Vigenère cipherdecoder

1.0

Generated by Doxygen 1.9.6

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 shift_value Struct Reference

A struct containing shift.

`#include <structure.h>`

Public Attributes

- int Structshift

3.1.1 Detailed Description

A struct containing shift.

3.1.2 Member Data Documentation

3.1.2.1 Structshift

`int shift_value::Structshift`

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

- structure.h

Chapter 4

File Documentation

4.1 function.cpp File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <map>
#include <deque>
#include <iomanip>
#include <cmath>
#include "function.h"
#include "structure.h"
```

Functions

- void action (int &number, int &can_decrypt_files, char *params[], const std::vector< std::string > &inputSw, const std::vector< std::string > &inputFi, bool &switch_case_one, bool &can_decrypt_switches)

  A function that is responsible for checking the correctness of program input data.
- void print_Instruction ()

  A function that prints short program instructions.
- bool switches_check (const std::vector< std::string > &input_value, bool &Option_One)

  A function that checks the correctness of program input switches.
- int Valid_file (const std::string &File_Name)

  A function that checks the correctness of input file.
- bool is_Alphabetic_Character (char &letter)

  A function that dtermines if input characte is alphabetic.
- std::deque< double > count_Letters (const std::string &file_name)

  A function that calculates the average of the number of each letter in the input file.
- std::deque< std::deque< char > > create_deque_Sepparated_letters (const std::string &file_name, const int &spacing)

  A function that generates deques containing letters from the input file.
- std::deque< double > count_Letters (std::deque< char > &letters)

  A function that calculates the average of the number of each letter in the input deque.
- shift_value find_shift (std::deque< double > &sample_average, std::deque< double > &decrypted_average)

A function that looks for an shift between a single letter in the encoded file and a letter in the plaintext file using their averages.

- double Sum_of_shift (std::vector< double > &vector_Of_averages)

  A functionthat sums all values from the input vector.
- shift_value min_difference (const std::vector< double > &vector_Of_averages)

  A function that finds the smallest value in the input vector.
- int findkeyLength (const std::string &file_name)

  A function that determines the most likely length of the key used to encrypt the message.
- double calculate_index_of_coincedence (std::deque< double > input_averages)

  A function hat calculate index of coincedence of the input deque.
- void decrypt (std::string &Return_File_Name, std::vector< shift_value > &key, std::string &Input_File_name, std::string &Key_file)

  A function that decrypts an encrypted message based on a previously found shift.

4.1.1 Function Documentation

4.1.1.1 action()

```
void action (
            int & number,
            int & can_decrypt_files,
            char * params[],
            const std::vector< std::string > & inputSw,
            const std::vector< std::string > & inputFi,
            bool & switch_case_one,
            bool & can_decrypt_switches )
```

A function that is responsible for checking the correctness of program input data.

Parameters

number	holds number of input parameters
can_decrypt_files	holds number of correct files on which the program operates.
params	holds input parameters
inputSw	vector of input switches
inputFi	vector of input and output files
switch_case_one	bool confirming that the first combination of input switches is met
can_decrypt_switches	bool which confirms that all input switches are valid.

Returns

Function does not return any value

```
00017 {
00018     // Printiong short manual of program
00019     if (number == 1)
00020     {
00021         print_Instruction();
00022     }
00023
00024     // when all inputs entered
```

```
00025     else if (number == 9)
00026     {
00027         std::cout « "Number of parameters is OK" « std::endl;
00028 
00029         can_decrypt_switches = switches_check(inputSw, switch_case_one);
00030 
00031         for (int i = 0; i < inputFi.size(); i++)
00032         {
00033             can_decrypt_files += Valid_file(inputFi[i]);
00034         }
00035 
00036     }
00037 
00038     // Wrong number of inputs
00039     else
00040     {
00041         std::cout « "Wrong number of input parameters!";
00042     }
00043 
00044 
00045 }
```

4.1.1.2 calculate_index_of_coincedence()

```
double calculate_index_of_coincedence (
            std::deque< double > input_averages )
```

A function hat calculate index of coincedence of the input deque.

Parameters

input_averages	deque containing average occurance of every alphabetic character.

Returns

Function returns index of coincede of input deque.

```
00386 {
00387     double index = 0;
00388     for (int i = 0; i < input_averages.size(); i++)
00389     {
00390         index += ((input_averages[i] * input_averages[i]));
00391 
00392     }
00393     return index;
00394 
00395 }
```

4.1.1.3 count_Letters() [1/2]

```
std::deque< double > count_Letters (
            const std::string & file_name )
```

A function that calculates the average of the number of each letter in the input file.

Parameters

file_Name	name of the file containing the text whose average number of letters will be counted

Returns

The function returns a deque containing the average of the occurrences.

```
00145 {
00146     const int SIZE = 'z' - 'a' + 1;
00147     std::deque<double> Letters_Counted;
00148     std::ifstream file(file_name);
00149     int file_size = 0;
00150     Letters_Counted.assign(26,0);
00151     bool exists = false;
00152
00153     if (file)
00154     {
00155
00156         char letterread = ' ', letter = ' ';
00157         while (file » letterread)
00158         {
00159             letter = tolower(letterread);
00160             exists = is_Alphabetic_Character(letter);
00161
00162             if (exists)
00163             {
00164                 file_size++;
00165                 Letters_Counted[letter - 'a']++;
00166             }
00167         }
00168         for (int i = 0; i < Letters_Counted.size(); i++)
00169         {
00170             Letters_Counted[i] = Letters_Counted[i] / file_size;
00171         }
00172
00173     }
00174     return Letters_Counted;
00175 }
```

4.1.1.4 count_Letters() [2/2]

```
std::deque< double > count_Letters (
            std::deque< char > & letters )
```

A function that calculates the average of the number of each letter in the input deque.

Parameters

letters	name of the deque containing the encrypted message which letter average we want to calculate.

Returns

The function returns a deque containing the average of the occurrences.

