EGR 112 Practice Exam

Guidelines

1. **This is an open book and notes exam.** You may use a textbook and notes. You may **not** use any electronic reference material.
2. **You must write your source code from scratch.** You may not copy code from another file or use another file as a starting point unless provided by the examiner. You must type all the source code yourself.
3. **All work is to be done independently.** You may not ask or talk to any other student during the exam. You may only ask the instructor to clarify the exam.
4. **Source code must be submitted in both electronic and hardcopy form.** A *twenty-minute warning* will be given towards the end of the exam period. At this time you must wrap up what you are doing, submit your source code (*only* your `.c` file and *not* your `.exe`, `.tds`, `.c~` or `.obj` files), and print out a copy of your source code.
5. **When creating your source code, you must follow the specified file naming convention** (filename: `yourname.c`).
6. **Source code will be graded on compilation, execution, structure, and documentation.**. The source code must compile without any warnings or errors (using the default compile options). The program must perform as described here. The source code must be properly documented and written in a well-structured manner. **Follow the directions exactly** – incorrect source name, using the wrong file pointer, skipping details, etc. may result in a non-passing score.
7. **Consideration for a passing score will only be given for code that compiles.** Complete the exam in stages. If you cannot complete the entire exam, turn in the last version that compiled and did something. **Save** backup copies as necessary. Write comments as you go so that you will always have a well written, although incomplete, program to turn in at the end of the exam.

Sample Rubric:

Part 1	10 pts
Part 2	20 pts
Part 3	30 pts
Part 4	20 pts
Comments and formatting of code	15 pts
Miscellaneous	5pts

Problem: You are to write a C program which includes the use of an array and 3 functions and follows the instructions below.

Follow these detailed instructions:

1. First, read this document in its entirety.
2. Include the usual (detailed) comment block including program name, author, date, inputs, outputs and description, followed by your preprocessor directives.
3. An outline (Skeleton-code) of the source code is shown below and you are REQUIRED to follow this outline. Insert appropriate `printf` statements where needed to get the desired output. The outline has four sections where you are to enter in your code and complete the program.
    a. **Part 1**: You are required to open a file for input and read the data in that file into an array (named data). Create the file yourself in notepad and call it `data.txt` so that it has the following numbers in order:
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
        Note: You may choose any of the following ways to open this file:
            1. Hardcode the filename into the `fopen` statement.
            2. Ask the user for a filename – dynamic filenames

    b. **Part 2**: You are required to write the function called `display` which accepts an array, and a parameter named size which indicates the size of the array. This function prints the array (using the following format. List ==> # # # # # #) to `stdout`. A sample output is shown below. Display the original data array.

    c. **Part 3**: You are required to write the function call `reverse` which accepts an array, and a parameter named size which indicates the size of the array. This function reverses the array. In other words, flip the array from front to back. Display the flipped array using the function `display` created in part 2. A sample output is shown below.

    d. **Part 4**: You are required to write the function call `removeZeros` which accepts an array, and a parameter named size which indicates the size of the array. This function removes the zeros from the array and compacts the array. The resulting array would be smaller than the original array assuming there were some zeros in the original. The function MUST return the size of the NEW COMPACTED array. **You are not allowed to create an extra integer array to solve this problem, i.e., only the array that was sent from the main function is to be manipulated.** Display the compacted array (using the function `display` created in part 2) AND indicate number of zeros. The number of zeros MUST be printed from `main`(). A sample output is below.

Turn in:

- Hard copy, i.e., print your source code.
- Hard copy of your output.
- Submit your **source code** electronically.

Input file: data.txt
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

Sample output:

```
Original Data:
List ==> 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

Flipped Data:
List ==> 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0

Data with zeros removed
List ==> 9 8 7 6 5 4 3 2 1 9 8 7 6 5 4 3 2 1

Number of zeros in original array = 3.
```

OUTLINE OF THE SOURCE CODE:

```
#include <stdio.h>
// include other header files that you need

void reverse (int data[], int size);
int removeZeros (int data[], int size);
void display (int data[], int size);

int main (void) {

      FILE *fp;
      int i, total, status;
      int data[100];

      //    Code for part 1 goes here, include ALL error checks.

      display(data,i);
      reverse (data, i);
      display(data,i);
      total = removeZeros(data,i);
      display(data, total);
      return 0;
}

// Code for part 2 goes here

// Code for part 3 goes here

// Code for part 4 goes here
```