

Vigenere-EDB

Generated by Doxygen 1.9.8



<b>1 File Index</b>	<b>1</b>
1.1 File List	1
<b>2 File Documentation</b>	<b>3</b>
2.1 main.cpp File Reference	3
2.1.1 Function Documentation	4
2.1.1.1 main()	4
2.2 src/breaker.cpp File Reference	4
2.2.1 Function Documentation	5
2.2.1.1 breakFile()	5
2.3 src/breaker.h File Reference	6
2.3.1 Function Documentation	6
2.3.1.1 breakFile()	6
2.4 breaker.h	7
2.5 src/decryptor.cpp File Reference	7
2.5.1 Function Documentation	8
2.5.1.1 decrypt()	8
2.6 src/decryptor.h File Reference	9
2.6.1 Function Documentation	10
2.6.1.1 decrypt()	10
2.7 decryptor.h	10
2.8 src/encryptor.cpp File Reference	11
2.8.1 Function Documentation	11
2.8.1.1 encrypt()	11
2.9 src/encryptor.h File Reference	12
2.9.1 Function Documentation	13
2.9.1.1 encrypt()	13
2.10 encryptor.h	13
2.11 src/utility.cpp File Reference	14
2.11.1 Function Documentation	15
2.11.1.1 decrypt()	15
2.11.1.2 fillFrequencyTableFromText()	15
2.11.1.3 fillRawText()	15
2.11.1.4 getCaesarKey()	16
2.11.1.5 getIndexOfCoincidence()	16
2.11.1.6 getKeyDataFromFile()	16
2.11.1.7 getKeyLength()	17
2.11.1.8 getTextDataFromFile()	17
2.11.1.9 nextValidKeyIndex()	18
2.11.1.10 parceArguments()	18
2.11.1.11 printHelp()	19
2.11.1.12 saveToFile()	19

---

2.11.1.13 stringToLowercase()	19
2.12 src/utility.h File Reference	20
2.12.1 Function Documentation	21
2.12.1.1 decrypt()	21
2.12.1.2 fillFrequencyTableFromText()	23
2.12.1.3 fillRawText()	23
2.12.1.4 getCaesarKey()	24
2.12.1.5 getIndexOfCoincidence()	24
2.12.1.6 getKeyDataFromFile()	25
2.12.1.7 getKeyLength()	25
2.12.1.8 getTextDataFromFile()	26
2.12.1.9 nextValidKeyIndex()	26
2.12.1.10 parceArguments()	27
2.12.1.11 printHelp()	28
2.12.1.12 saveToFile()	28
2.12.1.13 stringToLowercase()	29
2.12.2 Variable Documentation	29
2.12.2.1 ASCII_TEXT_MARGIN	29
2.12.2.2 FREQUENCY_TABLE	29
2.13 utility.h	30
<b>Index</b>	<b>31</b>

# Chapter 1

## File Index

### 1.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.cpp</a>	3
<a href="#">src/breaker.cpp</a>	4
<a href="#">src/breaker.h</a>	6
<a href="#">src/decryptor.cpp</a>	7
<a href="#">src/decryptor.h</a>	9
<a href="#">src/encryptor.cpp</a>	11
<a href="#">src/encryptor.h</a>	12
<a href="#">src/utility.cpp</a>	14
<a href="#">src/utility.h</a>	20



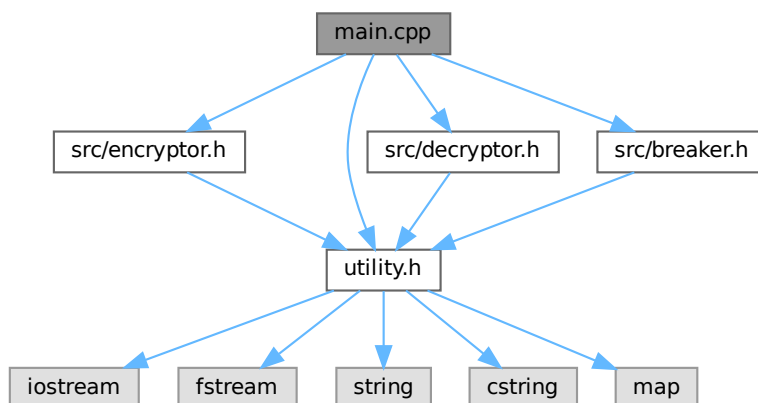
## Chapter 2

# File Documentation

### 2.1 main.cpp File Reference

```
#include "src/encryptor.h"  
#include "src/decryptor.h"  
#include "src/breaker.h"  
#include "src/utility.h"
```

Include dependency graph for main.cpp:



#### Functions

- `int main (int argc, char *argv[ ])`  
*Entry point of the application.*

## 2.1.1 Function Documentation

### 2.1.1.1 main()

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

Entry point of the application.

This program supports three modes of operation:

- Encryption: Encrypts the input file using a specified key.
- Decryption: Decrypts the input file using a specified key.
- Breaking: Attempts to break the encryption on the input file without a key.

#### Parameters

<i>argc</i>	Number of command-line arguments.
<i>argv</i>	Array of command-line arguments. <ul style="list-style-type: none"><li>• Mode: Specify "-en" for encryption, "-de" for decryption, or "-br" for breaking.</li><li>• InputFile: Path to the input file to process.</li><li>• OutputFile: Path to save the processed output.</li><li>• KeyFile: Path to the key file (required for encryption and decryption).</li></ul>

#### Returns

Exit status of the program.

