National University of Computer & Emerging Sciences (NUCES),

Islamabad Department of Computer Science

CS201: Data Structures (Fall 2018)

Assignment # 1

Submission: You are required to submit a single header file for the assignment. The h**eader file should contain N-Gram class as well as the template.** Name the header files as *Name_RollNumber_Section* (i.e. *Adam_i170999_A.h*) and submit it on the slate.

Deadline: Deadline to submit the assignment is **30th September 11:55 PM**. No submission will be considered for grading outside the slate or after 30th September 11:55 PM. **Correct and timely submission of the assignment is the responsibility of the student; hence no relaxation will be given to anyone.**

Plagiarism: -50% marks in the assignment if any part of the assignment is found plagiarized

Assignment Description:

In this problem, you will write a program that reads an input file and uses it to build a large data structure of word groups called "N-grams" as a basis for randomly generating new text that sounds like it came from the same author as that file.

What are n-grams?

The general idea is that you can look at each pair (or triple, set of four, etc.) of words that occur next to each other. In a sufficiently-large corpus (text file), you're likely to see "the red" and "red apple" several times, but less likely to see "apple red" and "red the". This is useful to know if, for example, you're trying to figure out what someone is more likely to say to help decide between possible output for an automatic speech recognition system or sentence correction.

These co-occurring words are known as "n-grams", where "n" is a number saying how long a string of words you considered. (Unigrams are single words, bigrams are two words, trigrams are three words, and 4-grams are four words)

In this assignment, we will consider unigrams, bigrams, and trigrams.

Let’s understand n-gram using an example:

Suppose we have following text file (*input.txt*):

I am Sam. Sam I am. I am Sam. I do not like, green “eggs” and Sam. But I do like white eggs.

Steps for n-grams:

1. Read the *input.txt* file and load the data into a “char” array.
2. Process (or Preprocess, to be technically correct) the char array as follows:

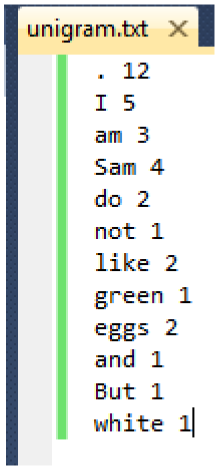
a. Remove all the punctuations marks **(~ ! @ # $ %** ^ **& * ( ) _** + = **“ ; : / ? >** , < and **‘\n’),** except dot (“.”) from the data.
Output: I am Sam. Sam I am. I am Sam. I do not like green eggs and Sam. But I do like white eggs.

b. Remove any space(s) after a dot (“.”).
Output: I am Sam.Sam I am.I am Sam.I do not like green eggs and Sam.But I do like white eggs.

c. Insert full stop (dot “.”) at the start and end of each sentence. Note that there are two full stops at the beginning of the text.
Output: ..I am Sam..Sam I am..I am Sam..I do not like green eggs and Sam..But I do like white eggs..

3. Create the unigrams, bigrams, and trigrams along with their counts (how many times word or group of words repeats) from the char array and save them in their respective files.

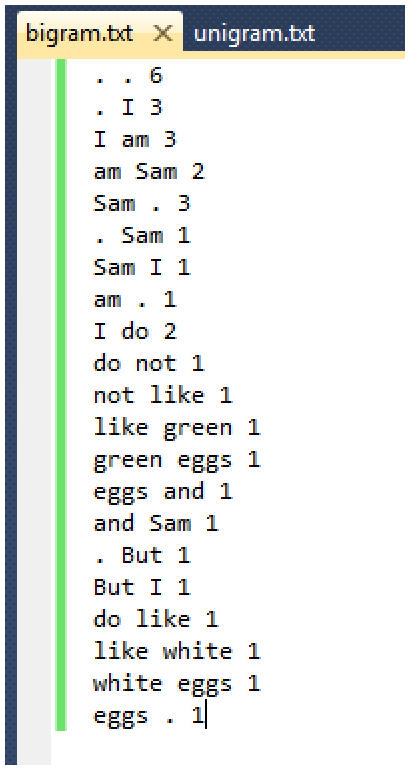
unigram.txt



```
. 12
I 5
am 3
Sam 4
do 2
not 1
like 2
green 1
eggs 2
and 1
But 1
white 1
```

Figure 1: Word and its repetition count, separated by a space.

