CIS*2500 – Assignment 2

Question 1: Swapping Rows and Columns

1. Write a function, `double_array(int row, int col)`, which returns a pointer to a struct called `Double_Array`. The `Double_Array` struct holds an uninitialized 2 dimensional double array, the row size, and the column size (accessed using .array, .rowsize, .colsize respectively).

   Note:
   - *From this point on, all functions will be referred to by their parameter signature*
     - *i.e. the types of the parameters will be given, but not their variable names.*
   - *This gives you design control over the local variable names used in the function.*
   - *However, in some cases parameter names are provided in the question.*
     - *In these cases, please use them.*

2. Using the function `rand_double()` you used in Lab Assignment 2, write a function `randomize_array(struct Double_Array *, double, double)` that takes a Double_Array struct, and two values of type double and initializes each element of the array in Double_Array to a randomly generated value between the two input values (inclusive).

3. Create a function called `print_array()` that takes a `Double_Array struct` pointer and prints the array with each element displaying 1 decimal place, with the column elements lining up vertically.

4. Create a function called `free_array()` which takes a `Double_Array struct` pointer and frees the struct as well as the array within it.

5. Create a function called `swap_rows(struct Double_Array *, int, int)` where the two integer are row numbers to be swapped. Make sure each row number is valid according to the structure. If valid the rows are swapped and the function returns 1, otherwise the rows are not swapped and the function returns 0.

   Note: There are two possible approaches to swapping rows.
   One is faster to execute, and even to code.
   Bonus ½ mark if you implement the faster approach.
   Place your explanation as both a comment in the code and in your readme file as Q1a

6. Create a function called `swap_columns(struct Double_Array *, int, int)` where the two integer are column numbers to be swapped. Make sure each column number is valid according to the structure. If valid the columns are swapped and the function returns 1, otherwise the columns are not swapped and the function returns 0.

   Note: The "fast" approach from `swap_rows` cannot be applied here.
   Bonus ½ mark if you explain why.
   Place your explanation as both a comment in the code and in your readme file as Q1b.

7. Create a `main()` called `a2_q1.c` that performs the following:

   - Prints the following header
     ```
     ------------------------------------
             Question 1
     ------------------------------------
     ```
   - Creates a `Double_Array` struct that holds a 6 row by 9 column array
   - Randomly initializes the array in the struct
   - Prints out the array from the struct
   - Randomly pick two rows to swap, and swaps them
   - Inform the user which two rows were swapped and then print out the array
   - Randomly pick to columns to swap, and swap them
   - Inform the user which two columns were swapped and then print out the array
   - Print 3 blank lines

   Make sure you free all malloced memory before exiting the program.

Question 2: Pointer, Shallow and Deep Copy

1. Write a function `shallow_copy(struct Double_Array *)`
   that returns a copy of the `Double_Array` struct that is passed in (has a different pointer value) but holds the same content. Any change to the interior of either the new or the old struct should be reflected in the other automatically, even though the pointers to the two structs are different.

2. Write a function `deep_copy(struct Double_Array *)`
   that returns a copy of the Double_Array struct that is passed in (has a different pointer value) but holds the same content. Now, however, any change to the interior of one struct should **not** be reflected in the other.

3. Write a function `print_struct(struct Double_Array *, char *)` that prints out
   - a header string on the first line (the string that was passed in as the second parameter)
   - "struct address = %p" on the second line
   - "row_size = %d, col_size = %d" on the third line
   - "array address = %p, with contents:" on the fourth line
   - leave the fifth line blank
   - print the array from the sixth line on (printed in the same way you did in print_array from Q1)
   - after the array is printed, leave two blank lines

4. Create a `main()` called `a2_q2.c` that performs the following:

   Q2a
   - Print the following header

     Question 2a

   - Declare a variable called a1 that holds a Double_Array struct that contains a 6 x 9 array (as in Q1)
     note: a1 should hold a struct pointer, not a struct itself
   - Initialize a1 to random values between 0.0 and 10.0
   - Print the address of a1 using the printf command "the address of a1 is %p"
   - Print the structure pointed to by a1 with the header "The structure pointed to by a1 is:"
   - Declare a variable called a2 that can hold the same struct pointer as a1, and set a2 to a1
   - Print the address of a2 using the printf command "the address of a1 is %p"
   - Print the structure pointed to by a2 with the header "The structure pointed to by a2 is:"
   - Create a shallow copy of a1 and store it in a variable called a_shallow
   - Print the address of a_shallow using the printf command "the address of a_shallow is %p"
   - Print the structure pointed to by a2 with the header "The structure pointed to by a_shallow is:"
   - Create a deep copy of a1 and store it in a variable called a_deep
   - Print the address of a_deep using the printf command "the address of a_deep is %p"
   - Print the structure pointed to by a2 with the header "The structure pointed to by a_ deep is:"
   - Print 3 blank lines

Q2b

- Print the following header

  Question 2b

- In a1, set [0][1] to 100.0

  note: the above instruction, for the purpose of this assignment, is shorthand for the following: set the element at the 0^{th} row and 1^{st} column of the array in the struct held by a1 to the value 100.0

- In a2, set [1][2] to 200.0
- In a_shallow, set [2][3] to 300.0
- In a_deep, set [3][4] to 400.0
- Print a1, a2, a_shallow and a_deep
- Print 3 blank lines

Explain the results observed in the readme file. Label your answer as Q2b

Q2c

- Print the following header

  Question 2c

- Declare a variable called b1 that holds a Double_Array struct that contains a 6 x 9 array (as in Q2a)
- Initialize b1 to random values between 10.0 and 20.0
- Set `a2-> array = b1 -> a`
- Print a1, a2, a_shallow, a_deep, and b1

- In a1, set [0][1] to 5000.0
- In a2, set [1][2] to 6000.0
- In a_shallow, set [2][3] to 700.0
- In a_deep, set [3][4] to 8000.0
- In b1, set [4][5] to 9000.0
- Print a1, a2, a_shallow, a_deep, and b1

Explain the results observed in the readme file. Label your answer as Q2c

Q2d

- Free all malloc'd pointers. Make sure you do not have a memory leak.

In the readme file under Q2d explain which pointers need to be freed, in what order, and why the order is important (be specific).