Program Design Project 3

Problem 1 (50 points)

Write a C program to determine if the parity (even or odd) of each index matches the parity (even or odd) element at that index. The inputs are the length of an array of positive integers, and the elements the array. For example, the parities of the elements array [3 5 1] are [odd, odd, odd], the parities of the indices 0 1 2 are even odd even. The parities of the elements do not match the parities of the indices.

Example input/output:

```
Enter the length of the array: 7

Enter the elements of the array: 2 5 4 1 6 9 12

Output: parities of elements match parities of indices
```

1) Name your program *project3_parity.c.*
2) The program reads in the number of elements of the array, for example, 4, then read in the numbers in the array, for example, 3 6 8 9.
3) In the main function, declare the input array after reading in the number of elements of the array, then read in the elements.
4) The program determines if the parity (even or odd) of each index matches the parity (even or odd) element at that index and displays the result.

Problem 2 (50 points)

Write a program that removes k number of smallest element in an input array.

Example input/output:

```
Enter the number of elements in set A: 7

Enter the numbers in set A: 3 5 2 7 8 1 4

Enter the number of smallest elements to be removed: 4

Output:

5 7 8
```

The program should include the following functions:

```
int remove_smallest_numbers(int a[], int n, int k)
```

The function removes k number of smallest element in an input array a[] of length n and return the new actual length of the array after the removal, which is n-k.

The function removes the smallest element by shifting elements to the right of the smallest element and repeats it for k times.

For example, if the input array a contains [3, 6, 8, 2, 9, 4] of length 6, suppose k is 3.

The function will remove the smallest element, 2, at index 3, by shifting 9 and 4 to the left. The result is [**3, 6, 8, 9, 4**, 4], with an actual length of 5.

The function will then remove the smallest element, 3, at index 0, by shifting 6, 8, 9, and 4 to the left. The result is [**6, 8, 9, 4**, 4, 4], with an actual length of 4.

The function will then remove the smallest element, 4, at index 3, by shifting no element to the left. The result is [**6, 8, 9,** 4, 4, 4], with an actual length of 3.

The program should also include the following function, which is a helper function to the remove_smallest_numbers function.

```
int find_smallest_index(int a[], int n)
```

The function finds and returns the index of smallest element in an input array a[] of length n.

5) Name your program *`project3_remove_smallest.c.`*
6) The program will read in the number of elements of the array, for example, 4, then read in the numbers in the array, for example, 3 6 8 9.
7) In the main function, declare the input array after reading in the number of elements of the array, then read in the elements.
8) **Your program should only use one array in the main function and the `remove_smallest_numbers` function.**
9) In the main function, call `remove_smallest_numbers` function to remove k number of smallest element.
10) The main function should display the resulting array with length n-k.

Other requirements and submission:

1. Compile on ***student cluster (sc.rc.usf.edu)***:

gcc –Wall project3_parity.c

gcc –Wall project3_remove_smallest.c

2. Test your program with the shell script on the student cluster:

   ```
   chmod +x try_project3_parity
   ./try_project3_parity


   chmod +x try_project3_remove_smallest
   ./try_project3_remove_smallest
   ```

3. Download the programs (.c files) from student cluster and submit it on Canvas>Assignments.

Grading:

Total points: 100

1. A program that does not compile will result in a zero.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality requirement 80%

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **name**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. **Variable names** and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.

4. Use consistent **indentation** to emphasize block structure.
5. Full line comments inside function bodies should conform to the indentation of the code where they appear.
6. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **`#define PI 3.141592`**
7. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
8. Use either underscores or capitalization for compound names for variable: **`tot_vol`**, **`total_volumn`**, or **`totalVolumn`**.