CPSC 350: Data Structures and Algorithms
Fall 2021

Programming Assignment 1: Tutnese Translation
Due: Sept. 20th, 2021. 11:59pm

The Assignment

Most of you are probably familiar with the popular language game, Pig Latin. In fact, some of you may also recall a similar game called Double Dutch, which is akin to Pig Latin but with a different set of rules that focus on consonants instead of vowels. What you might not know is that Double Dutch is a contemporary name for Tutnese, or the Tut language, which dates back to the horrific period of American history prior to the abolition of slavery. Tutnese served as a language obfuscation scheme so that the enslaved could communicate freely with one another and teach each other to write and spell without reprisal…these activities were prohibited and punished.

In this assignment you will build a program that translates American English text stored in a plain text file to Tutnese. This assignment serves two purposes. The first is to provide some historical context for language encoding and obfuscation techniques, some of which, like Tutnese, came about due to human necessity in extreme circumstances. The second is to exercise your basic C++ skills before moving on to more advanced data structure implementations.

Those interested in additional historical information on Tutnese may find this website useful: http://www.tutlanguage.com/

Tutnese Language Rules (courtesy of Wikipedia)

In Tutnese, vowels are pronounced normally, but each consonant is replaced with a syllable from the following table:

Letter	Syllables	Letter	Syllables	Letter	Syllables
B	Bub	K	Kuck	S	Sus
C	Cash	L	Lul	T	Tut
D	Dud	M	Mum	V	Vuv
F	Fuf	N	Nun	W	Wack
G	Gug	P	Pub	X	Ex
H	Hash	Q	Quack	Y	Yub
J	Jay	R	Rug	Z	Zub

Double letters in a word, rather than being repeated, are preceded by the syllable *squa* to indicate doubling. For doubled vowels, the prefix becomes *squat* instead—thus, *OO* would be spoken as *squat-oh* and written as *squato*.

Word Example: "Tree" becomes "Tutrugsquate". Sentence Example: "I took a walk to the park yesterday" becomes "I tutsquatokuck a wackalulkuck tuto tuthashe pubarugkuck yubesustuterugdudayub".

Program Design

Your program should implement the following classes EXACTLY per the specification (including naming and capitalization) below. FAILURE TO FOLLOW THE SPECIFICATION will lead to an automatic 25% deduction in your score. Each class should consist of a .h file and a .cpp file.

The Model Class

You will build a class named Model that will encode the rules of the Tutnese language. The class will contain the following public methods:

- A default constructor
- A default destructor
- translateSingleCharacter – takes a single character as input and returns a string representing its encoding in Tutnese. **Capitalization should be preserved.**
- translateDoubleCharacter – takes a single character as input that appears twice in a row and returns a string representing its encoding in Tutnese. **Capitalization should be preserved.**

The Translator Class

You will build a class named Translator that will translate English sentences to Tutnese sentences using the Model class. The class will contain the following public methods:

- A default constructor
- A default destructor
- translateEnglishWord– takes a single string representing a single English word as input and returns a string representing the Tutnese translation.
- translateEnglishSentence– takes a single string representing a single English sentence as input and returns a string representing the Tutnese translation. Make sure to account for punctuation.

The FileProcessor Class

You will build a class named FileProcessor that will take txt files containing English text and produce a txt file containing the equivalent Tutnese text. The class will contain the following public methods:

- A default constructor
- A default destructor
- processFile – takes a string representing the input file (English) and a string representing the output file (where the Tutnese translation will be written). This method has a void return type.