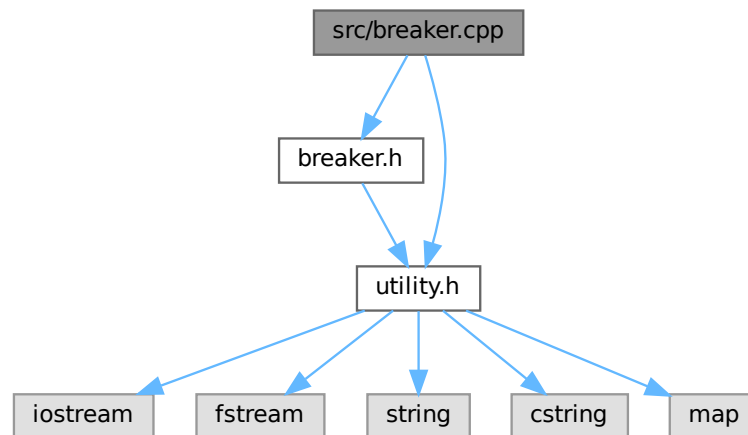
- 0: Success.
- 1: Invalid mode specified.
- -1: Error in parsing arguments or execution.

## 2.2 src/breaker.cpp File Reference

```
#include "breaker.h"
#include "utility.h"
```



Include dependency graph for breaker.cpp:



## Functions

- int [breakFile](#) (const std::string &targetFile, const std::string &outputFile)  
*Attempts to break the encryption of a file and write the decrypted output.*

## 2.2.1 Function Documentation

### 2.2.1.1 breakFile()

```

int breakFile (
    const std::string & targetFile,
    const std::string & outputFile )

```

Attempts to break the encryption of a file and write the decrypted output.

Attempts to break the encryption of a file.

This function analyzes the encrypted file, determines the likely key length, derives the encryption key, and decrypts the file. The result is saved to the output file.

#### Parameters

<i>targetFile</i>	The path to the encrypted file to be analyzed and decrypted.
<i>outputFile</i>	The path to the file where decrypted content will be saved.

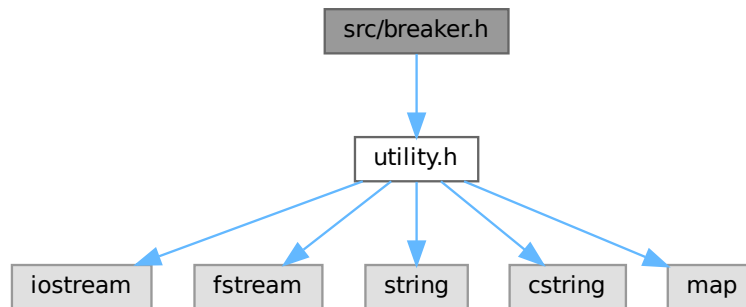
#### Returns

int Returns 0 on success, or a non-zero error code on failure.

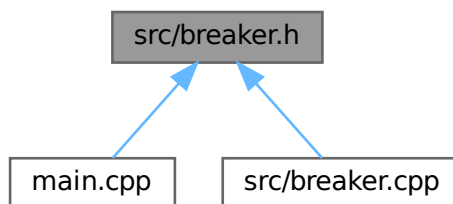
## 2.3 src/breaker.h File Reference

```
#include "utility.h"
```

Include dependency graph for breaker.h:



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



### Functions

- `int breakFile (const std::string &targetFile, const std::string &outputFile)`  
*Attempts to break the encryption of a file.*

### 2.3.1 Function Documentation

#### 2.3.1.1 breakFile()

```
int breakFile (
    const std::string & targetFile,
    const std::string & outputFile )
```

Attempts to break the encryption of a file.

This function analyzes the target file and attempts to break its encryption, saving the results to the specified output file. The exact breaking method depends on the implementation in the source file.

**Parameters**

<i>targetFile</i>	The path to the file to analyze and attempt to break.
<i>outputFile</i>	The path where the results will be saved.

**Returns**

int Returns 0 on success, or a non-zero error code on failure.

Attempts to break the encryption of a file.

This function analyzes the encrypted file, determines the likely key length, derives the encryption key, and decrypts the file. The result is saved to the output file.

**Parameters**

<i>targetFile</i>	The path to the encrypted file to be analyzed and decrypted.
<i>outputFile</i>	The path to the file where decrypted content will be saved.

**Returns**

int Returns 0 on success, or a non-zero error code on failure.

## 2.4 breaker.h

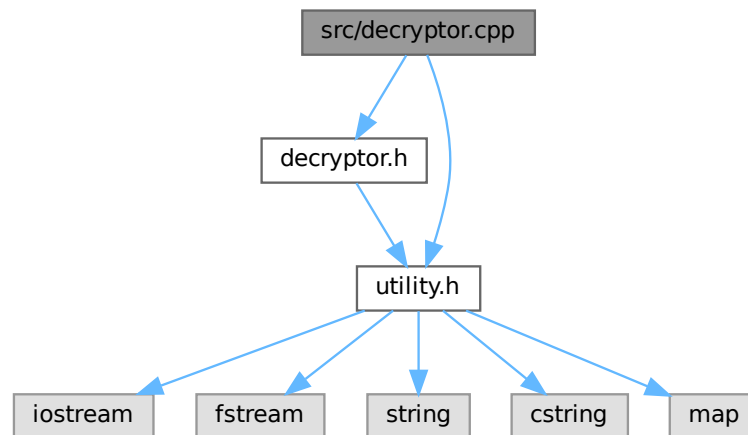
[Go to the documentation of this file.](#)

```
00001
00002 #pragma once
00003 #include "utility.h"
00004
00016 int breakFile(const std::string& targetFile, const std::string& outputFile);
```

## 2.5 src/decryptor.cpp File Reference

```
#include "decryptor.h"
#include "utility.h"
```

Include dependency graph for decryptor.cpp:



## Functions

- `int decrypt (const std::string &inputFile, const std::string &outputFile, const std::string &keyFile)`  
*Decrypts the input file data using a specified key and saves the result to an output file.*

## 2.5.1 Function Documentation

### 2.5.1.1 `decrypt()`

```

int decrypt (
    const std::string & inputFile,
    const std::string & outputFile,
    const std::string & keyFile )

```

Decrypts the input file data using a specified key and saves the result to an output file.

