CSE/ECE 474 C-Programming Assignment 1:
C programming with structs and pointers

April 1, 2024

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

Instructions

We have provided a template C file, **`c_prog1.c`** which contains comments which indicate where to put each part of your code as well as directions and sample outputs as well as a corresponding header file **`c_prog1.h`**. In addition to this we have provided **`c_prog1_arduino.ino`** which contains a set of function calls in the setup() function that will run a set of test cases. You do not need to modify or turn in **`c_prog1_arduino.ino`**. A sample output is provided in **`sample_output.txt`**.

Please read the entire file before starting work so you understand where things go. Please do not modify the test code. 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) should behave the same with the same seed. Note that the random numbers for a given seed may change from one computer architecture or C compiler to the next.

Running your code. This code is designed to run in the Arduino IDE which is available to download here: https://www.arduino.cc/en/software
Clicking the "Upload" icon will compile and upload your code to the Arduino board. The output will be displayed in the Serial monitor (Tools > Serial Monitor). The code will run once and will be repeated if you press the RESET button on the board.

If you need to run your code and do not have physical access to your arduino board you can use this online simulator Wokwi. This tool emulates the Arduino and prints the same output: https://wokwi.com/

For those with an existing C development environment you are welcome to use other tools as well (Visual Studio etc), however we will not go over this setup extensively in class and will evaluate the code based using the Arduino environment described above. To run this on your computer you will need to write a new **`main.c`** file with the testing code in **`setup()`** and redefine the printing functions at the bottom of **`c_prog1_arduino.ino`** to use **`printf`**.

Coding style: For this assignment you do NOT need to follow the coding style guidelines for the labs. All we ask is that you write your code in the "c_prog1.c" file and that code you write does not have compile errors or warnings. Warnings generated by our starter code (not a problem on most computers) are OK. Comments about how your code works can help us award partial credit if needed.

Printing. We have defined a set of 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);
```

The code for these functions is given at the bottom of **`c_prog1_arduino.ino`**.

1 C Basics

1.1 Looping

Write the **`count`** function that prints out integers one per line counting from 1… N. If you define additional variables please place these at the top of the file with a comment for the problem they are used for.

```
SAMPLE OUTPUT:
 1
 2
 3
 (continued)...
```

1.2 Summations

Write the **`sums_and_squares1`** function to print numbers counting up from 1…N as before but also print the sum and the sum of the squares at the bottom of the output.

```
SAMPLE OUTPUT:
 1
 2
 3
 (continued)...
 [sum(1,2,3 ... N)] [sum(1^2,2^2,3^2 ... N^2)]
```

1.3 Text Output

Write the **`sums_and_squares2`** function that prints the sum and sum of squares for numbers 1…N as before but are also labeled with text: “sum: ” and “sum of squares: ” on separate lines.

```
SAMPLE OUTPUT:
 sum: [sum(1,2,3 ... N)]
 sum of squares: [sum(1^2,2^2,3^2 ... N^2)]
```

1.4 Text Manipulation

Write a function **`length_pad`** 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 in **`st_buffer`**. Hint: look at the definition for **`st_buffer`** and modify it as needed. You may use the strlen(char* string) function (which comes from #include <string.h>) but NOT any other string.h functions.

If the length of the input is greater than N, truncate the string. The function should return a new string containing the modified text and leave the original string intact. Fill in the following function:

```
char* length_pad(char *st, char* st_buffer, int n);
```

Update the function **sums_and_squares3** as needed to use the **length_pad** to print the output such that the numbers are all aligned starting in col 21 (i.e. all values have length 20).

```
SAMPLE OUTPUT:
 sum:                  [sum(1,2,3 ... N)]
 sum of squares:       [sum(1^2,2^2,3^2 ... N^2)]
```

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 the 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 to make the output more readable. Note that the sample output below has some spaces truncated to fit on the page.

To generate a random number use the helper function **int randN(int n)** defined at the bottom of this file that returns a random integer between 1 and N.

```
SAMPLE OUTPUT (some spaces truncated):
[ 11 2 ]  [ 11 4 ]  [ 12 1 ]  [ 5 4 ]  [ 4 3 ]  [ 7 3 ]  [ 5 3 ]
[ 13 4 ]  [ 9 3 ]  [ 2  3 ]  [ 1 1 ]  [ 2 4 ]  [ 3 2 ]  [ 2 1 ]
[ 13  1 ]  [ 7 4 ]  [ 8 2 ]  [ 4 4 ]  [ 6 4 ]  [ 4 2 ]  [ 1 4 ]
[ 7 1 ]  [ 3 3 ]  [ 12 2 ]  [ 6  1 ]  [ 12  4 ]  [ 13 3 ]  [ 9 2 ]
[ 9 1 ]  [ 3 4 ]  [ 9 4 ]  [ 10 2 ]  [ 4 1 ]  [ 8 1 ]  [ 1 3 ]
[ 3 1 ]  [ 11 3 ]  [ 8 4 ]  [ 10 3 ]  [ 5 1 ]  [ 10 1 ]  [ 13 2 ]
[ 11 1 ]  [ 7 2 ]  [ 6 3 ]  [ 8 3 ]  [ 12 3 ]  [ 2 2 ]  [ 5 2 ]
[ 1 2 ]  [ 10 4 ]  [ 6 2 ]
```

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 the following functions:

a) **unsigned char convert(int card, int suit)** which converts two integers (from a shuffle for example) into a char as above.

b) **`int valid_card(unsigned char card)`** to test if a char equals a valid integer pair in the range of playing cards (numbers 1-13 and suits 1-4)
c) **`int gcard(unsigned char card)`** to get the integer suit from a char
d) **`int gsuit(unsigned char card)`** to get the integer card from a char

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).

3 Pointers and Array Manipulation

3.1 Displaying card names

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) → "Jack 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.
We have defined the following arrays at the top of the file:

```
card_names[]={"Ace","2","3","4","5","6","7","8","9","10","Jack","
Queen","King"};
suit_names[]={"Hearts","Diamonds","Clubs","Spades"};
```

```
Part 3.1 Test Results:
 >>>> 0 :  1  1 Ace of Hearts
 >>>> 1 :  1  2 Ace of Diamonds
 >>>> 2 :  1  3 Ace of Clubs
 >>>> 3 :  1  4 Ace of Spades
 >>>> 4 :  11  2 Jack of Diamonds
 >>>> 5 :  12  2 Queen of Diamonds
 >>>> 6 :  13  4 King of Spades
```

3.1 Dealing cards

Write a function **`void deal(int M, unsigned char hand[7], int deck[N_DECK][2])`** 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.
Test your deal function by dealing three hands of 7 cards from a shuffled deck. Also write a function **`printhand(int M, unsigned char* hand, char* buff1)`** to display the cards.

```
SAMPLE OUTPUT:
----testing deal: hand:  0
Deck count:  0
--------dealt hand:
the hand:
10 of Diamonds
King of Diamonds
7 of Clubs
```

```
5 of Diamonds
3 of Hearts
4 of Diamonds
7 of Hearts



----testing deal: hand:  1
Deck count:  7
--------dealt hand:
the hand:
7 of Spades
9 of Diamonds
10 of Hearts
5 of Hearts
3 of Clubs
6 of Hearts
Ace of Clubs
```

3.3 Evaluating hands

The next step is evaluating the strength of the hands! Deal three hands from a deck.

Write a the functions **`pairs`**, **`trip_s`** and **`four_kind`**, which finds out if the hands contains any pairs (two cards with the same number), three-of-a-kinds (three cards with the same number), or 4 of a kind (four cards with the same number) respectively.