My Project

AUTHOR

Version

05/07/2020

Table of Contents

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

Class Documentation

Caesar Class Reference

`#include <caesar.h>`

Public Member Functions

Caesar ()

original, uses e to find the key

void **print_characterFrequency** (**Lettertype** *arrayout, int my_size)

default constructor, sets shift data to 0

void **decrypt** (int key, ifstream &inputFile, ofstream &outputFile)

protytyep to cprint the frequency of each character

Lettertype * **character_count** (vector< char > array, int size)

prototype for decrypt function , decypt uses the key found in max index to decrypt string

---

Detailed Description

class , which will be used in all fucntions that needs it Class has a default of Private CLASS can be used and OO

Constructor & Destructor Documentation

Caesar::Caesar ()

original, uses e to find the key

the default constructor

Member Function Documentation

Lettertype * Caesar::character_count (vector< char > *array*, int *size*)

prototype for decrypt function , decypt uses the key found in max index to decrypt string

could be in private, hence user will not need it

dynamic memory allocation, using the keyword new

find_character_in_table will find the character in the characterFrequency table, if it exist return it else return -1

index == -1 tests if the character does not exists in the table of characterfrequency

then add the letter to the characterFrequency and then count 1

for character frequency

increment the value of the frequency count of characterFrequency

void Caesar::decrypt (int *key*, ifstream & *inputFile*, ofstream & *outputFile*)

protytyep to cprint the frequency of each character

TEST IF USER WANTS TO DECRYPT FILE OR NOT, IF NOT THEN DO NOTHING, IF YEST DECRYPT FILE AND OUTPUT IT IN ANOTHER FILE.

read file by each character

if the character is an Alphabet

mycharacter = mycharacter - key; ///Apply the shifts

mycharacter = mycharacter - 'Z' + 'A' - 1;

used when the applied shift is negative

///decrypt uppercase characters

if a character is UPPERCASE, i.e. Z then subtract 90 from it and the key

then take the remainder of 26 add that to 90

90 is lowercase Z in ascii value

ADDED ELSE IF STATEMENT

decrypt lowecase characters

if a character isn't uppercase, i.e. a then subtract 122 from it and the key

then take the remainder of 26 add that to 122

122 is lowercase z in ascii value

outputFile.flush();

cout << std::flush;

used to flush the buffer in memory, tells the program to flush the output file;

if the character is not a alphabet (comma, space, !, ...)

outputFile.flush();

cout << std::flush;

void Caesar::print_characterFrequency (Lettertype * *arrayout*, int *my_size*)

default constructor, sets shift data to 0

arrayout is an array of a **Lettertype** and a paremeter of the size of the table

if (arrayout[i].count != -842150451)

add the value of arrayout[i] to based on the letter found

The documentation for this class was generated from the following files:

0 **caesar.h**
1 **caesarfunction.cpp**

Lettertype Class Reference

```
#include <caesar.h>
```

Public Attributes

char **letter**
int **count**

Member Data Documentation

int Lettertype::count

char Lettertype::letter

The documentation for this class was generated from the following file:

2 **caesar.h**

File Documentation

caesar.h File Reference

```
#include <vector>
#include <fstream>
```

Include dependency graph for caesar.h:

IMAGE

This graph shows which files directly or indirectly include this file:

IMAGE

Classes

class **Lettertype**
class **Caesar**

Functions

int **find_character_in_table** (**Lettertype** *array, int size, char letter_to_search)
int **max_index** (**Lettertype** *array, int size)
bool **is_upper** (char input_char)

function prototype

char **to_upper** (char input_char)
int **keyCalculator** (char **LanguageFrequentCharacter**, char mostFrequentCharacter)
bool **is_alpha** (char input_char)

prtototype to find key for character

void **read_decrypted_file** (string outputFile)

function prototype need to be in ***caesar.h*** *later on*

bool **is_alphabet** (char input_char)

read decrypt file prototype

void **print_vector** (vector< char > arrayout)

function prototype for is_alphabet

vector< char > **read_file** (string inputFile)

prototype of print_vector function, whch will print the vector passed as an argument in the funcntion

