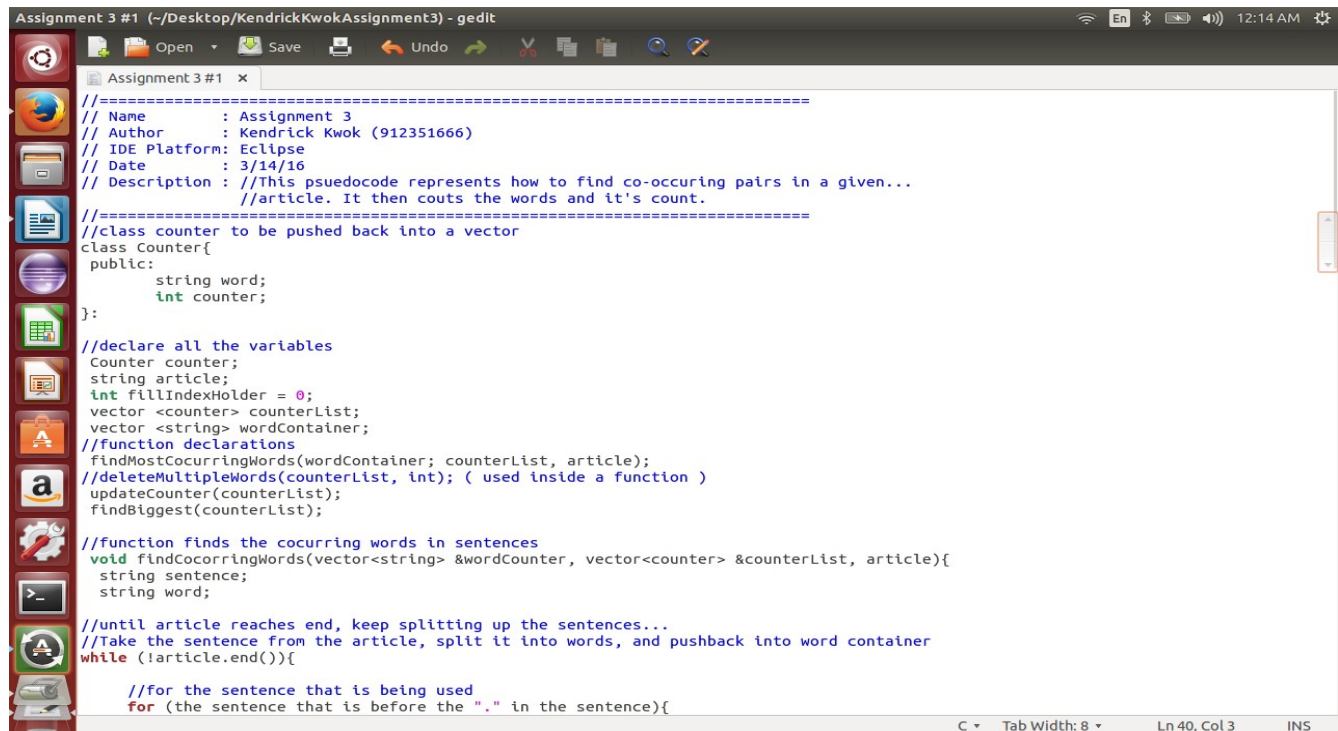


```
//=====
=====
// Name      : Assignment 3
// Author     : Kendrick Kwok (912351666)
// Platform  : Linux text file, used GEDIT to open these text files
// Date      : 3/14/16
// Description : //This assignment was to write pseudocode for the different prompts.
//=====
```

#1: //This pseudocode represents how to find co-occurring pairs in a given article. It then counts the words and its count. I wrote the pseudocode using a LINUX text file, using GEDIT to open the files. Here should how the file should look when it is opened.

