Section:	BCS-2B
	Assignment -1

Submission Instructions

Course:	Object Oriented Prc

(Please read the following instructions carefully before submitting the assignment).

1. Save your file with your Roll Number (format: xxL-xxxx_A1.cpp).
2. There should be only one cpp file submitted for this assignment. Do not submit multiple copies or empty files.
3. It should be clear that your assignment will **not get any credit** if:
   - The assignment is submitted **after the due date**.
   - Assignment is **plagiarized** from any source. (online/other students/etc.)

Important Notes:

I. There should not be any memory leakage in your program, otherwise it will result in negative marks.

II. Make separate functions for input and output. Your program must be as modular as possible. Your main() should be a sequence of function calls only.

III. You are not allowed to use global variables and go-to instruction.

IV. All allocations of pointers should be dynamic. Use of static arrays is prohibited. V. Remember to delete the allocated memory when it is no longer needed. VI. All the data will be given by the user. Do not submit a program that runs on hard coded inputs.

VII. Pass the pointers to functions instead of subscript notation [].

VIII. Make proper functions to solve the problems.

IX. Debug your code extensively to find errors/bugs. Test modules against the sample input given in the assignment.

X. Use of **"string"** datatype and all built-in functions like strcmp() and strtok(), etc., is strictly prohibited .

XI. Write readable, well-formatted and well-commented code.

XII. No matter how hard things seem, be confident and believe in your own abilities. XIII. Do not panic and remember to take breaks from your device screen while attempting this assignment!

Task 1: [Tokenization]

Taking input:

Get user input (a paragraph) using a char array (each sentence ends with either "." or "?").

You can declare a dynamic 1D character array large enough to store 2000 to 3000 characters (new int[3000]). Alternatively, you can read the data from a text file to save the input time.

Tokenization:

Your task is to Tokenize the words (here blank space should be treated as a delimiter) entered by the user and store these words in another 2D dynamic character array. This 2D array shall be our "data dictionary". Make sure there are no duplicates in the data dictionary, in case of multiple occurrences of the same word in the user input (hint: functions to search and match would make this task easier). Number of rows in the data dictionary should be equal to the unique words entered by the user and the number of columns in each row should be one greater than the word length (to store the null terminator).

Tokenize the entire paragraph and store all unique words in the data dictionary. Remember to not lose the original user input. We shall need it in later parts. "." and "?" are of course not unique words and hence, they should not be present in the data dictionary. **Printing:**

After tokenization, print the number of unique words in the original input, i.e., the number of rows of the data dictionary and print all the identified unique words in a readable manner.

Example#1 **Input:** I love to be a FASTIAN. Being a FASTIAN is love.	**Example#2** **Input:** Never gonna give you up? Never gonna let you down. //extra spaces are intentional
Output: Unique words: 8 I love to be A FASTIAN being is	**Output:** Unique words: 7 Never gonna give you up let down

Task 2: [Synonyms]

Initialization:

Now you need to create a 3D character array called "synonyms" equal to the size of unique words n in the data dictionary created in the previous task and initialize each index of "synonyms" with a null pointer.

Input:

Your program should now iterate for each word of "data dictionary" and ask the user whether he/she wants to store synonyms of that word or not. If the user is interested to store the synonym then first ask for the count of synonyms, i.e., how many synonyms he/she wants to store against that word. Get the count of synonyms from the user and allocate the required memory at that index of the 3D "synonyms" array. Now iterate to get the input synonym for that unique word of the data dictionary. To get the input for synonyms you can use a “temp 1D static character array”. Once you get the synonym from user in a “temp” character array you need to allocate the memory at that particular index of “3D synonym array” equal to the size (strlen+1) of “temp” character array then copy the data character by character and then prompt the user to enter the next synonym if any. Remember! The dynamic array allocated at a particular index of the 3D “synonyms” array for one synonym should not have extra indices with garbage value. You need to repeat this task for each word in the “data dictionary”.

(hint: you may want to store the number of synonyms against each word of the dictionary for future use).

Example#1 **Input:** The unique word “FASTIAN” in the dictionary.	**Example#2** **Input:** The unique word “Never” in the dictionary.
Output: Do you want to store synonyms for “FASTIAN”? (Y/N) : Y How many synonyms do you want to store for “FASTIAN” ? : 1 Enter synonym 1 : Pakistani Synonyms stored successfully! Proceed to next word*	**Output:** Do you want to store synonyms for “Never”? (Y/N) : Y How many synonyms do you want to store for “cake”? : 2 Enter synonym 1 : ever Enter synonym 2 : forever Synonyms stored successfully! Proceed to next word*

Task 3: [Matching Words In The Dictionary]

Input:

Prompt the user to enter another paragraph (same as in task 1). You can also use a .txt file as well for this task for convenience. We shall refer to this input as “p2”.

Searching:

Iterate through p2 and identify each unique word in p2 separated by white spaces and search up that word in the data dictionary. Your searching algorithm should not be case sensitive, i.e., “FASTIAN”, “FASTian” and “fastian” are the same word and hence should match if present in p2 as well as the data dictionary. If that word is not found in the data dictionary, then append the dictionary to add that word as well.

Example#1 **Input:** I am a proud fastian. Fastian are the smartest people in pakistan.
Output: **Print all the words present in the dictionary after including new unique words of p2 as done at the end of task 1.**

Task 4: [Replacing Synonyms]

Replacing:

Now modify the searching algorithm of task 3 to also allow the user to replace any word in p2 with its synonyms (if present in “synonyms”). If any unique word of p2 matches with a word in “data dictionary” and also happens to have one or more synonyms stored against it in “synonyms” then ask the user if he/she wishes to replace that particular instance of the word with one of its synonyms (Y/N). if the user selects “Y”, then offer the user a menu of all options to replace the word with (all the synonyms stored “synonyms”). Now replace that particular instance of the said word in p2 with the selected user choice and proceed to the next word.

Printing:

Now print the updated p2. Also print the updated data dictionary alongside it (after addition of the new words of p2).

Example#1

Input: I am a proud fastian. Fastian are the smartest people in pakistan.

Output:

```
“fastian” is found in the dictionary. Do you wish to replace it? (Y/N) : Y
Enter choice: (enter -1 to not replace the word)
   1) pakistani
: 1
“fastian” was replaced with “pakistani” successfully!
“Fastian” is found in the dictionary. Do you wish to replace it? (Y/N) : N //for the 2nd instance of

“Fastian” Updated input: I am a proud pakistani. Fastian are the smartest people in pakistan.
```

****BONUS: A user-friendly interface will land you bonus points!!!****