You can form a project group of 2 to 3 persons to solve the problem in project.

Project 4 - Text Analysis

(Credit: The following questions come from the School of Computing)

The descriptions are based on the compiler gcc and the program runs on the UNIX system. You can run your program on the PC system.

Problem Statement:

The objective of this Assessment is to assess your ability to manipulate texts, using arrays and character data types. You will write a program to read in a database of words (comprising English alphabets), collect bigrams out of them, and perform some statistics on the bigrams collected.

A **bigram** is a consecutive pair of alphabets occurring in a word. For instance, the word "`chill`" consists of 4 bigrams:"`ch`", "`hi`", "`il`" and "`ll`".

The task is divided into several levels. Read through all the levels (from first to last, then from last to first) to see how the different levels are related. **You may start from any level.**

Summary

Now, we give a summary of the objectives to be completed at each level.

- **Level 1**: Read in a database of words (comprising English alphabets only). The input will be terminated when a full-stop ('.') is read. For every word read, immediately print out that word.

- **Level 2**: Continuing from level 1, store the words read, in the same order as they are read, in a 2D array -- which will be referred to as a **word-bank** -- with each row storing one word -- in lowercase. Repeated words read will be kept as distinct entries in the word-bank -- no word read will be discarded. Determine the length of the longest word and the number of words read.

- **Level 3**: Find and display all the bigrams from the words in the word-bank.

- **Level 4**: Enhance the task described in Level 3 by determining the number of occurrences of each bigram found in the word-bank. We call the number of occurrences "**frequency**". Then, build and display a bigram-frequency table associating each bigram with its frequency.

- **Level 5**: Modify the presentation of bigram-frequency table constructed by the task in Level 4, so that the **table presented is in compact format**, in which no rows and columns in the presentation contain only zeros.

- **Level 6**: List down all the **repeated bigrams**. For instance, the bigram "`ti`" is a "repeated bigram" with respect to the word "`Initialization`", since it occurs twice in the word. Identify all the repeated bigrams by re-examining the word-bank.

  For each repeated bigram, display a message indicating the number of words in the database for which the bigram is found to be repeated. Arrange all these messages in the following order: first, in descending order of the number of words a repeated bigram was found; and then, if two repeated bigrams have the same number of words found, arrange these bigrams in ascending lexicographical order.

Before you begin your coding tasks, here are some boundaries that we set for your solution:

i. The maximum number of English words in a database that your program will read is assumed to be 100.
ii. The minimum number of English words in a database that your program will read is assumed to be 1.
iii. The maximum length of an English word that your program will read and process is assumed to be 20 characters.
iv. While the database may contains words of 1 character long, it **must** contain at least one word with length of at least 2 characters.
v. The maximum number of bigrams that can be found from all the words in the database is assumed to be 250.
vi. The minimum number of bigrams that can be found from all the words in the database is at least 1; a fact that can be derived from point (iv) above.
vii. The maximum number of occurrences of a bigram in the word database is at most 99.

You will need the C Character function library to build your code. Please the header file is `ctype.h` in your code. The three C Character functions you may want to use are described below:

Character Functions	Descriptions
`isspace()`	Check whether a character (its argument) is a white-space character or not.
`isalpha()`	Check whether a character (its argument) is an alphabet or not.
`tolower()`	Takes an uppercase alphabet (as its argument) and convert it to the corresponding lowercase alphabet. If the argument passed to this function is other than an uppercase alphabet, it returns the same character that is passed to the function.

♦ ♦ ♦ ♦ ♦

Name your program as `bigram1.c`.

Read here regarding the assumptions made about the tasks.

Read in a database comprising only words (comprising English alphabets only). Input comprises individual words with at least one whitespace interspersed between them. Input terminates when a full-stop ('.') is read. Output the words individually as they are read.

Following are some sample runs (user input are in **blue and in bold**). The first column shows the sample runs, and the second column shows the database read. Note that lines containing whitespaces are accepted, but whitespaces read are not printed.

Sample Run 1

```
$ a.out
Read in words:
   in
in
on
on
         toTOloTTO
toTOloTTO
   sinGING  .
sinGING
```

Input database used for this run:

```
   in
on
         toTOloTTO
   sinGING  .
```

Sample Run 2

```
$a.out
Read in words:
The
The
       embodiment
embodiment


radiator
radiator
attendence.
attendence
```

Input database used for this run:

```
The
       embodiment


radiator
attendence.
```

Click here to submit to CodeCrunch.