```
00223 {
00224     std::deque<double> Letters_Counted;
00225     Letters_Counted.assign(26, 0);
00226
00227     for (int i = 0; i < letters.size(); i++)
00228     {
00229         Letters_Counted[letters[i] - 'a']++;
00230     }
00231     for (int i = 0; i < Letters_Counted.size(); i++)
00232     {
00233         Letters_Counted[i] = Letters_Counted[i] / letters.size();
00234     }
00235
00236     return Letters_Counted;
00237 }
```

4.1.1.5 create_deque_Sepparated_letters()

```
std::deque< std::deque< char > > create_deque_Sepparated_letters (
            const std::string & file_name,
            const int & spacing )
```

A function that generates deques containing letters from the input file.

these letters are divided by spaces of length n

Parameters

file_Name	name of the file containing the text from which characters will be taken
spacing	is the value which represents te leng of spaces between letters.

Returns

The function returns a deque containing deques with separated characters.

```
00178 {
00179     std::ifstream file(file_name);
00180     bool exists = false;
00181     std::deque<char> letters_separated;
00182     std::deque<std::deque<char> > letters_separated_deques;
00183 
00184     if (file)
00185     {
00186         char letterread = ' ', letter = ' ';
00187         while (file » letterread)
00188         {
00189             letter = tolower(letterread);
00190             exists = is_Alphabetic_Character(letter);
00191 
00192             if (exists)
00193             {
00194                 letters_separated.push_back(letter);
00195             }
00196         }
00197 
00198     }
00199 
00200     for (int j = 0; j < spacing; j++)
00201     {
00202         std::deque<char> separated_chars;
00203 
00204 
00205         for (int i = 0; i < letters_separated.size(); i += spacing)
00206         {
00207             separated_chars.push_back(letters_separated[i]);
00208         }
00209 
00210 
00211         letters_separated_deques.push_back(separated_chars);
00212 
00213 
00214         letters_separated.pop_front();
00215 
00216     }
00217 
00218 
00219     return letters_separated_deques;
00220 }
```

4.1.1.6 decrypt()

```
void decrypt (
            std::string & Return_File_Name,
```

```
            std::vector< shift_value > & key,
            std::string & Input_File_name,
            std::string & Key_file )
```

A function that decrypts an encrypted message based on a previously found shift.

Parameters

Return_File_Name	name of the file where the decrypted message is to be saved
key	vector with strict shift containing shift value of every key character.
Input_File_name	the name of the file we want to decrypt.
Key_file	file where the key to decrypt the message is to be saved

Returns

Function does not return any value

```
00398 {
00399     std::ifstream input_file(Input_File_name);
00400     std::ofstream output_file(Return_File_Name);
00401     std::ofstream key_file(Key_file);
00402     const int sta = 1;
00403 
00404     int shift = 0,incrementer = 0;
00405 
00406     if (input_file && output_file)
00407     {
00408         std::string line = "";
00409         char letter = ' ', helper =' ';
00410         while (std::getline(input_file, line, '\0')) {
00411             bool exists = false;
00412             incrementer = 0;
00413 
00414             for (int i = 0; i < line.size(); i++) {
00415 
00416                 letter = tolower(line[i]);
00417                 exists = is_Alphabetic_Character(letter);
00418 
00419                 if (exists)
00420                 {
00421                     shift = key[incrementer % key.size()].Structshift;
00422 
00423                     if (int(letter - shift) < int('a'))
00424                     {
00425                         helper = char(int('z' - (shift - (letter - 'a') - sta)));
00426                     }
00427                     else
00428                     {
00429                         helper = char(int(letter - shift));
00430 
00431                     }
00432                     output_file « helper;
00433                     incrementer++;
00434                 }
00435                 else
00436                 {
00437                     output_file « letter;
00438                 }
00439             }
00440         }
00441 
00442         std::cout « "Decryption successful! Check file: " « Return_File_Name « " to see decrypted text
      and file: " «Key_file « " to see shift." « std::endl;
00443 
00444     }
00445     key_file « "Shift is equal: ";
00446     for (int i = 0; i < key.size(); i++)
00447     {
00448 
00449         key_file « key[i].Structshift « " ";
00450 
00451     }
00452     key_file « " Key size is equal: " « key.size();
00453 
00454     input_file.close();
00455     output_file.close();
00456     key_file.close();
```

```
00457
00458
00459 }
```

4.1.1.7 find_shift()

```
shift_value find_shift (
            std::deque< double > & sample_average,
            std::deque< double > & decrypted_average )
```

A function that looks for an shift between a single letter in the encoded file and a letter in the plaintext file using their averages.

Parameters

sample_average	deque cantaining average of the occurrences of sample file.
decrypted_average	deque cantaining average of the occurrences of decrypted file.

Returns

The function returns an int that corresponds to the shift of the encrypted message

```
00240 {
00241     size_t cycles = sample_average.size();
00242     int  step = 0;
00243
00244     double suma_z_roznic_pomiedzy_2_dequami = 0;
00245     std::vector<double> sumy;
00246     shift_value shift_min;
00247
00248
00249
00250
00251     while (step < cycles)
00252     {
00253
00254         std::vector<double> WektorRoznic;
00255
00256
00257         for (int i = 0; i < sample_average.size(); i++)
00258         {
00259             double absolute_value = fabs(decrypted_average[i] - sample_average[i]);
00260
00261             WektorRoznic.push_back(absolute_value);
00262         }
00263
00264             double holder = 0;
00265             holder = sample_average.back();
00266             sample_average.pop_back();
00267             sample_average.push_front(holder);
00268
00269         suma_z_roznic_pomiedzy_2_dequami = Sum_of_shift(WektorRoznic);
00270
00271         sumy.push_back(suma_z_roznic_pomiedzy_2_dequami);
00272
00273         step++;
00274     }
00275
00276     shift_min = min_difference(sumy);
00277
00278     return shift_min;
00279 }
```

4.1.1.8 findkeyLength()

```
int findkeyLength (
            const std::string & file_name )
```

A function that determines the most likely length of the key used to encrypt the message.

Parameters

file_name	name of the file containing the encrypted message whose key length we are looking for.

Returns

the function returns an int which is the most likely length of the searched key.

```
00314 {
00315     std::ifstream file(file_name);
00316     int file_size = 0;
00317     char letterread = ' ', letter = ' ';
00318     std::deque<std::deque<double» coincedence_indexes;
00319     std::deque<double> coincedence_indexes_of_spacing_i;
00320     const double proportion = 0.40;
00321
00322     while (file » letterread)
00323     {
00324         letter = tolower(letterread);
00325         bool exists = is_Alphabetic_Character(letter);
00326
00327         if (exists)
00328         {
00329             file_size++;
00330         }
00331     }
00332
00333
00334
00335
00336     for (int i = 1; i < (file_size / 2) + 1; i++)
00337     {
00338         std::deque<std::deque<char» letters_separated = create_deque_Sepparated_letters(file_name, i);
00339
00340         std::deque<std::deque<double» averages_counted;
00341
00342         for (int j = 0; j < letters_separated.size(); j++)
00343         {
00344                 averages_counted.push_back(count_Letters(letters_separated[j]));
00345         }
00346
00347         double average_of_index = 0;
00348
00349         for (int k = 0; k < averages_counted.size(); k++)
00350         {
00351             average_of_index += calculate_index_of_coincedence(averages_counted[k]);
00352         }
00353
00354         average_of_index = (average_of_index / averages_counted.size());
00355         coincedence_indexes_of_spacing_i.push_back(average_of_index);
00356
00357         if (coincedence_indexes_of_spacing_i.size() > 2)
00358         {
00359             for (int k = 1; k < coincedence_indexes_of_spacing_i.size(); k++)
00360             {
00361                 double compare = (coincedence_indexes_of_spacing_i[k - 1] +=
      (coincedence_indexes_of_spacing_i[k - 1] * proportion));
00362
00363                 if (compare < coincedence_indexes_of_spacing_i[k])
00364                 {
00365                     int key = k + 1;
00366
00367                     //std::cout «"Key: "« key « std::endl;
00368
00369                     return key;
00370                 }
00371
00372             }
00373         }
00374
00375     }
```

```
00376     /*for (int k = 0; k < coincedence_indexes_of_spacing_i.size(); k++)
00377     {
00378         std::cout « coincedence_indexes_of_spacing_i[k] « " ";
00379     }
00380     std::cout « std::endl;*/
00381     //std::cout « "Key: 1" « std::endl;
00382     return 1;
00383 }
```

4.1.1.9 is_Alphabetic_Character()

```
bool is_Alphabetic_Character (
             char & letter )
```

A function that dtermines if input characte is alphabetic.

Parameters

letter	character we evaluate.

Returns

true if input character is alphabetic

```
00135 {
00136     char allowed_Letters[] = {
      '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'
      };
00137     char letterevaluated = ' ';
00138
00139     bool exists = std::find(std::begin(allowed_Letters), std::end(allowed_Letters), letter) !=
      std::end(allowed_Letters);
00140
00141     return exists;
00142 }
```

4.1.1.10 min_difference()

```
shift_value min_difference (
             const std::vector< double > & vector_Of_averages )
```

A function that finds the smallest value in the input vector.

Parameters

vector_Of_averages	the vector in which we want to find the smallest value.

Returns

Function return smallest value in vector

```
00294 {
00295     double comparing_min_value = 100;
00296     int shift = 0;
00297     shift_value p;
00298
```

```
00299     for (int i = 0; i < vector_Of_averages.size(); i++)
00300     {
00301         if (vector_Of_averages[i] < comparing_min_value)
00302         {
00303             shift = i;
00304             comparing_min_value = vector_Of_averages[i];
00305         }
00306     }
00307     p.Structshift = shift;
00308     p.min_value = comparing_min_value;
00309     return p;
00310
00311 }
```

4.1.1.11 print_Instruction()

```
void print_Instruction ( )
```

A function that prints short program instructions.

Parameters

Function	does not require any parameters

Returns

Function does not return any value

```
00048 {
00049     std::cout « "Program breaks a cyphertext encrypted with an unknown key with the Vigenere method."
      « std::endl;
00050     std::cout « "The program elaborates the unknown key and decrypts the encrypted file.\n The program
      is run in command line with switches:" « std::endl;
00051     std::cout « " -i input text file (cyphertext)\n -w sample text in the same language as the
      cyphertext \n -k output text file with the elaborated key \n -o output text file (plainext)" «
      std::endl;
00052
00053 }
```

4.1.1.12 Sum_of_shift()

```
double Sum_of_shift (
            std::vector< double > & vector_Of_averages )
```

A functionthat sums all values from the input vector.

Parameters

vector_Of_averages	the vector in which we want to sum all the values

Returns

Function return sum of vector

```
00282 {
00283     double added_avg = 0, final_average = 0;
```

```
00284
00285     for (int i = 0; i < vector_Of_averages.size(); i++)
00286     {
00287         added_avg += vector_Of_averages[i];
00288     }
00289     final_average = (added_avg/vector_Of_averages.size());
00290     return final_average;
00291 }
```

4.1.1.13 switches_check()

```
bool switches_check (
            const std::vector< std::string > & input_value,
            bool & Option_One )
```

A function that checks the correctness of program input switches.

Parameters

input_value	vector of input switches, the correctness of which we check
Option_One	bool evaluates whether we have encountered the first possibility of switches

Returns

Function return true if all input switches are correct

```
00056 {
00057     std::string Possibility_One[4] = {"-i","-o","-w","-k"};
00058     std::string Possibility_Two[4] = {"-o","-w","-k","-i"};
00059     int Return_true = 0;
00060
00061     if (input_value[0] == "-i")
00062     {
00063         Return_true += 1;
00064         Option_One = true;
00065         for (int i = 1; i < input_value.size(); i++)
00066         {
00067             if (input_value[i] != Possibility_One[i])
00068             {
00069                 std::cout « "Wrong switch: " « input_value[i] « std::endl;
00070             }
00071             else
00072             {
00073                 Return_true += 1;
00074             }
00075         }
00076         if (Return_true == 4)
00077         {
00078             return true;
00079         }
00080         else
00081             return false;
00082     }
00083     else if (input_value[0] == "-o")
00084     {
00085         Return_true += 1;
00086         for (int i = 1; i < input_value.size(); i++)
00087         {
00088             if (input_value[i] != Possibility_Two[i])
00089             {
00090                 std::cout « "Wrong switch: " « input_value[i] « std::endl;
00091             }
00092             else
00093             {
00094                 Return_true += 1;
00095             }
00096         }
00097         if (Return_true == 4)
00098         {
00099             return true;
```

```
00100         }
00101         else
00102             return false;
00103     }
00104     else
00105     {
00106         std::cout « "Switch Error! you typed: " « input_value[0] « std::endl;
00107         for (int i = 1; i < input_value.size(); i++)
00108         {
00109             if (input_value[i] != Possibility_One[i] || input_value[i] != Possibility_One[i])
00110             {
00111                 std::cout « "Switch Error! you typed: " « input_value[i] « std::endl;
00112             }
00113         }
00114         return false;
00115     }
00116
00117 }
```

4.1.1.14 Valid_file()

```
int Valid_file (
            const std::string & File_Name )
```

A function that checks the correctness of input file.

Parameters

File_Name	the name of the file we are checking for correctness

Returns

Function return 1 if input file exists if not then 0

```
00119                                           {
00120
00121     int cycle_counter = 0;
00122
00123     std::fstream file(File_Name);
00124     if (!file)
00125         std::cout « "File " « File_Name « " can't be found! Check if name is proper and/or if file
      exists." « std::endl;
00126     else
00127     {
00128         cycle_counter += 1;
00129         file.close();
00130     }
00131     return cycle_counter;
00132 }
```

4.2 function.h File Reference

```
#include "structure.h"
```

Macros

- #define FUNCTION_H

Functions

- void action (int &number, int &can_decrypt_files, char *params[], const std::vector< std::string > &inputSw, const std::vector< std::string > &inputFi, bool &switch_case_one, bool &can_decrypt_switches)

  A function that is responsible for checking the correctness of program input data.
- void print_Instruction ()

  A function that prints short program instructions.
- bool switches_check (const std::vector< std::string > &input_value, bool &Option_One)

  A function that checks the correctness of program input switches.
- int Valid_file (const std::string &File_Name)

  A function that checks the correctness of input file.
- std::deque< double > count_Letters (const std::string &file_name)

  A function that calculates the average of the number of each letter in the input file.
- std::deque< std::deque< char > > create_deque_Sepparated_letters (const std::string &file_name, const int &spacing)

  A function that generates deques containing letters from the input file.
- std::deque< double > count_Letters (std::deque< char > &letters)

  A function that calculates the average of the number of each letter in the input deque.
- shift_value find_shift (std::deque< double > &sample_average, std::deque< double > &decrypted_average)

  A function that looks for an shift between a single letter in the encoded file and a letter in the plaintext file using their averages.
- double Sum_of_shift (std::vector< double > &vector_Of_averages)

  A functionthat sums all values from the input vector.
- shift_value min_difference (const std::vector< double > &vector_Of_averages)

  A function that finds the smallest value in the input vector.
- bool is_Alphabetic_Character (char &letter)

  A function that dtermines if input characte is alphabetic.
- double calculate_index_of_coincedence (std::deque< double > input_averages)

  A function hat calculate index of coincedence of the input deque.
- int findkeyLength (const std::string &file_name)

  A function that determines the most likely length of the key used to encrypt the message.
- void decrypt (std::string &Return_File_Name, std::vector< shift_value > &key, std::string &Input_File_name, std::string &Key_file)

  A function that decrypts an encrypted message based on a previously found shift.

4.2.1 Macro Definition Documentation

4.2.1.1 FUNCTION_H

```
#define FUNCTION_H
```

4.2.2 Function Documentation

4.2.2.1 action()

```
void action (
             int & number,
             int & can_decrypt_files,
             char * params[],
             const std::vector< std::string > & inputSw,
             const std::vector< std::string > & inputFi,
             bool & switch_case_one,
             bool & can_decrypt_switches )
```

A function that is responsible for checking the correctness of program input data.

Parameters

number	holds number of input parameters
can_decrypt_files	holds number of correct files on which the program operates.
params	holds input parameters
inputSw	vector of input switches
inputFi	vector of input and output files
switch_case_one	bool confirming that the first combination of input switches is met
can_decrypt_switches	bool which confirms that all input switches are valid.

Returns

Function does not return any value

```
00017 {
00018     // Printiong short manual of program
00019     if (number == 1)
00020     {
00021         print_Instruction();
00022     }
00023
00024     // when all inputs entered
00025     else if (number == 9)
00026     {
00027         std::cout « "Number of parameters is OK" « std::endl;
00028
00029         can_decrypt_switches = switches_check(inputSw, switch_case_one);
00030
00031         for (int i = 0; i < inputFi.size(); i++)
00032         {
00033             can_decrypt_files += Valid_file(inputFi[i]);
00034         }
00035
00036     }
00037
00038     // Wrong number of inputs
00039     else
00040     {
00041         std::cout « "Wrong number of input parameters!";
00042     }
00043
00044
00045 }
```

4.2.2.2 calculate_index_of_coincedence()

```
double calculate_index_of_coincedence (
             std::deque< double > input_averages )
```

A function hat calculate index of coincedence of the input deque.

Parameters

input_averages	deque containing average occurance of every alphabetic character.

Returns

Function returns index of coincede of input deque.

```
00386 {
00387     double index = 0;
00388     for (int i = 0; i < input_averages.size(); i++)
00389     {
00390         index += ((input_averages[i] * input_averages[i]));
00391 
00392     }
00393     return index;
00394 
00395 }
```

4.2.2.3 count_Letters() [1/2]

```
std::deque< double > count_Letters (
            const std::string & file_name )
```

A function that calculates the average of the number of each letter in the input file.

Parameters

file_Name	name of the file containing the text whose average number of letters will be counted

Returns

The function returns a deque containing the average of the occurrences.

```
00145 {
00146     const int SIZE = 'z' - 'a' + 1;
00147     std::deque<double> Letters_Counted;
00148     std::ifstream file(file_name);
00149     int file_size = 0;
00150     Letters_Counted.assign(26,0);
00151     bool exists = false;
00152 
00153     if (file)
00154     {
00155 
00156         char letterread = ' ', letter = ' ';
00157         while (file » letterread)
00158         {
00159             letter = tolower(letterread);
00160             exists = is_Alphabetic_Character(letter);
00161 
00162             if (exists)
00163             {
00164                 file_size++;
00165                 Letters_Counted[letter - 'a']++;
00166             }
00167         }
00168         for (int i = 0; i < Letters_Counted.size(); i++)
00169         {
00170             Letters_Counted[i] = Letters_Counted[i] / file_size;
00171         }
00172 
00173     }
00174     return Letters_Counted;
00175 }
```

4.2.2.4 count_Letters() [2/2]

```
std::deque< double > count_Letters (
            std::deque< char > & letters )
```

A function that calculates the average of the number of each letter in the input deque.

Parameters

letters	name of the deque containing the encrypted message which letter average we want to calculate.

Returns

The function returns a deque containing the average of the occurrences.

```
00223 {
00224     std::deque<double> Letters_Counted;
00225     Letters_Counted.assign(26, 0);
00226
00227     for (int i = 0; i < letters.size(); i++)
00228     {
00229         Letters_Counted[letters[i] - 'a']++;
00230     }
00231     for (int i = 0; i < Letters_Counted.size(); i++)
00232     {
00233         Letters_Counted[i] = Letters_Counted[i] / letters.size();
00234     }
00235
00236     return Letters_Counted;
00237 }
```

4.2.2.5 create_deque_Sepparated_letters()

```
std::deque< std::deque< char > > create_deque_Sepparated_letters (
            const std::string & file_name,
            const int & spacing )
```

A function that generates deques containing letters from the input file.

these letters are divided by spaces of length n

Parameters

file_Name	name of the file containing the text from which characters will be taken
spacing	is the value which represents te leng of spaces between letters.

Returns

The function returns a deque containing deques with separated characters.

```
00178 {
00179     std::ifstream file(file_name);
00180     bool exists = false;
00181     std::deque<char> letters_separated;
00182     std::deque<std::deque<char> > letters_separated_deques;
00183
00184     if (file)
00185     {
00186         char letterread = ' ', letter = ' ';
```

```
00187         while (file » letterread)
00188         {
00189             letter = tolower(letterread);
00190             exists = is_Alphabetic_Character(letter);
00191 
00192             if (exists)
00193             {
00194                 letters_separated.push_back(letter);
00195             }
00196         }
00197 
00198     }
00199 
00200     for (int j = 0; j < spacing; j++)
00201     {
00202         std::deque<char> separated_chars;
00203 
00204 
00205         for (int i = 0; i < letters_separated.size(); i += spacing)
00206         {
00207             separated_chars.push_back(letters_separated[i]);
00208         }
00209 
00210 
00211         letters_separated_deques.push_back(separated_chars);
00212 
00213 
00214         letters_separated.pop_front();
00215 
00216     }
00217 
00218 
00219     return letters_separated_deques;
00220 }
```

4.2.2.6 decrypt()