Decrypts an input file using the provided key.

This function reads the encrypted data and the decryption key from the specified files, performs the decryption by shifting each character based on the key, and then writes the decrypted data to the output file.

#### Parameters

<i>inputFile</i>	The path to the file containing the encrypted data.
<i>outputFile</i>	The path to the file where the decrypted data will be saved.
<i>keyFile</i>	The path to the file containing the decryption key.

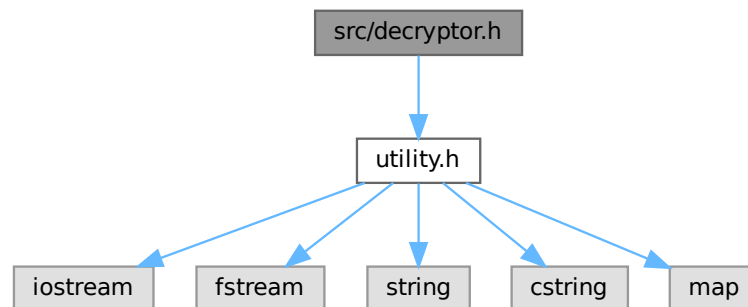
### Returns

Returns 0 if the decryption was successful, or -1 if an error occurred during the process (e.g., invalid key or file reading issues).

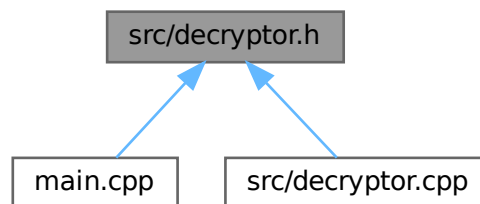
## 2.6 src/decryptor.h File Reference

```
#include "utility.h"
```

Include dependency graph for decryptor.h:



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



### Functions

- int `decrypt` (const std::string &inputFile, const std::string &outputFile, const std::string &keyFile)  
*Decrypts an input file using the provided key.*

## 2.6.1 Function Documentation

### 2.6.1.1 decrypt()

```
int decrypt (
    const std::string & inputFile,
    const std::string & outputFile,
    const std::string & keyFile )
```

Decrypts an input file using the provided key.

This function reads an encrypted file and decrypts its contents using the key specified in the key file. The decrypted output is saved to the specified output file.

#### Parameters

<i>inputFile</i>	The path to the encrypted input file.
<i>outputFile</i>	The path to the output file where decrypted content will be saved.
<i>keyFile</i>	The path to the key file containing the decryption key.

#### Returns

int Returns 0 on successful decryption, or a non-zero error code on failure.

Decrypts an input file using the provided key.

This function reads the encrypted data and the decryption key from the specified files, performs the decryption by shifting each character based on the key, and then writes the decrypted data to the output file.

#### Parameters

<i>inputFile</i>	The path to the file containing the encrypted data.
<i>outputFile</i>	The path to the file where the decrypted data will be saved.
<i>keyFile</i>	The path to the file containing the decryption key.

#### Returns

Returns 0 if the decryption was successful, or -1 if an error occurred during the process (e.g., invalid key or file reading issues).

## 2.7 decryptor.h

[Go to the documentation of this file.](#)

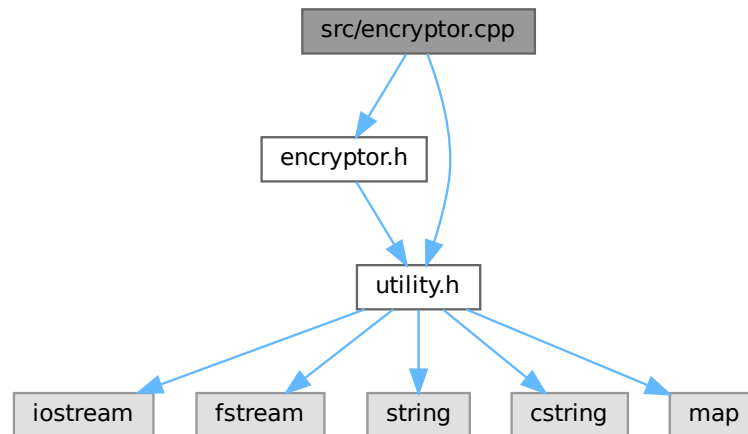
```
00001
00002 #pragma once
00003 #include "utility.h"
00004
00016 int decrypt(const std::string& inputFile, const std::string& outputFile, const std::string& keyFile);
```

## 2.8 src/encryptor.cpp File Reference

```
#include "encryptor.h"
```

```
#include "utility.h"
```

Include dependency graph for encryptor.cpp:



### Functions

- `int encrypt (const std::string &inputFile, const std::string &outputFile, const std::string &keyFile)`  
*Encrypts the input file data using a specified key and saves the result to an output file.*

### 2.8.1 Function Documentation

#### 2.8.1.1 encrypt()

```
int encrypt (
    const std::string & inputFile,
    const std::string & outputFile,
    const std::string & keyFile )
```

Encrypts the input file data using a specified key and saves the result to an output file.

