C++ Test Documentation for each function

**Blue – have not used yet, or have used a slightly different of it**

**Orange = wrote test cases but need to test it with argc and argv, i.e. ./caesarFunction.cpp a**

void zero\_analysis\_array(…);

void character\_count(…);

void print\_analysis\_array(…);

* void print\_array(int \* array, int size) – own – should print the array character to the user
* void max\_index(int \* array, int length) – own – displays the max index for the array
* void print\_vector (vector <char> arrayout) – own – prints the vector in the table arrayout in the function characterFrequency
* vector <char> read file (string input file) – own – function to read the data in the file and put each character in a table which is dynamic table, hence we don’t know the limit of the table size
* letterType \* charactercount (vector <char> array, int size) – own - tests if the character does not exists in the table of characterFrequency then add the letter to the characterFrequency and then count 1. It finds the character in the characterFrequency table, if it exist return it, else return -1
* void print\_characterFrequency (lettertype \* arrayout, int my\_size) – own – prints out character array frequency
* void decrypt(…); [uses the key to decrypt the encrypted string]
* char to\_upper(char input\_char);
* bool is\_alpha(char input\_char);
* bool is\_upper(char input\_char);
* bool is\_alphabet(char input\_char); - my own function

\*\*test each function to make sure it shows what it is supposed to\*\*

bool is\_upper(char input\_char) -> checks if the input character is in Uppercase or not, if it isn’t return False, if it is return True

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Passed? |
| A | 1 | 1 |  |
| c | 0 | 0 |  |
| “ ” [space] | 0 | 0 |  |
| ? | 0 | 0 |  |

bool is\_alpha(char input\_char) -> checks if the input character is alpha or not, if it isn’t return False, if it is return True

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | * Passed? or X |
| A | 1 |  |  |
| C | 1 | 1 |  |
| “ ” | 0 | 0 |  |
| $ | 0 | 0 |  |
| c | 1 | 1 |  |

char to\_upper(char input\_char) -> if the input char is Uppercase then don’t do nothing return the character, else if the character isn’t Uppercase, then put the character in uppercase

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Passed? |
| a | A | A |  |
| zdAA | ZDAA | ZDAA |  |
| Acccv | ACCCV | A |  |
| BravoBE | BRAVOBE | BRAVOBE |  |

bool is\_alphabet(char input\_char) -> function will check if the character is alphabet or not, returning either true if it is alpha character or false if it isn’t.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Passed? |
| A | 1 | 1 |  |
| z | 1 | 1 |  |
| a | 1 | 1 |  |
| Z | 1 | 1 |  |
| “ “ | 0 | 0 | X |
| £ | 0 | 0 | X |

int max\_index(int \*array, int size) – prints out the maximum index in the array with biggest number

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Passed? |
| [1564879] | 6[index] | 6[index] |  |
| [11157863258] | 10[index] | 10[index] |  |
| [aaa] | aaa | aaa | X |
| “ “ | “ “ | “ “ | X |

letterType \* charactercount (vector <char> array, int size)

tests if the character does not exists in the table of characterFrequency then add the letter to the characterFrequency and then count 1. It finds the character in the characterFrequency table, if it exist return it, else return -1

|  |  |  |  |
| --- | --- | --- | --- |
| Input in table characterFrequency | Expected output | Actual output | Passed? |
| helloworld | h🡺 1, L🡺1,  o🡺1, w🡺1, r🡺1, d🡺1 |  |  |
| programming | m🡺 1 | m🡺 1 |  |
| programming | m🡺2 | m🡺2 |  |
| helloworld | L🡺2, | L🡺2, |  |
| helloworld | L🡺3, | L🡺3, |  |
| programming  helloworld | I🡺1  o🡺2, | I🡺1  o🡺2, |  |
| helloworld | e🡺1 | e🡺1 |  |
|  |  |  |  |

* void decrypt(int key, ifstream& inputFile, ofstream& outputFile)
* takes in the key, the encrypted file, decrypts it and writes the output to a file

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |

Figure 1.2 , shows a table of alpha characters with an index beginning at 0, the key would be shifted of which a specific letter shows decrypt character, i.e. ‘P’ with a key of 3 gives a character of ‘M’

|  |  |  |  |
| --- | --- | --- | --- |
| Input string (Letter + Key) | Expected output | Actual output | Passed? |
| “PFT 7” (key = 3) | “MCQ 7” | “MCQ 7 ” |  |
| HADP (key = 3) | EXAM | EXAM |  |
| ZHHN (key = 3) | WEEK | WEEK |  |
| V2 (key = 3) | S2 | S2 |  |
| Aol (key = 7) | the | the |  |
| Ibpsa (key = 7) | built | built |  |
| Puav (key = 7) | into | into |  |
| Hss (key = 7) | all | all |  |
| Yhaoly (key = 7) | Rather | Rather |  |

* void print\_characterFrequency (lettertype \* arrayout, int my\_size) – own – prints out character array frequency
* print character frequency outputs the frequencies of each character in the array by firstly reading the encrypted text in the file

test @ aaaa ==> 4

test @ helloworld = o==> 2 , l==> 3, h==> 1 , e==> 1, w==> 1 , r==> 1, d==> 1

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Passed? |
| Aaaaa | * 5 | * 5 |  |
| helloworld | o==> 2 , l==> 3, h==> 1 , e==> 1, w==> 1 , r==> 1, | o==> 2 , l==> 3, h==> 1 , e==> 1, w==> 1 , r==> 1, |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Passed? |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Passed? |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |