not five).

Straight Five cards in a sequence. This characterization is checked by the function **isStraight()**. In the interest of simplicity, for this assignment we changed the definition of a straight to "all cards are in a sequence" (this distinction only matters if the number of cards in the hand is not five). We further re-defined an Ace to be adjacent only to 2 (in traditional poker, an Ace can be adjacent to 2 or to King but not both at the same time).

Three of a Kind Three cards all have the same value. This characterization is checked by the function **isThreeOfKind()**.

Two Pair The hand holds two different pairs. This characterization is checked by the function **isTwoPair()**.

Pair Two cards with the same value. This characterization is checked by the function **isPair()**.

High Card If the hand cannot be better characterized, it is characterized by the greatest-value card in the hand. The starter code does not have a function to check for this since this is the characterization if all of the other functions return a **0**.

4.2 Study the Code

Look at the code for **isPair()**. Notice that the parameter **hand**'s type is **card***; that is, **hand** is a pointer to a **card**. In the code, though, we treat **hand** as though it were an array. This is because in C, arrays are pointers and we can treat pointers as arrays. Now look at the rest of the code in **isPair()**. Why does this return a **1** when the hand contains at least one pair? Why does it return a **0** when the hand contains no pairs? If you can't determine this on your own, you may talk it over with other students or the TA.

Look at the code for **isFlush()**. Why does this return a **1** when all cards in the hand have the same suit? Why does it return a **0** when at least two cards have different suits? If you can't determine this on your own, you may talk it over with other students or the TA.

Look at the code for **isStraight()**. This is a little more challenging to understand than **isPair()** and **isFlush()**. Why does it return a **1** when all cards in the hand are in sequence? Why does it return a **0** when they are not in sequence?[3] If you can't determine this on your own, you may talk it over with other students or the TA.

Look at the code for **isTwoPair()**. Recall that in C, arrays are pointers. The assignment $partialHand = hand + i$ makes use of *pointer arithmetic*. If the assignment were $partialHand = hand$ then it would assign **hand**'s base address to **partialHand**, and so **partialHand** would point to the 0^{th} element of **hand**. The expression $hand + i$ generates the address for the i^{th} element of **hand**, and so $partialHand = hand + i$ assigns to **partialHand** the address of the i^{th} element of **hand**. This effectively makes **partialHand** an array such that $\forall j : partialHand[j] = hand[i + j]$.

Examine the remaining starter code in *poker.c* to make sure you understand it.

4.3 Complete the code

Write the code in *poker.c*'s **main()** function to generate a hand of five cards by calling **getHand()**.[4] (Uncomment the `int sizeOfHand = 5` line if you previously commented it.) Then have the program print out the five cards in the hand. Finally, by calling the **is...()** functions, determine the best-possible characterization of the hand and print out that information.

[3] There's actually a bug in this code. It will always return a **1** when the cards are in sequence, but it's possible to construct a hand in which the cards are not in sequence but the function will still return a **1**, such as 2,2,4,5,6. You do *not* need to fix this bug.

[4] When you test the other functions you need to write, you might want to temporarily bypass **getHand()** and explicitly assign specific cards to an array of five **card**s.

Now write the code for **isThreeOfKind()**, **isFullHouse()**, and **isFourOfKind()**. When you have completed this assignment, upload your code to Canvas.