```
void decrypt (
              std::string & Return_File_Name,
              std::vector< shift_value > & key,
              std::string & Input_File_name,
              std::string & Key_file )
```

A function that decrypts an encrypted message based on a previously found shift.

Parameters

Return_File_Name	name of the file where the decrypted message is to be saved
key	vector with strict shift containing shift value of every key character.
Input_File_name	the name of the file we want to decrypt.
Key_file	file where the key to decrypt the message is to be saved

Returns

Function does not return any value

```
00398 {
00399     std::ifstream input_file(Input_File_name);
00400     std::ofstream output_file(Return_File_Name);
00401     std::ofstream key_file(Key_file);
00402     const int sta = 1;
00403 
00404     int shift = 0,incrementer = 0;
00405 
00406     if (input_file && output_file)
00407     {
00408         std::string line = "";
00409         char letter = ' ', helper =' ';
```

```
00410         while (std::getline(input_file, line, '\0')) {
00411             bool exists = false;
00412             incrementer = 0;
00413 
00414             for (int i = 0; i < line.size(); i++) {
00415 
00416                 letter = tolower(line[i]);
00417                 exists = is_Alphabetic_Character(letter);
00418 
00419                 if (exists)
00420                 {
00421                     shift = key[incrementer % key.size()].Structshift;
00422 
00423                     if (int(letter - shift) < int('a'))
00424                     {
00425                         helper = char(int('z' - (shift - (letter - 'a') - sta)));
00426                     }
00427                     else
00428                     {
00429                         helper = char(int(letter - shift));
00430 
00431                     }
00432                     output_file « helper;
00433                     incrementer++;
00434                 }
00435                 else
00436                 {
00437                     output_file « letter;
00438                 }
00439             }
00440         }
00441 
00442         std::cout « "Decryption successful! Check file: " « Return_File_Name « " to see decrypted text
      and file: " «Key_file « " to see shift." « std::endl;
00443 
00444     }
00445     key_file « "Shift is equal: ";
00446     for (int i = 0; i < key.size(); i++)
00447     {
00448 
00449         key_file « key[i].Structshift « " ";
00450 
00451     }
00452     key_file « " Key size is equal: " « key.size();
00453 
00454     input_file.close();
00455     output_file.close();
00456     key_file.close();
00457 
00458 
00459 }
```

4.2.2.7 find_shift()

```
shift_value find_shift (
             std::deque< double > & sample_average,
             std::deque< double > & decrypted_average )
```

A function that looks for an shift between a single letter in the encoded file and a letter in the plaintext file using their averages.

Parameters

sample_average	deque cantaining average of the occurrences of sample file.
decrypted_average	deque cantaining average of the occurrences of decrypted file.

Returns

The function returns an int that corresponds to the shift of the encrypted message

```
00240 {
00241     size_t cycles = sample_average.size();
00242     int  step = 0;
00243 
00244     double suma_z_roznic_pomiedzy_2_dequami = 0;
00245     std::vector<double> sumy;
00246     shift_value shift_min;
00247 
00248 
00249 
00250 
00251     while (step < cycles)
00252     {
00253 
00254         std::vector<double> WektorRoznic;
00255 
00256 
00257         for (int i = 0; i < sample_average.size(); i++)
00258         {
00259             double absolute_value = fabs(decrypted_average[i] - sample_average[i]);
00260 
00261             WektorRoznic.push_back(absolute_value);
00262         }
00263 
00264             double holder = 0;
00265             holder = sample_average.back();
00266             sample_average.pop_back();
00267             sample_average.push_front(holder);
00268 
00269         suma_z_roznic_pomiedzy_2_dequami = Sum_of_shift(WektorRoznic);
00270 
00271         sumy.push_back(suma_z_roznic_pomiedzy_2_dequami);
00272 
00273         step++;
00274     }
00275 
00276     shift_min = min_difference(sumy);
00277 
00278     return shift_min;
00279 }
```

4.2.2.8 findkeyLength()

```
int findkeyLength (
            const std::string & file_name )
```

A function that determines the most likely length of the key used to encrypt the message.

Parameters

file_name	name of the file containing the encrypted message whose key length we are looking for.

Returns

the function returns an int which is the most likely length of the searched key.

```
00314 {
00315     std::ifstream file(file_name);
00316     int file_size = 0;
00317     char letterread = ' ', letter = ' ';
00318     std::deque<std::deque<double» coincedence_indexes;
00319     std::deque<double> coincedence_indexes_of_spacing_i;
00320     const double proportion = 0.40;
00321 
00322     while (file » letterread)
00323     {
00324         letter = tolower(letterread);
00325         bool exists = is_Alphabetic_Character(letter);
00326 
00327         if (exists)
00328         {
00329             file_size++;
```

