Project 4, Program Design

1. (50 points) Suppose you are given an array of integers with a zero as one of elements. You want to split the array into two arrays, one containing the elements before zero and one containing the elements after zero.

Sample input/output #1:

```
Enter the length of the array: 7
Enter the elements of the array: 3 5 0 4 1 3 9

Output:
Array 1: 3 5
Array 2: 4 1 3 9
```

Sample input/output #2:

```
Enter the length of the array: 6
Enter the elements of the array: 0 6 13 4 1 7

Output:
Array 1:
Array 2: 6 13 4 1 7
```

1) Name your program `arrays1.c`
2) Include the `compute()` function to compute and return the length of the array 1 by searching for 0:

```
int compute(int *a, int n);
```

`a` represents the input array with length `n`. **The compute function should use pointer arithmetic– not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.**

3) In the main function, ask the user to enter the length of the input array, declare the input array, call the compute function to find the length of array 1 and calculate the lengths of array 2, declare array 1 and array 2 and store the elements, then display the two output arrays. Pointer arithmetic in NOT required in the main function.

2. (50 points) Write a program `arrays2.c` that checks if an integer array contains three identical consecutive elements. Assume the number of identical consecutive elements are no more than three if they exist.

```
Sample input/output #1:
Enter the length of the array: 11
Enter the elements of the array:  -12 0 4 2 2 2 36 7 7 7 43
Output: The array contains 2 of three identical consecutive
elements: 2 7

Sample input/output #2:
Enter the length of the array: 4
```

```
Enter the elements of the array:  -12 0 4 36
Output: The array does NOT contain identical consecutive
elements

Sample input/output #3:
Enter the length of the array: 11
Enter the elements of the array:  -12 0 4 2 2 2 36 7 7 43 43
Output: The array contains 1 of identical consecutive
elements: 2
```

1) Name your program `arrays2.c`
2) Include and implement the following function in the program. **Do not modify the function prototype.**

```
void search_three(int *a1, int *a2, int n, int *num_three);
```

The function takes two integer array parameter `a1` and `a2` of size `n`. `a1` is the input array. `a2` is the output array containing the integers that are in the three consecutive integers. `num_three` as a pointer variable, pointing to the variable for the total number of the three consecutive integers (also the actual size of the output array). For example, the total number of the three consecutive integers for sample input #1 is 2, 0 for sample input #2, and 1 for sample input #3.

The function should use pointer arithmetic – not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.

3) In the `main` function, ask the user to enter the size of the input array, declare the input array and output array of the same size, and then ask the user to enter the elements of the input array. The `main` function calls `search_three` function, and displays the result.

Before you submit

1. Compile both programs with –Wall. –Wall shows the warnings by the compiler. Be sure it compiles on ***student cluster*** with no errors and no warnings.

   gcc –Wall arrays1.c
   gcc –Wall arrays2.c

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

   chmod 600 arrays1.c
   chmod 600 arrays2.c

3. Test your fraction program with the shell scripts on Unix:

chmod +x try_arrays1
./try_arrays1

chmod +x try_arrays2
./try_arrays2

4. Submit *arrays1.c* and *arrays2.c* on Canvas.

Grading

Total points: 100 (50 points each problem)

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 80% (**Including functions implemented as required)**

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. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. 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.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: `tot_vol` or `total_volumn` is clearer than totalvolumn.