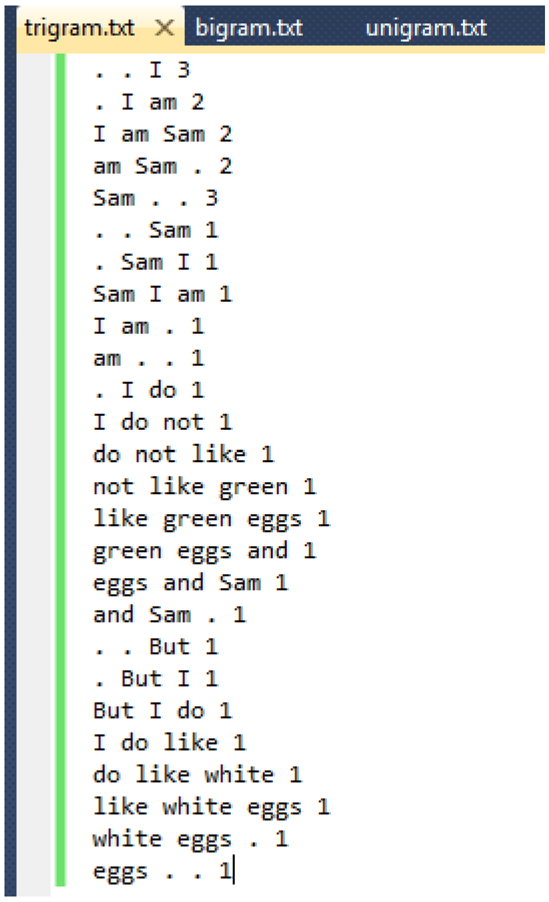
bigram.txt



```
. . 6
. I 3
I am 3
am Sam 2
Sam . 3
. Sam 1
Sam I 1
am . 1
I do 2
do not 1
not like 1
like green 1
green eggs 1
eggs and 1
and Sam 1
. But 1
But I 1
do like 1
like white 1
white eggs 1
eggs . 1
```

Figure 2: Bigram and its repetition count, separated by a space.

Trigram.txt:



```
. . I 3
. I am 2
I am Sam 2
am Sam . 2
Sam . . 3
. . Sam 1
. Sam I 1
Sam I am 1
I am . 1
am . . 1
. I do 1
I do not 1
do not like 1
not like green 1
like green eggs 1
green eggs and 1
eggs and Sam 1
and Sam . 1
. . But 1
. But I 1
But I do 1
I do like 1
do like white 1
like white eggs 1
white eggs . 1
eggs . . 1
```

Figure 3: Trigram and its repetition count, separated by a space.

4. Using unigrams, bigrams, and trigrams, apply the following functions:
   a. Validation of a Sentence:
      Given a sentence, you need to check if the sentence is valid according to the given corpus (text file) or not. The Naive Bayes Rule (given as under) is applied to compute the joint probability of the words in the sentence.

      Example: Check if the sentence “Sam I am” is valid or not in the given corpus.
      Validation: P(“Sam I am”) = P(“..Sam”| "..") x P(“.Sam I” | ".Sam") x P(“Sam I am” | "Sam I") x P(“I am.” | "I am") x P("am.." | "am.")
      where
      P(“..Sam”| "..") = count(“..Sam”) / count("..") = 1/6

P(“.Sam I” | ".Sam") = count(“.Sam I”) / count(".Sam") = 1/1
P(“Sam I am” | "Sam I") = count(“Sam I am”) / count("Sam I") = 1/1
P(“I am.” | "I am") = count(“I am.”) / count("I am") = 1/3
P("am.." | "am.") = count("am..") / count("am.") = 1/1
i.e.
P(“Sam I am”) = 1/6 x 1/1 x 1/1 x 1/3 x 1/1 = 1/18

As the P(”Sam I am”) is greater than zero, so the sentence is a valid sentence. Note that the answer can be a very small ratio, so you will apply the “log” function to both sides of the equation of the probability. The updated formula will be:
log(P(“I am Sam”)) = log(P(“..Sam”| "..") x P(“.Sam I” | ".Sam") x P(“Sam I am” | "Sam I") x P(“I am.” | "I am") x P("am.." | "am."))