```
00330         }
00331     }
00332
00333
00334
00335
00336     for (int i = 1; i < (file_size / 2) + 1; i++)
00337     {
00338         std::deque<std::deque<char» letters_separated = create_deque_Sepparated_letters(file_name, i);
00339
00340         std::deque<std::deque<double» averages_counted;
00341
00342         for (int j = 0; j < letters_separated.size(); j++)
00343         {
00344                 averages_counted.push_back(count_Letters(letters_separated[j]));
00345         }
00346
00347         double average_of_index = 0;
00348
00349         for (int k = 0; k < averages_counted.size(); k++)
00350         {
00351             average_of_index += calculate_index_of_coincedence(averages_counted[k]);
00352         }
00353
00354         average_of_index = (average_of_index / averages_counted.size());
00355         coincedence_indexes_of_spacing_i.push_back(average_of_index);
00356
00357         if (coincedence_indexes_of_spacing_i.size() > 2)
00358         {
00359             for (int k = 1; k < coincedence_indexes_of_spacing_i.size(); k++)
00360             {
00361                 double compare = (coincedence_indexes_of_spacing_i[k - 1] +=
      (coincedence_indexes_of_spacing_i[k - 1] * proportion));
00362
00363                 if (compare < coincedence_indexes_of_spacing_i[k])
00364                 {
00365                     int key = k + 1;
00366
00367                     //std::cout «"Key: "« key « std::endl;
00368
00369                     return key;
00370                 }
00371
00372             }
00373         }
00374
00375     }
00376     /*for (int k = 0; k < coincedence_indexes_of_spacing_i.size(); k++)
00377     {
00378         std::cout « coincedence_indexes_of_spacing_i[k] « " ";
00379     }
00380     std::cout « std::endl;*/
00381     //std::cout « "Key: 1" « std::endl;
00382     return 1;
00383 }
```

4.2.2.9 is_Alphabetic_Character()

```
bool is_Alphabetic_Character (
             char & letter )
```

A function that dtermines if input characte is alphabetic.

Parameters

letter	character we evaluate.

Returns

true if input character is alphabetic

```
00135 {
```

```
00136     char allowed_Letters[] = {
      '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'
      };
00137     char letterevaluated = ' ';
00138
00139     bool exists = std::find(std::begin(allowed_Letters), std::end(allowed_Letters), letter) !=
      std::end(allowed_Letters);
00140
00141     return exists;
00142 }
```

4.2.2.10 min_difference()

```
shift_value min_difference (
            const std::vector< double > & vector_Of_averages )
```

A function that finds the smallest value in the input vector.

Parameters

vector_Of_averages	the vector in which we want to find the smallest value.

Returns

Function return smallest value in vector

```
00294 {
00295     double comparing_min_value = 100;
00296     int shift = 0;
00297     shift_value p;
00298
00299     for (int i = 0; i < vector_Of_averages.size(); i++)
00300     {
00301         if (vector_Of_averages[i] < comparing_min_value)
00302         {
00303             shift = i;
00304             comparing_min_value = vector_Of_averages[i];
00305         }
00306     }
00307     p.Structshift = shift;
00308     p.min_value = comparing_min_value;
00309     return p;
00310
00311 }
```

4.2.2.11 print_Instruction()

```
void print_Instruction ( )
```

A function that prints short program instructions.

Parameters

Function	does not require any parameters

Returns

Function does not return any value

```
00048 {
00049     std::cout « "Program breaks a cyphertext encrypted with an unknown key with the Vigenere method."
      « std::endl;
00050     std::cout « "The program elaborates the unknown key and decrypts the encrypted file.\n The program
      is run in command line with switches:" « std::endl;
00051     std::cout « " -i input text file (cyphertext)\n -w sample text in the same language as the
      cyphertext \n -k output text file with the elaborated key \n -o output text file (plainext)" «
      std::endl;
00052
00053 }
```

4.2.2.12 Sum_of_shift()

```
double Sum_of_shift (
            std::vector< double > & vector_Of_averages )
```

A functionthat sums all values from the input vector.

Parameters

vector_Of_averages	the vector in which we want to sum all the values

Returns

Function return sum of vector

```
00282 {
00283     double added_avg = 0, final_average = 0;
00284
00285     for (int i = 0; i < vector_Of_averages.size(); i++)
00286     {
00287         added_avg += vector_Of_averages[i];
00288     }
00289     final_average = (added_avg/vector_Of_averages.size());
00290     return final_average;
00291 }
```

4.2.2.13 switches_check()

```
bool switches_check (
            const std::vector< std::string > & input_value,
            bool & Option_One )
```

A function that checks the correctness of program input switches.

Parameters

input_value	vector of input switches, the correctness of which we check
Option_One	bool evaluates whether we have encountered the first possibility of switches

Returns

Function return true if all input switches are correct

```
00056 {
00057     std::string Possibility_One[4] = {"-i","-o","-w","-k"};
00058     std::string Possibility_Two[4] = {"-o","-w","-k","-i"};
00059     int Return_true = 0;
00060
00061     if (input_value[0] == "-i")
00062     {
00063         Return_true += 1;
00064         Option_One = true;
00065         for (int i = 1; i < input_value.size(); i++)
00066         {
00067             if (input_value[i] != Possibility_One[i])
00068             {
00069                 std::cout « "Wrong switch: " « input_value[i] « std::endl;
00070             }
00071             else
00072             {
00073                 Return_true += 1;
00074             }
00075         }
00076         if (Return_true == 4)
00077         {
00078             return true;
00079         }
00080         else
00081             return false;
00082     }
00083     else if (input_value[0] == "-o")
00084     {
00085         Return_true += 1;
00086         for (int i = 1; i < input_value.size(); i++)
00087         {
00088             if (input_value[i] != Possibility_Two[i])
00089             {
00090                 std::cout « "Wrong switch: " « input_value[i] « std::endl;
00091             }
00092             else
00093             {
00094                 Return_true += 1;
00095             }
00096         }
00097         if (Return_true == 4)
00098         {
00099             return true;
00100         }
00101         else
00102             return false;
00103     }
00104     else
00105     {
00106         std::cout « "Switch Error! you typed: " « input_value[0] « std::endl;
00107         for (int i = 1; i < input_value.size(); i++)
00108         {
00109             if (input_value[i] != Possibility_One[i] || input_value[i] != Possibility_One[i])
00110             {
00111                 std::cout « "Switch Error! you typed: " « input_value[i] « std::endl;
00112             }
00113         }
00114         return false;
00115     }
00116
00117 }
```

4.2.2.14 Valid_file()

