## PROBLEM. Sets (contributed by N. R. Lim-Cheng) [Score: 55/50 → This means if you solve this problem perfectly, you can get 110 for Hands-on 1]

Many processes we do that involve collections of data, whether numbers or strings, involve set processing based in Math. In fact, looking for substring is also a form of set processing (intersection) that looks at the presence of consecutive sequence of characters. For this problem, the collection of data are strings. Your task is to create the following functions to generate set union, set intersection, and set difference in 1D array of strings and to generate the set where string key appears as a substring in the 1D array of strings. For this problem, **consider elements same only if they have the same case** (i.e., entries in small letters are not same to entries in capital letters). You may refer to the following link https://www.mathsisfun.com/sets/venn-diagrams.html to understand more about union, intersection, and difference.

The following subtasks are identified for you, but there are certain requirements and restrictions indicated to test your programming capabilities (also provided in the file templates).

(1) Search [5 pts]
- Implement function Search() that will perform a linear search on the elements in array A, starting from the first element. Should the key value be found, this function returns the index where it was first found. Otherwise, this function returns -1.

(2) Set Union [10 pts]
- Implement function setUnion() that will produce the union (combination) of unique values from the array A and array B. In math, union of sets will be the elements from both sets. This function returns the resulting total number of elements in array C. Do not assume that the arrays have same number of elements**. You are NOT allowed to use array indexing notation** in this function. You are required to meaningfully **use Search() as part of the solution** for this function.

(3) Set Difference [10 pts]
Implement function setDiff() that will produce the of difference of array A and array B. In math, the resulting set will contain those unique elements **in A that are not in B**. Do not assume that the arrays have same number of elements. This function returns the resulting total number of elements in array C. You are required to meaningfully **use Search() as part of the solution** for this function.

(4) Set Intersection [10 pts]
Implement the function intersect() that will produce the intersection of unique values from the array A and array B. In math, intersection of sets will be the **unique common elements only from both sets**. This function returns the resulting total number of elements in array C. Do not assume that the arrays have same number of elements. You are required to meaningfully **use Search() as part of the solution** for this function.

(5) Substring [20 pts]
Implement substringList() that will produce the collection of words from the 1D array B that has substrings equivalent to the string key. The resulting collection should be stored in array C and this should contain only the unique set of words. The function returns the resulting total number of elements in array C. For your reference, a substring is a contiguous sequence of characters within a string. As an example, the strings "app", "ppl", "apple", and "e" are among the substrings from the string "apple". But, strings "ale" are "elppa" are not substrings from string "apple".

Suppose array A contains the following, with nElemA = 6:

| | | | | | | |
|---|---|---|---|---|---|---|
| "act" | "now" | "before" | "it's" | "late" | "now" | |

Suppose array B contains the following, with nElemB = 7:

| | | | | | | |
|---|---|---|---|---|---|---|
| “it’s” | “today” | “now” | “or” | “forever” | “today” | “ACT” |

For the following example results, technically in Math, the sequence of contents do not matter. However, **for your solution**, it is required that the results **follow the same sequence**.

If setUnion() is called, array C should contain the following by the end of the function and function returns 9:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| “act” | “now” | “before” | “it’s” | “late” | “today” | “or” | “forever” | “ACT” | |

Referring to contents of arrays A and B above, the following should be the contents of array C by the end of the function setDiff() and the function returns :

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| “act” | “before” | “late” | | | | | | | |

Referring to contents of arrays A and B above, the following should be the contents of array C by the end of the function intersect() and the function returns 2:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| “now” | “it’s” | | | | | | | | |

Referring to contents of array B above and suppose key = “or”, the following should be the contents of array C by the end of the function substringList() and the function returns 2 :

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| “or” | “forever” | | | | | | | | |

## General Instructions and Restrictions:

1. Your solution should NOT include any input or output statements (e.g., scanf() or printf())
2. You are only allowed to use strlen(), strcmp(), strcat(), and strcpy() from string.h. No other functions from string.h can be used.
3. You are not allowed to include other header files in the templates, other than those indicated.
4. Contents of arrays A and B should not be modified after the function calls to the functions you are tasked to implement.
5. Your submitted file LASTNAME-sets.c should not contain main().

You are given five files for this problem:

**(1)** `sets.h` which contains the type declaration and function prototypes that will be used in the program. This file cannot be modified.
**(2)** `LASTNAME-sets.c` which contains some initial code that you’ll need to complete. Comments indicate the requirement and restrictions imposed for the solution. For this file, the student’s implementation should not include any scanf) or printf(). This file should not be modified to contain the main().
**(3)** `setMain.c` which contains the main() that can be used to test the requirement
**(4)** `INPUT.txt` contains all the sequence of [sample] inputs needed by setMain.c
**(5)** `EXPECTED.txt` contains the expected output of running the executable file generated from setMain.c and using the inputs from INPUT.txt. For ease in debugging and testing, it is recommended that the

student use input and output redirection. Assuming you have a.exe as the executable file from compiling setMain.c, to do input and output redirection, type the following in the command prompt (in the same folder where the a.exe and the INPUT.txt are):
**For Windows:** `a < INPUT.txt > LASTNAME-ACTUAL.txt`
**For Linux/Mac:** `./a < INPUT.txt > LASTNAME-ACTUAL.txt`

You can then open LASTNAME-ACTUAL.txt and compare the contents to EXPECTED.txt. Note that even the spacing should be exactly the same.

**DELIVERABLE:** Your C program source file **`LASTNAME-sets.c`** with your own last name as filename. For example, if your lastname is SANTOS, then the source file should be named as `SANTOS-sets.c`. Upload your source file in Canvas before the indicated submission time.

**TESTING & SCORING:**

- Your program will be compiled via gcc -Wall. Thus, for each function that does not compile successfully, the score for that function is 0.
- Your program will be tested by your instructor with other TEXT FILES (contents are different from the ones given to you) and with function calls of different parameter values.
- Full credit will be given for the function only if the student's implementation are all correct for all the test values used by the instructor during checking AND only if the student's implementation complied with the requirement and did not violate restrictions. Deductions will be given if not all test cases produce correct results OR if restrictions were not followed.