ENIGMA I Project

0.1.0

Generated by Doxygen 1.12.0

Chapter 1

Enigma I Cipher Machine Simulator in C++

1.1 Overview

This project simulates the Enigma cipher machine, a device used by the German military during World War II for encrypting and decrypting messages. Actually, from all the different types of Enigma that existed, this project simulates a 1930 Enigma I. This enigma had the following components:

- **Plugboard**: Addition for this specific model, greatly increased it's cryptographic strength. The plugboard had cables that allowed the connection between two letters, swaping them. Although 13 pairs were available, only 10 were used normally.
- **Reflector**: The reflector connected outputs of the last rotor in pairs, redirecting current back through the rotors by a different route. The reflector ensured that Enigma would be self-reciprocal; thus, with two identically configured machines, a message could be encrypted on one and decrypted on the other, without the need for a bulky mechanism to switch between encryption and decryption modes. The reflector allowed a more compact design, but it also gave Enigma the property that no letter ever encrypted to itself. This was a severe cryptological flaw that was subsequently exploited by codebreakers. For our enigma, three reflectors were available: UKW A, UKW B and UKW C.
- **Rotors**: Each rotor is a disc approximately 10 cm (3.9 in) in diameter made from Ebonite or Bakelite with 26 brass, spring-loaded, electrical contact pins arranged in a circle on one face, with the other face housing 26 corresponding electrical contacts in the form of circular plates. The pins and contacts represent the alphabet — typically the 26 letters A–Z, as will be assumed for the rest of this description. When the rotors are mounted side by side on the spindle, the pins of one rotor rest against the plate contacts of the neighbouring rotor, forming an electrical connection. Inside the body of the rotor, 26 wires connect each pin on one side to a contact on the other in a complex pattern. When the Enigma I was created by 1930, it had available 3 different rotors: 'I', 'II' and 'III'. Our project is based on an Enigma I after 1934, when 2 more rotors were added: 'IV' and 'V'. Those rotors could be configured in any order, position or alphabet tyre configuration.

With an Enigma ready to work, each letter followed a closed wired circuit. After the keyboard, the signal goes to the plugboard, were it's swapped only if a cable is present on the letter pressed on the keyboard. From that the signal goes from right to left for all three rotors, being changed on all of them. After the rotors it reaches the reflector, were is changed by the appropriate pair, and then goes back to the rotors, but this time from left to right. For last, it goes one last time to the plugboard, and then to the light bulbs that show the result of the encryption.

To know more details about the different components of a Enigma and how they work, please check the following links:

- General information from it's Wikipedia.
- Technical information about it's wiring, possible configurations, and other particularities on Crypto_↩ Museum or on CIPHER_MACHINES_AND_CRYPTOLOGY.
- Video with a great visual explanation of how Enigma works from Jared Owen Youtube's channel.
- Information about Bletchley Park and how the code was broken on Britannica.

1.2 Goals

This C++ implementation wanted to provide an exploration of the Enigma machine's functionalities, including rotors, reflectors, and plugboard configurations. It was done out of curiosity to know how the machine worked and to know if I would be able to replicte it. It was done with C++ to use and mantain my skills on this languange while I go through a very innactive period of time. I'm aware it can have some improvements, like how the input is manage or error handling, but perfection was never the goal of this project.

1.3 Features

- ☑ **Rotor Configuration**: Select from multiple rotors and set their initial positions and ring configuration.
- ☑ **Reflector Configurtation**: Select from multiple reflectors.
- ☑ **Plugboard Setup**: Customize the plugboard for letter swaps.
- ☑ **Change or Reset**: The program will show the configuration entered and will ask if it's correct to allow changes. Also, the program includes a Reset option to go back to the initial configuration.
- ☑ **Encryption & Decryption**: Input either files or type message to encrypt and ciphertext to decrypt. The message can be in capital, undercase or both. Output both on a file "Output.txt" and on terminal. The program allows to encrypt a message, use the reset option, and then enter the result of the first encryption to see how it give us the first message entered.
- ☑ **Historical Accuracy**: Simulates the original Enigma I design and mechanisms.

1.4 Requirements

- C++11 or higher
- A compatible compiler (e.g., g++, clang++)

1.5 Installation

Open a terminal or navigate into the project directory and compile the source code:

```
g++ -o enigma *.cpp header/*.h src/*.cpp
```

1.6 Usage

1. Run the program:
   ```
   ./enigma
   ```
2. Follow the prompts to configure the machine:
   - Select rotors and their initial positions and ring configuration.
   - Select the reflector.
   - Set up the plugboard.
   - Choose between the available options to use a message for encryption or decryption.

1.7 Example

To encrypt the message "hello":

```
START ENIGMA CONFIGURATION:
ENTER TYPE FOR THE REFLECTOR (UKW) { A , B , C }
B
RIGHT ROTOR: ENTER TYPE { I , II , III , IV , V}
II
RIGHT ROTOR: ENTER RING CONFIGURATION { A ... Z }
A
RIGHT ROTOR: ENTER INITIAL POSITION { A ... Z }
R
MIDDLE ROTOR: ENTER TYPE { I , II , III , IV , V}
IV
MIDDLE ROTOR: ENTER RING CONFIGURATION { A ... Z }
C
MIDDLE ROTOR: ENTER INITIAL POSITION { A ... Z }
D
LEFT ROTOR: ENTER TYPE { I , II , III , IV , V}
I
LEFT ROTOR: ENTER RING CONFIGURATION { A ... Z }
B
LEFT ROTOR: ENTER INITIAL POSITION { A ... Z }
A
ENTER PAIR OF LETTERS TO SWAP ON THE PLUGBOARD { (A ... Z)(A ... Z) }
ENTER "DONE" TO FINISH THIS STEP
BD
TN
ZI
DONE

CONFIGURATION:
UKW: B
RIGHT ROTOR: TYPE: II, RING CONFIG: A, INITIAL POS: R
MIDDLE ROTOR: TYPE: IV,RING CONFIG: C, INITIAL POS: D
LEFT ROTOR: TYPE: I, RING CONFIG: B, INITIAL POS: A
PLUGBOARD CONFIG: B->D D->B I->Z N->T T->N Z->I

FINISH CONFIGURATION { Y } OR START AGAIN { N } ?
Y

OPTIONS:
--- 1 --- PROCESS MESSAGE TO ENCRYPT/DECRYPT
--- 2 --- RESET ENIGMA
--- 3 --- SHOW ENIGMA CONFIGURATION
--- 0 --- FINISH PROGRAM
1
OPTIONS:
--- 1 --- ENTER MESSAGE MANUALLY
--- 2 --- ENTER MESSAGE AS A FILE
--- 0 --- GO BACK
1
ENTER MESSAGE: hello
ENCRYPTION/DECRYPTION: BJTYG
NUMBER OF CHARACTERS: 5

OPTIONS:
--- 1 --- ENTER MESSAGE MANUALLY
--- 2 --- ENTER MESSAGE AS A FILE
--- 0 --- GO BACK
0
OPTIONS:
--- 1 --- PROCESS MESSAGE TO ENCRYPT/DECRYPT
--- 2 --- RESET ENIGMA
--- 3 --- SHOW ENIGMA CONFIGURATION
--- 0 --- FINISH PROGRAM
0
```

1.8 License

This project is licensed under the MIT License. See the LICENSE file for details.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 ConfigData Struct Reference

Struct used to contain the necessary values to configurate a Rotor, element of the cipher machine Enigma.

`#include <Rotor.h>`

Public Attributes

- string rotor_type

  Contains type of the rotor (for Enigma I it can be "I", "II", "III", "IV" or "V"). From it depends the internal wiring, so the order of the letters of the alphabet .
- int initial_pos = 0

  Contains the inital pos of the rotor (it can be 1 to 26).
- int ring_config = 0

  Contains the position of the alphabet tyre relative of the internal wiring of the rotor (it can be 1 to 26).

4.1.1 Detailed Description

Struct used to contain the necessary values to configurate a Rotor, element of the cipher machine Enigma.

To know more about how each element works see the specific class or go back to the main page.

Definition at line 26 of file Rotor.h.

4.1.2 Member Data Documentation

4.1.2.1 rotor_type

`string ConfigData::rotor_type`

Contains type of the rotor (for Enigma I it can be "I", "II", "III", "IV" or "V"). From it depends the internal wiring, so the order of the letters of the alphabet .

Definition at line 27 of file Rotor.h.

4.1.2.2 initial_pos

```
int ConfigData::initial_pos = 0
```

Contains the inital pos of the rotor (it can be 1 to 26).

Definition at line 28 of file Rotor.h.

4.1.2.3 ring_config

```
int ConfigData::ring_config = 0
```

Contains the position of the alphabet tyre relative of the internal wiring of the rotor (it can be 1 to 26).

Definition at line 29 of file Rotor.h.

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

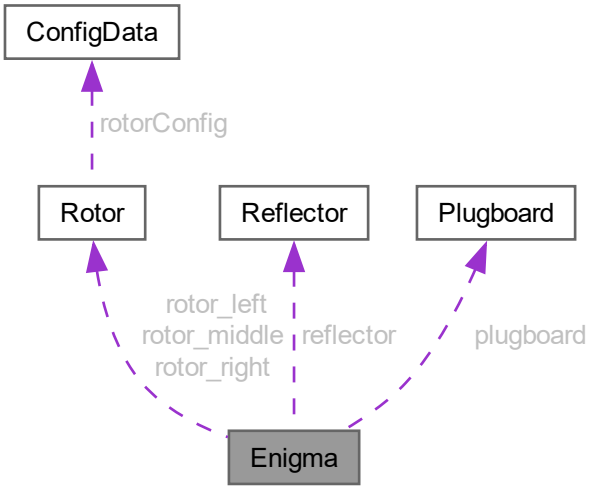
- ENIGMA_I/header/Rotor.h

4.2 Enigma Class Reference

Class meant to represent the functionalities and use of the electromechanical cipher machine developed by Germany in the early to mid 20th century.

```
#include <Enigma.h>
```

Collaboration diagram for Enigma:



Public Member Functions

- Enigma ()

  Construct a new Enigma object.
- void configRightRotor (const string type, const int ring_pos, const int ini_pos)

  Creates the rotor_right with the configuration entered as parameters.
- void configMiddleRotor (const string type, const int ring_pos, const int ini_pos)

  Creates the rotor_middle with the configuration entered as parameters.
- void configLeftRotor (const string type, const int ring_pos, const int ini_pos)

  Creates the rotor_left with the configuration entered as parameters.
- void configReflector (const string type)

  Creates the reflector of the enigma with the type entered.
- void configPlugboard (const int letter1, const int letter2)

  Configures the plugboard of the enigma adding the pair of letters entered as input.
- void reset ()

  Sets the enigma configuration back to the initial one.
- int processLetter (int letter)

  Advances enigma in one rotation and processes the letter entered as input throught all the steps for it to be encrypted.
- const string getReflectorConfig ()

  Get the Reflector configuration type.
- const ConfigData getRightRotorConfig ()

  Get the right Rotor configuration as a ConfigData object.
- const ConfigData getMiddleRotorConfig ()

  Get the middle Rotor configuration as a ConfigData object.
- const ConfigData getLeftRotorConfig ()

  Get the left Rotor configuration as a ConfigData object.
- const vector< int > & getPlugboardConfig ()

  Get the Plugboard configuration as a reference to its vector letterPair.

Private Member Functions

- void mechRotation ()

  Rotates the necessary rotors of the Enigma every time a letter is introduced.

Private Attributes

- Rotor rotor_right

  Rotor object to represent the right rotor of an ENIGMA I.
- Rotor rotor_middle

  Rotor object to represent the middle rotor of an ENIGMA I.
- Rotor rotor_left

  Rotor object to represent the left rotor of an ENIGMA I.
- Reflector reflector

  Reflector object to represent the reflector of an ENIGMA I.
- Plugboard plugboard

  Plugboard object to represent the plugboard of an ENIGMA I.

