CST8234 – C Programming F17 (Lab 3)

Programming Exercise

In this lab, we will gain experience with

- Using arrays
- Using random numbers
- Use of multiple source files, and a 'makefile'

Statement of the problem

We'll be building the foundation of a simple card game. But at this point, all we're going to do is shuffle a deck and deal the cards (i.e., no actual playing of any game!)

To start, you will ask the user how many players there are, and how many cards should be dealt out to the players. Then you'll randomly shuffle a standard deck of playing cards, deal the players' hands, and finally show what was dealt to each player.

Here are some sample executions.

```
$ ./lab3.exe
Enter the number of players: 2
Enter the number of cards per player: 5
Hand #1 is:
QS 4D KS KD QD
Hand #2 is:
AS 5D 9D 2C JH

$ ./lab3.exe
Enter the number of players: 3
Enter the number of cards per player: 16
Hand #1 is:
AH 7S QH 5S 10H 3D 3H QC 8S 6C 8H JC 9D 10S JH 10C
Hand #2 is:
KS 8D 2H QS 5C 6D JD JS AS 3C 9S 8C 5H 9H 2S 9C
Hand #3 is:
KH 4C 10D AD 7C KC KD 4H 7H 6S AC QD 7D 5D 6H 2D

$ ./lab3.exe
Enter the number of players: 4
Enter the number of cards per player: 16
Error: Invalid number of cards
```

Random Numbers

C has a random number function, '`rand()`'. Read up on it. C also has a random number "randomizer" that can ensure a unique random sequence, '`srand()`'. Read up on it.

Passing Multi-dimensional Arrays

One *possible* implementation of the solution is to use two-dimensional arrays, i.e., define

```
int hands[4][13];
```

where the first index represents the number of players (0-3), and the second index represents the number of cards (0-12).

When passing such arrays to a function, in the arguments you have to specify the second column's number of elements (i.e., 13) , but you can/should omit the first column's number of elements (i.e., 4).

```
int main() {
       int hands[4][13]; /* four hands, with 13 cards each */
       playGame( hands )
}
void playGame( int hands[][13] ) {        ←note the syntax!
       /* do stuff */
}
```

The logic behind this syntax will become more evident as we get into arrays in more depth, but for now, I don't want you to be tripping over this syntax issue.

Makefile

Makefiles have a flexible and inherently-complicated syntax! But they are really useful for managing multiple source files. For example I created the following simple '`makefile`' to build my solution to this lab.

```
CC = gcc
CC_FLAGS = -g -ansi -pedantic -Wall
FILES = main.c deal.c format.c      ←my source files... yours will likely be different!
OUT_EXE = lab3

build: $(FILES)
	$(CC) $(CC_FLAGS) -o $(OUT_EXE) $(FILES)

clean:
	rm -f *.o core *.exe *~

rebuild: clean build
```

With this file, I can just type '`make`' at the command line, and it will build lab3.exe. If I want to get rid of all but the source files (.c, .h), I can type '`make clean`'. If I ever decide to add a fourth file, all I have to do is change the definition of "`FILES`".

*Important hint: the white space in the makefile before the "`$(CC)` …" and "`rm` …" lines must be a **TAB** character. If you type in SPACES, your makefile will fail to run!*

Requirements

1. Create a folder called algonquinUserID_L3 (e.g., "mynam00123_L3"). Do all of your work in this folder, and when complete, submit the zipped folder as per the "Lab Instructions" posted on Blackboard.
2. Write a program that will ask the use for the number of players (min = 2, max = 4), and number of cards (any number, given the limit of cards in the deck), and
   a. Check the validity of the input, and report any problems and exit if not valid
   b. Generate a randomly shuffled deck, where the randomization is different from execution-to-execution. Your deck must contain ***all*** 52 cards (i.e., no duplicates, and no missing cards)
   c. Distribute the specified number of cards to the specified number of players' hands
   d. Print the hands in the format illustrated in the sample executions, e.g., the ace of hearts would be depicted by "AH", the ten of diamonds would be represented by "10D" and the queen of spades would be represented by "QS", etc.
3. You must distribute your functions in a meaningful manner across **at least three** .c files.
   a. One should be called main.c, and should only contain the 'main' function
   b. The others are at your discretion, but should represent a sensible grouping of functionality
   c. You must define .h files as appropriate, and use `#include`'s rather than manually typing in `extern` declarations.

Marking

This assignment is out of 40

- 10 for coding correctness (i.e., correct results)
- 10 for appropriate and efficient logic (i.e., no spaghetti code!)
- 10 for sensible distribution of well-contained functions between files
- 10 for clear comments and coding convention (not necessarily K&R, but it should be clean and consistent)

You can also lose marks for incorrect submission (e.g., including unnecessary files in the zipped folder), compiler warnings, typographic errors (including in comments).

Submission

When you are done, submit your program to Blackboard. Make sure that you have the appropriate header in your source file(s), and have zipped up the appropriately name directory. Only include the source code (.c, .h) and your makefile (mandatory!).

You ***must*** include a makefile, as part of your submission! When grading your reports I will unpack your zip file and type '`make`'… and if I don't end up with a 'lab3.exe' to run, ***I will not grade your assignment***.