Lab 1

C Programming Familiarization Lab

Due: Week of January 24, before the start of your lab section

Except as noted in Section 6.2, This is an individual-effort project. You may discuss concepts and syntax with other students, but you may discuss solutions only with the professor and the TAs. Sharing code with or copying code from another student or the internet is prohibited.

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 your account on the *csce.unl.edu* Linux server. Except for demonstrating that you can connect to your account on the *csce.unl.edu* Linux server, you may edit and run the code in a different environment if you wish; 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++.

Learning Objectives

After successful completion of this assignment, students will be able to:

- Connect to your account on the *csce.unl.edu* Linux server.
- Edit and compile a C program.
- Understand the similarities between Java and C code.
- Adapt to some differences between Java and C code, specifically those associated with arrays, strings, and boolean values.

Continuing Forward

Even though the C code in this assignment can be edited, compiled, and run anywhere, there will be future lab assignments that must be completed on your account on the *csce.unl.edu*

Linux server. Having learned how to connect to your account on the *csce.unl.edu* Linux server, you will be able to complete future labs.

This lab is a first step in understanding the C language. Upcoming labs will build upon this understanding, both to improve your familiarity with the C language and also to apply that understanding to learning CSCE 231's concepts.

During Lab Time

During your lab period, the TAs will help you connect to your account on the *csce.unl.edu* Linux server, and they will show you how to edit and compile a C program. During the remaining time, the TAs will be available to answer questions.

1 Connecting to your account on the *csce.unl.edu* Linux server

You will need to be able to set up a secure shell terminal to your account on the *csce.unl.edu* Linux server. You will also need to be able to edit files either directly on your account on the *csce.unl.edu* Linux server, or on your personal computer (or a lab computer) and to transfer files to and from your account on the *csce.unl.edu* Linux server.

1.1 Secure Shell Terminal

You will need to run commands on your account on the *csce.unl.edu* Linux server.

If your personal computer (or the lab computer you're using) is a Windows machine, the most popular option is PuTTY. See https://computing.unl.edu/faq-section/working-remotely#node-29471 for instructions. The principal difference is that instead of using *cse.unl.edu* has the Host Name, use *csce.unl.edu*.

If your personal computer (or the lab computer you're using) is a Mac or a Linux box, the simplest option is to open a terminal window on your computer and type `ssh` *`username`*`@csce.unl.edu`, where *username* is your School of Computing login ID. See https://computing.unl.edu/faq-section/working-remotely#node-300 for Mac, or https://computing.unl.edu/faq-section/working-remotely#node-30086 for Linux.

For a broad variety of platforms, you can use NoMachine (see https://computing.unl.edu/faq-section/working-remotely#node-30855). Note that NoMachine will only connect to *cse.unl.edu*. After you have connected to *cse.unl.edu* through NoMachine, you can open a terminal window and `ssh` to *csce.unl.edu* just as you would from any other Linux system.

If you already have an IDE on your personal computer, that IDE may provide the option of opening a secure shell terminal on a remote system. I do not guarantee that the TAs or the School of Computing tech support team can help you with connecting your IDE to your account on the *csce.unl.edu* Linux server.

1.2 Editing Files

You might choose to edit files directly on your account on the *csce.unl.edu* Linux server. If you do so, your options are `vim`, an enhanced version of the classic `vi` editor, GNU Emacs, or Pico (or its clone, GNU nano). Be aware that `vim` and Emacs have non-trivial learning curves: knowing how to use them will pay dividends in your future careers, but you may be frustrated if you're using to using more conventional editors. Pico, an editor derived from the classic `pine` email client, has a helpful list of available commands always shown at the bottom of the terminal.

If you're using NoMachine, you can use Atom, a general-purpose text editor that will have a more-familiar style user interface. Because *cse.unl.edu* and *csce.unl.edu* share a file server, you can edit files on *cse.unl.edu* and use them on *csce.unl.edu* without having to take any action to transfer the files.