Encrypts an input file using the provided key.

This function reads the plain text data and the encryption key from the specified files, performs the encryption by shifting each character based on the key, and then writes the encrypted data to the output file.

#### Parameters

<i>inputFile</i>	The path to the file containing the plain text data.
<i>outputFile</i>	The path to the file where the encrypted data will be saved.
<i>keyFile</i>	The path to the file containing the encryption key.

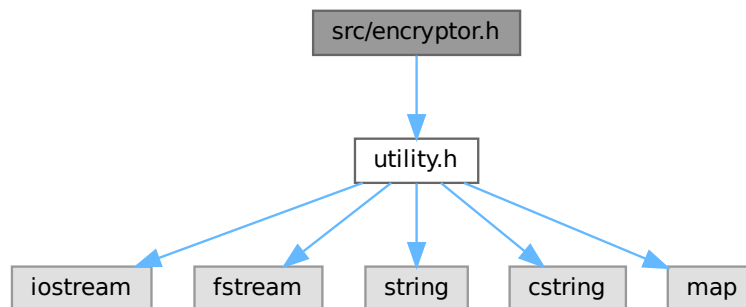
### Returns

Returns 0 if the encryption was successful, or -1 if an error occurred during the process (e.g., invalid key or file reading issues).

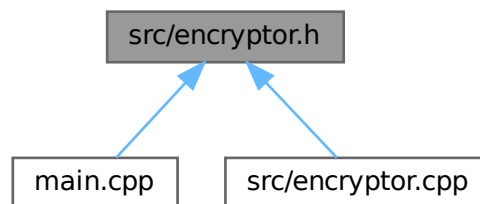
## 2.9 src/encryptor.h File Reference

```
#include "utility.h"
```

Include dependency graph for encryptor.h:



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



### Functions

- int [encrypt](#) (const std::string &inputFile, const std::string &outputFile, const std::string &keyFile)  
*Encrypts an input file using the provided key.*



## 2.9.1 Function Documentation

### 2.9.1.1 encrypt()

```
int encrypt (
    const std::string & inputFile,
    const std::string & outputFile,
    const std::string & keyFile )
```

Encrypts an input file using the provided key.

This function reads a plaintext file and encrypts its contents using the key specified in the key file. The encrypted output is saved to the specified output file.

#### Parameters

<i>inputFile</i>	The path to the plaintext input file.
<i>outputFile</i>	The path to the output file where encrypted content will be saved.
<i>keyFile</i>	The path to the key file containing the encryption key.

#### Returns

int Returns 0 on successful encryption, or a non-zero error code on failure.

Encrypts an input file using the provided key.

This function reads the plain text data and the encryption key from the specified files, performs the encryption by shifting each character based on the key, and then writes the encrypted data to the output file.

#### Parameters

<i>inputFile</i>	The path to the file containing the plain text data.
<i>outputFile</i>	The path to the file where the encrypted data will be saved.
<i>keyFile</i>	The path to the file containing the encryption key.

#### Returns

Returns 0 if the encryption was successful, or -1 if an error occurred during the process (e.g., invalid key or file reading issues).

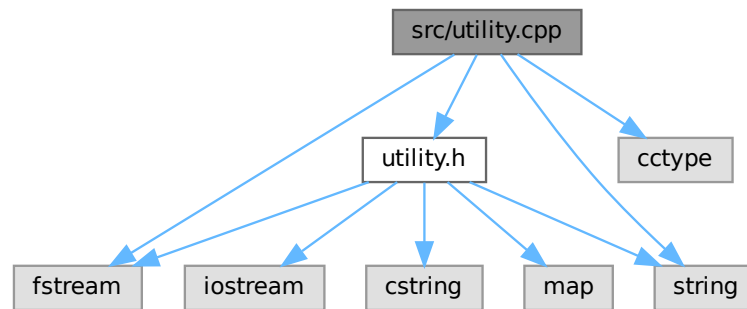
## 2.10 encryptor.h

[Go to the documentation of this file.](#)

```
00001
00002 #pragma once
00003 #include "utility.h"
00004
00016 int encrypt(const std::string& inputFile, const std::string& outputFile, const std::string& keyFile);
```

## 2.11 src/utility.cpp File Reference

```
#include "utility.h"
#include <cctype>
#include <fstream>
#include <string>
Include dependency graph for utility.cpp:
```



### Functions

- long long int [nextValidKeyIndex](#) (const std::string &key, long long int currentIndex)
 

*Finds the next valid index in the key string.*
- bool [getKeyDataFromFile](#) (const std::string &filename, std::string &key)
 

*Reads the key data from a file into a string.*
- bool [getTextDataFromFile](#) (const std::string &filename, std::string &textData)
 

*Reads text data from a file into a string.*
- void [stringToLowercase](#) (std::string &str)
 

*Converts a string to lowercase.*
- bool [saveToFile](#) (const std::string &filename, const std::string &data)
 

*Saves data to a file.*
- void [fillRawText](#) (const std::string &filename, std::string &rawText)
 

*Extracts raw alphabetical text from a file.*
- void [fillFrequencyTableFromText](#) (const std::string &rawText, long long unsigned int \*frequencyTable)
 

*Fills a frequency table based on the text.*
- void [decrypt](#) (std::string &inputData, const std::string &key)
 

*Decrypts text data using a Vigenère cipher key.*
- char [getCaesarKey](#) (std::string &inputData, long long unsigned int \*frequencyTable)
 

*Determines the Caesar cipher key for a given text group.*
- double [getIndexOfCoincidence](#) (long long unsigned int \*frequencyTable)
 

*Calculates the index of coincidence for a frequency table.*
- long long unsigned int [getKeyLength](#) (const std::string &rawText)
 

*Determines the likely key length for a Vigenère cipher.*
- void [printHelp](#) ()
 

*Prints help information about the program's usage.*
- bool [parseArguments](#) (int argc, char \*\*argv, std::string &mode, std::string &inputFile, std::string &outputFile, std::string &keyFile)
 

*Parses command-line arguments to configure the program.*

## 2.11.1 Function Documentation

### 2.11.1.1 decrypt()

```
void decrypt (
    std::string & inputData,
    const std::string & key )
```

Decrypts text data using a Vigenère cipher key.

