Overview

The purpose of this program is to count characters supplied by a data file as an argument when the program is run and then display output containing three points of data: the character, it ASCII code, and the frequency of the character in the file.

### PROCESSING LOGIC

The Main method of the Program class is executed first and performs one and only one function: to call the ControlManager class and pass to it its own arguments.

ControlManager contains the following variables and methods:

private Type ex; //private Type used in another portion of the program

private UserInterface TheUserInterface { get; } // a UserInterface class to handle output

private string CurrentDir { get; } // string, to hold the current directory

public CharacterFrequency[] CharacterFrequencyObjectArray;

public CounterManager(string[] args){  
 Instantiate UserInterface.

check for number of args.

{

Create a custom exception to handle unsupported arguments.

Call ExceptionMessageOutput and pass it the exception ex.

Create a new string[] to hold the single argument we support.

Assign args = newArgs.

}

Set CurrentDir equal to the directory the program ran from.

Call ReadFile(args).

Exit the program.}

private void ReadFile(string[] args)

if args is not null

if the file exists

try to make a string of the full path

use StreamReader to open a stream.

Call ProcessFile.

Catch the exception

Catch the exception

private void ProcessFile(StreamReader file)

create a char[] array

create a string and assign it the contents of the file

assign var chars to hold the contents

Call HandleTheFile and pass it chars

private void HandleTheFile(char[] chars)

This is one of nearly ten versions of the same method I wrote in my attempts to create the counting functionality. None of these worked correctly, several of them crashed the program for no apparent reason, and at least two caused an execution jump to the exceptions area of the UserInterface class without being told to do so. I provide this in lieu of actual documentation because I prefer not to document failure. The single universal sticking point was the apparent impossibility of incrementing the frequency of the characters such that the value remained in place following exit of the loop used to iterate through the array. I was not able to prevent the value from being reset. I have no idea at all why I couldn’t get that to persist or how to prevent it from being reset. Oddly, the first array item **always** persisted outside the loop.

As written, the method creates a counter

foreach (char aChar in chars)

declare a local int, charToCheckIndex, and set it equal to variable aChar, cast to int so we can get the ASCII value of the character.

Declare a for (i = 0; i < chars.Length; i++)

Look for matches between chars[i] and charToCheckIndex. If equal,

CharacterFrequencyObjectArray[charToCheckIndex].IncrementFrequency();

(note that neither this call to the class method nor any variation on a

.Frequency++ declaration, nor any other effort or experiment, resulted in any outcome other than the value being reset upon loop exit. Most efforts resulted in (at best) no apparent change at all; some simply crashed the program.

Instantiate a new CharacterFrequency characterFrequencyObject object.

Set the CharacterFrequencyObjectArray[i] equal to characterFrequencyObject.

Call TheUserInterface.DisplayOutput(CharacterFrequencyObjectArray) in order to display the output from the program to the user.

Pogram output is broken and any output it might produce is missing ASCII codes and frequency,

OTHER CLASSES

class UnsupportedArgsException : Exception

Class to extend the Exception class to provice a way to throw a custom exception related to the user supplying an unsupported argument at runtime.This is one of the few things that works correctly.

class UserInterface

public string Output {get; set; } // supplies messages and program output to the user.

private string Convert(CharacterFrequency[] freq) // intended convert the frequency of a given character to a string so it can be manipulated.

public void ExceptionMessageOutput(Type ex) // Contains a switch/case statement containing a number of possible program exceptions and supplying messages that describe what happened for the user’s benefit. I find I strongly prefer this approach now that I have used it.

public void DisplayOutput(CharacterFrequency[] freqArray) // provides final output to the user. This cde also does not function, probably because storedChr and oneCharFreqObj were broken beyond help.

The main program flow must be supplemented with the flow of subroutines/methods/functions that are called from the main program.

Any specific algorithms to be used should be stated or referenced.

### DATA (INPUT/OUTPUT)

The program accepts a single input in the form of a text file.

The text file is entirely printable ASCII.

Each character in the text file must be found on the ASCII table.

Values of each element are their ASCII values

### COMPONENTS (SOURCE CODE NAMES, CLASSES, METHODS)

Program class- entry point. Contains a single call to ControlManager class.

UnsupportedException class : Esception

Contains three constructors, none of which require implementation due to the use of the switch/case in UserInterface:

public UnsupportedArgsException()

public UnsupportedArgsException(string message)

: base(message)

public UnsupportedArgsException(string message, Exception inner)

: base(message, inner)

ControlManager class: its methods are named and described above.

UserInterface class:

public string Output { get; set; } Stores the Output messages as strings for retrieval.

private string Convert(CharacterFrequency[] freq)

public void ExceptionMessageOutput(Type ex)

public void DisplayOutput(CharacterFrequency[] freqArray) // these are described above.

Include UML class diagram showing the properties and methods of the class.

Describe the properties of the class and any constraints upon their value.

Indicate where the algorithms defined in processing logic are implemented.

### TESTING

Present one or more named scenarios including the input data, expected output and the success criteria desired that will be utilized to test the application. The testing plan should be repeatable so discuss any ‘clean up’ that might be required to do so.

Scenario 1 – output file arg test

Steps to test

Supply an output file argument in addition to the input file when the program is run.

Expected reaction

Expected results is a message saying “You have supplied an argument not supported in this version of \n" + "this program. Writing output to a file may be implemented in a\n "future version.";

Scenario 2 – FileLoadException

Steps to test

Move the input file out of the directory the programs runs from.

Expected results is a message saying "The file has failed to load. Please check the file\n" +"name and path to be certain that they are correct and that " "the file that you specified does in fact exist at that location.\n";

Scenario 3 – ArgumentNullException

Steps to test

Fail to provide any arguments when the program is run.

Expected results is a message saying You have failed to specify an argument.