A sample test case is provided for you to test for format correctness using the `diff` command:

```
./a.out < bigram.in | diff - bigram1.out
```

To proceed to the next level (say level 2), copy your program by typing the Unix command

```
cp bigram1.c bigram2.c
```

♦ ♦ ♦ ♦ ♦

Name your program as `bigram2.c`.

Read here regarding the assumptions made about the tasks.

Read in a database comprising only words. Input comprises individual words with at least one whitespace interspersed between them. Input terminates when a full-stop ('.') is read. All words in the database contain only English alphabets, either uppercase or lowercase or a mixture of both. For practicality, modify your code in Level 1 so that reading is performed via a procedure named `readWords`, the prototype of which is as follows:

```
   void readWords(char bank[][MAX_WORD_SIZE], int* maxWordSize, int*
wordCount) ;
```

Here, the first parameter is the resulting word-bank (a 2D array) constructed to contain all words read (duplicated words are included as separate words in the word-bank), such that each row in the array contains one word, words kept in the array are left-justified (ie., every word begins at column 0 of the word-bank), arranged according to the order in which they are read by your program, and converted to lowercase.

The second parameter (passed by reference) of procedure `readWords` is the resulting maximum word length found, and the third parameter (again passed by reference) enables returning of the total number of words read from database.

Once reading of words is done, your program will display the words stored in the word-bank, the total number of words read, and the maximum word length. Construct a procedure to print out all the words in the word-bank. The procedure can have the following prototype:

```
     void prtWords(char bank[][MAX_WORD_SIZE], int maxWordSize, int
wordCount) ;
```

Following are some sample runs (user input are in **blue and in bold**):

Sample Run 1

```
$ a.out

Read in words:

   in SINGING

on singing

   barberer toTOloTTO inkling

an sinGING  .

Word Bank:

++++++++++

in

singing
```

Sample Run 2

```
$a.out

Read in words:

      abcdefg

abcdef

abcde     abcd

abc    ab

a

    .

Word Bank:

++++++++++
```

```
on

singing

barberer

totolotto

inkling

an

singing

Number of words read: 9

The length of the longest word: 9
```

```
abcdefg

abcdef

abcde

abcd

abc

ab

a

Number of words read: 7

The length of the longest word: 7
```

Click here to submit to CodeCrunch.

A sample test case is provided for you to test for format correctness using the `diff` command:

```
./a.out < bigram.in | diff - bigram2.out
```

To proceed to the next level (say level 3), copy your program by typing the Unix command

```
cp bigram2.c bigram3.c
```

♦ ♦ ♦ ♦ ♦

Name your program as `bigram3.c`.

1. Read here regarding the assumptions made about the tasks.

2. Read the detail of Level 2 for the purpose of reading in words and forming a word-bank. Print out the content of the word-bank as described in Level 2.

3. Develop a procedure named `findBigrams` which goes through the entire word-bank and print out all the bigrams found in each word. The procedure has the following prototype:

```
4.                 void findBigrams(char bank[][MAX_WORD_SIZE], int
   maxWordSize, int wordCount) ;
```

Following are some sample runs (user input are in **blue and in bold**):

Sample Run 1

```
Read in words:
```

Sample Run 2

```
Read in words:
```

```
   in singing
on singing
   barberer totolotto inkling
an singing.
Word Bank:
+++++++++
in
singing
on
singing
barberer
totolotto
inkling
an
singing
Number of words read: 9
The length of the longest word: 9

Bigrams found:
++++++++++++++
in: in
singing: si in ng gi in ng
on: on
singing: si in ng gi in ng
barberer: ba ar rb be er re er
totolotto: to ot to ol lo ot tt to
inkling: in nk kl li in ng
an: an
singing: si in ng gi in ng
```

```
abcdefg
abcdef
abcde
abcd
abc
ab
a
   .
Word Bank:
+++++++++
abcdefg
abcdef
abcde
abcd
abc
ab
a
Number of words read: 7
The length of the longest word: 7

Bigrams found:
++++++++++++++
abcdefg: ab bc cd de ef fg
abcdef: ab bc cd de ef
abcde: ab bc cd de
abcd: ab bc cd
abc: ab bc
ab: ab
a:
```

Click here to submit to CodeCrunch.