Decrypts input data using the given key.

This function decrypts the input data using the provided key and modifies the data in place.

#### Parameters

<i>inputData</i>	The encrypted text to decrypt.
<i>key</i>	The Vigenère cipher key.

### 2.11.1.2 fillFrequencyTableFromText()

```
void fillFrequencyTableFromText (
    const std::string & rawText,
    long long unsigned int * frequencyTable )
```

Fills a frequency table based on the text.

Populates a frequency table from the given text.

This function updates the provided frequency table based on the occurrence of each letter in the raw text.

#### Parameters

<i>rawText</i>	The input text to analyze.
<i>frequencyTable</i>	An array to store the frequency of each letter.

### 2.11.1.3 fillRawText()

```
void fillRawText (
    const std::string & filename,
    std::string & rawText )
```

Extracts raw alphabetical text from a file.

Reads raw text from a file into a string.

This function reads a file, filters out non-alphabetical characters, and converts uppercase characters to lowercase.

**Parameters**

<i>filename</i>	The name of the file to process.
<i>rawText</i>	The string where the raw text will be stored.

**2.11.1.4 getCaesarKey()**

```
char getCaesarKey (
    std::string & inputData,
    long long unsigned int * frequencyTable )
```

Determines the Caesar cipher key for a given text group.

Determines the Caesar cipher key from input data.

Analyzes the input data and determines the most likely Caesar cipher key by comparing letter frequencies.

**Parameters**

<i>inputData</i>	The text to analyze.
<i>frequencyTable</i>	A frequency table to be used during analysis.

**Returns**

char The most likely Caesar cipher key.

**2.11.1.5 getIndexOfCoincidence()**

```
double getIndexOfCoincidence (
    long long unsigned int * frequencyTable )
```

Calculates the index of coincidence for a frequency table.

Computes the Index of Coincidence for a frequency table.

This metric is used to analyze the likelihood of a particular key length in cryptographic analysis.

**Parameters**

<i>frequencyTable</i>	A table of letter frequencies.
-----------------------	--------------------------------

**Returns**

double The calculated index of coincidence.

**2.11.1.6 getKeyDataFromFile()**

```
bool getKeyDataFromFile (
```

```
const std::string & filename,  
std::string & key )
```

Reads the key data from a file into a string.

Reads key data from a file.

This function reads the contents of the specified file and stores the data in the provided key string.

#### Parameters

<i>filename</i>	The name of the file containing the key.
<i>key</i>	The string where the key data will be stored.

#### Returns

bool True if the key data is successfully read, false otherwise.

#### 2.11.1.7 getKeyLength()

```
long long unsigned int getKeyLength (  
    const std::string & rawText )
```

Determines the likely key length for a Vigenère cipher.

Determines the key length of the given text.

Analyzes the input text and determines the most likely key length based on statistical analysis.

#### Parameters

<i>rawText</i>	The text to analyze.
----------------	----------------------

#### Returns

long long unsigned int The most likely key length, or 0 if none is found.

#### 2.11.1.8 getTextDataFromFile()

```
bool getTextDataFromFile (  
    const std::string & filename,  
    std::string & textData )
```

Reads text data from a file into a string.

Reads text data from a file.

This function reads the contents of a file line by line and appends it to the provided string.

**Parameters**

<i>filename</i>	The name of the file containing the text data.
<i>textData</i>	The string where the text data will be stored.

**Returns**

bool True if the text data is successfully read, false otherwise.

**2.11.1.9 nextValidKeyIndex()**

```
long long int nextValidKeyIndex (
    const std::string & key,
    long long int currentIndex )
```

Finds the next valid index in the key string.

Finds the next valid key index.

This function increments the index until it points to a valid lowercase alphabetical character. Returns -1 if no valid characters are found.

**Parameters**

<i>key</i>	The key string to iterate through.
<i>currentIndex</i>	The current index in the key string.

**Returns**

long long int The next valid index, or -1 if none exist.

**2.11.1.10 parseArguments()**

```
bool parseArguments (
    int argc,
    char ** argv,
    std::string & mode,
    std::string & inputFile,
    std::string & outputFile,
    std::string & keyFile )
```

Parses command-line arguments to configure the program.

Parses command-line arguments and extracts required values.

Parses and validates command-line arguments for setting operation mode, input file, output file, and key file paths.

**Parameters**

<i>argc</i>	The number of command-line arguments.
-------------	---------------------------------------

**Parameters**

<i>argv</i>	The array of command-line arguments.
<i>mode</i>	A reference to store the chosen mode.
<i>inputFile</i>	A reference to store the input file path.
<i>outputFile</i>	A reference to store the output file path.
<i>keyFile</i>	A reference to store the key file path.

**Returns**

bool True if arguments are successfully parsed, false otherwise.

**2.11.1.11 printHelp()**

```
void printHelp ( )
```

Prints help information about the program's usage.

