Overview: In this project, you will write a C program that reinforces programming fundamentals with respect to linked lists involving set operations.

Premise: Sett wants to learn more about sets. He’s asked if you could help him and, without a doubt, you are happy to help him. You decide to briefly brush up on your knowledge on set operations and theory by reading a book titled "Sets and You: Finding Your Inner Convergence" which gives you the following information:

Sets are a collection of unordered unique elements in which there are a few main operations that operate on sets. The four main set operations are union, intersection, difference, and complement.

For this project, you will be working with sets that follow alphabetical ordering using union, intersection, and symmetric difference which work as follows:

**Union**: Given two sets A and B, A ∪ B (A Union B) results in a set of distinct elements that belong to both, set A and set B.

For example, consider A = {1, 2, 3, 4} and B = {4, 5, 6, 7}, the union of A and B results in A ∪ B = {1, 2, 3, 4, 5, 6, 7}

**Intersection**: Given two sets A and B, A ∩ B (A Intersection B) results in a set of distinct elements that are common to both, set A and set B

For example, consider A = {1, 2, 3, 4} and B = {3, 4, 5, 7}, the intersection of A and B results in A ∩ B = {3, 4}

**Symmetric difference**: Given two sets A and B, A Δ B (A symmetric difference B) results in a set of distinct elements that are either of the sets A and B, but not in their intersection.

For example, consider A = {2, 6, 7, 9} and B = {2, 4, 6, 10}, the symmetric difference of A and B results in A Δ B = {4, 7, 9, 10}

You have decided to implement a C program to help Sett gain more knowledge about sets by representing the sets as linked lists.

**Restrictions: No global variables allowed**

**Requirements: Your program must have no memory leaks as verified by running Valgrind**

**Required functions:**

**Choose the appropriate parameters for each of the following functions.**

**void addNode(…)**

This function adds a node to a set

**void clear(…)**

This function clears out the contents of a set (making the set become an empty set)

**void print(…)**

This function prints out the contents of a set

**void printSets(…)**

This function prints out the contents of all the nonempty sets

**void copy(…)**

This function copies the contents of one set to another set

**void removeNode(…)**

This function will remove a node from a set

**void unionSets(…)**

This function takes the union of two sets and store the result into a destination set

**void intersection(…)**

This function takes the intersection of two sets and store the result into a destination set

**void symmetricdifference(…)**

This function takes the symmetric difference of two sets and store the result into a destination set.

**Running the program:**

First, ask the user to enter a number. This number will indicate the number of sets to create from which we will operate on using set operations.

Afterwards, your program will continuously print out a menu from which the user selects the different options. Upon making a selection, your program will perform the selected operation. If the user enters in anything other than a valid option, you will prompt for input until a valid prompt is given.

For the menu choices, you must print out the following choices from which the user can select from:

\*\*\*\*\* Menu Options \*\*\*\*\*

Add String: a or A

Remove String: r or R

Union: u or U

Intersection: i or I

Symmetric Difference: s or S

Copy: c or C

Clear: z or Z

Print Set: p

Print All Sets: P

Quit: q or Q

If the user enters in ‘a’ or ‘A’, you must:

Prompt for a set to add to, and prompt for a string that the user would like to add to the set. Afterwards, call addNode(…) with the appropriate arguments to add the string to the set.

If the user enters in ‘p’, you must:

Prompt for a set to print. Afterwards, call print(…) with the appropriate arguments to print out the set.

If the user enters in ‘P’, you must:

Call printSets(…) with the appropriate arguments to print out the contents of all the nonempty sets

If the user enters in ‘z’ or ‘Z’, you must:

Prompt for a set to clear. Afterwards, call clear(…) with the appropriate arguments to clear out the contents of the set.

If the user enters in ‘r’ or ‘R’, you must:

Prompt for a set to remove from, and the string to be removed. Afterwards, call removeNode(…) with the appropriate arguments to remove the string from the set.

If the user enters in ‘c’ or ‘C’, you must:

Prompt for a set to copy to, and a set to copy from. Afterwards, call copy(…) with the appropriate arguments to copy from the source set to the destination set.

If the user enters in ‘u’ or ‘U’, you must:

Prompt for a destination set, a first set to union, and a second set to union. Call unionSets(…) with the appropriate arguments to store the result of the first and second set into the destination set.

If the user enters in ‘I’ or ‘I’, you must:

Prompt the user for a destination set, a first set to intersect from, and a second set to intersect from. Afterwards, call intersection(…) with the appropriate arguments to store the results of the intersection of the first and second set into the destination set.

If the user enters in ‘s’ or ‘S”, you must:

Prompt the user for a destination set, a first set to take the symmetric difference from, and a second set to take the symmetric difference from. Afterwards, call symmetricdifference(…) with the appropriate arguments to store the results of the symmetric difference of the first and second set into the destination set.

Note that the destination set may be the same as the source set.

• Assume that the user input is a valid destination and source sets.

• Assume that the user will not enter in more than 50 characters for a string.

**Provided code:**

**To avoid duplicate strings in sets, use the following provided code for alphabetical ordering:**

int strcmpa(char \*s1, char \*s2){

while (\*s1 && tolower(\*s1) == tolower(\*s2)){

s1++; s2++;

}

return tolower(\*s1) - tolower(\*s2);

}