A sample test case is provided for you to test for format correctness using the `diff` command:

```
./a.out < bigram.in | diff - bigram3.out
```

To proceed to the next level (say level 4), copy your program by typing the Unix command

```
cp bigram3.c bigram4.c
```

♦ ♦ ♦ ♦ ♦

Name your program as `bigram4.c`.

1. Read here regarding the assumptions made about the tasks.

2. Read the details about Level 2 for the purpose of reading in words and forming a word-bank. Print out the content of the word-bank as described in Level 2.

3. Perform the task described in Level 3 in order to find all bigrams present in the word bank. Modify your procedure `findBigrams` so that it does not just display all bigrams found, but also produce a 26X26 table with the following properties:

    1. Each row represents the first alphabet of a bigram, with the first row (row 0) represents alphabet `a`, second row (row 1) represents `b`, up to the last row (row 25) represents `z`.

    2. Each column represents the second alphabet of a bigram, with the first row represents alphabet `a`, and second presents alphabet `b`, and so on.

    3. A cell (*i,j*) in the table keeps the number of occurrences of the corresponding bigram in the entire word bank.

    The modified procedure `findBigrams` has the following prototype:

```
       void findBigrams(char bank[][MAX_WORD_SIZE], int maxWordSize,
int wordCount,
                                          int bigramTable[26][26]) ;
```

Next, construct a procedure `bfInfo` to display three pieces of information about the bigrams found:

1. The table `bigramTable` received from procedure `findBigrams`; refer to the sample runs for display format, and the format specifier for printing each of the table value is `%3d`, as in:

```
2.                 printf("%3d", ...) ;
```

3. The total number of distinct bigrams found from the word-bank, and

4. The total number of occurrences of all bigrams in the word-bank.

The procedure `bfInfo` has the following prototype:

```
void bfInfo(int bigramTable[26][26]) ;
```

Following are some sample runs (user input are in **blue and in bold**):

Sample Run 1

```
Read in words:
abcdefg
abcdef
abcde
abcd
abc
ab
a
.
Word Bank:
+++++++++
abcdefg
abcdef
abcde
abcd
abc
ab
a
Number of words read: 7
The length of the longest word: 7


Bigrams found:
++++++++++++
```

```
abcdefg: ab bc cd de ef fg

abcdef: ab bc cd de ef

abcde: ab bc cd de

abcd: ab bc cd

abc: ab bc

ab: ab

a:


Bigram Frequency Table:

++++++++++++++++++++++

  |  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z

--+-----------------------------------------------------------------------------

 a|  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 b|  0  0  5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 c|  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 d|  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 e|  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 f|  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 g|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 h|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 i|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 j|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 k|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 l|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 m|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 n|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 o|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 p|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 q|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 r|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```
 s|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 t|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 u|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 v|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 w|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 x|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 y|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 z|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0


Bigram statistics:

++++++++++++++++++

Total number of bigrams: 6.

Total count of bigram occurrences: 21.
```

Sample Run 2

```
$ a.out

Read in words:

potatomato.

Word Bank:

++++++++++

potatomato

Number of words read: 1

The length of the longest word: 10


Bigrams found:

++++++++++++++

potatomato: po ot ta at to om ma at to


Bigram Frequency Table:

++++++++++++++++++++++
```

```
  |  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z
--+-------------------------------------------------------------------------------
 a|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0
 b|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 c|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 d|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 e|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 f|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 g|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 h|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 i|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 j|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 k|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 l|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 m|  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 n|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 o|  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0
 p|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
 q|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 r|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 s|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 t|  1  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0
 u|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 v|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 w|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 x|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 y|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 z|  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0


Bigram statistics:
```

```
++++++++++++++++++

Total number of bigrams: 7.

Total count of bigram occurrences: 9.
```

Click here to submit to CodeCrunch.

A sample test case is provided for you to test for format correctness using the `diff` command:

```
./a.out < bigram.in | diff - bigram4.out
```

To proceed to the next level (say level 5), copy your program by typing the Unix command

```
cp bigram4.c bigram5.c
```

♦ ♦ ♦ ♦ ♦

Name your program as `bigram5.c`.

1. Read here regarding the assumptions made about the tasks.

2. Read the details about Level 2 for the puRead the details about Level 2 for the purpose of reading in words and forming a word-bank. Print out the content of the word-bank as described in Level 2.