4.2.1 Detailed Description

Class meant to represent the functionalities and use of the electromechanical cipher machine developed by Germany in the early to mid 20th century.

The Enigma represented by this class is the Enigma I. Said Enigma had the following elements:

- 1 Reflector (UKW). The machine had 3 different reflectors available.
- 3 Rotors. There was 5 different rotors, and Enigma used 3 of them inserted on any order, with any configuration and with any initial pos.
- 1 Plugboard. Used to connect pairs of letters to be exchanged and add another level of complexity.

To know more about how each element works see the specific class or go back to the main page.

Warning

The methods of this class don't do a validation of its inputs.

Definition at line 33 of file Enigma.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Enigma()

```
Enigma::Enigma ()
```

Construct a new Enigma object.

Definition at line 13 of file Enigma.cpp.

4.2.3 Member Function Documentation

4.2.3.1 configRightRotor()

```
void Enigma::configRightRotor (
            const string type,
            const int ring_pos,
            const int ini_pos)
```

Creates the rotor_right with the configuration entered as parameters.

Parameters

in	*type*	Defines the type of the right rotor {I, II, III, IV, V}.
in	*ring_pos*	Defines the rotation of only the alphabet of the right rotor {1 to 26}.
in	*ini_pos*	Defines the initial position of the full right rotor {1 to 26}.

Definition at line 16 of file Enigma.cpp.

Here is the caller graph for this function:

4.2.3.2 configMiddleRotor()

```
void Enigma::configMiddleRotor (
            const string type,
            const int ring_pos,
            const int ini_pos)
```

Creates the rotor_middle with the configuration entered as parameters.

Parameters

in	*type*	Defines the type of the middle rotor {I, II, III, IV, V}.
in	*ring_pos*	Defines the rotation of only the alphabet of the middle rotor {1 to 26}.
in	*ini_pos*	Defines the initial position of the full middle rotor {1 to 26}.

Definition at line 20 of file Enigma.cpp.

Here is the caller graph for this function:

4.2.3.3 configLeftRotor()

```
void Enigma::configLeftRotor (
            const string type,
            const int ring_pos,
            const int ini_pos)
```

Creates the rotor_left with the configuration entered as parameters.

Parameters

in	*type*	Defines the type of the left rotor {I, II, III, IV, V}.
in	*ring_pos*	Defines the rotation of only the alphabet of the left rotor {1 to 26}.
in	*ini_pos*	Defines the initial position of the full left rotor {1 to 26}.

Definition at line 24 of file Enigma.cpp.

Here is the caller graph for this function:

4.2.3.4 configReflector()

```
void Enigma::configReflector (
            const string type)
```

Creates the reflector of the enigma with the type entered.

Parameters

in	*type*	Defines the type of the reflector {A, B, C}.

Definition at line 28 of file Enigma.cpp.

Here is the caller graph for this function:

4.2.3.5 configPlugboard()

```
void Enigma::configPlugboard (
            const int letter1,
            const int letter2)
```

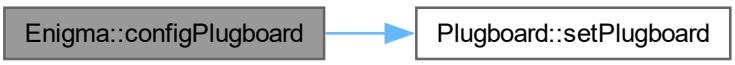
Configures the plugboard of the enigma adding the pair of letters entered as input.

Parameters

in	*letter1*	Position of the letter in the alphabet {1 to 26}.
in	*letter2*	Position of the letter in the alphabet {1 to 26}.

Definition at line 32 of file Enigma.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.6 reset()

```
void Enigma::reset ()
```

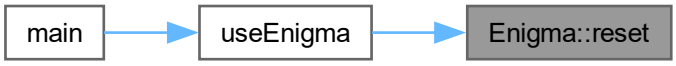
Sets the enigma configuration back to the initial one.

Reseting the enigma only implies reseting the three rotors, since the other elements are static.

Definition at line 36 of file Enigma.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



4.2.3.7 processLetter()

```
int Enigma::processLetter (
            int letter)
```

Advances enigma in one rotation and processes the letter entered as input throught all the steps for it to be encrypted.

Parameters

in	*letter*	Position of the letter in the alphabet {1 to 26} that we will encrypt.

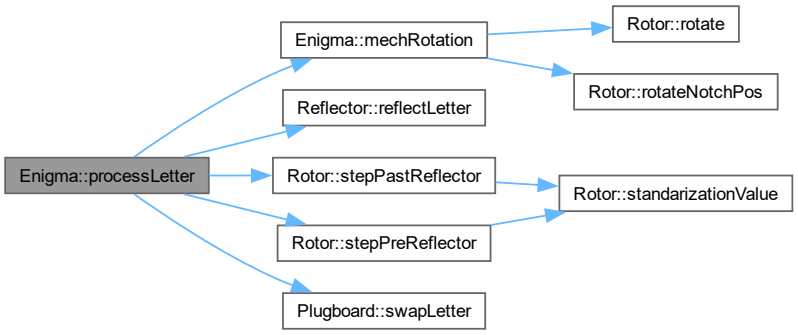
Returns

int with the position of the letter in the alphabet {1 to 26} resulting from encripting the param letter.

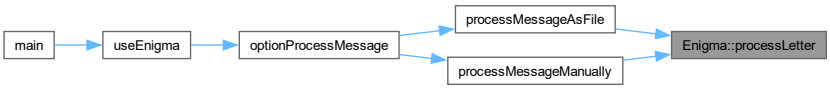
Every time a letter is entered, and after the rotation is done, it goes througt 9 different steps (2 times each rotor, 2 times through the plugboard, and once through the reflector). Also, the steps through the rotors differ if it's pre or after the reflector. That's because the signal on the real rotors goes right to left before getting to the reflector, and left to right once the reflector returns the signal.

Definition at line 68 of file Enigma.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.8 getReflectorConfig()

```
const string Enigma::getReflectorConfig ()
```

Get the Reflector configuration type.

Returns

const string with the Reflector type.

Definition at line 43 of file Enigma.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

4.2.3.9 getRightRotorConfig()

```
const ConfigData Enigma::getRightRotorConfig ()
```

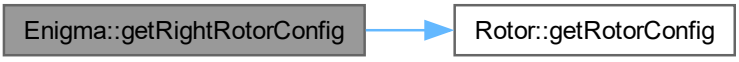
Get the right Rotor configuration as a ConfigData object.

Returns

const ConfigData object with the rotor_right configuration values.

Definition at line 47 of file Enigma.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.10 getMiddleRotorConfig()

```
const ConfigData Enigma::getMiddleRotorConfig ()
```

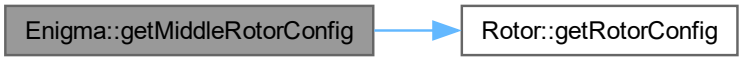
Get the middle Rotor configuration as a ConfigData object.

Returns

const ConfigData object with the rotor_middle configuration values.

Definition at line 51 of file Enigma.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.2.3.11 getLeftRotorConfig()

```
const ConfigData Enigma::getLeftRotorConfig ()
```

Get the left Rotor configuration as a ConfigData object.

Returns

const ConfigData object with the rotor_left configuration values.

Definition at line 55 of file Enigma.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

4.2.3.12 getPlugboardConfig()

```
const vector< int > & Enigma::getPlugboardConfig ()
```

Get the Plugboard configuration as a reference to its vector letterPair.

Returns

const vector<int>& that reflects the pairs of letters of the plugboard.

Definition at line 59 of file Enigma.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

4.2.3.13 mechRotation()

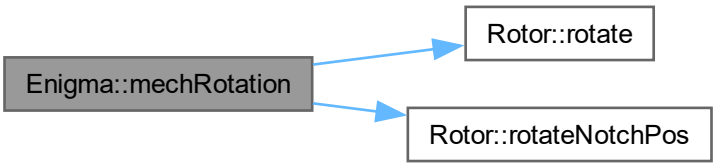
```
void Enigma::mechRotation () [private]
```

Rotates the necessary rotors of the Enigma every time a letter is introduced.

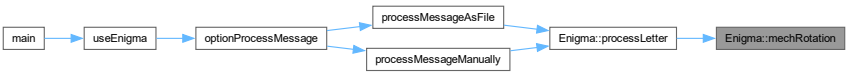
Every time processLetter(int letter) is called the Enigma makes a rotation. That process makes the rotor_right rotates every single time, and then checks for the notch position of the right and the middle rotor to decide if the middle and the left rotor have to rotate respectively. Also, this method takes in consideration the double stepping: once the middle rotor is in notch position, the mechanical process makes both the left and the middle one rotate. That means that the middle one, in this situation, rotates twice in a raw, first after the right rotor triggers it, an then because it's on the notch position. Last, for how the mechanical process works, the rotation has to be done from left to right.

Definition at line 95 of file Enigma.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4 Member Data Documentation

4.2.4.1 rotor_right

```
Rotor Enigma::rotor_right  [private]
```

Rotor object to represent the right rotor of an ENIGMA I.

Definition at line 125 of file Enigma.h.

4.2.4.2 rotor_middle

```
Rotor Enigma::rotor_middle  [private]
```

Rotor object to represent the middle rotor of an ENIGMA I.

Definition at line 126 of file Enigma.h.

4.2.4.3 rotor_left

```
Rotor Enigma::rotor_left  [private]
```

Rotor object to represent the left rotor of an ENIGMA I.

Definition at line 127 of file Enigma.h.

4.2.4.4 reflector

```
Reflector Enigma::reflector  [private]
```

Reflector object to represent the reflector of an ENIGMA I.

Definition at line 128 of file Enigma.h.

4.2.4.5 plugboard

```
Plugboard Enigma::plugboard  [private]
```

Plugboard object to represent the plugboard of an ENIGMA I.

Definition at line 129 of file Enigma.h.

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

- ENIGMA_I/header/Enigma.h
- ENIGMA_I/src/Enigma.cpp

4.3 Plugboard Class Reference

Class meant to represent a the plugboard of an Enigma mchine and its funcionalities.

`#include <Plugboard.h>`

Public Member Functions

- Plugboard ()

  Construct a new Plugboard object initializing its vector<int> letterPair with zeros.
- void setPlugboard (const int letter1, const int letter2)

  Adds the pair of letters entered in the vector letterPair.
- const int swapLetter (int letter)

  Checks the vector letterPair at the letter position, and swaps it with its content if necessary.
- const vector< int > & getConfig ()

  Get the configuration of the plugboard as a const reference to the vector<int> letterPair.

Private Attributes

- vector< int > letterPair

  Vector<int> with the pair of letters we want to swap, if any.

4.3.1 Detailed Description

Class meant to represent a the plugboard of an Enigma mchine and its funcionalities.

The Plugboard is the first and the last step of the encryption of a letter with Enigma. Is also the only one that can be skipped if it's not configured. The plugboard, if configured, works with pair of letters that will be exchange between them. It consist of a vector<int> with size =26, where cada position represents a letter, and its content is the letter that will be exchanged. Since it works in both directions, if we want to change 'A' with 'T', the position 'A' of the vector will contain a 'T', and the 'T' position an 'A'.

To know more about how each element works see the specific class or go back to the main page.

Warning

The methods of this class don't do a validation of its inputs.

Definition at line 31 of file Plugboard.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Plugboard()

```
Plugboard::Plugboard ()
```

Construct a new Plugboard object initializing its vector<int> letterPair with zeros.

Definition at line 14 of file Plugboard.cpp.

4.3.3 Member Function Documentation

4.3.3.1 setPlugboard()

```
void Plugboard::setPlugboard (
            const int letter1,
            const int letter2)
```

Adds the pair of letters entered in the vector letterPair.

Parameters

in	*letter1*	Letter 1 of a pair {1 ... 26}.
in	*letter2*	Letter 2 of a pair {1 ... 26}.

While setting the vector, it checks if any of the letters had a previous settings, and if so, it overrites it with the new one, setting the old one to 0.

Definition at line 23 of file Plugboard.cpp.

Here is the caller graph for this function:

4.3.3.2 swapLetter()

```
const int Plugboard::swapLetter (
            int letter)
```

Checks the vector letterPair at the letter position, and swaps it with its content if necessary.

Parameters

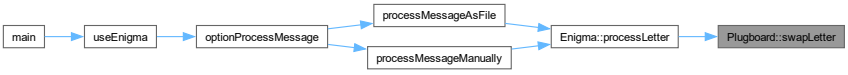
in	*letter*	Contains the letter to search {1 ... 26}.

Returns

const int with the new letter if (letterPair[letter - 1] != 0), or the same letter entered.

Definition at line 36 of file Plugboard.cpp.

Here is the caller graph for this function:



4.3.3.3 getConfig()

```
const vector< int > & Plugboard::getConfig ()
```

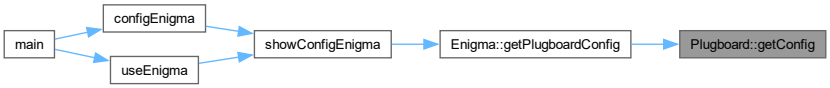
Get the configuration of the plugboard as a const reference to the vector<int> letterPair.

Returns

const vector<int>& to letterPair.

Definition at line 42 of file Plugboard.cpp.

Here is the caller graph for this function:



4.3.4 Member Data Documentation

4.3.4.1 letterPair

```
vector<int> Plugboard::letterPair  [private]
```

Vector<int> with the pair of letters we want to swap, if any.

Definition at line 65 of file Plugboard.h.

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

- ENIGMA_I/header/Plugboard.h
- ENIGMA_I/src/Plugboard.cpp

4.4 Reflector Class Reference

Class meant to represent the reflector of an Enigma and its functionalities.

```
#include <Reflector.h>
```

Public Member Functions

- Reflector ()
  Construct a new Reflector object.
- Reflector (const string type)
  Construct a new Reflector object acordingly to the type entered.
- const int reflectLetter (int pos)
  Returns the letter in the position entered.
- const string getConfig ()
  Get the configuration of the Reflector.

Private Attributes

- vector< int > notches

  Vector that contains the alphabet of the Reflector.
- string reflec_type

  Defines the type of alphabet.

4.4.1 Detailed Description

Class meant to represent the reflector of an Enigma and its functionalities.

For the Enigma I this reflector can have 3 types {'A', 'B', 'C'}, and the type decides what alphabet will contain. Similar to a plugboard fully configured, the reflector alphabet is set in pairs, where position x contains letter y and position y contains letter x. When the reflector gets a position, returns the letter in that position.

To know more about how each element works see the specific class or go back to the main page.

Warning

The methods of this class don't do a validation of its inputs.

Definition at line 31 of file Reflector.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Reflector() [1/2]

```
Reflector::Reflector ()
```

Construct a new Reflector object.

Definition at line 14 of file Reflector.cpp.

4.4.2.2 Reflector() [2/2]

```
Reflector::Reflector (
            const string type)
```

Construct a new Reflector object acordingly to the type entered.

Parameters

in	*type*	Defines the vector<int> notches of the Reflector with alphabet {'A', 'B', 'C'};

Definition at line 18 of file Reflector.cpp.

4.4.3 Member Function Documentation

4.4.3.1 reflectLetter()

```
const int Reflector::reflectLetter (
            int pos)
```

Returns the letter in the position entered.

Parameters

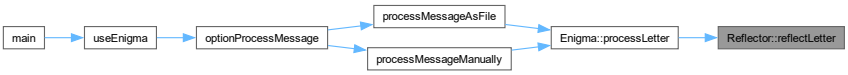
in	*pos*	Position to check in the vector<int> notches {1 ... 26}.

Returns

const int with the letter {1 ... 26}.

Definition at line 26 of file Reflector.cpp.

Here is the caller graph for this function:



4.4.3.2 getConfig()

```
const string Reflector::getConfig ()
```

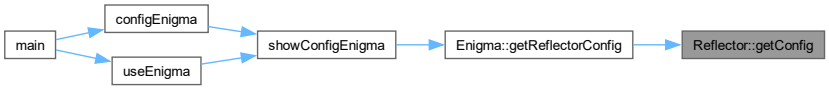
Get the configuration of the Reflector.

Returns

const string with the type of the Reflector.

Definition at line 30 of file Reflector.cpp.

Here is the caller graph for this function:



4.4.4 Member Data Documentation

4.4.4.1 notches

```
vector<int> Reflector::notches  [private]
```

Vector that contains the alphabet of the Reflector.

Definition at line 62 of file Reflector.h.

4.4.4.2 reflec_type

```
string Reflector::reflec_type  [private]
```

Defines the type of alphabet.

Definition at line 63 of file Reflector.h.

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

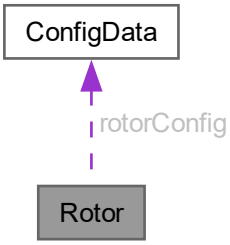
- ENIGMA_I/header/Reflector.h
- ENIGMA_I/src/Reflector.cpp

4.5 Rotor Class Reference

Class meant to represent the elements and functionalities of a rotor of the cipher machine Enigma.

```
#include <Rotor.h>
```

Collaboration diagram for Rotor:



Public Member Functions

- Rotor ()
  Construct a new Rotor object.
- Rotor (const string type, const int ring_pos, const int start_pos)
  Construct a new Rotor object using the parameters entered as input for its configuration.
- void reset ()
  Sets the configuration of the rotor back to the initial one.
- void rotate ()
  Adds one rotation to the rotor. Checks if full rotation has been done.
- const int stepPreReflector (int pos)
  Returns the letter contained in the position entered considering also rotations and ring configuration.
- const int stepPastReflector (int letter)
  Returns the position that contains the letter entered considering also rotations and ring configuration.
- const bool rotateNotchPos ()
  Returns if the rotor is in the turning_notch position.
- const ConfigData getRotorConfig ()
  Get the Rotor Config object.

Private Member Functions

- int standarizationValue (int value)

  Standarizes the value entered to a scale from 1 to 26.

Private Attributes

- ConfigData rotorConfig

  ConfigData object with the configuration of the rotor.
- int num_rotations = 0

  Keeps track of the rotor position with the number of rotations done by it.
- int turning_notch = 0

  Contains the rotor position that will affect the rotation of a different rotor at its left, if any.
- vector< int > notches

  Vector with the alphabet tyre of the rotor.

4.5.1 Detailed Description

Class meant to represent the elements and functionalities of a rotor of the cipher machine Enigma.

The rotors are the main element of an Enigma machine. They're responsable of changing the initial entry for another totally different. The output depends on the type of the rotor, the ring configuration, and the position of the rotor. From the type also depends what position is the notch position. Also, and because the real enigma just works as a wiring closed circuit, the rotor works different if the signal comes from the plugboard to the reflector, or comes from the reflector to the plugboard. To summarize, from the plugboard the rotor gets a position and returns the letter contained in that position. From the reflector, the rotor gets a letter and returns the position that contains that letter. In this class, the alphabet of the rotors are done numerically (1 to 26) inside a vector<int>. From those vectors, both the position of the element and its content represent the letters to exchange. For exemeple, with no other changes, the rotor "I" with 'A' as input will return 'E', but in our case 'A' is position 1 and 'E' is 5, the letter contained in that position. Also, since the alphabet never changes once it's set and the rotation of the rotor is always 1 position in the same direction, this class doesn't rotate the vector, it just keeps how many rotations has a rotor done and from that it calculates what position and output it should give. The rotations go from 0 (no rotations) to 25 (full roation - 1), and then starts again.

To know more about how each element works see the specific class or go back to the main page.

Warning

The methods of this class don't do a validation of its inputs.

Definition at line 45 of file Rotor.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Rotor() [1/2]

```
Rotor::Rotor ()
```

Construct a new Rotor object.

Definition at line 14 of file Rotor.cpp.

4.5.2.2 Rotor() [2/2]

```
Rotor::Rotor (
            const string type,
            const int ring_pos,
            const int start_pos)
```

Construct a new Rotor object using the parameters entered as input for its configuration.

Parameters

in	*type*	Defines the type of the rotor {I, II, III, IV, V}.
in	*ring_pos*	Defines the alphabet tyre position relative to the rotor {1 ... 26}.
in	*start_pos*	Defines the initial position of the rotor {1 ... 26}.

This method will define the differents elements of the rotor. Depending on the type entered the method will creat the vector<int> with the alphabet tyre of that type. Also, it will save the type, the ring position and the initial position on the rotor's rotorConfig element. Next, since setting the initial position works as rotating the rotor, the method will equal the num_rotations to the initial position. And last, since the ring position does change the alphabet tyre relative to the rotor, the method will modify the alphabet moving the positions of the letters as many times as the position entered.

Definition at line 25 of file Rotor.cpp.

4.5.3 Member Function Documentation

4.5.3.1 reset()

```
void Rotor::reset ()
```

Sets the configuration of the rotor back to the initial one.

The reset only affects a the rotations done by the rotor, so it equals the number of rotations to the initial pos of the rotor.

Definition at line 63 of file Rotor.cpp.

Here is the caller graph for this function:

4.5.3.2 rotate()

```
void Rotor::rotate ()
```

Adds one rotation to the rotor. Checks if full rotation has been done.

Definition at line 67 of file Rotor.cpp.

Here is the caller graph for this function:

4.5.3.3 stepPreReflector()

```
const int Rotor::stepPreReflector (
            int pos)
```

Returns the letter contained in the position entered considering also rotations and ring configuration.

Parameters

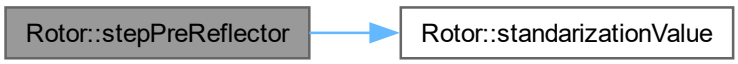
in	*pos*	Position with the letter we want to find {1 ... 26}.

Returns

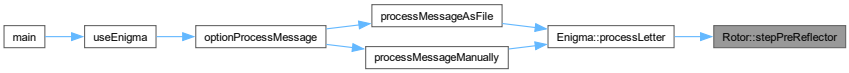
const int with the letter of that position {1 ... 26}.

Definition at line 73 of file Rotor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.4 stepPastReflector()

```
const int Rotor::stepPastReflector (
            int letter)
```

Returns the position that contains the letter entered considering also rotations and ring configuration.

Parameters

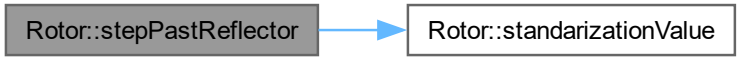
letter	Letter we want to find inside the vector<int> {1 ... 26}.

Returns

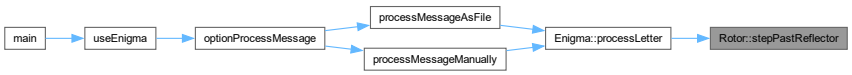
const int with the position of that letter {1 ... 26}.

Definition at line 81 of file Rotor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.5 rotateNotchPos()

```
const bool Rotor::rotateNotchPos ()
```

Returns if the rotor is in the turning_notch position.

Returns

true when rotor position (num_rotations) == notch position (turning_notch).

false when rotor position (num_rotations) != notch position (turning_notch).

Definition at line 91 of file Rotor.cpp.

Here is the caller graph for this function:

4.5.3.6 getRotorConfig()

```
const ConfigData Rotor::getRotorConfig ()
```

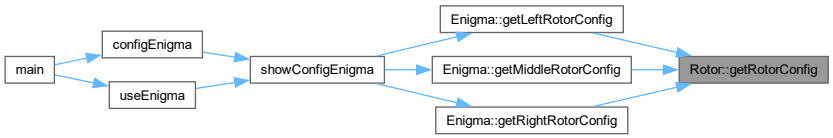
Get the Rotor Config object.

Returns

const ConfigData rotorConfig of this rotor.

Definition at line 95 of file Rotor.cpp.

Here is the caller graph for this function:



4.5.3.7 standarizationValue()

```
int Rotor::standarizationValue (
            int value)  [private]
```

Standarizes the value entered to a scale from 1 to 26.

Parameters

in	*value*	Element to standarize (value > -26 && value < 53).

Returns

int standarized value.

Definition at line 100 of file Rotor.cpp.

Here is the caller graph for this function:

4.5.4 Member Data Documentation

4.5.4.1 rotorConfig

```
ConfigData Rotor::rotorConfig  [private]
```

ConfigData object with the configuration of the rotor.

Definition at line 106 of file Rotor.h.

4.5.4.2 num_rotations

```
int Rotor::num_rotations = 0  [private]
```

Keeps track of the rotor position with the number of rotations done by it.

Definition at line 107 of file Rotor.h.

4.5.4.3 turning_notch

```
int Rotor::turning_notch = 0  [private]
```

Contains the rotor position that will affect the rotation of a different rotor at its left, if any.

Definition at line 108 of file Rotor.h.

4.5.4.4 notches

```
vector<int> Rotor::notches  [private]
```

Vector with the alphabet tyre of the rotor.

Definition at line 109 of file Rotor.h.

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

- ENIGMA_I/header/Rotor.h
- ENIGMA_I/src/Rotor.cpp

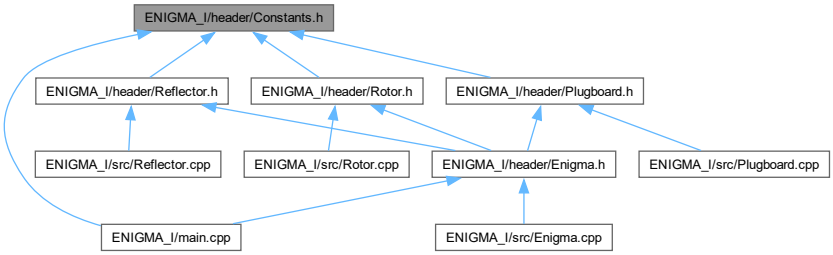
Chapter 5

File Documentation

5.1 ENIGMA_I/header/Constants.h File Reference

Declaration of constant parameters used in multiple files.

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



Variables

- const int ALPHABET_LENGTH = 26

  Size of the English alphabet {A ... Z}.
- const int LETTERS_ASCII_DIF = 64

  Number to scale the int ASCII values of capital letter from {65 ... 90} to {1 ... 26}.
- const int LETTERS_UNDERCASE_ASCII_DIF = 96

  Number to scale the int ASCII values of uppercase letter from {97 ... 122} to {1 ... 26}.

5.1.1 Detailed Description

Declaration of constant parameters used in multiple files.

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Definition in file Constants.h.

5.1.2 Variable Documentation

5.1.2.1 ALPHABET_LENGTH

```
const int ALPHABET_LENGTH = 26
```

Size of the English alphabet {A ... Z}.

Definition at line 14 of file Constants.h.

5.1.2.2 LETTERS_ASCII_DIF

```
const int LETTERS_ASCII_DIF = 64
```

Number to scale the int ASCII values of capital letter from {65 ... 90} to {1 ... 26}.

Definition at line 16 of file Constants.h.

5.1.2.3 LETTERS_UNDERCASE_ASCII_DIF

```
const int LETTERS_UNDERCASE_ASCII_DIF = 96
```

Number to scale the int ASCII values of uppercase letter from {97 ... 122} to {1 ... 26}.

Definition at line 18 of file Constants.h.

5.2 Constants.h

Go to the documentation of this file.

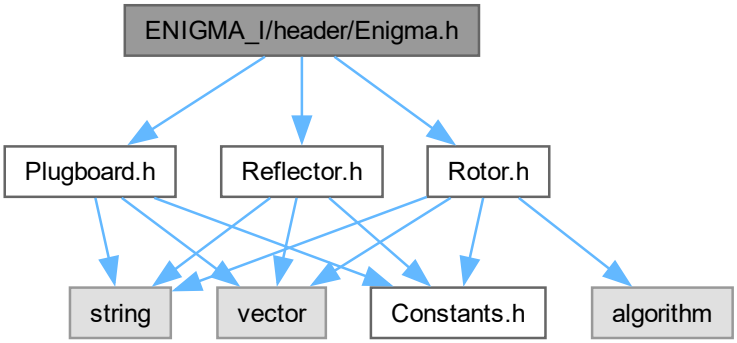
```
00001
00011 #ifndef CONSTANTS_H
00012 #define CONSTANTS_H
00013
00014 const int ALPHABET_LENGTH = 26;
00015
00016 const int LETTERS_ASCII_DIF = 64;
00017
00018 const int LETTERS_UNDERCASE_ASCII_DIF = 96;
00019
00020 #endif
```

5.3 ENIGMA_I/header/Enigma.h File Reference

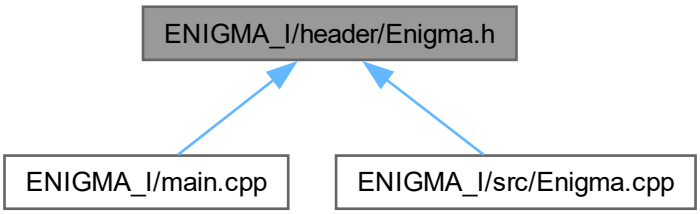
Contains the declaration of the class Enigma with its parameters and methods.

```
#include "Rotor.h"
#include "Reflector.h"
#include "Plugboard.h"
```

Include dependency graph for Enigma.h:



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



Classes

- class Enigma

  Class meant to represent the functionalities and use of the electromechanical cipher machine developed by Germany in the early to mid 20th century.

5.3.1 Detailed Description

Contains the declaration of the class Enigma with its parameters and methods.

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Definition in file Enigma.h.

5.4 Enigma.h

Go to the documentation of this file.

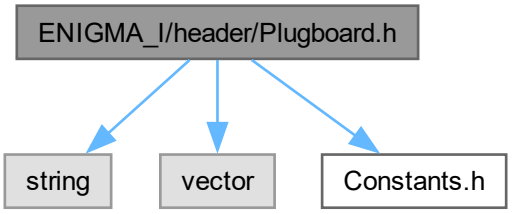
```
00001
00011 #ifndef _ENIGMA_H_
00012 #define _ENIGMA_H_
00013
00014 #include "Rotor.h"
00015 #include "Reflector.h"
00016 #include "Plugboard.h"
00017
00018 using namespace std;
00019
00033 class Enigma{
00034     public:
00035         //CONSTRUCTOR
00040         Enigma();
00041         //SETTERS
00049         void configRightRotor(const string type, const int ring_pos, const int ini_pos);
00057         void configMiddleRotor(const string type, const int ring_pos, const int ini_pos);
00065         void configLeftRotor(const string type, const int ring_pos, const int ini_pos);
00071         void configReflector(const string type);
00078         void configPlugboard(const int letter1, const int letter2);
00084         void reset();
00091         int processLetter(int letter);
00092         //GETTERS
00098         const string getReflectorConfig();
00104         const ConfigData getRightRotorConfig();
00110         const ConfigData getMiddleRotorConfig();
00116         const ConfigData getLeftRotorConfig();
00122         const vector<int>& getPlugboardConfig();
00123
00124     private:
00125         Rotor rotor_right;
00126         Rotor rotor_middle;
00127         Rotor rotor_left;
00128         Reflector reflector;
00129         Plugboard plugboard;
00130
00135         void mechRotation();
00136 };
00137
00138 #endif
```

5.5 ENIGMA_I/header/Plugboard.h File Reference

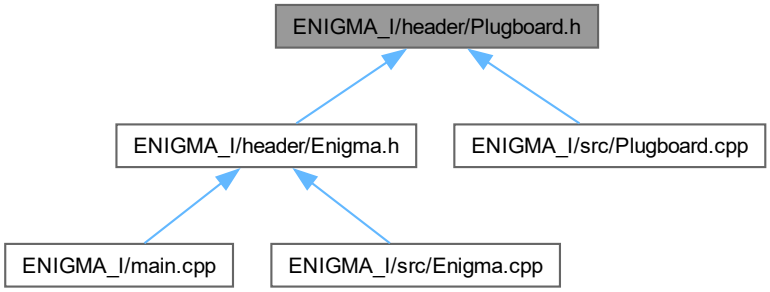
Contains the declaration of the class Plugboard with its parameters and methods.

```
#include <string>
#include <vector>
#include "Constants.h"
```

Include dependency graph for Plugboard.h:



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



Classes

- class Plugboard

  Class meant to represent a the plugboard of an Enigma mchine and its funcionalities.

5.5.1 Detailed Description

Contains the declaration of the class Plugboard with its parameters and methods.

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Definition in file Plugboard.h.

5.6 Plugboard.h

Go to the documentation of this file.

```
00001
00011 #ifndef _PLUGBOARD_H_
00012 #define _PLUGBOARD_H_
00013
00014 #include <string>
00015 #include <vector>
00016 #include "Constants.h"
00017
00018 using namespace std;
00019
00031 class Plugboard{
00032     public:
00033         //CONSTRUCTOR
00038         Plugboard();
00039
00040         //SETTER
00047         void setPlugboard(const int letter1, const int letter2);
00048
00049         //GETTERS
00056         const int swapLetter(int letter);
00062         const vector<int>& getConfig();
00063
00064     private:
00065         vector<int> letterPair;
00066 };
00067
00068 #endif
```

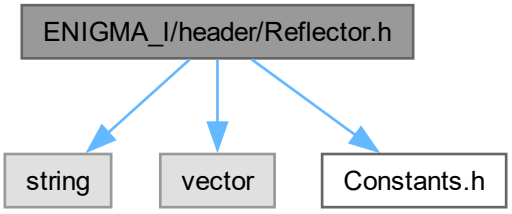
5.7 ENIGMA_I/header/Reflector.h File Reference

Contains the declaration of the class Reflector with its parameters and methods.

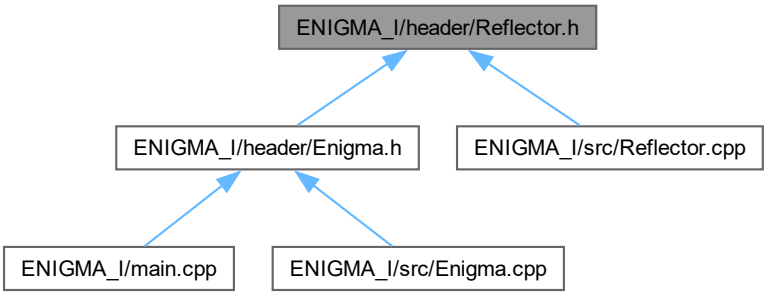
```
#include <string>
#include <vector>
```

```
#include "Constants.h"
```

Include dependency graph for Reflector.h:



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



Classes

- class Reflector

  Class meant to represent the reflector of an Enigma and its functionalities.

5.7.1 Detailed Description

Contains the declaration of the class Reflector with its parameters and methods.

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Copyright (c) 2024

Definition in file Reflector.h.

5.8 Reflector.h

Go to the documentation of this file.

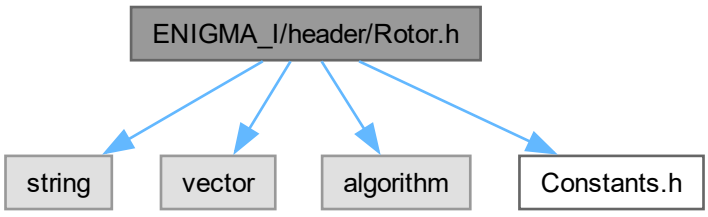
```
00001
00011 #ifndef _REFLECTOR_H_
00012 #define _REFLECTOR_H_
00013
00014 #include <string>
00015 #include <vector>
00016 #include "Constants.h"
00017
00018 using namespace std;
00019
00031 class Reflector{
00032     public:
00033         //CONSTRUCTORS
00038         Reflector();
00044         Reflector(const string type);
00045
00046         //GETTERS
00053         const int reflectLetter(int pos);
00059         const string getConfig();
00060
00061     private:
00062         vector<int> notches;
00063         string reflec_type;
00064 };
00065
00066 #endif
```

5.9 ENIGMA_I/header/Rotor.h File Reference

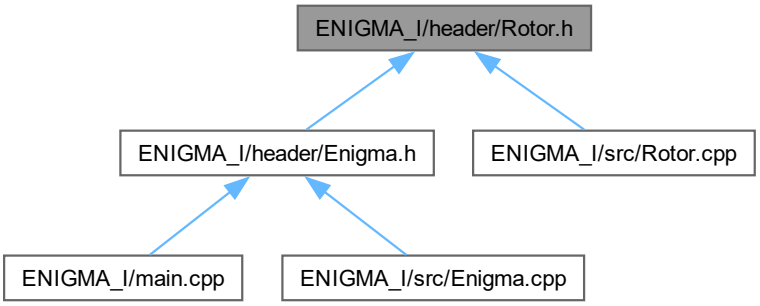
Contains the declaration of the class ConfigData and Rotor with its parameters and methods.

```
#include <string>
#include <vector>
#include <algorithm>
#include "Constants.h"
```

Include dependency graph for Rotor.h:



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



Classes

- struct ConfigData

  Struct used to contain the necessary values to configurate a Rotor, element of the cipher machine Enigma.
- class Rotor

  Class meant to represent the elements and functionalities of a rotor of the cipher machine Enigma.

5.9.1 Detailed Description

Contains the declaration of the class ConfigData and Rotor with its parameters and methods.

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Copyright (c) 2024

Definition in file Rotor.h.

5.10 Rotor.h

Go to the documentation of this file.

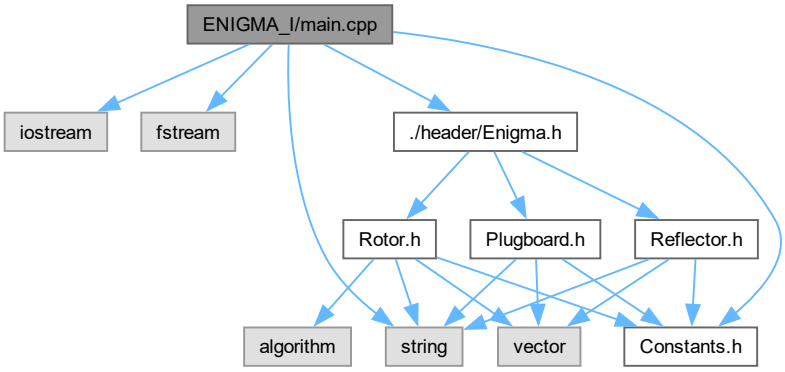
```
00001
00011 #ifndef _ROTOR_H_
00012 #define _ROTOR_H_
00013
00014 #include <string>
00015 #include <vector>
00016 #include <algorithm>
00017 #include "Constants.h"
00018
00019 using namespace std;
00020
00026 struct ConfigData{
00027     string rotor_type;
00028     int initial_pos = 0;
00029     int ring_config = 0;
00030 };
00031
00045 class Rotor{
00046     public:
00047         //CONSTRUCTORS
00052         Rotor();
00060         Rotor(const string type, const int ring_pos, const int start_pos);
00061
00062         //SETTERS
00069         void reset();
00074         void rotate();
00075
00076         //GETTERS
00083         const int stepPreReflector(int pos);
00090         const int stepPastReflector(int letter);
00097         const bool rotateNotchPos();
00103         const ConfigData getRotorConfig();
00104
00105     private:
00106         ConfigData rotorConfig;
00107         int num_rotations = 0;
00108         int turning_notch = 0;
00109         vector<int> notches;
00110
00117         int standarizationValue(int value);
00118 };
00119
00120 #endif
```

5.11 ENIGMA_I/main.cpp File Reference

Contains the main program with it's functions to use a class Enigma and encrypt/decrypt messages.

```
#include <iostream>
#include <fstream>
#include <string>
#include "./header/Enigma.h"
#include "./header/constants.h"
```

Include dependency graph for main.cpp:



Functions

- void configReflectorEnigma (Enigma &enigma)

  Ask to the user for a Reflector (UKW) configuration and calls the function to set that Enigma's component.
- void configRotorEnigma (Enigma &enigma)

  Ask to the user for three rotor configurations, one time for each rotor of an Enigma, and calls the function to set those components.
- void configPlugboardEnigma (Enigma &enigma)

  Ask to the user for a plugboard configuration and calls the function to set that Enigma's component.
- void showConfigEnigma (Enigma &enigma)

  Shows as a cout the actual configuration of our Enigma.
- bool finishConfig ()

  Ask the user if they want to finish the Enigma configuration.
- void configEnigma (Enigma &enigma)

  Calls the different functions that will be used to cofigure our Enigma.
- void processMessageManually (Enigma &enigma, ofstream &outputFile)

  Asks the user to enter a message manually for it to be used in our Enigma.
- void processMessageAsFile (Enigma &enigma, ofstream &outputFile)

  Asks the user to enter the name of the file with the message that has to be used in our Enigma.
- void optionProcessMessage (Enigma &enigma, ofstream &outputFile)

  This function controls an options menu related with how we want to enter a message to process.
- void useEnigma (Enigma &enigma)

  This function controls the main options menu with the Enigma functionalities.
- int main ()

  Creates an object Enigma and uses it.

5.11.1 Detailed Description

Contains the main program with it's functions to use a class Enigma and encrypt/decrypt messages.

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Definition in file main.cpp.

5.11.2 Function Documentation

5.11.2.1 configReflectorEnigma()

```
void configReflectorEnigma (
            Enigma & enigma)
```

Ask to the user for a Reflector (UKW) configuration and calls the function to set that Enigma's component.

The function will ask to the user to choose on of the three possible configurations for the reflector of an Enigma I. Those options can be A, B or C, and the function itself makes sure that the entry is correct before calling the Enigma method.

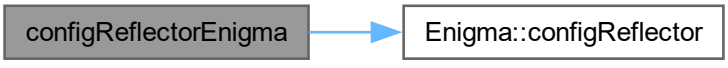
In case of an incorrect entry, it shows a message and ask for the configuration again. It has a loop that stays until a correct configuration is entered.

Parameters

in	*enigma*	Pass by reference of the object Enigma we're using.

Definition at line 27 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.11.2.2 configRotorEnigma()

```
void configRotorEnigma (
            Enigma & enigma)
```

Ask to the user for three rotor configurations, one time for each rotor of an Enigma, and calls the function to set those components.

The function will ask to the use all the parameters to set each one of the three rotors of an ENIGMA I, right, middle and left, and will check that entry before calling Enigma's method. The parameters for each rotor are:

1. TYPE of the rotor. For an ENIGMA I there are 5 types, named from 1 to 5 in roman numbers
2. RING CONFIGURATION sets an initial movement of the array of letters of a rotor before this in inserted in the Enigma
3. INITIAL POSITION sets an initial rotation of the rotor, not just the array of letters.

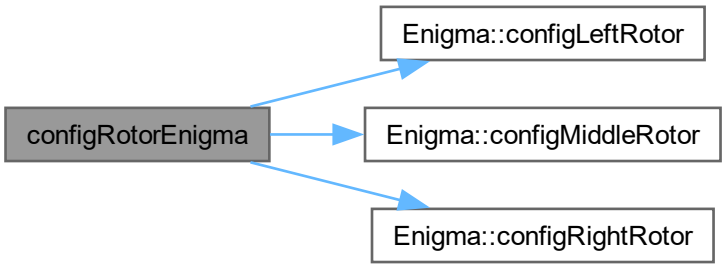
Both RING CONFIG. and INITIAL POS. are set with a letter inside the english alphabet, which the method will change to convert it in a number between 1 to 26. This function contain three loops that can't be solved until a correct configuration has been entered for all three rotors.

Parameters

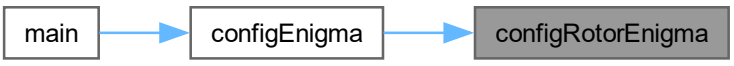
in	*enigma*	Pass by reference of the object Enigma we're using.

Definition at line 51 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.3 configPlugboardEnigma()

```
void configPlugboardEnigma (
            Enigma & enigma)
```

Ask to the user for a plugboard configuration and calls the function to set that Enigma's component.

The function will ask the user to enter pairs of letters inside an english alphabet, it'll convert them in numbers from 1 to 26 and it'll call the enigma's method to set this component.

The function will guarantee that the inputs are correct. Also it has a loop to keep entering pairs until the user wants to leave wrinting DONE as input. ENIGMA I works perfectly even if the user desides to skip the Plugboard configuration.

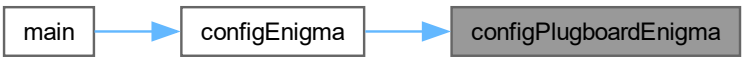
Parameters

in	*enigma*	Pass by reference of the object Enigma we're using.

Definition at line 126 of file main.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



5.11.2.4 showConfigEnigma()

```
void showConfigEnigma (
            Enigma & enigma)
```

Shows as a cout the actual configuration of our Enigma.

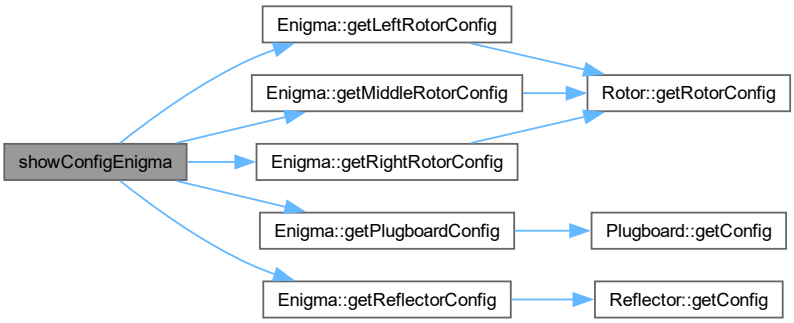
Gets from our Enigma the configuration we have and shows it with the definition of each element as a cout iostream on our terminal.

Parameters

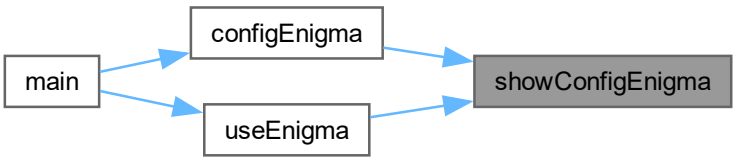
in	*enigma*	Pass by reference of the object Enigma we're using.

Definition at line 152 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.5 finishConfig()

```
bool finishConfig ()
```

Ask the user if they want to finish the Enigma configuration.

The function will ask the user if they are done with the configuration for the enigma. The entry has to be "Y" or "N".

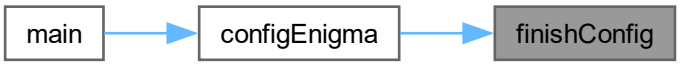
Contains a loop that can be left only with a correct entry.

Returns

true if input == "Y"
false if input == "N".

Definition at line 178 of file main.cpp.

Here is the caller graph for this function:



5.11.2.6 configEnigma()

```
void configEnigma (
            Enigma & enigma)
```

Calls the different functions that will be used to cofigure our Enigma.

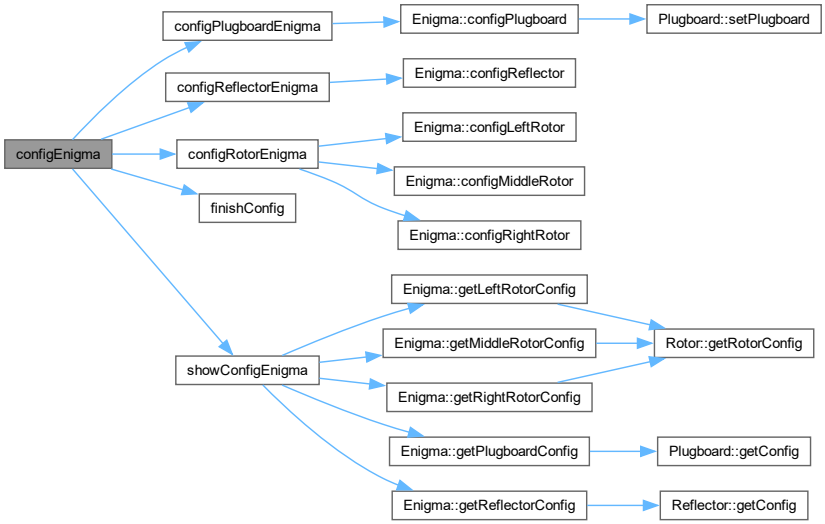
This function will call all the necessary functions to configure our ENIGMA I. It contains a loop that ends once the configuration is done (know with the function finishConfig()).

Parameters

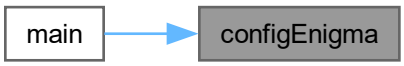
in	*enigma*	Pass by reference of the object Enigma we're using.

Definition at line 202 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.7 processMessageManually()

```
void processMessageManually (
            Enigma & enigma,
            ofstream & outputFile)
```

Asks the user to enter a message manually for it to be used in our Enigma.

This function will ask the user to enter using their terminal a message to be encrypted or decrypted by the Enigma. With the results the function will show them on the terminal and it'll write them on the given outputFile (at the end of it).

From the message, only the letters of the English alphabet (either capitals or uppercase). The function will process the message previously to ignore any character not valid for Enigma. The results will have the same format as the original ENIGMA I, groups of 5 characters with an space in between.

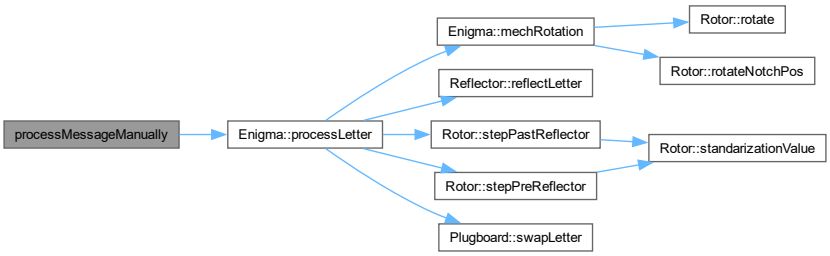
Parameters

in	*enigma*	Pass by reference of the object Enigma we're using.

out	*outputFile*	Pass by reference of a ofstream file already open that will be used to write the results of the function.

Definition at line 223 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.11.2.8 processMessageAsFile()

```
void processMessageAsFile (
            Enigma & enigma,
            ofstream & outputFile)
```

Asks the user to enter the name of the file with the message that has to be used in our Enigma.

This function will ask the user to enter using their terminal the name of the file with the message that has to be encrypted or decrypted by the Enigma. The function will open the file and process its contents. With the results the function will show them on the terminal and it'll write them on the given outputFile (at the end of it).

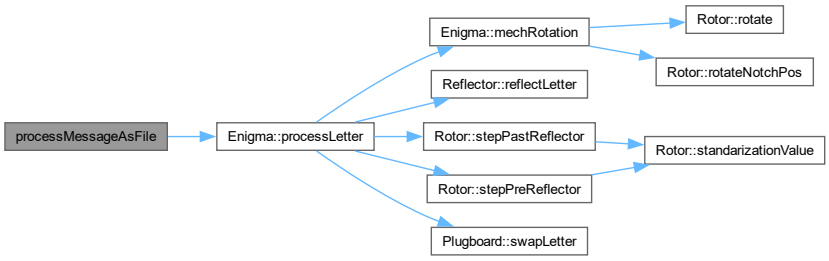
The function will check if the file with the message was opened properly. From the message, only the letters of the English alphabet (either capitals or uppercase). The function will process the message previously to ignore any character not valid for Enigma. The results will have the same format as the original ENIGMA I, groups of 5 characters with an space in between.

Parameters

in	*enigma*	Pass by reference of the object Enigma we're using.
out	*outputFile*	Pass by reference of a ofstream file already open that will be used to write the results of the function.

Definition at line 276 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.11.2.9 optionProcessMessage()

```
void optionProcessMessage (
            Enigma & enigma,
            ofstream & outputFile)
```

This function controls an options menu related with how we want to enter a message to process.

After showig the available options on the terminal, the function will ask the user which option they require.

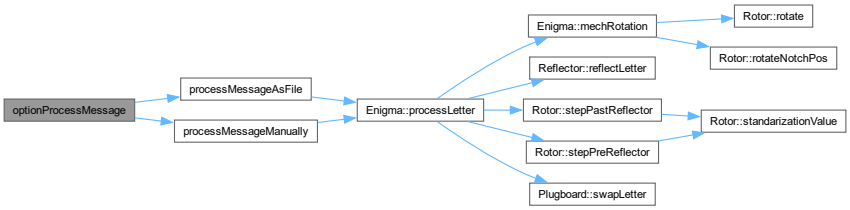
Entry must be "1" to proceed to processMessageManually(), "2" to proceed to processMessageAsFile(), or "0" to go back and exit the function. A loop prevents the user to leave until a valid entry.

Parameters

in	*enigma*	Pass by reference of the object Enigma we're using.
out	*outputFile*	Pass by reference of a ofstream file already open that will be used to write the results of the functions that will process the message.

Definition at line 337 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.11.2.10 useEnigma()

```
void useEnigma (
            Enigma & enigma)
```

This function controls the main options menu with the Enigma functionalities.

After showig the available options on the terminal, the function will ask the user which option they require. This function also opens or creates a file named "Output.txt" that will be used to have our results. The file will contain every message we want to process, and it wont be closed until we leave the function to stop the program.

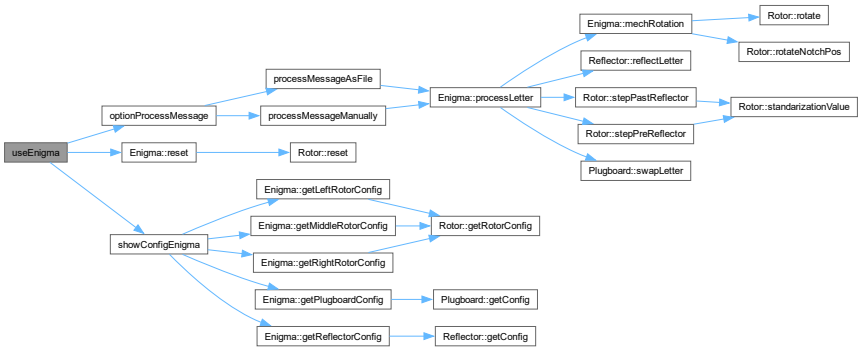
Entry must be "1" to proceed to processMessageManually(), "2" to proceed to reset our Enigma to it's initial configuration, "3" to show our Enigma configuration or "0" to go back to our main(). A loop prevents the user to leave until a valid entry.

Parameters

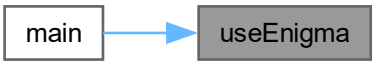
in	*enigma*	Pass by reference of the object Enigma we're using.

Definition at line 360 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.11 main()

```
int main ()
```

Creates an object Enigma and uses it.

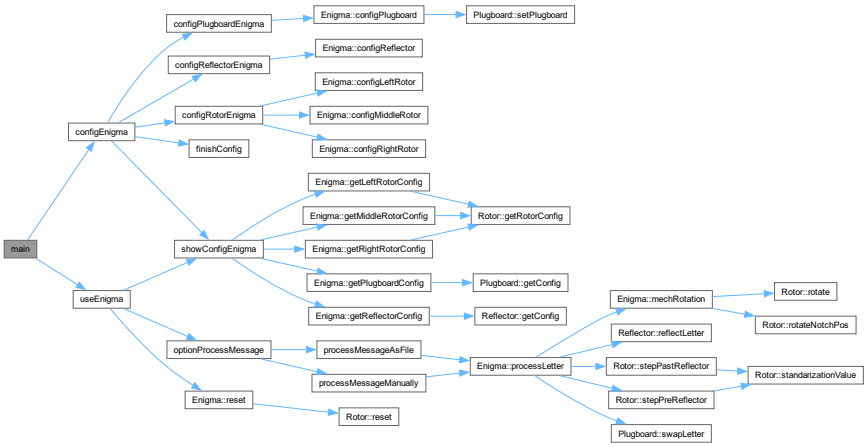
Our main will first create an object Enigma and then will call the functions configEnigma() and useEnigma().

Returns

0 to finish the program.

Definition at line 388 of file main.cpp.

Here is the call graph for this function:



5.12 main.cpp

Go to the documentation of this file.

```
00001
00011 #include <iostream>
00012 #include <fstream>
00013 #include <string>
00014
00015 #include "./header/Enigma.h"
00016 #include "./header/constants.h"
00017
00018 using namespace std;
00019
00027 void configReflectorEnigma(Enigma& enigma){
00028     bool config = false;
00029     string type;
00030     while(!config){
00031         cout « "ENTER TYPE FOR THE REFLECTOR (UKW) { A , B , C }" « endl;
00032         cin » type;
00033         if (type == "A" || type == "B" || type == "C"){
00034             enigma.configReflector(type);
00035             config = true;
00036         }
00037         else cout « "WRONG REFLECTOR TYPE" « endl;
00038     }
00039 }
00040
00051 void configRotorEnigma(Enigma& enigma){
00052     bool config = false;
00053     bool rotor1 = false;
00054     bool rotor2 = false;
00055     string type;
00056     string ringPos;
00057     string letterPos;
00058     while(!config){
00059         if (!rotor1){
00060             cout « "RIGHT ROTOR: ENTER TYPE { I , II , III , IV , V}" « endl;
00061             cin » type;
00062             if (type == "I" || type == "II" || type == "III" || type == "IV" || type == "V"){
00063                 cout « "RIGHT ROTOR: ENTER RING CONFIGURATION { A ... Z }" « endl;
00064                 cin » ringPos;
00065                 if (ringPos.size() == 1 && ringPos[0] - LETTERS_ASCII_DIF > 0 && ringPos[0] -
    LETTERS_ASCII_DIF <= ALPHABET_LENGTH){
00066                     cout « "RIGHT ROTOR: ENTER INITIAL POSITION { A ... Z }" « endl;
00067                     cin » letterPos;
00068                     if (letterPos.size() == 1 && letterPos[0] - LETTERS_ASCII_DIF > 0 && letterPos[0]
    - LETTERS_ASCII_DIF <= ALPHABET_LENGTH){
00069                         enigma.configRightRotor(type, ringPos[0] - LETTERS_ASCII_DIF, letterPos[0] -
    LETTERS_ASCII_DIF);
00070                         rotor1 = true;
00071                     }
00072                     else cout « "WRONG ROTOR POSITION" « endl;
```

```
00073                 }
00074                 else cout « "WRONG RING ROTOR PARAMETER" « endl;
00075             }
00076             else cout « "WRONG ROTOR TYPE" « endl;
00077         }
00078         else if (!rotor2){
00079             cout « "MIDDLE ROTOR: ENTER TYPE { I , II , III , IV , V}" « endl;
00080             cin » type;
00081             if (type == "I" || type == "II" || type == "III" || type == "IV" || type == "V"){
00082                 cout « "MIDDLE ROTOR: ENTER RING CONFIGURATION { A ... Z }" « endl;
00083                 cin » ringPos;
00084                 if (ringPos.size() == 1 && ringPos[0] - LETTERS_ASCII_DIF > 0 && ringPos[0] -
    LETTERS_ASCII_DIF <= ALPHABET_LENGTH){
00085                     cout « "MIDDLE ROTOR: ENTER INITIAL POSITION { A ... Z }" « endl;
00086                     cin » letterPos;
00087                     if (letterPos.size() == 1 && letterPos[0] - LETTERS_ASCII_DIF > 0 && letterPos[0]
    - LETTERS_ASCII_DIF <= ALPHABET_LENGTH){
00088                         enigma.configMiddleRotor(type, ringPos[0] - LETTERS_ASCII_DIF, letterPos[0] -
    LETTERS_ASCII_DIF);
00089                         rotor2 = true;
00090                     }
00091                     else cout « "WRONG ROTOR POSITION" « endl;
00092                 }
00093                 else cout « "WRONG RING ROTOR PARAMETER" « endl;
00094             }
00095             else cout « "WRONG ROTOR TYPE" « endl;
00096         }
00097         else {
00098             cout « "LEFT ROTOR: ENTER TYPE { I , II , III , IV , V}" « endl;
00099             cin » type;
00100             if (type == "I" || type == "II" || type == "III" || type == "IV" || type == "V"){
00101                 cout « "LEFT ROTOR: ENTER RING CONFIGURATION { A ... Z }" « endl;
00102                 cin » ringPos;
00103                 if (ringPos.size() == 1 && ringPos[0] - LETTERS_ASCII_DIF > 0 && ringPos[0] -
    LETTERS_ASCII_DIF <= ALPHABET_LENGTH){
00104                     cout « "LEFT ROTOR: ENTER INITIAL POSITION { A ... Z }" « endl;
00105                     cin » letterPos;
00106                     if (letterPos.size() == 1 && letterPos[0] - LETTERS_ASCII_DIF > 0 && letterPos[0]
    - LETTERS_ASCII_DIF <= ALPHABET_LENGTH){
00107                         enigma.configLeftRotor(type, ringPos[0] - LETTERS_ASCII_DIF, letterPos[0] -
    LETTERS_ASCII_DIF);
00108                         config = true;
00109                     }
00110                     else cout « "WRONG ROTOR POSITION" « endl;
00111                 }
00112                 else cout « "WRONG RING ROTOR PARAMETER" « endl;
00113             }
00114             else cout « "WRONG ROTOR TYPE" « endl;
00115         }
00116     }
00117 }
00118
00126 void configPlugboardEnigma(Enigma& enigma){
00127     bool config = false;
00128     string pair;
00129     int letter1;
00130     int letter2;
00131     cout « "ENTER PAIR OF LETTERS TO SWAP ON THE PLUGBOARD { (A ... Z)(A ... Z) }" « endl;
00132     cout « "ENTER \"DONE\" TO FINISH THIS STEP" « endl;
00133     while(!config){
00134         cin » pair;
00135         if (pair.size() == 2 && pair[0]!=pair[1]){
00136             letter1 = pair[0] - LETTERS_ASCII_DIF;
00137             letter2 = pair[1] - LETTERS_ASCII_DIF;
00138             if ((letter1 > 0 && letter1 <= ALPHABET_LENGTH) && (letter2 > 0 && letter2 <=
    ALPHABET_LENGTH)) enigma.configPlugboard(letter1,letter2);
00139             else cout « "WRONG CHARACTERS" « endl;
00140         }
00141         else if (pair == "DONE") config = true;
00142         else cout « "ENTRY MUST BE A PAIR OF DIFERENT CHARACTERS -> ENTER \"DONE\" TO FINISH THIS
    STEP" « endl;
00143     }
00144 }
00145
00152 void showConfigEnigma(Enigma& enigma){
00153     cout « endl « "CONFIGURATION:" « endl;
00154     cout « "UKW: " « enigma.getReflectorConfig() « endl;
00155     ConfigData rotorData = enigma.getRightRotorConfig();
00156     cout « "RIGHT ROTOR: TYPE: " « rotorData.rotor_type « ", RING CONFIG: " «
    char(rotorData.ring_config+LETTERS_ASCII_DIF+1) « ", INITIAL POS: " «
    char(rotorData.initial_pos+LETTERS_ASCII_DIF+1) « endl;
00157     rotorData = enigma.getMiddleRotorConfig();
00158     cout « "MIDDLE ROTOR: TYPE: " « rotorData.rotor_type « ",RING CONFIG: " «
    char(rotorData.ring_config+LETTERS_ASCII_DIF+1) « ", INITIAL POS: " «
    char(rotorData.initial_pos+LETTERS_ASCII_DIF+1) « endl;
00159     rotorData = enigma.getLeftRotorConfig();
00160     cout « "LEFT ROTOR: TYPE: " « rotorData.rotor_type « ", RING CONFIG: " «
```

```
    char(rotorData.ring_config+LETTERS_ASCII_DIF+1) « ", INITIAL POS: " «
    char(rotorData.initial_pos+LETTERS_ASCII_DIF+1) « endl;
00161     cout « "PLUGBOARD CONFIG: ";
00162     vector<int> plugboardConfig = enigma.getPlugboardConfig();
00163     for(auto i = plugboardConfig.begin(); i != plugboardConfig.end(); ++i) {
00164         if(*i != 0) cout « char(i-plugboardConfig.begin()+LETTERS_ASCII_DIF+1) « "->" «
    char(*i+LETTERS_ASCII_DIF) « " ";
00165         else;
00166     }
00167     cout « endl « endl;
00168 }
00169
00178 bool finishConfig(){
00179     bool done = false;
00180     string entry;
00181     while(!done){
00182         cout « "FINISH CONFIGURATION { Y } OR START AGAIN { N } ?" « endl;
00183         cin » entry;
00184         if(entry.size() == 1 && (entry == "Y" || entry == "N")){
00185             if(entry == "Y"){
00186                 done = true;
00187                 return true;
00188             }
00189             else done = true;
00190         }
00191         else cout « "WRONG ENTRY" « endl;
00192     }
00193     return false;
00194 }
00195
00202 void configEnigma(Enigma& enigma){
00203     bool config = false;
00204     while(!config){
00205         cout « "START ENIGMA CONFIGURATION:" « endl;
00206         configReflectorEnigma(enigma);
00207         configRotorEnigma(enigma);
00208         configPlugboardEnigma(enigma);
00209         showConfigEnigma(enigma);
00210         config = finishConfig();
00211         cout « endl;
00212     }
00213 }
00214
00223 void processMessageManually(Enigma& enigma, ofstream& outputFile){
00224     string message;
00225     char letter;
00226     int counter = 0;
00227     int addSeparator = 0; //Counter that will be used to add a space each 5 processed characters
00228     cout « "ENTER MESSAGE: ";
00229     cin.ignore();
00230     getline(cin, message);
00231     if (message.empty())cout « "EMPTY MESSAGE" « endl;
00232     else {
00233         outputFile « "MESSAGE ENTERED MANUALLY: " « message « endl « "ENCRYPTION/DECRYPTION: ";
00234         cout « "ENCRYPTION/DECRYPTION: ";
00235         for(string::iterator it = message.begin(); it != message.end(); it++){
00236             letter = *it - LETTERS_ASCII_DIF;
00237             if ((letter > 0 && letter <= ALPHABET_LENGTH) && (letter > 0 && letter <=
    ALPHABET_LENGTH)){
00238                 letter = enigma.processLetter(letter);
00239                 outputFile « char(letter+LETTERS_ASCII_DIF);
00240                 cout « char(letter+LETTERS_ASCII_DIF);
00241                 counter++;
00242                 addSeparator++;
00243                 if(addSeparator == 5){
00244                     outputFile « " ";
00245                     cout « " ";
00246                     addSeparator = 0;
00247                 }
00248             }
00249             else if ((*it - LETTERS_UNDERCASE_ASCII_DIF > 0 && *it - LETTERS_UNDERCASE_ASCII_DIF <=
    ALPHABET_LENGTH) && (*it - LETTERS_UNDERCASE_ASCII_DIF > 0 && *it - LETTERS_UNDERCASE_ASCII_DIF <=
    ALPHABET_LENGTH)){
00250                 letter = enigma.processLetter(*it - LETTERS_UNDERCASE_ASCII_DIF);
00251                 outputFile « char(letter+LETTERS_ASCII_DIF);
00252                 cout « char(letter+LETTERS_ASCII_DIF);
00253                 counter++;
00254                 addSeparator++;
00255                 if(addSeparator == 5){
00256                     outputFile « " ";
00257                     cout « " ";
00258                     addSeparator = 0;
00259                 }
00260             }
00261             else ;
00262         }
00263         outputFile « endl « "NUMBER OF CHARACTERS: " « counter « endl « endl;
```

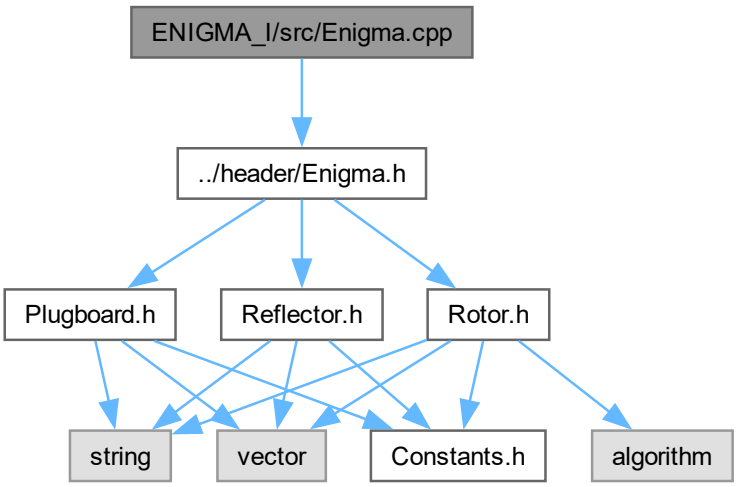
```
00264         cout « endl « "NUMBER OF CHARACTERS: " « counter « endl « endl;
00265     }
00266 }
00267
00276 void processMessageAsFile(Enigma& enigma, ofstream& outputFile){
00277     string fileName;
00278     char letter;
00279     int counter;
00280     int addSeparator; //Counter that will be used to add a space each 5 processed characters
00281     cout « "ENTER FILE NAME: ";
00282     cin » fileName;
00283     if (fileName.empty())cout « "EMPTY FILENAME" « endl;
00284     else {
00285         ifstream inputFile(fileName);
00286         if(!inputFile.is_open()) cout « "ERROR OPENING THE FILE" « endl;
00287         else{
00288             string line;
00289             while(getline(inputFile,line)){
00290                 outputFile « "MESSAGE ENTERED FROM FILE " « fileName « " : " « line « endl «
      "ENCRYPTION/DECRYPTION: ";
00291                 cout « "MESSAGE ENTERED FROM FILE " « fileName « " : " « line « endl «
      "ENCRYPTION/DECRYPTION: ";
00292                 counter = addSeparator = 0;
00293                 for(string::iterator it = line.begin(); it != line.end(); it++){
00294                     letter = *it - LETTERS_ASCII_DIF;
00295                     if ((letter > 0 && letter <= ALPHABET_LENGTH) && (letter > 0 && letter <=
      ALPHABET_LENGTH)){
00296                         letter = enigma.processLetter(letter);
00297                         outputFile « char(letter+LETTERS_ASCII_DIF);
00298                         cout « char(letter+LETTERS_ASCII_DIF);
00299                         counter++;
00300                         addSeparator++;
00301                         if(addSeparator == 5){
00302                             outputFile « " ";
00303                             cout « " ";
00304                             addSeparator = 0;
00305                         }
00306                     }
00307                     else if ((*it - LETTERS_UNDERCASE_ASCII_DIF > 0 && *it -
      LETTERS_UNDERCASE_ASCII_DIF <= ALPHABET_LENGTH) && (*it - LETTERS_UNDERCASE_ASCII_DIF > 0 && *it -
      LETTERS_UNDERCASE_ASCII_DIF <= ALPHABET_LENGTH)){
00308                         letter = enigma.processLetter(*it - LETTERS_UNDERCASE_ASCII_DIF);
00309                         outputFile « char(letter+LETTERS_ASCII_DIF);
00310                         cout « char(letter+LETTERS_ASCII_DIF);
00311                         counter++;
00312                         addSeparator++;
00313                         if(addSeparator == 5){
00314                             outputFile « " ";
00315                             cout « " ";
00316                             addSeparator = 0;
00317                         }
00318                     }
00319                     else ;
00320                 }
00321             outputFile « endl « "NUMBER OF CHARACTERS: " « counter « endl « endl;
00322             cout « endl « "NUMBER OF CHARACTERS: " « counter « endl « endl;
00323             }
00324             inputFile.close();
00325         }
00326     }
00327 }
00328
00337 void optionProcessMessage(Enigma& enigma, ofstream& outputFile){
00338     string option;
00339     bool finish = false;
00340     while(!finish){
00341         cout « "OPTIONS:" « endl;
00342         cout « "--- 1 --- ENTER MESSAGE MANUALLY" « endl;
00343         cout « "--- 2 --- ENTER MESSAGE AS A FILE" « endl;
00344         cout « "--- 0 --- GO BACK" « endl;
00345         cin » option;
00346         if(option == "1") processMessageManually(enigma,outputFile);
00347         else if(option == "2") processMessageAsFile(enigma,outputFile);
00348         else if(option == "0") finish = true;
00349         else cout « "ENTER A VALID OPTION" « endl;
00350     }
00351 }
00352
00360 void useEnigma(Enigma& enigma){
00361     ofstream outputFile("Output.txt");
00362     string option;
00363     bool finish = false;
00364     while(!finish){
00365         cout « "OPTIONS:" « endl;
00366         cout « "--- 1 --- PROCESS MESSAGE TO ENCRYPT/DECRYPT" « endl;
00367         cout « "--- 2 --- RESET ENIGMA" « endl;
00368         cout « "--- 3 --- SHOW ENIGMA CONFIGURATION" « endl;
```

```
00369         cout « "--- 0 --- FINISH PROGRAM" « endl;
00370         cin » option;
00371         if(option == "1") optionProcessMessage(enigma,outputFile);
00372         else if(option == "2") enigma.reset();
00373         else if(option == "3") showConfigEnigma(enigma);
00374         else if(option == "0") {
00375             finish = true;
00376             outputFile.close();
00377         }
00378         else cout « "ENTER A VALID OPTION" « endl;
00379     }
00380 }
00381
00388 int main(){
00389
00390     Enigma enigma1;
00391
00392     configEnigma(enigma1);
00393
00394     useEnigma(enigma1);
00395
00396     return (0);
00397 }
```