Your classes may contain any private methods you wish to implement to achieve the requirements of the public methods. String should use the std::string type. (Not c-style char*)

In addition to the above, you should provide a main method in a file called main.cpp. It should only contain a main method. The main method will be invoked with 2 command line arguments representing the input (English) file and the output (Tutnese) file as strings. Your main method should do the following:

- Instantiate a FileProcessor
- Translate the provided input file to Tutnese using the file processor.
- Exit

Rules of Engagement

- You may **NOT** use any non-primitive data structures. (No arrays, Vectors, Lists, etc) Just use individual primitive variables (int, double, etc) and std strings. Hopefully this will convince you that data structures make programs more efficient and easier to write. (Though I suspect you know this already…) Of course, to do the file processing you may use any of the C++ IO classes.
- For this assignment, you must work individually.
- Develop using any IDE you want, but make sure your code runs correctly with g++ using the course docker container. Make sure to provide a Makefile so I can build your code easily. (See Canvas for a sample Makefile.)
- Feel free to use whatever textbooks or Internet sites you want to refresh your memory with C++ IO operations, just cite them in a README file turned in with your code. All code you write, of course, must be your own.

Due Date

This assignment is due at 11:59pm on 9-20-2021. Submit all your commented code to your GitHub repository following the submission instructions discussed in class.

Grading

Grades will be based on correctness, adherence to the guidelines, and code quality (including the presence of meaningful comments). An elegant, OO solution will receive much more credit than procedural spaghetti code. I assume you are familiar with the standard style guide for C++, which you should follow. (See the course page on Canvas for a C++ style guide and Coding Documentation Requirements.)

Again, code that does not follow the specification EXACTLY will receive an automatic 25% deduction. Code that does not compile will receive an automatic 50% deduction.

CPSC 350: Data Structures and Algorithms
Fall 2021

Programming Assignment 2: Tutnese Translation Continued
Due: Sept. 28th, 2021. 11:59pm

The Assignment

In assignment 1 you learned all about Tutnese, and wrote a program to translate files written in English into files written in Tut. Now it's time to go the other way. You will write a program that takes input files in Tut, and outputs their English translation to a file.

The rules of the Tut language are exactly as they were in assignment 1, so refer to that spec as needed for a refresher.

Program Design

For the first assignment we told you exactly how to organize your files to achieve a high level of OO. Not so for this assignment. You have one simple design criteria – to implement this new functionality by reusing what you wrote for assignment 1 and modifying it in the cleanest and most OO way possible. All the classes from assignment 1 should be present, and existing methods should remain untouched. Your goal is to add to these classes and methods to be able to translate from Tut to English.

Your main method will follow the same general outline as assignment 1, with a minor change. **It will now take a 3rd command line argument**, which can be either **"E2T"** or **"T2E".** If the user specifies **"E2T"** the program should translate from English to Tut as in assignment 1. If the user specifies **"T2E"** the translation should be Tut to English.

Again, the goal is to use your existing classes and write the least amount of OO code possible to implement the additional functionality.

Rules of Engagement

- You may **NOT** use any non-primitive data structures. (No arrays, Vectors, Lists, etc) Just use individual primitive variables (int, double, etc) and std strings. Hopefully this will convince you that data structures make programs more efficient and easier to write. (Though I suspect you know this already…) Of course, to do the file processing you may use any of the C++ IO classes.
- For this assignment, you must work individually.
- Develop using any IDE you want, but make sure your code runs correctly with g++ using the course docker container. Make sure to provide a Makefile so I can build your code easily. (See Canvas for a sample Makefile.)
- Feel free to use whatever textbooks or Internet sites you want to refresh your memory with C++ IO operations, just cite them in a README file turned in with your code. All code you write, of course, must be your own.

Due Date

This assignment is due at 11:59pm on 9-28-2021. Submit all your commented code to your GitHub repository following the submission instructions discussed in class.

Grading

Grades will be based on correctness, adherence to the guidelines, and code quality (including the presence of meaningful comments). An elegant, OO solution will receive much more credit than procedural spaghetti code. I assume you are familiar with the standard style guide for C++, which you should follow. (See the course page on Canvas for a C++ style guide and Coding Documentation Requirements.)

Code that does not compile will receive an automatic 50% deduction.