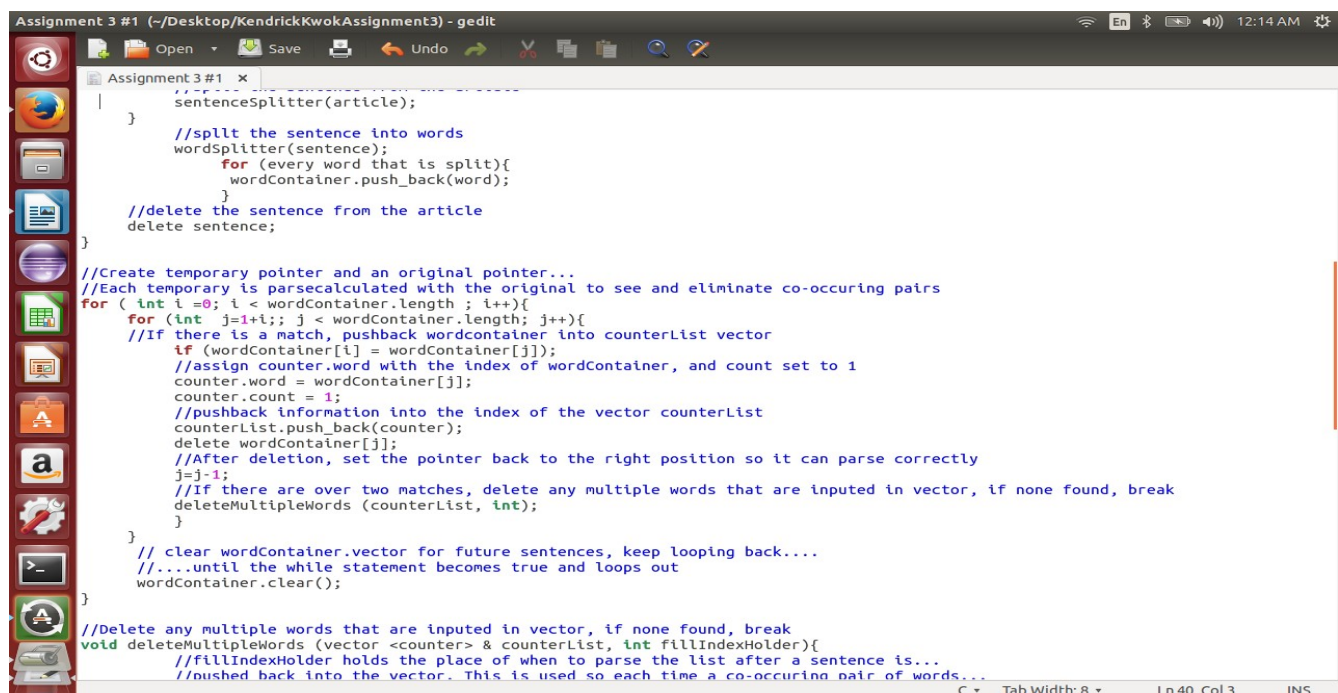


```
Assignment 3 #1 (~/Desktop/KendrickKwokAssignment3) - gedit
//=====
// Name      : Assignment 3
// Author     : Kendrick Kwok (912351666)
// IDE Platform: Eclipse
// Date      : 3/14/16
// Description : //This pseudocode represents how to find co-occurring pairs in a given...
//              //article. It then counts the words and its count.
//=====
//class counter to be pushed back into a vector
class Counter{
public:
    string word;
    int counter;
};

//declare all the variables
Counter counter;
string article;
int fillIndexHolder = 0;
vector <counter> counterList;
vector <string> wordContainer;
//function declarations
findMostCoccurringWords(wordContainer, counterList, article);
//deleteMultipleWords(counterList, int); ( used inside a function )
updateCounter(counterList);
findBiggest(counterList);

//function finds the cocurring words in sentences
void findCocurringWords(vector<string> &wordContainer, vector<counter> &counterList, article){
    string sentence;
    string word;

    //until article reaches end, keep splitting up the sentences...
    //Take the sentence from the article, split it into words, and pushback into word container
    while (!article.empty()){
        //for the sentence that is being used
        for (the sentence that is before the "." in the sentence){
```



```
        sentenceSplitter(article);
    }

    //spllt the sentence into words
    wordsplitter(sentence);
    for (every word that is split){
        wordContainer.push_back(word);
    }

    //delete the sentence from the article
    delete sentence;
}

//Create temporary pointer and an original pointer...
//Each temporary is parsecalculated with the original to see and eliminate co-occurring pairs
for ( int i=0; i < wordContainer.length; i++){
    for (int j=i+1; j < wordContainer.length; j++){
        //If there is a match, pushback wordContainer into counterList vector
        if (wordContainer[i] == wordContainer[j]){
            //assign counter.word with the index of wordContainer, and count set to 1
            counter.word = wordContainer[j];
            counter.count = 1;
            //pushback information into the index of the vector counterList
            counterList.push_back(counter);
            delete wordContainer[j];
            //After deletion, set the pointer back to the right position so it can parse correctly
            j=j-1;
            //If there are over two matches, delete any multiple words that are inputed in vector, if none found, break
            deleteMultipleWords (counterList, int);
        }
    }

    // clear wordContainer.vector for future sentences, keep looping back...
    //....until the while statement becomes true and loops out
    wordContainer.clear();
}

//Delete any multiple words that are inputed in vector, if none found, break
void deleteMultipleWords (vector <counter> & counterList, int fillIndexHolder){
    //fillIndexHolder holds the place of when to parse the list after a sentence is...
    //pushed back into the vector. This is used so each time a co-occurring pair of words...
```

```
Assignment 3 #1 (-/Desktop/KendrickKwokAssignment3) - gedit
//... added from a different sentence, it would not remove the co-occurring pair of words...
//found from the previous sentence
for (int i = fillIndexHolder; i < counterList.length; i++){
    for (int j=1+i; j < counterList.length; j++){
        if (there is more than one index with the same words duplicated in counterList vector){
            counterList(fillIndexHolder+i) = counterList(j)
            delete counterList(j);
            j=j-1;
            fillIndexHolder = fillIndexHolder + counterList.length();
        }
        else{
            break;
        }
    }
}

//update the counter so each similar words would be added to count after all the co-occurring words...