Many students choose to edit files on their personal computer. If you do so, and if your personal computer is a Windows machine, try to configure your editor to use "Unix-style" end-of-line characters. This won't matter for most labs, but a couple of upcoming labs will need "Unix-style" line separators. If you cannot change this setting, then get in the habit of running `dos2unix` ***`filename`*** on your files after transferring them to your account on the *csce.unl.edu* Linux server to convert "DOS-style" line separators to "Unix-style" line separators (this utility makes a few other changes, too, that have no bearing on CSCE 231assignments).

Some IDEs allow you to edit files on a remote system. Bear in mind that these IDEs may add metafiles to the remote system in sufficient quantity to exceed your disk quota on the School of Computing's file server (VS Code is particularly notorious for this). I do not guarantee that the TAs or the School of Computing tech support team can help you with connecting your IDE to your account on the *csce.unl.edu* Linux server.

If you edit files on your personal computer but store files on your personal computer (that is, you aren't editing remote files), then you will need to transfer files between your personal computer and your account on the *csce.unl.edu* Linux server.

1.3 Transferring Files

If you are editing your files on the School of Computing's file server, whether from within a secure shell terminal, from within NoMachine, or by configuring an IDE on your personal computer to do so, then you do not need to transfer files between the file server and your local computer – but you may wish to set up the ability to do so anyway.

Similarly, if you are editing files on a lab computer, you do not need to transfer files because the "Z drive" shares a file server with *cse.unl.edu* and *csce.unl.edu*.

Otherwise, if you are editing local files on your personal computer, then you will need to be able to transfer files. If you are using Windows, then the most popular option is FileZilla – see https://computing.unl.edu/faq-section/working-remotely#node-291. It does not matter whether you specify *cse.unl.edu* or *csce.unl.edu* as the host because they share the same file server.

If you are using Mac or Linux, you have options. The School of Computing's FAQ suggests Cyberduck for Mac – see https://computing.unl.edu/faq-section/working-remotely#node-3459, but FileZilla works fine on Mac and Linux. Another option, since you're already using terminal windows to `ssh` into *csce.unl.edu* is to use the `scp` command. Basic use of the `scp` command is very much the same as basic use of the `cp` command, except that you specify the remote host. Copying files from your computer to the server:
`scp` ***`file1 file2 ...`***`username@csce.unl.edu:`***`filepath`*** copies the files from your local computer to the *filepath* on your account on the *csce.unl.edu* Linux server, where *filepath* is relative to your home directory. For example,
`scp answers.txt username@csce.unl.edu:.` copies *answers.txt* to the top-level of your home directory. Or, working the other direction:
`scp username@csce.unl.edu:`***`file filepath`*** copies *file* from the remote server to *filepath* on your local computer. Just as with `cp`, you can use the `-r` argument to copy directories:

```
scp -r pokerlab username@csce.unl.edu:.
scp -r username@csce.unl.edu:pokerlab .
```

2 Scenario

You're relaxing at your favorite hangout when another customer catches your attention. He's rather large (dare I say, *mammoth*), a bit hairy, and looking frustrated in front of his laptop. "I'm Archie," he says, "and I'm trying to teach myself this card game called *Poker*. I found this source code that I thought I could use to understand Poker better, but the code is incomplete, and I don't entirely understand what's there. Could you explain the code to me, please?"

3 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 Jack, 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 in Section 6.1), which are ranked according

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

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.

No Spaghetti Code Allowed

In the interest of keeping your code readable, you may *not* use any **`goto`** statements, nor may you use any **`break`** statements to exit from a loop, nor may you have any functions **`return`** from within a loop.

4 Getting Started

Download *pokerlab.zip* or *pokerlab.tar* from Canvas or ~cse231 on *csce.unl.edu* and copy it to your account on the *csce.unl.edu* Linux server. Once copied, unzip the file. The three source code files (*card.h*, *card.c*, *poker.c*) contain the starter code for this assignment, and the text file (*answers.txt*) is where you'll provide some answers to demonstrate your ability to understand part of the starter code. In the *equivalent-java* directory you will find *Card.java* and *Poker.java* that has Java code that is equivalent to the C code. You do not need to use the Java files, but you may find them useful as a reference to help you understand some of the differences between Java and C.

