CST8234 – C Programming W19 (Lab 2)

Programming Exercise

In this lab, you will gain experience with

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

You will work in pairs on this lab. If you don't have a partner, ask on the CST8234 Discord. Your partner must be from the same lab section as you.

The Application

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.

```
$ ./lab2.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
Remainder of deck:
3S AS 6D …  ← remainder of list omitted for brevity


$ ./lab2.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
Remainder of deck:
4S AH 2C …  ← remainder of list omitted for brevity
```

```
$ ./lab2.exe
Enter the number of players: 10
Enter the number of cards per player: 6
Error: Invalid number of cards
```

Representing Cards

Each card in the deck can be represented by an integer, 0..51.

You will need a formatting function that will convert the number of the card (i.e., 0..51) into a nice string (e.g., “2C”, “AD”, “10S”, “QH”). Note that the 10’s should be represented as “10” (two characters, plus suit), whereas all the other cards (e.g., “7” or “K”) can be represented by a single character, plus suit.

You will also need a function that can take a pointer to an integer, and then print N elements (e.g., each hand, or the remainder of the deck), where the pointer is pointing somewhere into the deck.

Shuffling Algorithm

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.

There are many possible shuffling algorithms, but this is one of the simplest and fastest.

```
Define an array of 52 cards
Initialize the array such that it contains the numbers 0..51 (sequentially)
Iterate over the array, swapping each element with a random other element.
```

Obviously, your deck may not contain duplicate cards, or be missing cards.

Your program must generate different results each time it is run.

Dealing

What does it mean to “deal” a hand of cards? In a fully-developed game, we would take sequentially take cards off the top of the deck and move them to a dynamic list of cards for each player. We’d have to maintain a different list for each player, since their hand might shrink or grow independently as the game unfolds. Similarly, the pile of remaining cards after dealing would be reduced by the cards distributed to the players’ lists.

But in this lab, it is permitted to simply say that the top group of C cards in the deck belong to player #1, the second group of C cards in the deck belong to player #2, the third group of C cards in the deck belong to player #3, etc.

Printing out the remainder of the deck then just involves printing whatever doesn’t belong to any of the P players.

Organizing your Code

You will have a function main.c, that will define only the main() function.

The other functions will be then be logically grouped and put into an appropriately named file. E.g., you may want to put all the deck and shuffling code into a file called "deck.c", and all the formatting code into a file called "format.c". You may also opt to put the prompts/input scanning in a file called "input.c". Those are just examples of how I might divide up my functionality into files.

Just having two files, with all non-main() functions stuffed into a single file won't get you full marks.

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 -std=c99 -pedantic -W -Wall
FILES = main.c deal.c format.c      ← my source files… yours will likely be different!
OUT_EXE = lab2

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 lab2.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 based on the user names of you and your partner (e.g., "smit9112_sing0003"). Do all of your work in this folder, and when complete, submit the zipped folder as per the "Lab Instructions" posted on Brightspace.
2. Write a program that will ask the use for the number of players (min = 1), and number of cards (min 1, and respecting 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

There is a rubric posted on Brightspace. You are strongly encouraged to read it to make sure you know what you can lose marks for.

Submission

When you are done, submit your program to Brightspace. 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 ‘lab2.exe’ to run, ***I will not grade your assignment***.