Faculty of Engineering Technology

Department of Electrical and Computer Engineering

Please fill this form accurately and annex it to the first page of your assignment at the point indicated on the marked assignment.

Student Registration No	**519220940**	**OFFICE USE ONLY**
Student Name	**P.W.N. Hansaka**	
Course Code	**EEX4465**	
Course Name	**Data Structures and Algorithms**	
Mini project/~~Case study or TMA Title~~	**Mini Project**	
Mini project	**1**	
Due Date	**11/09/2021**	

Please write your name & Address on the dotted lines below

From: Bachelor of Software Engineering program,
Dept. of Electrical and Computer Engineering

the Open University of Sri Lanka

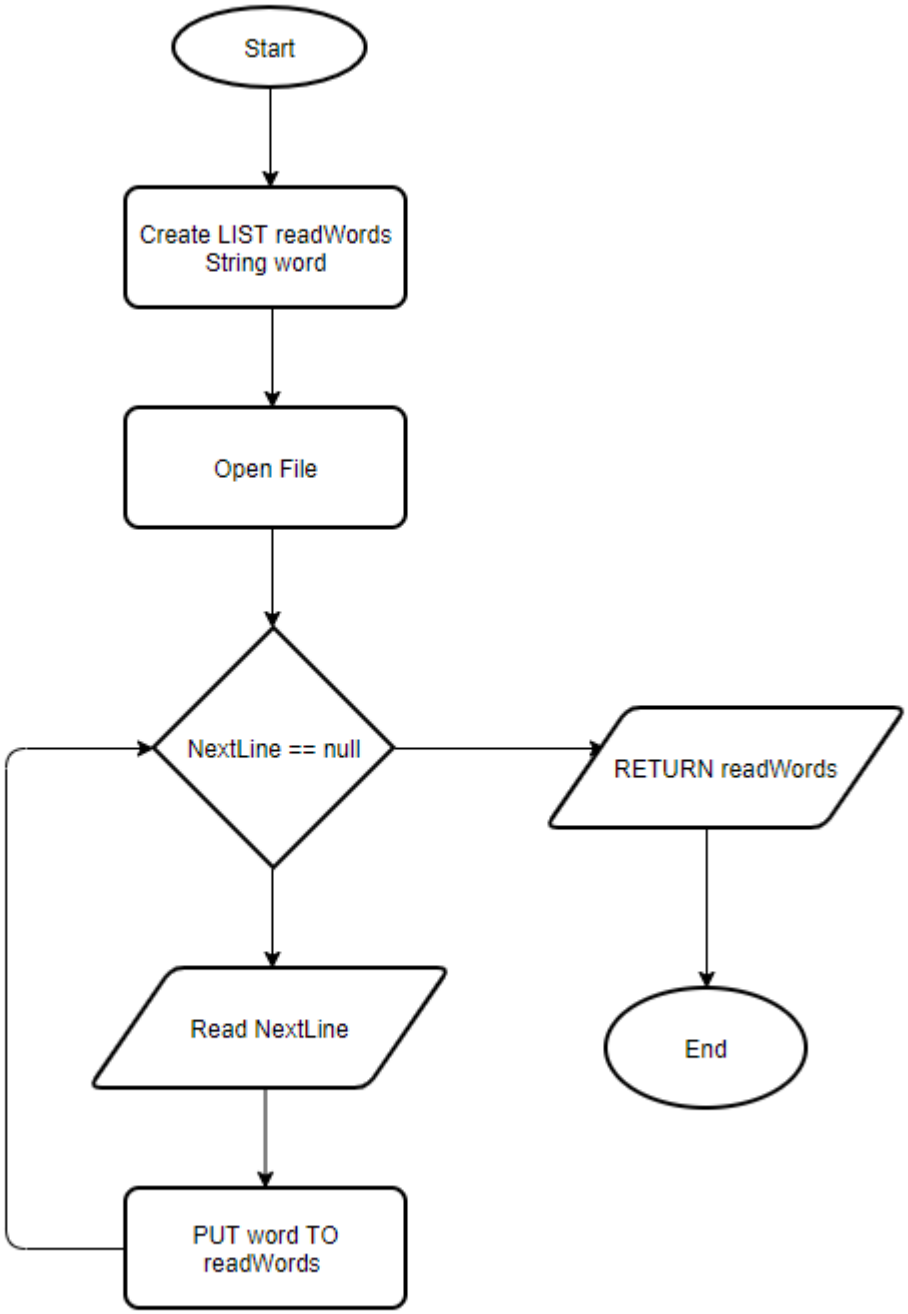
P. O. Box 21

Nawala, Nugegoda

To: 534/A, Makola North,

Makola.

A)


Start
Create LIST readWords
String word
Open File
NextLine == null
RETURN readWords
Read NextLine
End
PUT word TO
readWords

B) i)

```
BEGIN fileReader()
        GET List readWords[]
        OPENFILE kywrdsOdd.txt FOR READ
        WHILE NOT End Of File kywrdsOdd.txt
                READLINE IN kywrdsOdd.txt as word
                ADD word TO readWords
        END WHILE
        CLOSE FILE kywrdsOdd.txt
        RETURN readWords
END
```

B) ii)

```
BEGIN letterFrequencyCounter()
        SET List words = fileReader()
        GET HashMap (key, value) lettersValue
        GET count
        FOR w = 0 TO length of words
                SET word = GET STRING in words w position
                FOR i = 0 TO length of word
                        IF i == 0 OR i == length of word – 1 THEN
                                IF lettersValue NOT contains character IN word i position as key THEN
                                        PUT CHARACTER in word i position as Key, 1 as Value TO
lettersValue
                                ELSE IF lettersValue contains character IN word i position as key THEN
                                        SET count = GET(Key= CHARACTER IN word i position)
VALUE FROM lettersValue
                                        INCREMENT count
                                        REPLACE( Key = CHARACTER in word i position) Value as
count IN lettersValue
                                END IF
                        END IF
                END FOR
        END FOR
        RETURN lettersValue
END

BEGIN wordValueCalculator()
        SET HashMap(key, value) charactersValue = letterFrequencyCounter()
        SET List words = fileReader()
        GET HashMap(key, value) wordValues
        GET firstLetterValue
        GET lastLetterValue
        GET totalLetterValue
        FOR w = 0 TO length of words
                SET word = GET STRING in words w position
                firstLetterValue = GET(Key = CHARACTER in word 0 position ) Value FROM
keyWordsCount
                lastLetterValues = GET(Key = CHARACTER in word length of word -1 position ) Value
FROM keyWordsCount
                totalLetterValue = firstLetterValue + lastLetterValue
                PUT word as Key, totalLetterValue as Value TO wordsValue
        END FOR
        RETURN wordValues
```

```
END


BEGIN mapSortByValue()

        SET LinkedHashMap(Key, Value) sortedMap = sort by Decending order wordsValueCalculator()
END

BEGIN dataStructureConverter()
        SET LIST sortedWords[] = PUT Keys FROM mapSortByValue()



BEGIN setLetterFrequencyZero()
        SET HashMap(Key, Value) letterValues = letterFrequencyCounter()
        REPLACE ALL Values LINK TO Keys AS 0
        RETURN letterValues
END