Prints help information to the console.

Displays the supported flags, their purposes, and example usage.

**2.11.1.12 saveToFile()**

```
bool saveToFile (
    const std::string & filename,
    const std::string & data )
```

Saves data to a file.

Saves data to a file with a chosen separator.

Writes the provided data to a file with the specified filename.

**Parameters**

<i>filename</i>	The name of the file to save the data.
<i>data</i>	The string data to be written to the file.

**Returns**

bool True if the data is successfully saved, false otherwise.

**2.11.1.13 stringToLowercase()**

```
void stringToLowercase (
    std::string & str )
```

Converts a string to lowercase.

Converts every character of a string to its lowercase version.

This function modifies the provided string by converting all uppercase letters to their lowercase equivalents.

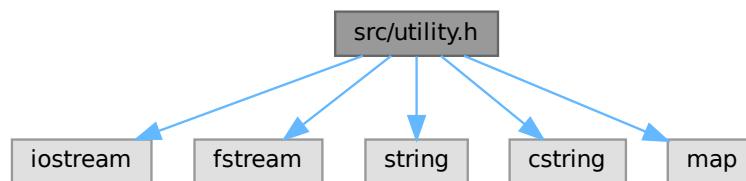
#### Parameters

<i>str</i>	The string to convert to lowercase.
------------	-------------------------------------

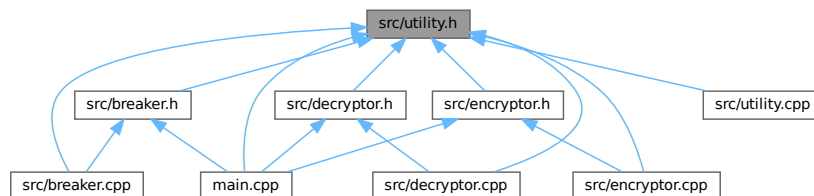
## 2.12 src/utility.h File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <map>
```

Include dependency graph for utility.h:



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



#### Functions

- double [getIndexOfCoincidence](#) (long long unsigned int \*frequencyTable)  
*Computes the Index of Coincidence for a frequency table.*
- void [decrypt](#) (std::string &inputData, const std::string &key)



- Decrypts input data using the given key.*
- char [getCaesarKey](#) (std::string &inputData, long long unsigned int \*frequencyTable)  
*Determines the Caesar cipher key from input data.*
- void [fillRawText](#) (const std::string &filename, std::string &rawText)  
*Reads raw text from a file into a string.*
- void [fillFrequencyTableFromText](#) (const std::string &rawText, long long unsigned int \*frequencyTable)  
*Populates a frequency table from the given text.*
- long long unsigned int [getKeyLength](#) (const std::string &rawText)  
*Determines the key length of the given text.*
- long long int [nextValidKeyIndex](#) (const std::string &key, long long int currentIndex)  
*Finds the next valid key index.*
- bool [saveToFile](#) (const std::string &filename, const std::string &data)  
*Saves data to a file with a chosen separator.*
- void [stringToLowercase](#) (std::string &str)  
*Converts every character of a string to its lowercase version.*
- bool [getTextDataFromFile](#) (const std::string &filename, std::string &textData)  
*Reads text data from a file.*
- bool [getKeyDataFromFile](#) (const std::string &filename, std::string &key)  
*Reads key data from a file.*
- void [printHelp](#) ()  
*Prints help information to the console.*
- bool [parseArguments](#) (int argc, char \*\*argv, std::string &mode, std::string &inputFile, std::string &outputFile, std::string &keyFile)  
*Parses command-line arguments and extracts required values.*

## Variables

- const int [ASCII\\_TEXT\\_MARGIN](#) = 97  
*ASCII margin value for text processing.*
- const double [FREQUENCY\\_TABLE](#) []  
*Percent frequency of every letter in English text.*

## 2.12.1 Function Documentation

### 2.12.1.1 decrypt()

```
void decrypt (
    std::string & inputData,
    const std::string & key )
```

Decrypts input data using the given key.

#### Parameters

<i>inputData</i>	The data to be decrypted.
<i>key</i>	The decryption key.

Decrypts input data using the given key.

This function decrypts the input data using the provided key and modifies the data in place.

## Parameters

<i>inputData</i>	The encrypted text to decrypt.
<i>key</i>	The Vigenère cipher key.

**2.12.1.2 fillFrequencyTableFromText()**

```
void fillFrequencyTableFromText (
    const std::string & rawText,
    long long unsigned int * frequencyTable )
```

Populates a frequency table from the given text.

## Parameters

<i>rawText</i>	The text to analyze.
<i>frequencyTable</i>	The frequency table to populate.

Populates a frequency table from the given text.

This function updates the provided frequency table based on the occurrence of each letter in the raw text.

## Parameters

<i>rawText</i>	The input text to analyze.
<i>frequencyTable</i>	An array to store the frequency of each letter.

**2.12.1.3 fillRawText()**

```
void fillRawText (
    const std::string & filename,
    std::string & rawText )
```

Reads raw text from a file into a string.

## Parameters

<i>filename</i>	The name of the file to read.
<i>rawText</i>	The string to store the raw text.

Reads raw text from a file into a string.

This function reads a file, filters out non-alphabetical characters, and converts uppercase characters to lowercase.

## Parameters

<i>filename</i>	The name of the file to process.
<i>rawText</i>	The string where the raw text will be stored.

#### 2.12.1.4 getCaesarKey()

