Data Structures

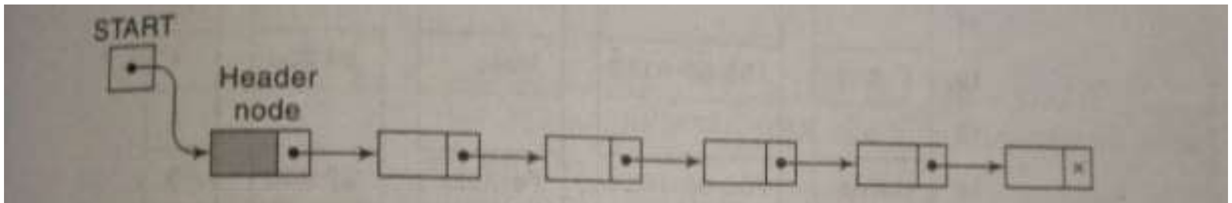
Assumptions

Reading in a text document per line requires some specific assumptions, which will eventually assist one to adequately come up with an algorithm which clearly illustrates how the process is undertaken. First assumption is that the beginning of the linked lists will remain the front and points to the first element of the text line in the queue. Secondly, is that every time an elements is added to the list, it is added at the end of the queue and the number of elements is added by one; for instance, **End = End + 1**.

Third assumption is that whenever there is deletion of elements from the list, the lists elements at the end of the queue are subtracted by one from the rear. For instance, **Rear = Rear - 1**. Lastly is that the initial or the beginning location, LOC, of the queue is where the end points out to the last entry location of the list.

Data Structure Chosen

The linked list data structure is best suited for this kind of program to implement reading in of text document line by line. To be precise, the grounded header list type of linked list. This is because of the unidirectional nature of the grounded header list kind of the graph, which is unidirectional. The text lines in a document are always unidirectional and therefore this data structure model is best suited for this scenario. Another reason for this choice is that the grounded list utilize space economically.



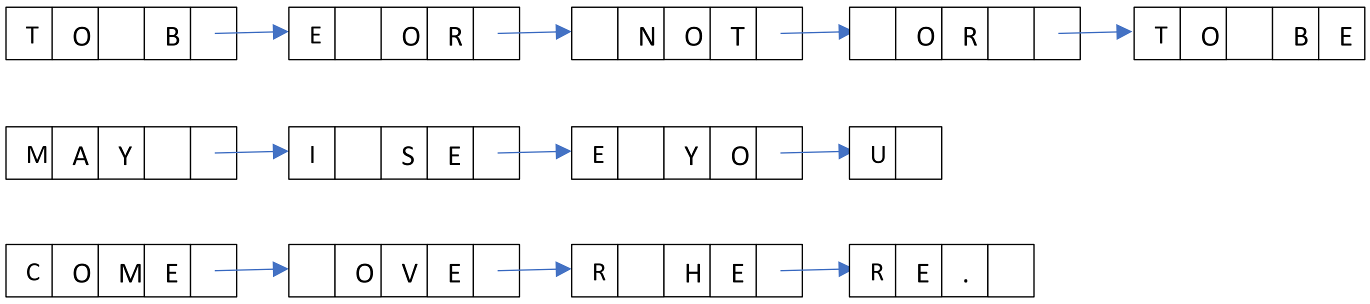
(a) Grounded header list

Code

```
#include <iostream>
#include <fstream>

using namespace std;
int main(int argc, char *argv[])
{
      ifstream in("test.txt");
      ofstream outfile("encfile.txt");

if(!in) {
cout << "Cannot open input file.\n";
return 1;
}
      char str[10000];
      char encStr[10000];
      while(in) {
      in.getline(str, 10000); // delim defaults to '\n'

if(in)
{
      int i=0;
      cout << str << endl;
      for(i=0;str[i]!='\0';i++)

//first we find ascii value by (int)str[i] and then
// adding 4 to it then again converting it to charecter
//by type casting it to (char)
encStr[i] = (char)((int)str[i] + 6);
//add a empty backslash zero to end the charecters array
encStr[i]='\0';
outfile << encStr << '\n';
}
}
```

The grounded header list data structure shown above will help solve the question at hand as shown in the examples shown below. The example will have three lines with a maximum of six words per line. The lines are as shown below:

1. TO BE OR NOT TO BE
2. MAY I SEE YOU
3. COME OVER HERE.



From the above scenarios it is evident that the strings of text in each line are stored in sets of linked list with fixed four elements per data. The data the white spaces represent the spaces between words and therefore, it is possible to read each word in each line considering the link arrow pointing to where the next letter is found (Hardiyana et al., 2020). This is despite the number of elements allowed per node being fixed.

Justification of the choice

Linked lists are normally used in situations where the time is paramount. They are first and in situations like reading in and identifying specific words in text documents, time is crucial. Another main reason why I chose linked lists for this task is that we may not know how many elements are in every line alike in many other data structure models which have predefined number of elements.

Operations

1. To print all the line numbers on which a given word occurs, the linked list data structure utilizes indexing, which refers to finding the position where the string pattern first appears, given by:

   ```
   INDEX (text.pattern)
   ```
2. This data structure works by first initializing the node pointer i.e. the current head and performing a set of operation when the head is not null. The operations include

checking every word in the list and if there is one equivalent to the search key it prints “TRUE” else “FALSE”

3. The linked list initializes the count to zero and loops through the lines to check the occurrence of a specific word and when it finds one it changes the count to one and so forth. This takes place until the end of the list and the last count number printed becomes the total number of occurrences of the specified word.
4. The linked list operation to print all the words that occur in a line, involves use of count of the head and key of the line elements. Whenever the head is not null the contents are extracted and the word listed and the next element is proceeded to until the end of the list.

References

Hardiyana, B., Fadilah, L., & Effendi, D. (2020, July). Application of Linked List Algorithm Based on Multimedia. In IOP Conference Series: Materials Science and Engineering (Vol. 879, No. 1, p. 012087). IOP Publishing.