3. Read the details about Level 4 for displaying all the bigrams found in the word-bank and showing the bigram-frequency table and statistics. However, modify the `bfInfo` procedure so that it does not display the full bigram-frequency table. Instead, **display only those rows and columns having some non-zeros in their entries**

Following are some sample runs (user input are in **blue and in bold**):

Sample Run 1

```
$ a.out

Read in words:

   in singing

on singing

   barberer totolotto inkling

an singing.

Word Bank:
```

Sample Run 2

```
Read in words:

abcdefg

abcdef

abcde

abcd

abc

ab
```

```
++++++++++
in
singing
on
singing
barberer
totolotto
inkling
an
singing
Number of words read: 9
The length of the longest word: 9


Bigrams found:
++++++++++++++
in: in
singing: si in ng gi in ng
on: on
singing: si in ng gi in ng
barberer: ba ar rb be er re er
totolotto: to ot to ol lo ot tt to
inkling: in nk kl li in ng
an: an
singing: si in ng gi in ng


Bigram Frequency Table:
++++++++++++++++++++++
  |  a  b  e  g  i  k  l  n  o  r  t
--+-----------------------------------
 a|  0  0  0  0  0  0  0  1  0  1  0
```

```
a
   .
Word Bank:
++++++++++
abcdefg
abcdef
abcde
abcd
abc
ab
a
Number of words read: 7
The length of the longest word: 7


Bigrams found:
++++++++++++++
abcdefg: ab bc cd de ef fg
abcdef: ab bc cd de ef
abcde: ab bc cd de
abcd: ab bc cd
abc: ab bc
ab: ab
a:


Bigram Frequency Table:
++++++++++++++++++++++++
  |  b  c  d  e  f  g
--+-------------------
 a|  6  0  0  0  0  0
 b|  0  5  0  0  0  0
```

```
 b|  1  0  1  0  0  0  0  0  0  0  0

 e|  0  0  0  0  0  0  0  0  0  2  0

 g|  0  0  0  0  3  0  0  0  0  0  0

 i|  0  0  0  0  0  0  0  9  0  0  0

 k|  0  0  0  0  0  0  1  0  0  0  0

 l|  0  0  0  0  1  0  0  0  1  0  0

 n|  0  0  0  7  0  1  0  0  0  0  0

 o|  0  0  0  0  0  0  1  1  0  0  2

 r|  0  1  1  0  0  0  0  0  0  0  0

 s|  0  0  0  0  3  0  0  0  0  0  0

 t|  0  0  0  0  0  0  0  0  3  0  1


Bigram statistics:

++++++++++++++++++

Total number of bigrams: 20.

Total count of bigram occurrences: 42.
```

```
 c|  0  0  4  0  0  0

 d|  0  0  0  3  0  0

 e|  0  0  0  0  2  0

 f|  0  0  0  0  0  1


Bigram statistics:

++++++++++++++++++

Total number of bigrams: 6.

Total count of bigram occurrences: 21.
```

Click here to submit to CodeCrunch.

A sample test case is provided for you to test for format correctness using the `diff` command:

```
./a.out < bigram.in | diff - bigram5.out
```

To proceed to the next level (say level 6), copy your program by typing the Unix command

```
cp bigram5.c bigram6.c
```

♦ ♦ ♦ ♦ ♦

Name your program as `bigram6.c`.

1. Read here regarding the assumptions made about the tasks.

2. Read the details about Level 2 for the puRead the details about Level 2 for the purpose of reading in words and forming a word-bank. Print out the content of the word-bank as described in Level 2.

3. Read the details about Level 5 for displaying all bigram-frequency pairs found in the word-bank and presenting the two statistics regarding the number of distinct bigrams found and the total number of occurrences of all bigrams.

4. Construct a procedure to list down all **repeated bigrams**. A repeated bigram is a bigram that **appears more than once in a word**. For instance, for the word "`Initialization`", the bigram "`ti`" (underlined) has appeared two times in the word. It is therefore a repeated bigram.

More specifically, given the following database:

```
                  in singing
               on singing
                  barberer totolotto inkling
               an singing.
```

We see that:

- "`in`" is a repeated bigram, as it is repeated twice in each of the four words: `singing` in first, second and fourth lines, and `inkling` in third line.
- "`er`" is a repeated bigram, as it is repeated twice in one word: `barberer`.
- "`ng`" is a repeated bigram which is repeated twice in three words: all the three occurrences of `singing` in the database. However, `ng` is **not** a repeated bigram in `inkling`, because it only appears once in the word. So, `ng` is found repeated in three words, not four.
- The other repeated bigrams are: "`ot`" (found repeated in one word: `totolotto`) and "`to`" (found repeated in one word: `totolotto`).

The procedure for listing down all repeated bigrams has the following prototype:

```
       void repeatBigram(char bank[][MAX_WORD_SIZE], int maxWordSize,
int wordCount,
                                                  int
bigramTable[26][26]);
```

The procedure displays the repeated bigrams in the following fashion: These bigrams will firstly be arranged in descending order by the number of words in which a bigram is found repeated; and then secondly (if the numbers of words reported for two repeated bigrams are equal) be arranged in ascending (lexicographical) order by the repeated bigrams.

Following are some sample runs (user input are in **blue and in bold**):

Sample Run 1	Sample Run 2
`$ a.out`	`$ a.out`

```
Read in words:
abcdefg
abcdef
abcde
abcd
abc
ab
a
   .
Word Bank:
++++++++++
abcdefg
abcdef
abcde
abcd
abc
ab
a
Number of words read: 7
The length of the longest word: 7


Bigrams found:
++++++++++++++
abcdefg: ab bc cd de ef fg
abcdef: ab bc cd de ef
abcde: ab bc cd de
abcd: ab bc cd
abc: ab bc
ab: ab
a:
```

```
Read in words:
   in singing
on singing
   barberer totolotto inkling
an singing.
Word Bank:
++++++++++
in
singing
on
singing
barberer
totolotto
inkling
an
singing
Number of words read: 9
The length of the longest word: 9


Bigrams found:
++++++++++++++
in: in
singing: si in ng gi in ng
on: on
singing: si in ng gi in ng
barberer: ba ar rb be er re er
totolotto: to ot to ol lo ot tt to
inkling: in nk kl li in ng
an: an
singing: si in ng gi in ng
```

```
Bigram Frequency Table:
++++++++++++++++++++++
  |  b  c  d  e  f  g
--+------------------
 a|  6  0  0  0  0  0
 b|  0  5  0  0  0  0
 c|  0  0  4  0  0  0
 d|  0  0  0  3  0  0
 e|  0  0  0  0  2  0
 f|  0  0  0  0  0  1

Bigram statistics:
++++++++++++++++++
Total number of bigrams: 6.
Total count of bigram occurrences: 21.

Repeated Bigrams Statistics:
++++++++++++++++++++++++++++
No bigram is found repeated in any word.
```

```
Bigram Frequency Table:
++++++++++++++++++++++
  |  a  b  e  g  i  k  l  n  o  r  t
--+---------------------------------
 a|  0  0  0  0  0  0  0  1  0  1  0
 b|  1  0  1  0  0  0  0  0  0  0  0
 e|  0  0  0  0  0  0  0  0  0  2  0
 g|  0  0  0  0  3  0  0  0  0  0  0
 i|  0  0  0  0  0  0  0  9  0  0  0
 k|  0  0  0  0  0  0  1  0  0  0  0
 l|  0  0  0  0  1  0  0  0  1  0  0
 n|  0  0  0  7  0  1  0  0  0  0  0
 o|  0  0  0  0  0  0  1  1  0  0  2
 r|  0  1  1  0  0  0  0  0  0  0  0
 s|  0  0  0  0  3  0  0  0  0  0  0
 t|  0  0  0  0  0  0  0  0  3  0  1

Bigram statistics:
++++++++++++++++++
Total number of bigrams: 20.
Total count of bigram occurrences: 42.

Repeated Bigrams Statistics:
++++++++++++++++++++++++++++
Number of words for which "in" is found repeated: 4
Number of words for which "ng" is found repeated: 3
Number of words for which "er" is found repeated: 2
Number of words for which "ot" is found repeated: 2
Number of words for which "to" is found repeated: 2
```

Click here to submit to CodeCrunch.

A sample test case is provided for you to test for format correctness using the `diff` command:

```
./a.out < bigram.in | diff - bigram6.out
```

♦ ♦ ♦ ♦ ♦

THE END

```
Repeated Bigrams Statistics:
++++++++++++++++++++++++++
Number of words for which "in" is found repeated: 4
Number of words for which "ng" is found repeated: 3
Number of words for which "er" is found repeated: 1
Number of words for which "ot" is found repeated: 1
Number of words for which "to" is found repeated: 1
```