```
char getCaesarKey (
    std::string & inputData,
    long long unsigned int * frequencyTable )
```

Determines the Caesar cipher key from input data.

##### Parameters

<i>inputData</i>	The data to analyze.
<i>frequencyTable</i>	The frequency table to use for analysis.

##### Returns

The determined Caesar cipher key.

Determines the Caesar cipher key from input data.

Analyzes the input data and determines the most likely Caesar cipher key by comparing letter frequencies.

##### Parameters

<i>inputData</i>	The text to analyze.
<i>frequencyTable</i>	A frequency table to be used during analysis.

##### Returns

char The most likely Caesar cipher key.

#### 2.12.1.5 getIndexOfCoincidence()

```
double getIndexOfCoincidence (
    long long unsigned int * frequencyTable )
```

Computes the Index of Coincidence for a frequency table.

##### Parameters

<i>frequencyTable</i>	The frequency table to analyze.
-----------------------	---------------------------------

##### Returns

The Index of Coincidence.

Computes the Index of Coincidence for a frequency table.

This metric is used to analyze the likelihood of a particular key length in cryptographic analysis.

**Parameters**

<i>frequencyTable</i>	A table of letter frequencies.
-----------------------	--------------------------------

**Returns**

double The calculated index of coincidence.

**2.12.1.6 getKeyDataFromFile()**

```
bool getKeyDataFromFile (
    const std::string & filename,
    std::string & key )
```

Reads key data from a file.

**Parameters**

<i>filename</i>	The name of the file to read.
<i>key</i>	The string to store the key data.

**Returns**

True if the operation succeeds, false otherwise.

Reads key data from a file.

This function reads the contents of the specified file and stores the data in the provided key string.

**Parameters**

<i>filename</i>	The name of the file containing the key.
<i>key</i>	The string where the key data will be stored.

**Returns**

bool True if the key data is successfully read, false otherwise.

**2.12.1.7 getKeyLength()**

```
long long unsigned int getKeyLength (
    const std::string & rawText )
```

Determines the key length of the given text.

**Parameters**

<i>rawText</i>	The text to analyze.
----------------	----------------------

**Returns**

The determined key length.

Determines the key length of the given text.

Analyzes the input text and determines the most likely key length based on statistical analysis.

**Parameters**

<i>rawText</i>	The text to analyze.
----------------	----------------------

**Returns**

long long unsigned int The most likely key length, or 0 if none is found.

**2.12.1.8 getTextDataFromFile()**

```
bool getTextDataFromFile (
    const std::string & filename,
    std::string & textData )
```

Reads text data from a file.

**Parameters**

<i>filename</i>	The name of the file to read.
<i>textData</i>	The string to store the text data.

**Returns**

True if the operation succeeds, false otherwise.

Reads text data from a file.

This function reads the contents of a file line by line and appends it to the provided string.

**Parameters**

<i>filename</i>	The name of the file containing the text data.
<i>textData</i>	The string where the text data will be stored.

**Returns**

bool True if the text data is successfully read, false otherwise.

**2.12.1.9 nextValidKeyIndex()**

```
long long int nextValidKeyIndex (
```

```
const std::string & key,  
long long int currentIndex )
```

Finds the next valid key index.

#### Parameters

<i>key</i>	The key string to analyze.
<i>currentIndex</i>	The current index to start from.

#### Returns

The next valid key index, or -1 if none are found.

Finds the next valid key index.

This function increments the index until it points to a valid lowercase alphabetical character. Returns -1 if no valid characters are found.

#### Parameters

<i>key</i>	The key string to iterate through.
<i>currentIndex</i>	The current index in the key string.

#### Returns

long long int The next valid index, or -1 if none exist.

### 2.12.1.10 `parceArguments()`

```
bool parceArguments (  
    int argc,  
    char ** argv,  
    std::string & mode,  
    std::string & inputFile,  
    std::string & outputFile,  
    std::string & keyFile )
```

Parses command-line arguments and extracts required values.

#### Parameters

<i>argc</i>	The number of arguments.
<i>argv</i>	The array of argument strings.
<i>mode</i>	The mode of operation (e.g., "-en", "-de", "-br").
<i>inputFile</i>	The input file path.
<i>outputFile</i>	The output file path.
<i>keyFile</i>	The key file path.

**Returns**

True if arguments are parsed successfully, false otherwise.

Parses command-line arguments and extracts required values.

Parses and validates command-line arguments for setting operation mode, input file, output file, and key file paths.

**Parameters**

<i>argc</i>	The number of command-line arguments.
<i>argv</i>	The array of command-line arguments.
<i>mode</i>	A reference to store the chosen mode.
<i>inputFile</i>	A reference to store the input file path.
<i>outputFile</i>	A reference to store the output file path.
<i>keyFile</i>	A reference to store the key file path.

**Returns**

bool True if arguments are successfully parsed, false otherwise.

**2.12.1.11 printHelp()**

```
void printHelp ( )
```

Prints help information to the console.

Prints help information to the console.

Displays the supported flags, their purposes, and example usage.

**2.12.1.12 saveToFile()**

```
bool saveToFile (
    const std::string & filename,
    const std::string & data )
```

Saves data to a file with a chosen separator.

**Parameters**

<i>filename</i>	The name of the file to save to.
<i>data</i>	The data to save.

**Returns**

True if the save operation succeeds, false otherwise.

Saves data to a file with a chosen separator.

Writes the provided data to a file with the specified filename.



**Parameters**

<i>filename</i>	The name of the file to save the data.
<i>data</i>	The string data to be written to the file.

**Returns**

bool True if the data is successfully saved, false otherwise.

**2.12.1.13 stringToLowercase()**