BEGIN hValueFinder(GET word, GET HashMap(Key, Value) characterValues, GET LIST
wordsArray[], GET LIST sortedWordList[])
        SET gFirstValue = GET(Key = CHARACTER in word 0 position) Value FROM characterValues
        SET gLastValue = GET(Key = CHARACTER in word length of word -1 position) Value FROM
characterValues
        SET wordLength = length of word
        SET hValue = (wordLength + gFirstValue + gLastValue) % 10
        IF wordsArray[hValue] != null AND gFirstValue <=4 THEN
                INCREMENT gFirstValue
                REPLACE(Key = = CHARACTER in word 0 position) Value as gFirstValue IN
hValueFinder
                hValue =  hValueFinder(word, characterValues, wordsArray[], sortedWordList[])
        ELSE IF wordsArray[hValue] != null AND gFirstValue > 4 THEN
                SET previousIndex = GET index of word -1 IN sortedWordList
                SET previousElement =  GET INDEX OF previousIndex IN sortedWordList
                PUT CHARACTER in word 0 position as Key, gFirstValue++ as Value
                hValue =  hValueFinder(word, characterValues, wordsArray[],  sortedWordList[])
                IF wordsArray[hValue] != null AND gFirstValue <= 4 THEN
                        PUT CHARACTER in previousElement 0 position as Key, gFirstValue++ as
        Value
                ELSE
                        SET wordsArray[hValue] = previousElement

                END IF
        END IF
        RETURN hValue
END

BEGIN hashTableGenerator()
        GET HashTable(Key, Value) perfectHashTable
        SET LIST sortedArrayList[] = dataStructureConverter()
        GET LIST wordsArray []
        FOR i = 0 TO length of sortedArrayList – 1
                SET hValue = hValueFinder(word, setLetterFrequencyZero(), wordsArray,
sortedArrayList)
        wordsArray[hValue] = word
        PUT hValue as Key, word as Value TO perfectHashTable
```

```
        END FOR
        RETURN perfectHashTable
END


B) iii)

BEGIN keyWordsCounter()
        GET HashTable(Key, Value) keyWordCount
        SET HashMap(Key, Value) keyWords = mapSortByValue()
        SET wordCount = 0
        OPENFILE tstOdd.txt FOR READ
        WHILE NOT End Of File tstOdd.txt
                READLINE IN tstOdd.txt as words
                        SET List textLine[] = words
                        FOR i = o TO length of textLine
                                IF keyWords contains textLine[i] as key THEN
                                        IF keyWordsCount contains textLine[i] as key THEN
                                                PUT textLine[i] as Key, 1 as Value TO keyWordsCount
                                        ELSE IF keyWordsCount NOT contains textLine[i] as Key THEN
                                                SET count = GET(textLine[i]) Value FROM
keyWordsCount
                                                INCREMNT count
                                                REPLACE(Key = textLine[i]) Value as count  IN
keyWordsCount
                                        END IF
                                END IF
                        END FOR
        END WHILE
END

B) iv)

BEGIN timeCalculator()
        SET startTime
        GET fileReader()
        GET letterFrequencyCounter()
        GET wordValueCalculator()
        GET mapSortByValue()
        GET dataStructureConverter()
        GET setLetterFrequencyZero()
        GET hashTableGenerator()
        SET finishedTime
        SET totalTime = finishedTime – startTime
        keyWordsCounter()
        RETURN totalTime
END
```

