Course:	**INFO3142**
Professor:	**Jim Cooper**
Project:	**Project #1 – N-gram Extractor – Version 1.0**
Due Date:	**Thursday Nov 5, 2020**
Submitting:	**Please see the last page for instructions**

How will my project be graded?

- This project counts for 25% of your final grade and will be evaluated using the following grid:

Marks Available	What are the Marks Awarded For?	Mark Assigned
2	Command line interface (input filename)	
2	Uses binary search	
8	Correct output for every input file	
3	n-gram construction is generalized and not hard-coded to 2,3, and 4 levels	
2	Proper submission	
15	Total	

Project #1 Requirements

Introduction:

Natural language processing (whether it's done by a human or a machine) requires two types of operations:

1. Decoding the language syntax
2. Understanding the meaning (semantics)

This project is mostly concerned with textual semantics and we'll be using a subset of the WordNet Lexical database to experiment with n-gram definitions.

One of the first steps necessary for "understanding" a piece of natural language text is the process of tokenizing the sentence into a collection (or array) of words. Using WordNet, we can somewhat easily extract definitions for single words although even this can be slightly more difficult when we don't know the part-of-speech for a specific word.

As difficult as meaning can be for single words, it's a much larger problem for compound words such as "ice cream," "venture capitalist" or even "Leaning Tower of Pisa."

In the world of NLP, these combined tokens are typically referred to as n-grams where n designates the number of words in the grouping. When we say 2-gram, we mean a pair of words, like “ice cream.” When we say 3-gram, we mean a triplet of words like “beyond the pale” or “Johann Sebastian Bach” etc.

Dealing with n-grams is an important part of extracting semantic meaning from text, because more often than not, n-grams contain information that’s greater than the sum of the individual words.

It’s also true that our app could be a lot more useful if we had a way to accurately tag all of our words with part-of-speech information because we would then know where to look for lexical definitions (adverbs, adjectives, verbs or nouns), not just for n-grams but also for individual words. Unfortunately, some individual words are both nouns and verbs and so we need to know how each word is being used grammatically within the sentence.

Details:

For this project, we’ll use the C# programming language to create either a .NET or .NET Core console application.

In addition, we want you to use the WordNet subset data that’s been provided with this project specification zip file. The index and data files for adverbs, adjectives, verbs and nouns are all included although you’ll only need the noun version for this project.

Note that we’ve also provide several sample sentence test files that you can use for debugging purposes along with sample corresponding outputs.

During the initialization of your console application, you should load the “noun” data into collections or object arrays, if you prefer. Arrays of struct types are also OK. See the addendum for detailed schema information. Despite the size of the index and data files, we’re using in memory data structures to optimize for performance.

Your executable app should be able to handle a single command line filename to indicate the name of the input file (containing the English text sentence or sentences)

Example run:

ngram test2.txt

In this case, the application is “ngram.exe” and the input file is “test2.txt” and it’s located in the same place as “ngram.exe”

If you're using Visual Studio 2019, it's relatively easy to pre-set a command line argument which you can then use for testing.

When the application is finished running it will have produced an output file named "debug.txt"

It's not required to have a command line option to set the output filename. We'll always output to debug.txt.

Note as well, that you only need to test against the set of sample input files provided with this spec although you may optionally, want to try your own.

Some of the test data contains multiple sentences and we want your final solution to be able to handle each sentence separately as shown in some of the example outputs.

Furthermore, the definition of tokenizing n-grams requires that you try all adjacent word combinations for each sentence. This will mean writing C# code that's capable of building n-grams that fit the following example:

I am flying to San Francisco to visit a venture capitalist

2 level n-gram

I_am
am_flying
flying_to
to_San
San_Francisco
Francisco_to
to_visit
visit_a
a_venture
venture_capitalist

3 level n-gram

I_am_flying
am_flying_to
flying_to_San
to_San_Francisco
San_Francisco_to
Francisco_to_visit
to_visit_a
visit_a_venture
a_venture_capitalist

As you can see the n-grams are constructed using underscores ("_") rather than spaces to separate the words because the WordNet tokens use underscores.

Also keep in mind that WordNet keeps everything in lower case and as such, you'll need to search with that in mind.

In addition to separating the input text into multiple sentences (where necessary), you will need to separate the sentence words into separate tokens in order to build the n-grams as shown above.

You only need to try n-grams 2 through 4 inclusive as seen in the sample output.

Anyone who prefers to work with the WordNet original database files may do so by downloading the latest versions from the WordNet site.

*** End of Requirements ***

Addendum

NounsIndex.txt

chesapeake_bay_retriever|02102501
cheshire_cat|09614850
cheshire_cheese|07869208
chess|12131755,00504248
chess_board|03017971
chess_club|08246196
chess_game|00504248
chess_master|09935109
chess_match|07481248
chess_move|00167176
chess_opening|00458914
chess_piece|03018094
chess_player|09935292
chess_set|08013780
chessboard|03017971
chessman|03018094
chest|05560240,03018359,05560921,03018908
chest_cavity|05560682
chest_of_drawers|03018908

All fields are delimited with vertical bar (“|”) characters

Left-most field contains word or n-gram token

Right-most field contains one or more comma delimited references to NounsData.txt file to retrieve the definition(s) for the word token or n-gram token.

Addendum

NounsData.txt

00941444|the act of arranging and adapting a piece of music
00941634|the act of arranging a piece of music for an orchestra and assigning parts to the different musical instruments
00941859|the completion or enrichment of a piece of music left sparsely notated by a composer
00942033|(music) the repetition of themes introduced earlier (especially when one is composing the final part of a movement)
00942228|the act of inventing
00942376|the act of inventing a word or phrase
00942525|the act of devising something

All fields are delimited with vertical bar (“|”) characters

Left-most field contains reference number token

Right-most field contains definition text referenced by the token in the index file.

Note that the definition field may contain multiple statements each separated with a semi-colon (“;”). In addition, examples may be included which are encapsulated with double quotes (“)

Each file is alleged to be sorted by the left-most field in both cases although this has not been verified.

How should I submit my project?

Electronic Submission:

Submit your project to the INFO3142 *"Project #1"* electronic submission folder in *FOL*. Your code should be submitted as a single "zip" file containing your Visual Studio console app solution.

Submit your project on time!

Project or essay submissions must be made on time! Late submissions will be subject to divisional policy on missed test and late projects. In accordance with this policy, no late submissions will be accepted without prior notification being received by the instructor from the student.

Submit your own work!

It is considered cheating to submit work done by another student or from another source. Helping another student cheat by sharing your work with them is also not tolerated. Students are encouraged to share ideas and to work together on practice exercises, but any code or documentation prepared for a project must be done by the individual student. Penalties for cheating or helping another student cheat may include being assigned zero on the project with even more severe penalties if you are caught cheating more than once. Just submit your own work and benefit from having made the effort on your own.