**Assignment: 2**

Write an object-oriented program to create a dictionary. The dictionary has 26 files (files). You may name the files (tables) as A.txt, B.txt,..., Z.txt. Your dictionary class should have at least the following attributes and methods:

**#include<iostream>**

**#include<string>**

**#include<vector>**

**using namespace std;**

**class Dictionary**

**{**

**private:**

**const int maxWordsInDict;**

**const int maxWordsPerFile;**

**int totalWordsInDict;**

**int numOfWordsInFile[26];**

**static bool failure;**

**static bool success;**

**public:**

**Dictionary();**

**Dictionary(int dictMaxWords, int fileMaxWords);**

**bool AddAWord(string word);**

**bool DeleteAWord(string word);**

**bool SearchForWord(string word);**

**bool PrintAFileInDict(string filename);**

**bool SpellChecking(string fileName);**

**bool InsertWordsIntoDict(string fileName);**

**void ProcessTransactionFile();**

**};**

**bool Dictionary::failure = false;**

**bool Dictionary::success = true;**

**Dictionary::Dictionary()**

**{**

/\* This routine sets the following values to the following attributes

maxWordsInDict = 10000;

maxWordsPerFile = 100;

totalWordsInDict = 0;

numOfWordsInFile[0] = 0; referring to A.txt

numOfWordsInFile[1] = 0; referring to B.txt

numOfWordsInFile[25] = 0; referring to Z.txt

\*/

**}**

**//------------------------------------------------------------------**

**Dictionary::Dictionary(int dictMaxWords, int fileMaxWords)**

**{**

/\* This routine sets the following values to the following attributes

maxWordsInDict = dictMaxWords;

maxWordsPerFile = fileMaxWords;

totalWordsInDict = 0;

numOfWordsInFile[0] = 0; referring to A.txt

numOfWordsInFile[1] = 0; referring to B.txt

numOfWordsInFile[25] = 0; referring to Z.txt

\*/

**}**

**//------------------------------------------------------------------**

**bool AddAWord(string word)**

**{**

/\* This routine opens the appropriate file, (If the first letter of the word is A, it should be in A.txt, or if the first letter of the word starts with B it should be in B.txt, and so on) and tries to add the word that is passed to this function into that file.

Before adding the word to the file, you should call the search function to ensure that the word is not already in the appropriate file

If you could add a word successfully into the appropriate file, you should increment the attribute TotalWordsInDict by 1 as long as you do not exceed the maxWordsInDict.

Further, if you add a word to A.txt you need to increment numOfWordsInFile[0] by 1. Similarly if you add a word to B.txt you need to increment numOfWordsInFile[1] and so on. Incrimination should be done as long as you do not exceed the maximum value.

If the word that you are trying to add, could not be added into the appropriate file for any reason, this routine should return failure

return (Dictionary::failure)

otherwise, if the word is successfully added, this routine should return success

return (Dictionary::success)

**HINT:  suppose the word you try to add is  "adam". The word "adam" should be added to A.txt. Therefore, you need to open "A.txt" first. To do that you can do the following**

**string fileName = "#.txt"; // all files should be have ".txt" extension**

**fileName[0] = toupper(word[0]);      // replaces the # sign with appropriate letter. Note that word is "adam" and. Letter 'a' is the first letter of the word and should be changed to upper case**

**ofstream fout;**

**fout.open(fileName.data(), ios::app);  // ios::app means appending the word to the file rather than overwriting the old information in the file**

**fout << word << endl;**

**\*/**

**}**

**//------------------------------------------------------------------**

**bool DeleteAWord(string word)**

**{**

/\* This routine opens the appropriate file where the word that you are looking for might be kept. (If the first letter of the word is A, it should be in A.txt, or if the first letter of the word starts with B it should be in B.txt, and so on. Then it places all the words into a vector and looks for the word in the vector. If the word is in the vector, it should be deleted and then the modified vector should be inserted back into the appropriate file

If you could remove a word successfully from the appropriate file, you should decrement the attribute TotalWordsInDict by 1.

Further, if you remove a word from A.txt you need to decrement numOfWordsInFile[0] by 1. Similarly if you remove a word from B.txt you need to decrement numOfWordsInFile[1] and so on.

If the word that you are trying to delete is not in the appropriate file, this routine should return failure

return (Dictionary::failure)

otherwise, when the word is successfully deleted, this routine should return success

return (Dictionary::success)

Note that before placing anything into the vector; it is a good idea to call the search function to ensure that the word is in the appropriate file

**\*/**

**}**

**//------------------------------------------------------------------**

**bool SearchForWord(string word)**

**{**

/\* This routine opens the appropriate file where the word that you are looking for might be kept. (If the first letter of the word is A, it should be in A.txt, or if the first letter of the word starts with B it should be in B.txt, and so on. If the word cannot be found in the appropriate file, this routine returns failure

return (Dictionary::failure)

otherwise, if the word does exist in the appropriate file, it returns success

return (Dictionary::success)

\*/

**}**

**//------------------------------------------------------------------**

**bool PrintAFileInDict(string filename)**

**{**

/\* This routine opens the file fileName, read the words one by one and print them on the screen. You should only print 5 words per line.

