Project 4, Program Design

1. (50 points) **This program will be graded based on whether the required functionality were implemented correctly instead of whether it produces the correct output, for the functionality part (80% of the grade).**

Modify set_operations.c (Project 3, problem #1) so that it includes the following functions:

```
void set_difference(int *a, int *b, int n, int
*difference);

void set_complement(int *a, int n, int *complement);
```

set_difference function: it should find the difference of the set represented by array *a* and set represented by array *b* and store the result in the set represented by array *difference*.

set_complement function: it should find the complement of the set represented by array *a* store the result in the set represented by array *complement*.

Both functions 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.

Your program should be named set_operations2.c. It calls set_difference and set_complement functions in the main function. The main function should also display the result.

2. (50 points)
A vector is an ordered collection of values in mathematics. An array is a very straightforward way to implement a vector on a computer. Two vectors are multiplied on an entry-by-entry basis, e.g. (1, 2, 3) * (4, 5, 6) = (4, 10, 18).

Write a program that include the following functions. The functions should use pointer arithmetic (instead of array subscripting). In other words, eliminate the loop index variables and all use of the [] operator in the functions.

```
    void multi_vec (int *v1, int *v2, int *v3, int n);

    int comp_vec(int *v1, int *v2, int n);
```

The multi_vec function multiplies vectors v1 and v2 and stores the result in v3. n is the length of the vectors.

The comp_vec function compares v1 and v2, return 1 if vectors v1 and v2 are equal (their corresponding components are equal), and 0 otherwise. n is the length of the vectors.

In the main function, ask the user to enter the length of the vectors, declare two arrays with the length, read in the values for two vectors, and call the two functions to compute the multiplication and comparison of them. The main function should display the result.

```
Enter the length of the vectors: 5
Enter the first vector: 3 4 9 1 4
Enter the second vector: 5 7 2 6 8

Output:
The multiplication of the vectors is: 15 28 18 6 32
The vectors are not the same.
```

Before you submit:

1. Compile with –Wall. Be sure it compiles on ***student cluster*** with no errors and no warnings.

```
gcc –Wall set_operations2.c
gcc –Wall vector.c
```

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

```
chmod 600 set_operations2.c
chmod 600 vector.c
```

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

```
chmod +x try_set_operations
./try_set_operations
```

```
chmod +x try_vector
./try_vector
```

4. Submit set_operations2.c and vector.c on Canvas.

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%

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.