= log(P(“..Sam”| "..")) + log(P(“.Sam I” | ".Sam")) + log(P(“Sam I am” | "Sam I")) + log(P(“I am.” | "I am")) + log(P("am.." | "am."))

=log(1.0/6.0) + log(1.0/1.0) + log(1.0/1.0) + log(1.0/3.0) + log(1.0/1.0)

Note: If a sentence contains any word (or sequence of words) which was not given in the corpus then the probability will be zero and the “log” function will be undefined. You need to handle this as a special case (**return 1**) in your code.

b. Sentence Creation:
   i) Randomly choose a trigram that has a dot (“.”) as the starting word, e.g. “. I do”.
   ii) To extend the sentence, skip the first word of the trigram so the remaining words will be “I do” (a bigram).
   iii) From your trigram.txt file, you can see that “like” and “not” are the possible words that can come after “I do” to make the next trigram. You will calculate conditional probability of every possible trigram that starts with “I do”, i.e.
   P(“like” | “I do”) = count (“I do like”) / count (“I do”)
   P(“not” | “I do”) = count (“I do not”) / count (“I do”)
   Here, you would pick the trigram which has the maximum probability. If probabilities are equal pick the first one.
   iv) Repeat the steps **ii and iii,** until you reach the end of the sentence i.e. the next trigram with maximum probability has a full stop (dot”.”) as the third word to indicate the completion of the sentence. Resultant string should not include dots (“.”) as start of sentence.

“I do not like green eggs and Sam.” can be a possible sentence from the given text.

Functions to be implemented (test cases):

1. Signature of "Ngram" class constructor will be

   Ngram(const char* Path)

   This will read file and save it in char array.

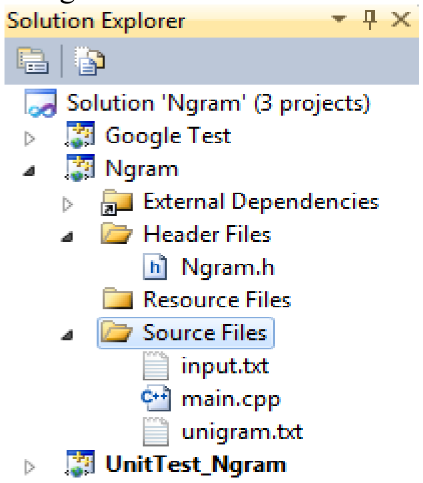
2. void removePunctucationMarks()

3. void removeSpacing()

4. void insertDots()

5. char * getText()

6. void generateUnigrams()
   This function will generate unigrams and save them in text file. Name of text will be "unigram.txt". Path of file will be same as the Project path. i.e.



```
Solution Explorer
Solution 'Ngram' (3 projects)
  ▷ Google Test
  ◢ Ngram
      ▷ External Dependencies
      ◢ Header Files
          Ngram.h
        Resource Files
      ◢ Source Files
          input.txt
          main.cpp
          unigram.txt
  ▷ UnitTest_Ngram
```

