CS 211 Homework 1

Jay Kania

Fall 2023

Introduction

This C programming assignment comprises three exercises designed to reinforce various fundamental concepts. Each exercise will be executed by discerning between the exercise name stated in the command line arguments.

```
./hw1 {pointers|structs|malloc}
```

Logistics

Your program must not utilize functions from any external C library. In other words, you must code all of the logic as per the standards set forth below. The libraries included in the skeleton code should be all that is required to complete a successful implementation.

You are able to create helper functions in order to complete the necessary implementations. However, ensure that the provided function signatures are the functions used for execution.

void nestedReverse(int** arr, int rows, int cols)

Write the code required to complete the implementation that will reverse the order of arrays in a 2D array as well as reverse the individual elements of each array inside the 2D array. You should modify the existing array in place – you will not return a value to be printed, but instead modify the existing variable by passing it by reference to the function (see skeleton code). Follow the steps outlined in main() to implement nestedReverse(). You must implement all logic yourself without any use of external library reverse functions. This is also the case with reading the command line arguments (do not use scanf() – we will check).

Input Format: Your program will accept the number of rows, the number of columns, and the elements to be inserted into the 2D array as a command line arguments. After parsing this data, your function will accept a double pointer reference to the 2D integer array, the number of rows as an integer, and the number of columns as an integer.

```
./hw1 pointers {rows} {cols} {elements}
```

```
./hw1 pointers 3 5 1 2 3 4 5 6 7 8 9 10 1 3 6 8 9
```

Number of Rows = 3, Number of Cols = 5, Elements: 1 2 3 4 5, 6 7 8 9 10, 1 3 6 8 9

Output Format: Your output will be a 2D array in which the order of the array has been reversed, along with each element of every array reversed. (**NOTE: The skeleton code already produces this format. Do not modify this code as it is required for the autograder.**

```
9 8 6 3 1
10 9 8 7 6
5 4 3 2 1
```

int deepCompare(student_t* students, int numStudents)

Write the code required to complete the implementation that will compare every possible combination pair of students in the students array. This comparison should compare all three attributes of the student struct. The function should return 1 when every student in the student array is exactly the same as every other student (duplicates). Otherwise, it will return 0. **NOTE: You are required to complete step 4 on line 161 to parse the GPA.**

Input Format: Your program will accept the number of students, and their respective data as command line arguments. After parsing, your function will accept a pointer of an array of student_t structs along with the total number of students in this array as an integer.

```
./hw1 structs {numStudents} {data}
```

```
./hw1 structs 3 1 aj 3.5 2 jay 4.0 3 zoe 2.5
```

Number of Students = 3

student_t {
    id: 1,
    name: “aj”
    gpa: 3.5
}
	student_t {
	    id: 2,
	    name: “jay”
	    gpa: 4.0
	}
		student_t {
		    id: 3,
		    name: “zoe”
		    gpa: 2.5
		}

Output Format: Based on your return value from your function, the output should print “Data is the same!” or “At least one student is different.” **NOTE: The skeleton code already produces this format. Do not modify this code as it is required for the autograder.**

```
At least one student is different!
```

void replaceNRandom(int* arr, int numToReplace)

Write the code required to complete the implementation that will replace the first N integers in an array with random numbers. This implementation slightly differs from the other functions as you will also need to code the groundwork to parse integers from the command line, check that the inputs are valid, load the array into memory, and perform any cleanup work (See L189 of skeleton).

In order to generate a random number, **you must use rand()**. Research its use and implement it in your function. There is no need to set a max limit of the random number to be returned.

ASSUME: numToReplace ≤ numElements

ASSUME: all elements from command line will be integers.

Input Format: Your program will accept the number of elements to be replaced, the total number of elements in the array, and the elements of the array. After parsing, your function will accept a pointer reference to the integer array, along with the number of elements to replace in this array as an integer.

```
./hw1 malloc {numToReplace} {numElements} {elements}
```

```
./hw1 malloc 2 5 1 2 3 4 5
```

Number to Replace = 2, Number of Elements = 5
Elements = [1, 2, 3, 4, 5]

Output Format: Your output will be an array that has had its first N elements replaced with a random number generated by rand(). (**NOTE: The skeleton code already produces this format. Do not modify this code as it is required for the autograder.**

```
441666038 469682187 3 4 5
```

Getting Started

You should first download and expand the provided files:

```
tar xvf hw1-provided.tar
```

You can also download these locally and then copy them to ilab using scp: **scp hw1-provided.tar yourNetID@kill.cs.rutgers.edu:~/cs211** (assuming you have a cs211 directory in your root directory).

NOTE: ALL COMPILATION, TESTING, AND EXECUTION MUST BE PERFORMED ON THE ILAB. This is compulsory to ensure a standardized testing and grading environment for all students.

To compile the program, you should use gcc:

```
gcc hw1.c -o hw1
```

And to run it you should:

```
./hw1 {pointers|structs|malloc}
```

Submission

ONLY SUBMIT hw1.c to Canvas. It is important that you only upload the completed C file. We will not accept a tar, zip, etc.

Execution Samples

Pointers

```
./hw1 pointers 4 7 8 6 3 4 5 2 1 3 1 0 2 8 6 1 0 4 2 6 3 2 2 9 6 3 1 4 7 8
```

```
8 7 4 1 3 6 9
2 2 3 6 2 4 0
1 6 8 2 0 1 3
1 2 5 4 3 6 8
```

```
./hw1 pointers 4 7 8 6 3 4 5 2 1 3 1 0 2 8 6 1 0 4 2 6 3 2 2 9 6 3 1 4 7 8
```

```
8 7 4 1 3 6 9
2 2 3 6 2 4 0
1 6 8 2 0 1 3
1 2 5 4 3 6 8
```

```
./hw1 pointers 6 6 5 0 4 8 0 8 8 6 0 0 5 6 9 3 1 0 1 5 0 9 3 3 9 4 5 6 6 4 8 3 9 5 9
5 0 8
```

```
8 0 5 9 5 9
3 8 4 6 6 5
4 9 3 3 9 0
5 1 0 1 3 9
6 5 0 0 6 8
8 0 8 4 0 5
```

```
./hw1 pointers 1 1 1
```

```
1
```

Structs

```
./hw1 structs 2 1 aj 3.5 1 aj 3.5
```

```
Data is the same!
```

```
./hw1 structs 5 1 aj 3.5 1 aj 3.5 1 aj 3.5 1 aj 3.5 1 aj 3.5
```

```
Data is the same!
```

```
./hw1 structs 6 1 aj 3.5 2 zoe 3.0 1 aj 3.5 1 aj 3.5 1 aj 3.5 1 aj 3.5
```

```
At least one student is different!
```

Malloc

```
./hw1 malloc 3 5 1 2 3 4 5
```

```
962913901 427303535 1986789340 4 5
```

```
/solution malloc 0 8 1 2 3 4 5 6 7 8
```

```
1 2 3 4 5 6 7 8
```

```
./hw1 malloc 4 4 0 0 0 0
```

```
68575226 239886508 233448575 194870288
```