CST8234 – C Programming W18 (Lab 3)

Programming Exercise

The purpose of this lab is to

- Demonstrate that you can use proper scanf arguments to protect against buffer overruns
- Demonstrate that you can write a program that uses arrays, and does proper input validation

There are two parts to this lab, and the attached makefile allows you to build one of two programs.

Part 1

Download 'lab3a.c' and the attached 'makefile'.

Build the program by running "make build-3a"

Run the program that you just built (i.e., 'lab3a' or 'lab3a.exe').

The program will output a brief overview of a college course, and then prompt you, three times, to enter a new course code for this course. After each input, the program will display the modified course overview.

For your first input, enter a valid course code (e.g., "CST8234"). For your second input, enter an invalid course code (e.g., "CST12345678"), and note how the course overview has been messed up.

Edit 'lab3a.c', and fix the program by improving on how the course code is being scanned (make use of the lecture slide decks relating to scanf). It should be a one-line fix.

Re-build and re-run the program until it can safely accept course codes without messing up the resulting course overview.

Include the corrected 'lab3a.c' in your submission

Part 2

Write a program (e.g., 'lab3b.c') that shuffles a deck of cards, and deals out a number of hands.

You will prompt the user for the number of players, and the number of cards. The inputs must be valid (e.g., you can't specify -1 players, or 93 cards), and must protect against bad inputs (e.g., 'hello').

You will then shuffle a deck, and prompt the user to display one of the hands. Upon receipt of a valid player #, you will print out that player's hand in the ***exact*** format specified in the sample below. If the user enters '0', as the player's hand, then the program will exit. You may enter the same player # several times, and naturally, the output for that player must be the same each time.

Your program should generate a random deck, but that randomization will be identical for each execution of the program. Your deck should be implemented as a 1-D array.

Consider the following sample output, which shows two successive invocations of the program

```
$ ./lab3b.exe
How many players? 1
How many cards per player? 52
Who's hand do you want to see (or 0 to quit)? 1
Player 1: [ JC, 7C, 6D, 8S, 8C, KD, JS, 4C, JD, 10D, 5D, 7H, QD, 5H, QS …

Who's hand do you want to see (or 0 to quit)? 0

$ ./lab3b.exe
How many players? 3
How many cards per player? 5
Who's hand do you want to see (or 0 to quit)? 1
Player 1: [ JC, 8S, JS, 10D, QD ]

Who's hand do you want to see (or 0 to quit)? 2
Player 2: [ 7C, 8C, 4C, 5D, 5H ]

Who's hand do you want to see (or 0 to quit)? 3
Player 3: [ 6D, KD, JD, 7H, QS ]

Who's hand do you want to see (or 0 to quit)? 4
Invalid hand... Who's hand do you want to see (or 0 to quit)? 1
Player 1: [ JC, 8S, JS, 10D, QD ]

Who's hand do you want to see (or 0 to quit)?
```

By entering 1 player and 52 cards, you can see what your entire shuffled deck is.

When there are multiple players, the sequence of cards in each player's hand matches what would have been dealt out with a real shuffled deck. I.e., In a three-player game (as shown above), player 1 will get cards 0, 3, 6, 9, … of the deck, whereas player 3 will get cards 2, 5, 8, 11, … In the example above, indeed player 1 gets (JC, 8S, JS, 10D…) which correspond to cards 0, 3, 6, 9, … in the entire shuffled deck.

Randomization

To enable the identical sequence of random numbers with each invocation, you'll put the following line of code at the top of your main() function

```
    srand( <your-student-id> );
```

where <*your-student-id*> is your numeric student ID (e.g. 83500323).

Shuffling

You'll find that a good way to shuffle an array of N elements is to implement the following pseudo-code

```
for ( i = 0..N-1 )
{
    r = random number between 0..N-1
    Swap element-i and element-r
}
```

Special Restrictions

In an effort to force you to pass information via parameters, rather than relying on global variables, you are not permitted to define any global variables.

In an effort to force you to divide your code into logical blocks of code, rather than putting everything in main(), you may not have any functions (including main()) that exceed 25 lines of code (where a line of code is considered to be a definition or statement, including those separated by semicolons. Attempts to jam multiple statements onto a single line will be penalized, as the goal is to come up with multiple logically-sized functions . To illustrate, my solution has the following methods:

```
void swap( int cards[], int i, int j );
void shuffle( int cards[] );
void printHand( int cards[], int nPlayers, int nCardsPerPlayer, int iPlayer );
int getInput( char *strPrompt );
void getParameters( int *pnPlayers, int *pnCardsPerPlayer );
```

Your program doesn't have to use the same methods, but it gives you an idea of how I chose to compartmentalize my code into functions that contained 15 lines of code or less.

If you want to use multiple source files, you may, but you'll have to edit the 'makefile' to specify a different set of source files (rather than just lab3b.c)

You will shuffle the deck exactly once, and all hands will be retrieved from that deck, such that the cards retrieve correspond to how a real deck would be dealt (i.e., one card to each player, in succession, until the desired number of cards are dealt out).

Error checking

If the user types in a non-numeric argument (e.g., "hello") for any of the values, or enters an inappropriate numeric value, then you'll print out a brief error message and re-prompt.

Submission

When you are complete submit your programs to Blackboard.   This will include

But… before you do, please check that you've satisfied the following submission requirements

1. Did you make sure that you have a header comment in your source files (e.g., lab3a.c, lab3b.c)?
2. Did you confirm that your zipped submission is a ".zip" file (not a '.rar' or '.tar.gz' or '.7z' file)?
3. Did you zip up a folder that includes your *user name*, and the lab/assignment indicator?   I.e., if you open up your own zipped submission, you should see a *folder* called (for example) "smit9112_L1". If you just see *file(s)*, you've done it wrong, and you'll need to go to the *parent* folder, and try zipping your lab folder.
4. Did you remove all the unnecessary files from the folder contained in your zipped submission? I.e., you open up your own zipped submission, and click on the folder called (for example) "smit9112_L1", you should see *just your makefile and your source file (e.g., lab3a.c and lab3b.c)*.
5. Is your makefile actually called 'makefile' or 'Makefile' (without any filename extensions?)

If you don't satisfy submission requirements #2 or #3 you will get ZERO on this lab.   If you realize afterwards that you've made a mistake, don't panic! you are allowed to correct your mistake and re-submit.   But you ONLY get TWO submissions per lab/assignment, so try to make sure you double-checked everything *before* doing your first submission… and only use the second submission in case of emergency.