**TASK1#**

**Paraphrasing Application**

Paraphrasing is the process of rewording a text, often done for simplification or clarity. Implement a simple C++ console-based application for paraphrasing purposes, in the following manner.

1. Read a file that contains a list of words and synonyms, where each synonym is separated by a single white space character, as in the following example:

|  |
| --- |
| abandon discontinue vacate absent missing unavailable cable wire  calculate compute determine measure safety security refuge |

Consider use of parallel arrays as well as (dynamically allocated) array of arrays for storing the synonyms, once the file is read. For instance, the above file may be represented in the memory in the following manner

**words array synonyms array of array**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| abandon | 0  1  2  3  4 | ptr0 | →  →  →  →  → | discontinue | vacate |  |
| absent | ptr1 | missing | unavailable |
| cable | ptr2 | wire |  |
| calculate | ptr3 | compute | determine | measure |
| safety | ptr4 | security | refuge |  |

0

1

2

3

4

Where, ptr0, ptr1, etc. are pointers assigned at run-time.

1. Take a text as input from the user and tokenize it into words.
2. For each word (after step 2) that exists in the list of synonyms loaded (in step-1), replace it with corresponding synonym (selected randomly in case of multiple options) **4.** Produce the output as paraphrased text

# Task#2

Write a program that creates 2D dynamic array in main function where the columns in each row should be of different length and both values should be positive. Create a function **fillArray**. This function should receive the 2D array from main function and prompt the user to provide data. Your program should only accept positive values to fill the array. Decide the remaining parameters and return type of this function at your own. Create a function **twoDimToOneDim**. This function should receive the 2D array from main function and creates a dynamic 1D array long enough to store the data of 2D array into this 1D array. This function should return the address of dynamically created 1D array to main function. Decide the remaining parameters of this function at your own. Create a function **SortArr**. This function should receive the 1D array from main function and sort its data in ascending order. Decide the remaining parameters and return type at your own. Create a function **showArr**. This function should receive the sorted 1D array and display its contents of console. Make sure that this function should not update the contents of array.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sample Output:**  Enter the size of rows: 3  Enter the columns for row#0: 4  Enter the columns for row#1: 2  Enter the columns for row#2: 3    Assume that following data is stored in 2D array:   |  |  |  |  | | --- | --- | --- | --- | | 87 | 61 | 92 | 14 | | 56 | 29 |  |  | | 5 | 78 | 45 |     Contents of Sorted 1D array are: 5, 14, 29, 45, 56, 61, 78, 87, 92 |