---

Function Documentation

int find_character_in_table (Lettertype * *array*, int *size*, char *letter_to_search*)

if the letter exists in the table, then return the position of the letter

if letter isn't found in the table then return -1

bool is_alpha (char *input_char*)

prtototype to find key for character

bool is_alphabet (char *input_char*)

read decrypt file prototype

bool is_upper (char *input_char*)

function prototype

casting input character to int , ascii value of input character

int keyCalculator (char *LanguageFrequentCharacter*, char *mostFrequentCharacter*)

convert all characters to uppercases before comparison

gives position of i in the alphabet position

returns absolute value of difference between the two characters in the alphabet without the negative

int max_index (Lettertype * *array*, int *size*)

determines the number of valid values in the array list of characters

dynamic memory allocation to store the results of valid character frequency

reinitialise the value of j reset to 0

determines the number of valid values in the array list of characters

copy the value of the character from previous table into new table

also copy the frequency of that letter from previous table to new table

calculate the max index of the valid character frequency

max changes and becomes the new current highest number of that character

key = keyCalculator(*LanguageFrequentCharacter, validCharacterFrequency[index].letter);

*langfrequency to get a list of most used letters based on figure 1 in assignment

cout << "The key to decrypt : " << key << endl;

cout << "The key to decrypt : " << key << endl;

cout << "The key to decrypt : " << key << endl;

becasue the alphabet start from 0 counting for the array

void print_vector (vector< char > *arrayout*)

function prototype for is_alphabet

gives user information before encrypted text is displayed

arrayout.size used to limit the for loop

output each character character with ' | '

void read_decrypted_file (string *outputFile*)

function prototype need to be in **caesar.h** later on

**

reads the decrypted file to print out to the user

vector<char> read_file (string *inputFile*)

prototype of print_vector function, whch will print the vector passed as an argument in the funcntion

void test(); vector, type of table i.e. char and name of table for dynamic tbales

declaring an empty dynamic table, to store the characterss tha are in the file

declaring file to read it using ifstream

getline is used to read the file line by line

if statement is used to check if the file is open or not, if the file isn't open, error message is sent,

if it is open, the while loop will run and display what is in the file

the method in the while loop allows us to get the file, read the file and store the results.

read each charater

cout << my_character;

must store the text in a table

then return the data of the table i.e. origin_table

use this then to order characters between highest and lowest

displayed if the file could not open

closes file

char to_upper (char *input_char*)

if the chracter input is in uppercase then don't do nothing return the character,

32 is the difference between A and a and Z and z on the ascii table

caesarfunction.cpp File Reference

```
#include <iostream>
#include <string>
#include <cctype>
#include <locale>
#include <fstream>
#include <math.h>
#include "caesar.h"
#include <vector>
```

Include dependency graph for caesarfunction.cpp:

IMAGE

Functions

int **find_character_in_table** (**Lettertype** *array, int size, char letter_to_search)
vector< char > **read_file** (string inputFile)

prototype of print_vector function, whch will print the vector passed as an argument in the funcntion

bool **is_upper** (char input_char)

function prototype

char **to_upper** (char input_char)
bool **is_alpha** (char input_char)

prtototype to find key for character

bool **is_alphabet** (char input_char)

read decrypt file prototype

int **max_index** (**Lettertype** *array, int size)
void **read_decrypted_file** (string outputFile)

*function prototype need to be in **caesar.h** later on*

int **keyCalculator** (char **LanguageFrequentCharacter**, char mostFrequentCharacter)
void **print_vector** (vector< char > arrayout)

function prototype for is_alphabet

Variables

char **alphabet** [26] = { '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' }

library used to limit the scope of variables in a program

const char **LanguageFrequentCharacter** = 'e'

const char LanguageFrequentCharacter [26] = { ' e ', ' t ', ' a ', ' o ', ' i ' , ' n ' , ' r ' , ' h ' , ' l ' , ' d ' , ' c ' , ' u ' , ' m ', ' f ', ' p ', ' g ' , ' w ', ' y ' ,' b ' ,' v ' ,' k ' ,' x ' ,' j ' ,' q ' ,' z ' };

Function Documentation

int find_character_in_table (Lettertype * *array*, int *size*, char *letter_to_search*)

if the letter exists in the table, then return the position of the letter

if letter isn't found in the table then return -1

bool is_alpha (char *input_char*)

prtototype to find key for character

bool is_alphabet (char *input_char*)

read decrypt file prototype

bool is_upper (char *input_char*)

function prototype

casting input character to int , ascii value of input character

int keyCalculator (char *LanguageFrequentCharacter*, char *mostFrequentCharacter*)

convert all characters to uppercases before comparison

gives position of i in the alphabet position

returns absolute value of difference between the two characters in the alphabet without the negative

int max_index (Lettertype * *array*, int *size*)

determines the number of valid values in the array list of characters

dynamic memory allocation to store the results of valid character frequency

reinitialise the value of j reset to 0

determines the number of valid values in the array list of characters

copy the value of the character from previous table into new table

also copy the frequency of that letter from previous table to new table

calculate the max index of the valid character frequency

max changes and becomes the new current highest number of that character

key = keyCalculator(*LanguageFrequentCharacter, validCharacterFrequency[index].letter);

