ECE/CSE 474
C-Programming Assignment 1
rev: December 2, 2021
C programming with structs and basic pointers

In this assignment you will write (or complete) C code to work with the standard deck of 52 playing cards. We will represent a card by two integers: $1 \ldots 13$ for the cards Ace ... King, and $1 \ldots 4$ for the suit, Hearts, Diamonds, Clubs, Spades.

Template You are given a C file, `c_asgn01_empty.c` which contains comments which indicate where to put each part of your code. There is already a `main()` which will run the different pieces of your code and `printf` statements which will separate the output from your different sections. The `main()` function also contains pre-written test code which will verify that your functions are working. Blank spaces for your functions are at the end of the file. **Please read the entire file before starting work so you understand where things go.** Please do not modify the test code. There is also a sample output file which shows you what the completed code should do.

Note that we are using the `rand()` function to generate random numbers. The random number generator is initialized with a "seed" number. The random number generator (and therefore your code) will always behave the same with the same seed. However, the random numbers for a given seed may change from one computer architecture or C compiler to the next.

Compiler Differences You may do this assignment on Linux or on Microsoft Visual C or any other C Compilers. There are a few small differences between C compilers. The sample code should compile on both linux (gnu gcc compiler) and Visual C if you set the appropriate `#define` statement near the top of the code file.

Printing Although we will study the `printf()` function and its formatting syntax later, at this point we will use some simplified wrapper functions for printing. Their names and function prototypes indicate which kind of data they can print:

```
void print_int(int);
void print_usi(unsigned int);
void print_newl();
void print_str(char*);
void print_dble(double);
```

Their code is given at the bottom of the file. From that you can figure out how `printf()` works and you may use `printf()` directly instead of these if you so desire.

1 C Basics

1.1 Looping

Write a short loop of code to print out the integers $1 \ldots 10$. Put this code in the `main()` routine where instructed in the comments. Put your variable declarations near the top of the file where the comment says:

```
/*********************************************************************
                All variable declarations
*********************************************************************/
```

Organize variable declarations by assignment part according to the comments in that section.

1.2 Summations

Modify 1.1 to print the sum and the sum of the squares at the bottom of the output. As before, insert your code directly in `main()`.

1.3 Text Output

Modify 1.2 so that the sum and sum of squares are labeled with text: "sum: " and "sum of squares: " on separate lines. For example:

```
sum: 27
sum of squares: 274
```

1.4 Text Manipulation

Write a function to make any string into a string of length n. If the string length is less than n, add spaces at the end. Do not modify the original string, instead create a new string. Use `malloc()` to assign memory for the new string. You may use the `strlen(char* string)` function (which comes from `#include <string.h>`) but NOT any other `string.h` functions.

If it is greater than n, truncate the string. The function should return a new string containing the modified text and leave the original string intact. The prototype of this function should be:

```
char * length_pad(char *string, int n);
```

Put this function *below* the `main()` function, were instructed with comments in the file.

Modify 1.3 to use `length_pad()` so that the numbers are all starting in col 21 (i.e. all card labels have length 20).

2 Card Games

A 'shuffle' is an array of 52 pairs of integers. The first of the pair is the card type (0-13 representing Ace, 2, 3, ... King) and the second representing the suit (hearts, diamonds, clubs, spades). Thus a pair of numbers describes a unique card in the deck. For example, {6,3} would describe the Six of Clubs, and {13,1} would be the King of Hearts.

2.1 Shuffling

Write a function (`void fill(shuffle[52][2])`) to fill a shuffle with 52 random integer pairs, BUT, as with your playing cards, there must be exactly one of each pair in the shuffle. Use your function to print out all the "cards" of the shuffle, 7 cards per line. Put brackets around each number pair so that the output is not so confusing.

Put your function below `main()` where instructed by comments.

2.2 Compact Data Representation

A 'hand' is an array of seven unsigned chars. Each char represents one card. We use a four bit field in the char for each of the two numbers above: the four most significant bits [bits 7...4] represent the card number (1-13) and the lower four [bits 3...0] represent the suit.

Write functions to:

a) convert two integers (from a shuffle for example) into a char as above.

```
unsigned char convert(int card, int suit);
```

b) get the integer suit from a char.

```
int gsuit(unsigned char card);
```

c) get the integer card from a char.

```
int gcard(unsigned char card);
```

All functions must print a warning message and return CARD_ERROR (already defined at top of file) if they get invalid input (such as suit > 4).

Write code which verifies these functions work properly and prints results.

3 Pointers and Array Manipulation

3.1

Create the array

```
char * card_names={"Ace", "2", "3", ...  ,"10", "Jack", "Queen", "King"};
```

and a similar array for the suits in the order hearts, diamonds, clubs, spades.

Write a function `names(int card, int suit, char answer[])` which places a string of the name and suit of a card in the array `answer[]`. For example:

`name(11,1)` → "Queen of Hearts"

`name(8,2)` → "8 of Diamonds".

hint: Use pointers to copy the characters one-by-one into the array `answer[]` to build up the final string.

3.2

Write a function to deal a hand of M (0<M<8) cards from a shuffle. Use a global variable `int dealer_deck_count` to keep track of how many cards have been dealt from the deck.

```
void deal(int M, unsigned char hand[7], shuffle deck[52][2]);
```

Keep track of which cards have been dealt. Test your deal function by dealing two hands of 7 cards from a shuffled deck. Use 2.2 and 3.1 to print out the dealt cards and the remaining deck.

3.3

The next step is evaluating the strength of the hands!

Deal three hands from a deck.

Write a function in `main()` which finds out if the hands contains any pairs, three-of-a-kinds, or 4 of a kinds.

3 points extra credit: Add code to find and print out all valid 5-card poker hands within a deal of 7 cards and to rank the hands from strongest to weakest.

For fun:

Once all your code is working, try to find a seed value which gives some player a cool hand like a full house. Can you find a seed value where no player gets anything?