C)

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class FileHandle {

    //Read key words from kywrdsOdd file and return it as a Array List
    public ArrayList<String> fileReader(){
        ArrayList<String> readWords = new ArrayList<String>();
        try {
            File kywrdsOdd = new File("D:\\STUDY\\L4\\EEX4465\\mp\\EEX4465-
2020_MP\\kywrdsOdd.txt");
            Scanner fileScan = new Scanner(kywrdsOdd);
            while (fileScan.hasNextLine()){
                String textWord = fileScan.nextLine();
                readWords.add(textWord);
            }
            fileScan.close();
        }
        catch (FileNotFoundException e){
            System.out.println(e.getMessage());
        }
        return  readWords;
    }
    // Calculate frequency of first letter and last letter of the keywords and return
the values as a hashmap
    public HashMap<Character, Integer> letterFrequencyCounter(){
        HashMap<Character, Integer> lettersValue = new HashMap<Character, Integer>();
        ArrayList<String> words = fileReader();
        int count;
        for (int w=0; w<words.size(); w++){
            String word = words.get(w);
            for (int i=0; i<word.length(); i++) {
               if (i==0 || i== word.length()-1){
                   if(!lettersValue.containsKey(word.charAt(i))){
                       lettersValue.put(word.charAt(i),1);
                   }
                   else if(lettersValue.containsKey(word.charAt(i))) {
                       count = lettersValue.get(word.charAt(i));
                       count = count +1;
                       lettersValue.replace(word.charAt(i), count);
                   }
               }
            }
        }
        return lettersValue;
    }
```