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 format specifiers you can put in the format string.

If you also get a warning for an unused variable

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

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

(Note that in future labs, we will provide a *Makefile* that can be used to build applications.)

Demonstrate that you can connect to your account on the *csce.unl.edu* Linux server

By whatever means you use to place files on your account on the *csce.unl.edu* Linux server, place *answers.txt* on your account on the *csce.unl.edu* Linux server. Open a secure shell terminal and navigate to the directory in which *answers.txt* is located. Type these commands:

`cat /etc/hostname`
The response should be ***csce.cs.unl.edu****. If it is* ***cse.unl.edu*** *then you connected to the wrong server. This will matter in two future lab assignments.*

`whoami`
This will print your School of Computing login ID.

`ls answers.txt`
The response should be ***answers.txt****. If it is* ***ls: cannot access 'answers.txt': No such file or directory*** *then you are not in the same directory as* answers.txt.

Take a screenshot and save the screenshot to submit to Canvas when you have completed the lab.

5 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 **create_card()** to populate a card's fields. Finally, change the first two lines of **display_card()** 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.

6 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 **populate_deck()**. Using **create_card()** 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 **`display_card()`**, to confirm that you wrote **`populate_deck()`** correctly.

6.1 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 **`is...()`** functions in *poker.c* are intentionally simple; they do not (and should not) check whether there is a better way to characterize the hand. 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. (Note also that a Royal Flush is not possible for this lab, based on our re-definition of a Straight, below.)

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 **`is_straight_flush()`**.

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

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 **`is_full_house()`**.

Flush Five cards all of the same suit. This characterization is checked by the function **`is_flush()`**. 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 **`is_straight()`**. 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 **`is_three_of_kind()`**.

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

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

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**.

6.2 Study the Code

Look at the code for **`is_pair()`**. 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 **`is_pair()`**. 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. Type your answer in *answers.txt*.

Look at the code for **`is_flush()`**. 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. Type your answer in *answers.txt*.

Look at the code for **`is_straight()`**. This is a little more challenging to understand than **`is_pair()`** and **`is_flush()`**. 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? If you can't determine this on your own, you may talk it over with other students or the TA. Type your answer in *answers.txt*.

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

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

6.3 Complete the code

Write the code in *poker.c*'s **`main()`** function to generate a hand of five cards by calling **`get_hand()`**.[3] (Uncomment the **`int`** `size_of_hand = 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] When you test the other functions you need to write, you might want to temporarily bypass **`get_hand()`** and explicitly assign specific cards to an array of five `card`s.

Now write the code for **`is_three_of_kind()`**, **`is_full_house()`**, and **`is_four_of_kind()`**.

7 Turn-in and Grading

When you have completed this assignment, upload *card.c*, *poker.c*, and *answers.txt* to Canvas.

This assignment is worth 10 points.

_____ **+1** The screenshot shows that the student has connected to your account on the *csce.unl.edu* Linux server and placed a copy of *answers.txt* there.

_____ **+1** The student's answers in *answers.txt* demonstrate an understanding of C's logical boolean operations.

_____ **+1** **`create_card`** populates a card's fields

_____ **+1** **`display_card`** generates the printable representation of a card

_____ **+1** **`populate_deck`** creates a deck of 52 cards

_____ **+1** **`is_three_of_kind`** determines whether a hand is a three of a kind

_____ **+1** **`is_full_house`** determines whether a hand is a full house

_____ **+1** **`is_four_of_kind`** determines whether a hand is a four of a kind

_____ **+2** **`main`** generates a hand of five cards, prints the hand, and determines the best-possible characterization of the hand

Penalties

_____ **-1** for each **`goto`** statement, **`break`** statement used to exit from a loop, or **`return`** statement that occurs within a loop.

Epilogue

Archie's face lights up in a very big smile. "Thanks!" After pausing in thought for a moment, he says, "Say, I've got a new startup company that could really use your help. Are you interested? It'll be exciting!"

To be continued...