```
void stringToLowercase (
    std::string & str )
```

Converts every character of a string to its lowercase version.

**Parameters**

<i>str</i>	The string to convert.
------------	------------------------

Converts every character of a string to its lowercase version.

This function modifies the provided string by converting all uppercase letters to their lowercase equivalents.

**Parameters**

<i>str</i>	The string to convert to lowercase.
------------	-------------------------------------

**2.12.2 Variable Documentation****2.12.2.1 ASCII\_TEXT\_MARGIN**

```
const int ASCII_TEXT_MARGIN = 97
```

ASCII margin value for text processing.

**2.12.2.2 FREQUENCY\_TABLE**

```
const double FREQUENCY_TABLE[]
```

**Initial value:**

```
= {
    0.08167, 0.01492, 0.02782, 0.04253, 0.02702, 0.02228, 0.02015, 0.06094,
    0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749, 0.07507, 0.01929,
    0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978, 0.02360, 0.00150,
    0.01975, 0.00074
}
```

Percent frequency of every letter in English text.

## 2.13 utility.h

[Go to the documentation of this file.](#)

```
00001
00002 #pragma once
00003 #include <iostream>
00004 #include <fstream>
00005 #include <string>
00006 #include <cstring>
00007 #include <map>
00008
00012 const int ASCII_TEXT_MARGIN = 97;
00013
00017 const double FREQUENCY_TABLE[] = {
00018     0.08167, 0.01492, 0.02782, 0.04253, 0.02702, 0.02228, 0.02015, 0.06094,
00019     0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749, 0.07507, 0.01929,
00020     0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978, 0.02360, 0.00150,
00021     0.01975, 0.00074
00022 };
00023
00029 double getIndexOfCoincidence(long long unsigned int*frequencyTable);
00030
00036 void decrypt(std::string& inputData, const std::string& key);
00037
00044 char getCaesarKey(std::string& inputData, long long unsigned int* frequencyTable);
00045
00051 void fillRawText(const std::string& filename, std::string& rawText);
00052
00058 void fillFrequencyTableFromText (const std::string& rawText , long long unsigned int* frequencyTable);
00059
00065 long long unsigned int getKeyLength(const std::string& rawText);
00066
00073 long long int nextValidKeyIndex(const std::string &key, long long int currentIndex);
00074
00081 bool saveToFile(const std::string& filename, const std::string& data);
00082
00087 void stringToLowercase(std::string& str);
00088
00095 bool getTextDataFromFile(const std::string& filename, std::string& textData);
00096
00103 bool getKeyDataFromFile(const std::string& filename, std::string& key);
00104
00108 void printHelp();
00109
00120 bool parceArguments(int argc, char** argv, std::string& mode, std::string& inputFile, std::string&
    outputFile, std::string& keyFile);
00121
```

# Index

ASCII\_TEXT\_MARGIN  
utility.h, 29

breaker.cpp  
breakFile, 5

breaker.h  
breakFile, 6

breakFile  
breaker.cpp, 5  
breaker.h, 6

decrypt  
decryptor.cpp, 8  
decryptor.h, 10  
utility.cpp, 15  
utility.h, 21

decryptor.cpp  
decrypt, 8

decryptor.h  
decrypt, 10

encrypt  
encryptor.cpp, 11  
encryptor.h, 13

encryptor.cpp  
encrypt, 11

encryptor.h  
encrypt, 13

fillFrequencyTableFromText  
utility.cpp, 15  
utility.h, 23

fillRawText  
utility.cpp, 15  
utility.h, 23

FREQUENCY\_TABLE  
utility.h, 29

getCaesarKey  
utility.cpp, 16  
utility.h, 24

getIndexOfCoincidence  
utility.cpp, 16  
utility.h, 24

getKeyDataFromFile  
utility.cpp, 16  
utility.h, 25

getKeyLength  
utility.cpp, 17  
utility.h, 25

getTextDataFromFile

utility.cpp, 17  
utility.h, 26

main  
main.cpp, 4  
main.cpp, 3  
main, 4

nextValidKeyIndex  
utility.cpp, 18  
utility.h, 26

parceArguments  
utility.cpp, 18  
utility.h, 27

printHelp  
utility.cpp, 19  
utility.h, 28

saveToFile  
utility.cpp, 19  
utility.h, 28

src/breaker.cpp, 4  
src/breaker.h, 6, 7  
src/decryptor.cpp, 7  
src/decryptor.h, 9, 10  
src/encryptor.cpp, 11  
src/encryptor.h, 12, 13  
src/utility.cpp, 14  
src/utility.h, 20, 30

stringToLowercase  
utility.cpp, 19  
utility.h, 29

utility.cpp  
decrypt, 15  
fillFrequencyTableFromText, 15  
fillRawText, 15  
getCaesarKey, 16  
getIndexOfCoincidence, 16  
getKeyDataFromFile, 16  
getKeyLength, 17  
getTextDataFromFile, 17  
nextValidKeyIndex, 18  
parceArguments, 18  
printHelp, 19  
saveToFile, 19  
stringToLowercase, 19  
utility.h  
ASCII\_TEXT\_MARGIN, 29  
decrypt, 21

fillFrequencyTableFromText, [23](#)  
fillRawText, [23](#)  
FREQUENCY\_TABLE, [29](#)  
getCaesarKey, [24](#)  
getIndexOfCoincidence, [24](#)  
getKeyDataFromFile, [25](#)  
getKeyLength, [25](#)  
getTextDataFromFile, [26](#)  
nextValidKeyIndex, [26](#)  
parceArguments, [27](#)  
printHelp, [28](#)  
saveToFile, [28](#)  
stringToLowercase, [29](#)