```
//Calculate the word value by sum of first letter value and last letter value of each
keyword and return it as a hashmap
public HashMap<String,Integer> wordValueCalculator(){

    HashMap<Character,Integer> characterValues = letterFrequencyCounter();
    ArrayList<String> words = fileReader();
    HashMap<String, Integer> wordValues = new HashMap<String, Integer>();
    int firstLetterValue, lastLetterValue, totalLetterValue;

    for (int w=0; w<words.size(); w++){
        String word = words.get(w);
         firstLetterValue = characterValues.get(word.charAt(0));
         lastLetterValue = characterValues.get(word.charAt(word.length()-1));
         totalLetterValue = firstLetterValue + lastLetterValue;
         wordValues.put(word, totalLetterValue);
    }
    return wordValues;
```

```
    //Key -> keyword | value -> sum of first letter value last letter value
}

public HashMap<String, Integer> mapSortByValue(){
    // Sort key words values by descending oder and put those values in to linked
hash map and return it
    HashMap<String, Integer> unSortedMap = wordValueCalculator();
    LinkedHashMap<String, Integer> sortedMap = new LinkedHashMap<>();

    unSortedMap.entrySet()
            .stream()
            .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
            .forEachOrdered(x -> sortedMap.put(x.getKey(), x.getValue()));
    return sortedMap;
}


public ArrayList<String> dataStructureConverter(){
    // Put keyword that sorted in to descending oder in to array list
    HashMap<String,Integer> sortedMap = mapSortByValue();
    Set<String> keyValues = sortedMap.keySet();
    ArrayList<String> sortedWords = new ArrayList<>(keyValues);
    return sortedWords;
}
public HashMap<Character, Integer> setLetterFrequencyZero(){
    //  Set all character values as zero and return it for hash table calculation
    HashMap<Character, Integer> letterValues = letterFrequencyCounter();
    letterValues.replaceAll((key, value) -> 0);
    return letterValues;
}
public Hashtable<Integer, String> hashTableGenerator(){
    // Create Perfect Hash Table
    Hashtable<Integer, String> perfectHashTable = new Hashtable<>();
    ArrayList<String>sortedArrayList = dataStructureConverter();
    String[] wordsArray = new String[sortedArrayList.size()];
    for (String word: sortedArrayList){
        int hValue = hValueFinder(word, setLetterFrequencyZero(),wordsArray,
sortedArrayList);
        wordsArray[hValue] = word;
        perfectHashTable.put(hValue, word);
    }
    return perfectHashTable;
}

public int hValueFinder(String word, HashMap<Character, Integer>characterValues,
String[] wordsArray, ArrayList<String>sortedWordList){
    //Find hash Value in each key word and return it

    int gFirstValue = characterValues.get(word.charAt(0));
    int gLastValue = characterValues.get(word.charAt(word.length()-1));
    int wordLength = word.length();
    int hValue = (wordLength + gFirstValue + gLastValue)%10;

    if (wordsArray[hValue] !=null && gFirstValue<=4){
        gFirstValue++;
        characterValues.replace(word.charAt(0),gFirstValue);
        hValue = hValueFinder(word, characterValues, wordsArray,sortedWordList);
    }
    else if (wordsArray[hValue] != null && gFirstValue>4){
        int previousIndex = sortedWordList.indexOf(word)-1;
        try {
            String previousElement = sortedWordList.get(previousIndex);
            characterValues.put(word.charAt(0), gFirstValue+1);
            hValue = hValueFinder(word, characterValues, wordsArray, sortedWordList);
            if (wordsArray[hValue] != null && gFirstValue<=4){
                characterValues.put(previousElement.charAt(0),gFirstValue+1);
            }
            else {
```

