**Overview**

**You may complete this task using any development environment you wish. You will need .NET 6 installed and the MSTest test framework package installed.**

Please complete the implementation of the TextSearcher class. In skeleton, the class looks like this:

public class TextSearcher

{

public TextSearcher(string *filePath*)

{

Data = System.IO.File.ReadAllText(filePath,

System.Text.Encoding.UTF8);

// file data is in Data variable

// YOUR CODE HERE

}

public string[] Search(string *queryWord*, int *contextWords*)

{

throw new NotImplementedException();

// return a list of the match contexts

// YOUR CODE HERE

}

public string Data { get; set; }

}

TextSearcher implements a search interface to an underlying text file, consisting of the single method Search(string,int). This method takes in a single query word, and returns all substrings found in the file consisting of the query word and a few words of context, given by contextWords, from either side of the query word.

For example, given an input file containing this text:

I will here give a brief sketch of the progress of opinion on the Origin of Species. Until recently the great majority of naturalists believed that species were immutable productions, and had been separately created. This view has been ably maintained by many authors. Some few naturalists, on the other hand, have believed that species undergo modification, and that the existing forms of life are the descendants by true generation of pre existing forms.

Consider this code fragment:

string file = "files/short\_excerpt.txt";

TextSearcher searcher = new TextSearcher(file);

string[] results = searcher.Search("naturalists", 3);

The results array will contain these two strings:

* “great majority of naturalists believed that species”
* “authors. Some few naturalists, on the other”

Note that each result string has 3 words to each side of the query word, “naturalists”, because contextWords was specified as 3. Also note that the result strings should be returned exactly as they appear in the document, including original punctuation and capitalization.

**Programming Environment**

You have been provided with a skeletal version of TextSearcher.cs. Comments in the source code provide additional information.

Unit tests using the MSTest test framework are provided which test the requirements on the next page. However, be aware that the tests do not test all requirements and you should add additional tests if needed. Get as many of the tests to pass as you can. You may use any development environment.

You may modify the tests, but we will be testing your code against the original tests.

A utility class is provided, Tokenizer, which you may find useful for processing the file. See the comments in the source code for details. You may use other third-party packages if you wish. Be sure to include everything with your solution.

**Performance Notes**

* Performance counts in the solution. The tests are each executed 1000 times and timeouts are set on the tests. If a test times out, it is a test failure and you will need to optimize your code to complete the test within the timeout.
* For the purposes of this test, it is safe to assume that the contents of the underlying file can be held in memory, i.e. it is not necessary to read the file in chunks or page it in and out. However, your solution should be able to handle multi gigabyte files in an efficient manner.
* You should assume that many searches will be done over the same document. A single instance of the TextSearcher class will be created once for each file, and then the search method will be called many times. You should build an implementation that is efficient for this usage pattern (the search method should be as fast as possible).

**Additional Requirements**

Look these over before you start coding:

* You should write “production” quality code.
* The query word must match the definition of a “word” as defined below or it should be rejected.
* A “word” consists of any sequence of the characters a-z, A-Z, 0-9, and apostrophe. Therefore, “animal’s” is a single word, as are “1844” and “xxxxx10x”. Any non-word character or sequence of characters is regarded as a delimiter and will be ignored by the search (but included in the results).
* Search should be case-insensitive. In the above example, the query words “species”, “SPECIES”, and “SpEcIeS” should all return the same three hits.
* If the query word occurs too close to the beginning or end of the file to permit the requested number of context words, start the result string at the beginning or end it at the end of the file as appropriate. So, search("forms",2) will include “pre existing forms.” as one of its hits.
* Strings should be returned in document order.
* Overlapping hits require no special treatment; each should be returned as a separate hit.
* The search method should never return null. If no hits are found it should return an empty array.