Project 4, Program Design

Problem 1 (50 points) Using pointers to process arrays

Write a C program that reads in two sets of numbers `A` and `B`, and calculates and print the difference of the sets $A - B$. $A - B$ is the set of elements that appear in A but not in B. The sets will be represented as arrays of `0`s and `1`s. That is, `a[i]!=0` if `i` is in the set, and `a[i] == 0` if it is not. For example, the array `a[5] = {0, 1, 1, 0, 1}` would represent the set `{1, 2, 4}` because `a[1]`, `a[2]`, and `a[4]` have the value `1`, and other elements of a are zeros. The values in the sets are restricted to the range `0…9`.

Example input/output:

```
Enter the number of elements in set A: 4
Enter the numbers in set A: 3 5 8 7
Enter the number of elements in set B: 3
Enter the numbers in set B: 7 5 9
Output:
The difference of set A and B is: 3 8
```

1) Name your program *`project4_sets1.c`*.
2) The program will read in the number of elements in the first set, then read in the numbers in the set. Then repeat for the second set.
3) **The sets are stored using arrays of 0s and 1s as described above.**
4) The program should include the following function. **Do not modify the function prototype.**

```
void find_set_difference(int *set_a, int *set_b, int n, int *set_difference);
```

`set_a` represents the input array for set A represented as an array of 0s and 1s of length `10`, and `set_b` represents the input array for set B represented as an array of 0s and 1s of length `10`, `n` is the length of arrays, which is 10. `set_difference` represents A - B, also of length 10. This function calculates A - B and store the result in `set_difference`.

This 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.

5) In the main function, call the `find_set_difference` function on the two sets and display the result.
6) Pointer arithmetic in NOT required in the main function.

Problem 2

(50 points) Add two functions to the program you wrote for problem 1:

1. Determine if two sets are mutually disjoint. Two sets are mutually disjoint if they do not share any common elements. For example, set A {*1, 2, 3}* and set B *{3, 2, 6, 7}* are not mutually disjoint. But set A {*1, 2, 3}* and set B *{4, 9, 6, 7}* are mutually disjoint.

2. Find the set of symmetric difference. The symmetric difference of two sets are the elements that are either in first set or the second set but not in both sets. For example, array a contains elements {*1, 2, 3}*, array b contains elements *{3, 2, 6, 7}*. The symmetric difference of A and B is {1, 6, 7}.

```
Enter the number of elements in set A: 4
Enter the numbers in set A: 3 5 8 7
Enter the number of elements in set B: 3
Enter the numbers in set B: 7 5 9
Output: The two sets are not mutually disjoint.
        The symmetric difference is 3 8 9
```

1) Name your program *project4_sets2.c.*
2) The program will read in the number of elements in the first set, then read in the numbers in the set. Then repeat for the second set.
3) **The sets are stored using arrays of 0s and 1s as described in problem 1.**
4) The program should include the following functions. **Do not modify the function prototype.**

```
int are_mutually_disjoint(int *set_a, int *set_b, int
n);
```

`set_a` represents the input array for set A represented as an array of 0s and 1s of length `10`, and `set_b` represents the input array for set B represented as an array of 0s and 1s of length `10`, `n` is the length of arrays, which is 10. This function returns 1 if the two sets are mutually disjoints, and returns 0 otherwise.

This 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.

```
void find_symmetric_difference (int *set_a, int
*set_b, int n, int *symmetric_difference);
```

`set_a` represents the input array for set A represented as an array of 0s and 1s of length `10,` and `set_b` represents the input array for set B represented as an array of 0s and 1s of length `10,` `n` is the length of arrays, which is 10. This function calculates the symmetric difference and store the result in `symmetric_difference.`

This 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.

5) In the main function, call the two functions explained above on the two sets and display the result.
6) Pointer arithmetic in NOT required in the main function.

Other requirements and submission:

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 project4_sets1.c
   gcc –Wall project4_sets2.c

2. Test your programs with the shell scripts on Unix:

   chmod +x try_project4_sets1
   ./try_project4_sets1

   chmod +x try_project4_sets2
   ./try_project4_sets2

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

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.