*langfrequency to get a list of most used letters based on figure 1 in assignment

cout << "The key to decrypt : " << key << endl;

cout << "The key to decrypt : " << key << endl;

cout << "The key to decrypt : " << key << endl;

becasue the alphabet start from 0 counting for the array

void print_vector (vector< char > *arrayout*)

function prototype for is_alphabet

gives user information before encrypted text is displayed

arrayout.size used to limit the for loop output each character character with ' | '

void read_decrypted_file (string *outputFile*)

function prototype need to be in **caesar.h** later on

**

reads the decrypted file to print out to the user

vector<char> read_file (string *inputFile*)

prototype of print_vector function, whch will print the vector passed as an argument in the funcntion

void test(); vector, type of table i.e. char and name of table for dynamic tbales

declaring an empty dynamic table, to store the characterss tha are in the file

declaring file to read it using ifstream

getline is used to read the file line by line

if statement is used to check if the file is open or not, if the file isn't open, error message is sent,

if it is open, the while loop will run and display what is in the file

the method in the while loop allows us to get the file, read the file and store the results.

read each charater

cout << my_character;

must store the text in a table

then return the data of the table i.e. origin_table

use this then to order characters between highest and lowest

displayed if the file could not open

closes file

char to_upper (char *input_char*)

if the chracter input is in uppercase then don't do nothing return the character,

32 is the difference between A and a and Z and z on the ascii table

---

Variable Documentation

**char alphabet[26] =
{ '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' }**

library used to limit the scope of variables in a program

const char LanguageFrequentCharacter = 'e'

const char LanguageFrequentCharacter [26] = { ' e ', ' t ', ' a ', ' o ', ' i ' , ' n ' , ' r ' , ' h ' , ' l ' , ' d ' , ' c ' , ' u ' , ' m ', ' f ', ' p ', ' g ' , ' w ', ' y ' , ' b ' , ' v ' , ' k ' , ' x ' , ' j ' , ' q ' , ' z ' };

a list of keys should be provided to the user, for them to choose a key to decrypt the text need to give a list of keys to the user, to allow them to choose a key language frequency char of [26] elelemt array uses all those letter as they are most used letters in english alphabet, to find the key

main.cpp File Reference

```
#include <iostream>
#include <string>
#include <cctype>
#include <locale>
#include <fstream>
#include <vector>
#include <math.h>
#include <stdlib.h>
#include "caesar.h"
```

Include dependency graph for main.cpp:

IMAGE

Functions

int **main** (int argc, char *argv[])

argc is the number of arguments , argv is the vector i.e the string of the array

---

Function Documentation

int main (int *argc*, char * *argv*[])

argc is the number of arguments , argv is the vector i.e the string of the array

declaration of variable key to store the key returned by max_index or user's own key

declaration of tables

storing value of max index returned by max_index function

ASK USER TO GIVE EXTENSION OF FILE LOCATION FOR ENCRYPTED FILE AND DECRYPT,

IF FILE EXTENSION GIVEN ISN'T VALID THEN PROMPT USER UNTIL CORRECT FILE EXTENSION IS ENTERED

IF CORRECT FILE IS ENTERED, THEN USE THAT LOCATION OF ENCRYPTED TEXT AND DECRYPT IT SEND IT TO A FILE

calling of default constructor

closes the program if can't open the file to read it

closes the program if can't open to write to the file }

the table arrayout will contain the results of the characters frequencies found in the function read file, characters without any numbers, spaces, symbols

prints the vector of characters returned by the read file function

this loop will continue until the user enters a digit

convert a char into a int

outputs the frequency of the characters in a table

key = max_index(characterFrequency, size);

caesar_main.character_count , caesar_main is used to access public member of the class **Caesar**,

caesar_main contains the methods (functions) of a class, character_count
outputs the frequency of the characters in a table
convert a char into int , using the user's input
gives the use choice to return back to the program and try again
cout << std::flush;
exit 1 will return to main menu
to read a file - ifstream
again = 1 will return to program menu
again = 0 means the user can exit the program

Index

INDE