Project 3, Program Design

1. Write three functions for the attached program *sets_of_numbers.c*. The sets are represented as arrays of integers. Assume the sets has no more than 50 elements. The zeroth element of each array has a special meaning: it indicates the number of elements in the set. The set elements themselves will be located in array elements index: 1 through N. Read the program and add the following functionalities:

   1) Write a function that deletes a specific element from a set.

      ```
      void delete_set(int v, int a[]);
      ```

      Add function declaration of `delete_set` after the other function declarations. Uncomment the statements in main function that calls `delete_set`.

   2) Write a function that calculated the set difference of set a and b, store the result in set c. The set difference is the set of all elements that are in set a, but not in b.

      ```
      void set_difference(int a[], int b[], int c[]);
      ```

      Add function declaration of `set_difference` after the other function declarations. Uncomment the statements in main function that calls `set_difference`.

   3) Write a function that calculates the symmetric difference between two sets. Symmetric difference between two sets a and b is defined as the set of all elements in either a or b, but not both. Hint: you'll find `set_difference` and `union_set` useful.

      ```
      void sym_difference(int a[], int b[], int c[]);
      ```

      Add function declaration of `sym_difference` after the other function declarations. Uncomment the statements in main function that calls `sym_difference`.

2. Assume the + operator is not available. Write a program that takes two numbers as input and display the addition of them. The program should include a function `add (int n, int m)` that will add two numbers **using only recursion** and the increment and decrement operators. Either of the two numbers can be zero, positive, or negative.

   Hint: `add(n, m)= add(++n, --m)`, if m is positive, and `add(n, 0)=n`.

Before you submit

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

   gcc –Wall sets_of_numbers.c

   gcc –Wall addition.c

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

   chmod 600 sets_of_numbers.c

   chmod 600 addition.c

3. Test your programs with the shell script try_sets and try_addition on Unix:

   chmod +x try_sets

   ./try_sets

   chmod +x try_addition

   ./try_addition

4. Submit *sets_of_numbers.c* and *addition.c* on Canvas.

Grading

Total points: 100 (problem 1: 60 points; problem 2: 40 points)

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. 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 underscores to make compound names easier to read: `tot_vol` or `total_volumn` is clearer than totalvolumn.