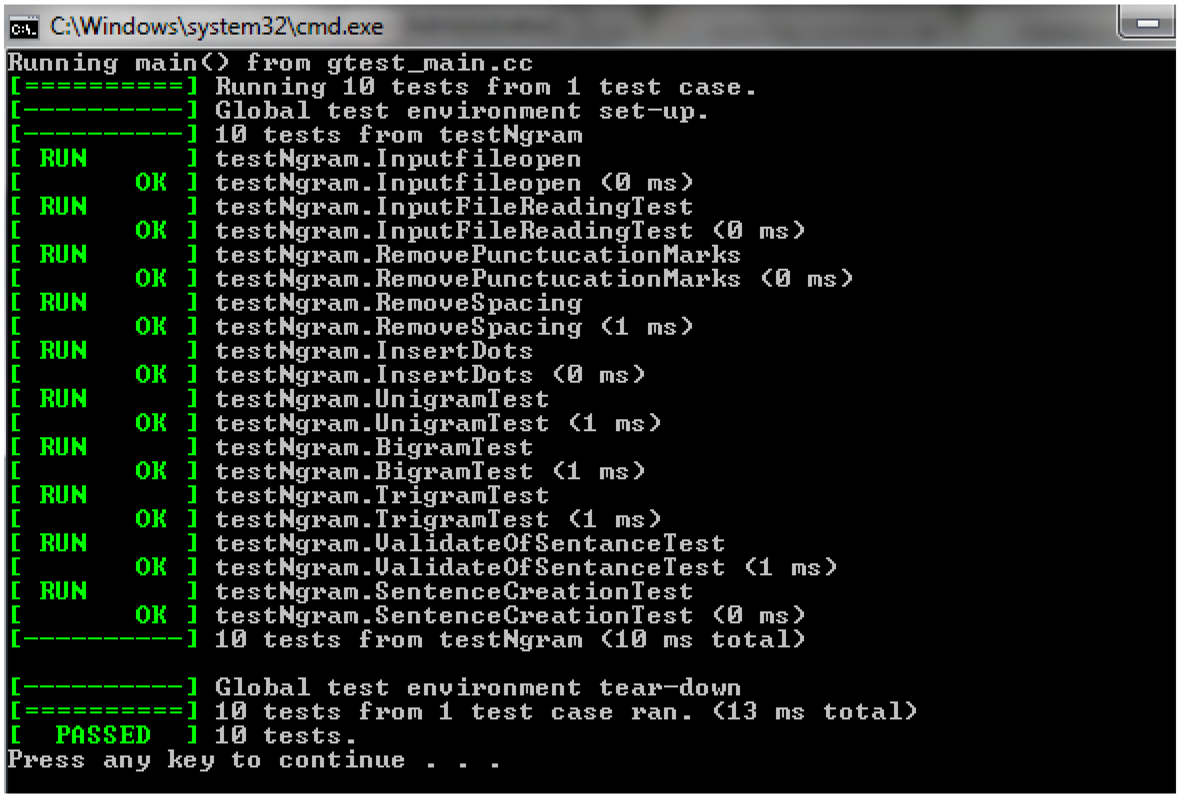
7. void generateBigrams()
   This function will generate bigrams and save them in text file ("bigram.txt")

8. void generateTrigrams()
   This function will generate bigrams and save them in text file ("trigram.txt")

9. double validateSentance(string str)
   This function will validates the given sentence as specified in Step 4a.

10. string sentenceCreation(int position)
    This function will generate the sentence as specified in Step 4b.
    For example:
    Suppose if we want to choose ". I do" as first trigram then we will pass integer 11 to the above given function.

Test Case output:



C:\Windows\system32\cmd.exe

```
Running main() from gtest_main.cc
[==========] Running 10 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 10 tests from testNgram
[ RUN      ] testNgram.Inputfileopen
[       OK ] testNgram.Inputfileopen (0 ms)
[ RUN      ] testNgram.InputFileReadingTest
[       OK ] testNgram.InputFileReadingTest (0 ms)
[ RUN      ] testNgram.RemovePunctucationMarks
[       OK ] testNgram.RemovePunctucationMarks (0 ms)
[ RUN      ] testNgram.RemoveSpacing
[       OK ] testNgram.RemoveSpacing (1 ms)
[ RUN      ] testNgram.InsertDots
[       OK ] testNgram.InsertDots (0 ms)
[ RUN      ] testNgram.UnigramTest
[       OK ] testNgram.UnigramTest (1 ms)
[ RUN      ] testNgram.BigramTest
[       OK ] testNgram.BigramTest (1 ms)
[ RUN      ] testNgram.TrigramTest
[       OK ] testNgram.TrigramTest (1 ms)
[ RUN      ] testNgram.ValidateOfSentanceTest
[       OK ] testNgram.ValidateOfSentanceTest (1 ms)
[ RUN      ] testNgram.SentenceCreationTest
[       OK ] testNgram.SentenceCreationTest (0 ms)
[----------] 10 tests from testNgram (10 ms total)

[----------] Global test environment tear-down
[==========] 10 tests from 1 test case ran. (13 ms total)
[  PASSED  ] 10 tests.
Press any key to continue . . .
```