Git TestDesign Document for the Alien Conversion program

By Jacob Boland

# Classes

## Program

This class acts as the entry point to the program and contains static methods that can be accessed without object initialization. It consists of three methods and acts as a menu and controller for the programs runtime.

### Methods:

* static void Main(string [] args): This is the starting point of the program. It begins by entering a while loop which asks user to input the pathname of the text file that contains the values to be converted. This loops until a valid pathname is given or the user enters ‘Q’ or ‘q’ to quit the application. If the pathname is valid it calls the readFile function and then calls contentReader method with the return given by the readFile method.
* private static List<string> readFile(string pathname): Takes a string representing the pathname of a text file and reads it line by line storing each in a list of strings until the end of file has been reached.
* private static void contentReader(List<string> fileContents): This method is responsible for taking the content read from a text file and call the appropriate method in a Converter object.

## Converter

This class is used to define the Converter object which consists of 8 methods and two member variables. It is used to hold the alien translation of each Roman numeral and the credit value of each resource that can be sold. Its methods are responsible for accessing/modifying the values in the numerals and resources dictionary as well as the conversion between alien, Roman, and Arabic numerals.

### Methods:

* public Converter(): Class constructor. Used to initialize an instance of the Converter class. Creates instances of the numerals and resources dictionary. Fill the numerals dictionary with the Roman numeral keys and sets the value to null since the translation is currently unknown (Numerals and resources are set as dictionaries instead of being designed as separate objects because the operations for validity still need to be checked which would be made more difficult if accessing member variables from another object was included).
* public void addAlienTranslation(char romanNumeral, string alienTranslation): Checks that Roman numeral given is valid and if so sets its value the alien translation. If not it throws an exception indicating the numeral is invalid.
* public void addOrUpdateResource(string resource, int credits): Adds a new resource and its credit value to the resources dictionary if it does not already exist. Otherwise it updates the credit value of the resource in the resources dictionary.
* public int getResourceValue(string resource): Returns the credit value of a resource if it exists in the resources dictionary otherwise return 0.
* public int romanToArabic(string romanNumerals): Takes a set of Roman numerals and converts them to the equivalent Arabic number. This is done by going letter by letter in the given string and using a switch statement to determine how much to add to the numeral based on the letter. It also ensures that the Roman numeral given is valid by making sure it does not have three or more of the same letter in a row or an invalid combination of letters. Throws an exception if invalid.
* public string romanToAlien(string romanNumerals): Converts a set of Roman numerals to the corresponding alien value stored in the numerals dictionary. This method is not necessary in the current scope of the project but has been implemented for use in later iterations.
* public string arabicToRoman(int arabicNumber): Converts an Arabic number between 0 and 4000 to its equivalent set of Roman numerals. This is done with a while loop and a series of if statement but could be changed to recursively call itself instead. However, this will still require the extensive if-else conditional statement.
* public string alienToRoman(string alienNumerals): Converts a set of alien numerals to its equivalent set of Roman numerals. This is done by looping through the numerals dictionary and checking each entry for the corresponding alien word. If the word is found it will concatenate the key to the string of Roman numerals otherwise it will throw an exception.

# Assumptions

* Each alien word is unique and no alien word is equivalent to the same Roman numeral. If this is not the case an issue will occur with the alienToRoman method.
* Since only Roman numerals from I to M it is assumes that no set alien numerals will exceed 3999 or be lower than 1.
* All question marks will be separated from the rest of the line by a space bar in the text file.

# Known Issues

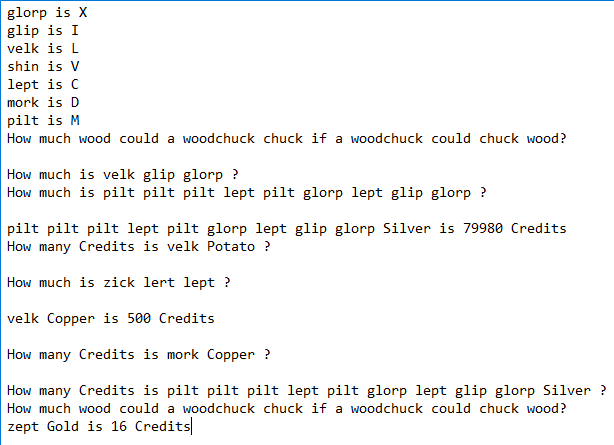
* Since the value of a resource is stored as an integer and not a double some accuracy can be lost in conversion due to the truncation of the decimal place. This can be altered to be supported.
* If the alien numerals are not unique an error will occur in the conversion from alien to Roman numerals. More than one key will be found and multiple alien numerals will be added for one Roman numeral.

# Building and Running the program

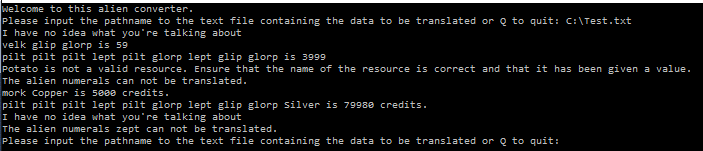
To build the program it is recommended you use the builder provided in Visual Studio 2010 or higher. To run the program either run the program through Visual Studio or run the executable, AlienConverter.exe, found in AlienConverter/bin/Debug/ after building the program through Visual Studio.

# Proof of Completion

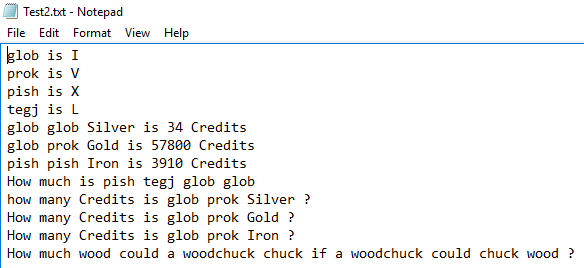
Test 1 Input



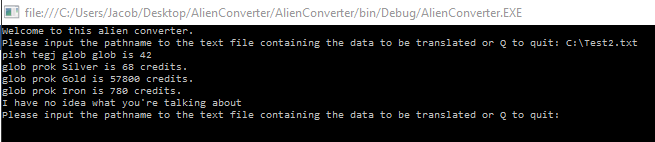
Test 2 Output



Test 2 Input



Test 2 output



Please note that the output of glob prok Iron is 780 not 782 because of the use of integers as mentioned in Known Issues.