```
int Valid_file (
            const std::string & File_Name )
```

A function that checks the correctness of input file.

Parameters

File_Name	the name of the file we are checking for correctness

Returns

Function return 1 if input file exists if not then 0

```
00119                                           {
00120
00121     int cycle_counter = 0;
00122
00123     std::fstream file(File_Name);
00124     if (!file)
00125         std::cout « "File " « File_Name « " can't be found! Check if name is proper and/or if file
      exists." « std::endl;
00126     else
00127     {
00128         cycle_counter += 1;
00129         file.close();
00130     }
00131     return cycle_counter;
00132 }
```

4.3 function.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #ifndef FUNCTION_H
00003 #define FUNCTION_H
00004
00005 #include "structure.h"
00006
00017 void action(int& number, int& can_decrypt_files, char* params[], const std::vector<std::string>&
      inputSw, const std::vector<std::string>& inputFi, bool& switch_case_one, bool& can_decrypt_switches);
00018
00019
00024 void print_Instruction();
00025
00031 bool switches_check(const std::vector<std::string>& input_value, bool& Option_One);
00032
00037 int Valid_file(const std::string& File_Name);
00038
00043 std::deque<double> count_Letters(const std::string &file_name);
00044
00050 std::deque<std::deque<char> > create_deque_Sepparated_letters(const std::string& file_name, const int&
      spacing);
00051
00056 std::deque<double> count_Letters(std::deque<char>& letters);
00057
00063 shift_value find_shift(std::deque<double>& sample_average, std::deque<double>& decrypted_average);
00064
00069 double Sum_of_shift(std::vector<double>& vector_Of_averages);
00070
00074 shift_value min_difference(const std::vector<double>& vector_Of_averages);
00075
00079 bool is_Alphabetic_Character(char& letter);
00080
00085 double calculate_index_of_coincedence(std::deque<double> input_averages);
00086
00091 int findkeyLength(const std::string& file_name);
00092
00100 void decrypt(std::string& Return_File_Name, std::vector<shift_value>& key, std::string&
      Input_File_name, std::string& Key_file);
00101 #endif
```

4.4 main.cpp File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <map>
#include <iomanip>
#include <deque>
#include "function.h"
#include "structure.h"
```

Functions

- int main (int number, char ∗params[ ])

4.4.1 Function Documentation

4.4.1.1 main()

```
int main (
              int number,
              char * params[] )
```

```
00013 {
00014
00015     std::vector<std::string> Input_Switches, Input_Files;
00016     int Can_Decrypt_Files = 0, keylength = 0;
00017     std::vector<shift_value> Shift_and_Value;
00018     const int NUMBER_OF_FILES { 4 };
00019     bool Can_Decrypt_Switches = false, Switch_case_One = false;
00020     std::string encrypted_text_file = "";
00021     std::string pattern_text_file = "";
00022     std::string key_file = "";
00023     std::string output_file = "";
00024
00025
00026     for (int i = 1; i < number; i++)
00027     {
00028         if (i % 2 == 1)
00029         {
00030             Input_Switches.push_back(params[i]);
00031         }
00032         else
00033             Input_Files.push_back(params[i]);
00034     }
00035
00036
      action(number,Can_Decrypt_Files,params,Input_Switches,Input_Files,Switch_case_One,Can_Decrypt_Switches);
00037
00038     if (Switch_case_One && Input_Switches.size() >= NUMBER_OF_FILES)
00039     {
00040         encrypted_text_file = Input_Files[0];
00041         pattern_text_file = Input_Files[2];
00042         key_file = Input_Files[3];
00043         output_file = Input_Files[1];
00044     }
00045     else if (Input_Switches.size() >= NUMBER_OF_FILES)
00046     {
00047         encrypted_text_file = Input_Files[3];
00048         pattern_text_file = Input_Files[1];
00049         key_file = Input_Files[2];
00050         output_file = Input_Files[0];
00051     }
00052
00053     if(Can_Decrypt_Files == NUMBER_OF_FILES && Can_Decrypt_Switches)
00054     {
00055
00056                 std::cout « "All files were found!" « std::endl;
00057                 std::cout « "Decryption in progress..." « std::endl;
00058                 std::ifstream file(encrypted_text_file);
00059                 keylength = findkeyLength(encrypted_text_file);
00060
00061                 std::deque<double> letters_Counted = count_Letters(pattern_text_file);
00062
00063                 std::deque<std::deque<char> > letters_separated_deques =
      create_deque_Sepparated_letters(encrypted_text_file, keylength);
00064                 for (int i = 0; i < letters_separated_deques.size(); i++)
00065                 {
00066                     std::deque<double> averages_spacing = count_Letters(letters_separated_deques[i]);
00067
00068                     Shift_and_Value.push_back(find_shift(letters_Counted, averages_spacing));
00069
00070                 }
00071                 decrypt(output_file, Shift_and_Value, encrypted_text_file, key_file);
00072
00073     }
```

```
00074     else
00075     {
00076         return 0;
00077     }
00078
00079     return 0;
00080 }
```

4.5 structure.h File Reference

Classes

- struct shift_value

  A struct containing shift.

4.6 structure.h

Go to the documentation of this file.

```
00001 #ifndef STRUCTURE_H
00002 #define STRUCTURE_H
00003
00005 struct shift_value
00006 {
00007     int Structshift;
00008 };
00009
00010
00011 #endif
```

Index