If the fileName could not be opened, this routine returns failure

return (Dictionary::failure)

otherwise, it returns success

return (Dictionary::success)

\*/

**}**

**//------------------------------------------------------------------**

**bool SpellChecking(string fileName)**

**{**

/\* This routine opens the file fileName, read the words one by one and does the spell checking of the words. Every word is searched and those that are not in the dictionary are reported on the screen

If the fileName could not be opened, this routine returns failure

return (Dictionary::failure)

otherwise, it returns success

return (Dictionary::success)

\*/

**}**

**//------------------------------------------------------------------**

**bool InsertWordsIntoDict(string fileName)**

**{**

/\* This routine opens the file fileName, read the words one by one and try to insert them into the dictionary. Note that before inserting a word into the dictionary, you should search the dictionary to ensure that the word is not there.

If the fileName could not be opened, this routine returns failure

return (Dictionary::failure)

otherwise, it returns success

return (Dictionary::success)

\*/

**}**

**//------------------------------------------------------------------**

**void ProcessTransactionFile()**

**{**

/\* In this routine, you need to ask the user to enter a transaction file. A transaction file has two information per line. The first information is a command telling the programmer what to do. The second information is the data associated with the command. For example, a possible transaction file may contain

AddW student

DeleteW play

SearchW medical

PrintF Q.txt

SpellCheck myEssay.txt

InsertF data.txt

....

....

....

AddW should add the word "student" into the dictionary. You should call the AddAWord(string word) function to add the word "student" into the dictionary

DeleteW should remove the word "play" from the dictionary. You should call the DeleteAWord(string word) function to remove the word "play" from the dictionary

SearchW should search for the word "medical" in the dictionary. You should call the SearchForWord(string word) function to search for the word "medical" in the dictionary

PrintF should print all the words in a particular fie on the screen. You should call the function PrintAFileInDict(string fileName) to print the content of file Q.txt on the screen

SpellCheck should do the spell checking of a file. All errors should be reported on the screen. You should call the function SpellChecking(string fileName) to do the spell checking of the file called "myEssay.txt"

InsertF should process the content of a file and insert word by word in the file into the dictionary. You should call the function InsertWordsIntoDict(string fileName) to add the content of the file called "data.txt" into the dictionary

Note that if the transaction file does not exist, you need to print appropriate failure message on the screen

**\*/**

**}**

**//------------------------------------------------------------------**

**int main()**

**{**

**Dictionary DictForCS211;**

**DictForCS211.ProcessTransactionFile();**

**return 0;**

**}**

/\*

Other Comments:

1) Do not worry about punctuations. Your program should only consider words and ignore all punctuations.

2) All words should be in lower case before it is placed in the dictionary

3) Do not write 26 statements to open/close A.txt to Z.txt files.

4) When a file name is a variable, you need to open it as follows:

fin.open(fileName.data());

\*/

***Place the following in your Transaction file***

AddW student

AddW teacher

AddW cs211

InsertF data.txt

PrintF S.txt

DeleteW student

AddW hello

SpellCheck myEssay.txt

DeleteW play

SearchW medical

SearchW cs211

PrintF C.txt

PrintF H.txt

PrintF T.txt

***Place the following in your data.txt file***

Current approaches to enhancing concurrency in database systems have been focused on developing new transaction models that typically demand changes to either the atomicity consistency or isolation properties of the transactions themselves Indeed much of this work has been successful but most of these attempts suffer from being either computationally infeasible or requiring the transaction designer be sufficiently knowledgeable to determine a priori how the application semantics must interface with the transaction model

Our approach exploits an object-oriented world but is different than these others in that the transaction designer is not expected to have intimate application knowledge this load is placed on the transaction system which must be able to determine by static analysis how to maintain database consistency This paper describes an overall architectural model to facilitate multiversion objects that are explicitly designed to enhance concurrency Within the context of concurrency the key aspects addressed by this paper are as follows

First A transaction model is introduced to show how transactions are managed in our multiversion environment

Second A new correctness criterion is described that emits more histories than conflict serializability and is computationally tractable

Third an architectural model is developed to support multiversioning that provides the well-known ACID transaction properties These build a framework for the implementation of an optimistic concurrency control algorithm Further they exploit related issues required to generate reconciliation procedures which have an important role for providing a reliable system

***Place the following in your myEssay.txt file***

Concrrent approaches to enhancing concurrency in database systems have been focused on developing new transaction models that typically demand changes to either the atomicity consistency or isolation properties of the transactions themeselves Indeed much of this work has been successful but most of these attempts suffer from being either computationally infeasible or requiring the transaction designer be sufficiently knowledgeable to determine a priori how the application semantics must interface with the transaction model

Our approach exploits an object-oriented world but is diferent than these others in that the transaction designer is not expected to have intimate application knowledge this load is placed on the transaction system wich must be able to determine by static analyziz how to maintain database consistency This paper describes an overall architectural model to facilitate multiversion objects that are explicitly designed to enhanc concurrency Within the context of concurrency the key aspects addressed by this paper are as follows

First A transactian model is introduced to show how transactions are managed in our multiversion environment

Second A new correctness criterion is described that emits more histories than conflict serializability and is computationally tractable

Third an architectural model is developed to support multiversioning that provides the well-known ACID transaction properties These build a framework for the implementation of an optimistic concurrency control algorithm Further they exploit relatted issues required to generate reconciliation procedures which have an important role for providing a reliable system

What to hand in:

* A hard copy of your source code
* A hard copy of all output files except the 26 files that you used for dictionary

Email me the source code only. Place the source code (all .h and .cpp files) into a folder, call the folder based on your first name and your last name (ex: JohnSmith-A2), zip the folder and email me the zipped file.