Overview

A program to count the characters in an ASCII file and display them in another ASCII file named by the user.

Introduction

This document holds the technical design of the CSCI312\_A1\_Schmidt program and serves as a pseudo lessons learned for the changes made to the initial design. The program is intended to process ASCII files and produce another ASCII file that shows the character frequencies of the original file in a fast and efficient method.

## Scope

To count all the characters in a given ASCII file using an array.

To produce a new ASCII file with the produced results.

To be fast an efficient.

### PROCESSING

The user enters an ASCII file address and a desired name for the soon-to-be created ASCII file.

The program then creates an array instance of CharacterFrequency objects with 255 slots. For every character in the given ASCII file address, the program checks if a CharacterFrequency object has been created for that character and if so increments() it, and if not, creates one. Once the program finishes all the characters, it then goes through the CharacterFrequency array and uses StreamWriter to write the CharacterFrequency instances toString() into the new ASCII file.

### DATA

The logical and physical data structure of files should be defined in detail.

Data structure definitions must include the:

description of each element, e.g. name, type, dimension;

relationships between the elements, i.e. the structure;

range of possible values of each element;

initial values of each element.

CharacterFrequency

Object, m\_Ch = byte and m\_Frequency integer.

The byte is the character the CharacterFrequency object represents; the integer is how many times that character was in the ASCII file.

m\_Ch = all possible characters, m\_Frequency > 0

Instance variables: m\_Ch = character used when instanced, m\_Frequency = 1.

### COMPONENTS

CSCI312\_A1\_Schmidt

* Holds the static main and runs the program through the process.

CharacterFrequency

* Data class needed by the program.

|  |
| --- |
| **CharacterFrequency** |
| -ch : char  -frequency : int |
| +getCharacter() : char  +setCharacter(in character : char)  +getFrequency() : int  +setFrequency(in frequency : int)  +increment()  +Equals() : bool  +toString() : string |

### TESTING

Present one or more named scenarios that will be utilized to test the application.

The testing plan should be repeatable.

Describe the scenario in detail, the steps required to execute the test, the input data, the output data, and the success criteria.

Present a summary of the testing scenarios before the details of each scenario.

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1st wap.txt run | Process wap.txt | Fail |
| 2nd wap.txt run | Process wap.txt | Pass |

Scenario #1- mystuff.txt test

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Enter mystuff.txt address | Input: .txt file to be counted |
| 2. | Enter desired file name | Output: .txt file with countings |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | To produce another .txt file with the results. |
| ACTUAL OUTPUT | | Took too long to find out |
| RESULTS – The desired output of a .txt file with the findings was produced. | | PASS |

Scenario #2- wap.txt test

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Enter wap.txt address | Input: .txt file to be counted |
| 2. | Enter desired file name | Output: .txt file with countings |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | To produce another .txt file with the results. |
| ACTUAL OUTPUT | | Took too long to find out |
| RESULTS – The initial algorithm was too inefficient to handle a roughly 3mb .txt file, mystuff.txt was a 1kb .txt file. After 1:30 min of processing the scenario was aborted and the algorithm was fixed. The issue was that the program was initially written to create an instance of the array for every single character in the document, and then check every single character against every instance in the array. | | FAIL |

Scenario #3 – wap.txt test

##### 

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Enter wap.txt address | Input: .txt file to be counted |
| 2. | Enter desired file name | Output: .txt file with countings |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | To produce another .txt file with the results. |
| ACTUAL OUTPUT | | Produced another .txt file with the results. |
| RESULTS – After changing the algorithm to instead create an array instance for all 255 possible ASCII character codes and running every character in the document against its own CharacterFrequency class instance, the wap.txt was processed in under a second. | | PASS |