//... are added into the vector
void updateCounter(vector <counter>&counterList){
    for (int i=0; i < counterList.length; i++){
        for (int j=1+i; j < counterList.length; j++){
            //parse through the list, if common word is found in vector, update counter +1
            if (counterList[i].word = counterList[j].word){
                counterList[i].counter = counter + 1;
                //erase the word and count from the list
                counterList.erase (counterList.begin(),counterList.begin()+j)
                //j is minused one so it can keep parsing after deletion
                j=j-1;
            }
        }
    }
}

//Find the biggest amount of count inside the vector counterList
void findBiggest(vector <counter>&counterList){
    //Original is put in the very first index
    original = counterList[0].count;
    for (unsigned int originalIndex=1; originalIndex < counterList.size(); originalIndex++){
```

```
Assignment 3 #1 (-/Desktop/KendrickKwokAssignment3) - gedit
//update the counter so each similar words would be added to count after all the co-occurring words...
//... are added into the vector
void updateCounter(vector <counter>&counterList){
    for (int i=0; i < counterList.length; i++){
        for (int j=1+i; j < counterList.length; j++){
            //parse through the list, if common word is found in vector, update counter +1
            if (counterList[i].word = counterList[j].word){
                counterList[i].counter = counter + 1;
                //erase the word and count from the list
                counterList.erase (counterList.begin(),counterList.begin()+j)
                //j is minused one so it can keep parsing after deletion
                j=j-1;
            }
        }
    }
}

//Find the biggest amount of count inside the vector counterList
void findBiggest(vector <counter>&counterList){
    //Original is put in the very first index
    original = counterList[0].count;
    for (unsigned int originalIndex=1; originalIndex < counterList.size(); originalIndex++){
        //if the counterList count starting from 0 is bigger the ones after, assign original to be...
        //biggest count
        if (counterList[originalIndex].count > original){
            original = list[originalIndex].count;
        }
    }
    //Loop through the whole vector, and if original (original = largest) is the same amount...
    //of counts as the other elements inside the vectors, cout the vector element with same value
    for (int i = 0; i < counterList.size(); i++){
        if (original == counterList[i].count){
            cout << "Your found the most frequent co-occurring word is: " counterList[i].word << endl;
            cout << "Your count is: " << original << endl;
        }
    }
}
```

#2. This pseudocode represents how to find products that is considered frequent (i.e products that are transactioned) more then 5% of the total amount of transactions. I used linux txtfile, and gedit to open it.

```
Assignment 3 #2 (~/Desktop/KendrickKwokAssignment3) - gedit
//=====//
// Name      : Assignment 3
// Author    : Kendrick Kwok (912351666)
// IDE Platform: Eclipse
// Date      : 3/14/16
// Description : //This psuedocode represents how to find products that is considered...
//              //frequent, i.e products that are transactioned...
//              //more then 5% of the total amount of transactions
//=====//

class productPairs{
    string products;
    int transactions;
}

//Declare variables used in the main function
int totalTransaction;
int calculatedFivePercent;
min_freq = .05;

//put the class into a vector
vector <productPairs> productVector;

//Declare an object for productPairs
productPairs productsBought;

//Assume that you have a list of products with the number of transactions already tied to it called "productPairs"
//Put every single product with the transactions into vector called productVector
for(int i =0; i < productVector(); i++){
    productsBought.products.push_back(productVector);
    productsBought.transactions(productVector);
}

//for every product inside the productVector, take each transaction and add it
for(int i = 0 ; i < productVector.length(); i++){
    totalTransaction = totalTransaction + productVector(i).transactions;
}
```

```
Assignment 3 #2 (~/Desktop/KendrickKwokAssignment3) - gedit
min_freq = .05;

//put the class into a vector
vector <productPairs> productVector;

//Declare an object for productPairs
productPairs productsBought;

//Assume that you have a list of products with the number of transactions already tied to it called "productPairs"
//Put every single product with the transactions into vector called productVector
for(int i =0; i < productVector(); i++){
    productsBought.products.push_back(productVector);
    productsBought.transactions(productVector);
}

//for every product inside the productVector, take each transaction and add it
for(int i = 0 ; i < productVector.length(); i++){
    totalTransaction = totalTransaction + productVector(i).transactions;
}

//find what is 5% of the totalTransactions made;
calculatedFivePercent = totalTransaction * min_freq;

//For every index of productVector, see if it is actually 5%. If it is under 5%, delete from the vector list
for (int i =0; i< productVector.length(); i++){
    //If the products total transaction is less then the calculated five percent, delete from vector
    if (productVector(i).totalTransaction < calculatedFivePercent){
        productVector.erase(productVector.begin + i;
        i=i-1;
    }
}

//Parse through the vector productVector and print out all the products that is more frequent
for (int i =0; i< productVector.length(); i++){
    cout << productVector(i).product << " is considered more frequent!" << endl;
}
```