```
                wordsArray[hValue] = previousElement;
            }
        }
        catch (IndexOutOfBoundsException e){
            System.out.println(e);
        }
    }
    return hValue;
}

public Hashtable<String, Integer> statisticsResults() {
        Hashtable <String, Integer> keyWordCount = new Hashtable<>();
        HashMap<String, Integer> keywords = mapSortByValue();
        int wordCount=0;
        int lineCount = 0;
        try {
            File tstOdd = new File("D:\\STUDY\\L4\\EEX4465\\mp\\EEX4465-
2020_MP\\tstOdd.txt");
            Scanner fileScan = new Scanner(tstOdd);
            while (fileScan.hasNextLine()) {
                String textWords =
fileScan.nextLine().replaceAll("[(),.=?]","").replaceAll("\\d+","");

                String[] textLine = textWords.split("\\s+");
                if (textWords.length() != 0){
                    lineCount++;
                }
                for (String w: textLine){
                    wordCount++;
                    if (keywords.containsKey(w)){
                        if (!keyWordCount.containsKey(w)){
                            keyWordCount.put(w, 1);
                        }
                        else if (keyWordCount.containsKey(w)){
                            int count = keyWordCount.get(w);
                            count++;
                            keyWordCount.replace(w, count);
                        }
                    }
                }
            }
            fileScan.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred");
            }
        System.out.println("    Statistics results :  ");
        System.out.println("--------------------------------");
        System.out.println("\tTotal lines Read " +lineCount);
        System.out.println("\tTotal words read " +wordCount);
        System.out.println("\tBreakdown by keyword");
        Set<String> keyCount = keyWordCount.keySet();
        int total = 0;
        for (String key: keyCount){
            System.out.println("\t"+key +" : "+ keyWordCount.get(key) );
            total = total + keyWordCount.get(key);
        }
        System.out.println("\tTotal keywords " + total);


        return keyWordCount;

    }


    public static void main(String[] args) {
        FileHandle fh = new FileHandle();
        long startTime = System.currentTimeMillis();
        fh.fileReader();
        fh.letterFrequencyCounter();
```

```
        fh.wordValueCalculator();
        fh.mapSortByValue();
        fh.setLetterFrequencyZero();
        fh.dataStructureConverter();
        fh.hashTableGenerator();
        long finishedTime = System.currentTimeMillis();
        long ExecutionTime = finishedTime - startTime;

        System.out.println(fh.fileReader());
        System.out.println(fh.letterFrequencyCounter());
        System.out.println(fh.wordValueCalculator());
        System.out.println(fh.mapSortByValue());
        System.out.println(fh.setLetterFrequencyZero());
        System.out.println(fh.dataStructureConverter());
        System.out.println(fh.hashTableGenerator());
        fh.statisticsResults();
        System.out.println("\tExecution time: "+ ExecutionTime+" milliseconds");

    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
[jitter, frequency, Chapter, camera, step, in, range, the, ToF, influence]
{a=1, C=1, c=1, e=3, f=1, F=1, i=2, j=1, n=1, p=1, r=3, s=1, t=1, T=1, y=1}
{the=4, jitter=4, in=3, Chapter=4, range=6, step=2, ToF=2, camera=2, frequency=2, influence=5}
{range=6, influence=5, the=4, jitter=4, Chapter=4, in=3, step=2, ToF=2, camera=2, frequency=2}
{a=0, C=0, c=0, e=0, f=0, F=0, i=0, j=0, n=0, p=0, r=0, s=0, t=0, T=0, y=0}
[range, influence, the, jitter, Chapter, in, step, ToF, camera, frequency]
{9=influence, 8=ToF, 7=Chapter, 6=jitter, 5=range, 4=step, 3=the, 2=in, 1=frequency, 0=camera}
    Statistics results :
--------------------------------
    Total lines Read 61
    Total words read 733
    Breakdown by keyword
    step : 3
    jitter : 15
    range : 15
    camera : 2
    ToF : 10
    the : 70
    frequency : 5
    influence : 7
    in : 23
    Chapter : 4
    Total keywords 154
    Execution time: 69 milliseconds

Process finished with exit code 0
```