5.13 ENIGMA_I/src/Enigma.cpp File Reference

Contains implementation of the class Enigma. It's documentation in the file Enigma.h .

`#include "../header/Enigma.h"`
Include dependency graph for Enigma.cpp:



5.13.1 Detailed Description

Contains implementation of the class Enigma. It's documentation in the file Enigma.h .

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Definition in file Enigma.cpp.

5.14 Enigma.cpp

Go to the documentation of this file.

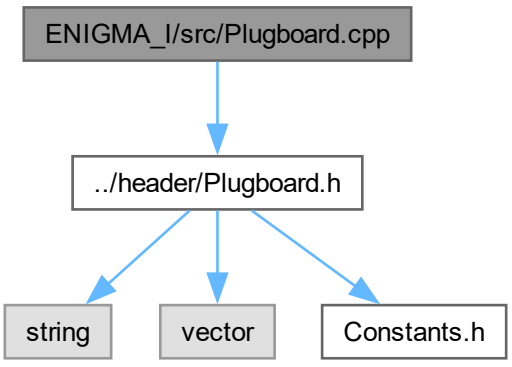
```
00001
00011 #include "../header/Enigma.h"
00012         //CONSTRUCTOR
00013         Enigma::Enigma(){}
00014
00015         //SETTERS
00016         void Enigma::configRightRotor(const string type, const int ring_pos, const int ini_pos){
00017             rotor_right = Rotor(type, ring_pos, ini_pos);
00018         }
00019
00020         void Enigma::configMiddleRotor(const string type, const int ring_pos, const int ini_pos){
00021             rotor_middle = Rotor(type, ring_pos, ini_pos);
00022         }
00023
00024         void Enigma::configLeftRotor(const string type, const int ring_pos, const int ini_pos){
00025             rotor_left = Rotor(type, ring_pos, ini_pos);
00026         }
00027
00028         void Enigma::configReflector(const string type){
00029             reflector = Reflector(type);
00030         }
00031
00032         void Enigma::configPlugboard(const int letter1, const int letter2){
00033             plugboard.setPlugboard(letter1,letter2);
00034         }
00035
00036         void Enigma::reset(){
00037             rotor_right.reset();
00038             rotor_middle.reset();
00039             rotor_left.reset();
00040         }
00041
00042         //GETTERS
00043         const string Enigma::getReflectorConfig(){
00044             return reflector.getConfig();
00045         }
00046
00047         const ConfigData Enigma::getRightRotorConfig(){
00048             return rotor_right.getRotorConfig();
00049         }
00050
00051         const ConfigData Enigma::getMiddleRotorConfig(){
00052             return rotor_middle.getRotorConfig();
00053         }
00054
00055         const ConfigData Enigma::getLeftRotorConfig(){
00056             return rotor_left.getRotorConfig();
00057         }
00058
00059         const vector<int>& Enigma::getPlugboardConfig(){
00060             return plugboard.getConfig();
00061         }
00062
00068         int Enigma::processLetter(int letter){
00069
00070             mechRotation();
```

```
00071             letter = plugboard.swapLetter(letter);
00072
00073             letter = rotor_right.stepPreReflector(letter);
00074             letter = rotor_middle.stepPreReflector(letter);
00075             letter = rotor_left.stepPreReflector(letter);
00076
00077             letter = reflector.reflectLetter(letter);
00078
00079             letter = rotor_left.stepPastReflector(letter);
00080             letter = rotor_middle.stepPastReflector(letter);
00081             letter = rotor_right.stepPastReflector(letter);
00082
00083             letter = plugboard.swapLetter(letter);
00084
00085             return letter;
00086         }
00087
00088         //PRIVATE METHOD
00095         void Enigma::mechRotation(){
00096             if(rotor_middle.rotateNotchPos()){
00097                 rotor_middle.rotate();
00098                 rotor_left.rotate();
00099             }
00100             else;
00101             if(rotor_right.rotateNotchPos())rotor_middle.rotate();
00102             else;
00103             rotor_right.rotate();
00104         }
```

5.15 ENIGMA_I/src/Plugboard.cpp File Reference

Contains implementation of the class Plugboard. It's documentation in the file Plugboard.h .

`#include "../header/Plugboard.h"`
Include dependency graph for Plugboard.cpp:



5.15.1 Detailed Description

Contains implementation of the class Plugboard. It's documentation in the file Plugboard.h .

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Definition in file Plugboard.cpp.

5.16 Plugboard.cpp

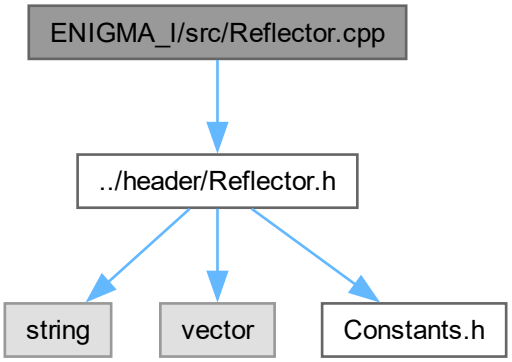
Go to the documentation of this file.

```
00001
00011 #include "../header/Plugboard.h"
00012
00013 //CONSTRUCTOR
00014 Plugboard::Plugboard(){
00015     letterPair = vector<int>(ALPHABET_LENGTH,0);
00016 }
00017
00018 //SETTER
00023 void Plugboard::setPlugboard(const int letter1, const int letter2){
00024     if (letterPair[letter1 - 1] != 0 || letterPair[letter2 - 1] != 0){
00025         int actLetter = letterPair[letter1 - 1];
00026         letterPair[actLetter - 1] = 0;
00027         actLetter = letterPair[letter2 - 1];
00028         letterPair[actLetter - 1] = 0;
00029     }
00030     else;
00031     letterPair[letter1 - 1] = letter2;
00032     letterPair[letter2 - 1] = letter1;
00033 }
00034
00035 //GETTERS
00036 const int Plugboard::swapLetter(int letter){
00037     if (letterPair[letter - 1] != 0) letter = letterPair[letter - 1];
00038     else;
00039     return letter;
00040 }
00041
00042 const vector<int>& Plugboard::getConfig(){
00043         return letterPair;
00044 }
```

5.17 ENIGMA_I/src/Reflector.cpp File Reference

Contains implementation of the class Reflector. It's documentation in the file Reflector.h .

```
#include "../header/Reflector.h"
```
Include dependency graph for Reflector.cpp:



5.17.1 Detailed Description

Contains implementation of the class Reflector. It's documentation in the file Reflector.h .

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Copyright (c) 2024

Definition in file Reflector.cpp.

5.18 Reflector.cpp

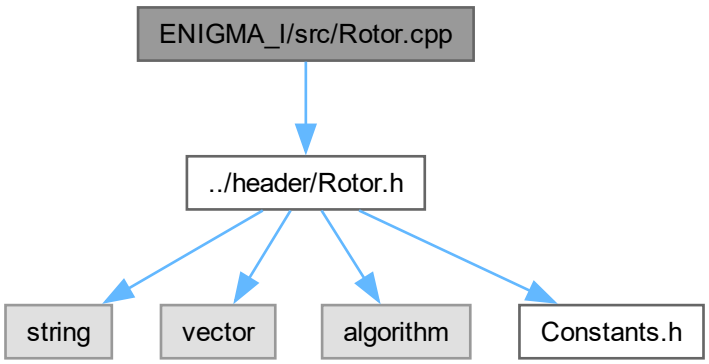
Go to the documentation of this file.

```
00001
00011 #include "../header/Reflector.h"
00012
00013 //CONSTRUCTORS
00014 Reflector::Reflector(){
00015
00016 }
00017
00018 Reflector::Reflector(string type){
00019     reflec_type = type;
00020     if (type == "A") notches = {5,10,13,26,1,12,25,24,22,2,23,6,3,18,17,21,15,14,20,19,16,9,11,8,7,4};
00021     else if (type == "B") notches =
      {25,18,21,8,17,19,12,4,16,24,14,7,15,11,13,9,5,2,6,26,3,23,22,10,1,20};
00022     else notches = {6,22,16,10,9,1,15,25,5,4,18,26,24,23,7,3,20,11,21,17,19,2,14,13,8,12};
00023 }
00024
00025 //GETTERS
00026 const int Reflector::reflectLetter(int pos){
00027     return notches[pos-1];
00028 }
00029
00030 const string Reflector::getConfig(){
00031     return reflec_type;
00032 }
00033
```

5.19 ENIGMA_I/src/Rotor.cpp File Reference

Contains implementation of the class Rotor. It's documentation in the file Rotor.h .

`#include "../header/Rotor.h"`
Include dependency graph for Rotor.cpp:



5.19.1 Detailed Description

Contains implementation of the class Rotor. It's documentation in the file Rotor.h .

Author

Lluis Torres (https://github.com/lluistp)

Version

0.1

Date

2024-09-17

Copyright

Definition in file Rotor.cpp.

5.20 Rotor.cpp

Go to the documentation of this file.

```
00001
00011 #include "../header/Rotor.h"
00012
00013 //CONSTRUTORS
00014 Rotor::Rotor(){
00015
00016 }
00017
00025 Rotor::Rotor(const string type, const int ring_pos, const int start_pos){
00026
00027     rotorConfig.initial_pos = start_pos-1;
00028     rotorConfig.ring_config = ring_pos-1;
00029     rotorConfig.rotor_type = type;
00030     num_rotations = start_pos-1;
00031
00032     if (type == "I"){
00033         notches = {5,11,13,6,12,7,4,17,22,26,14,20,15,23,25,8,24,21,19,16,1,9,2,18,3,10};
00034         turning_notch = 16;
00035     }
00036     else if (type == "II"){
00037         notches = {1,10,4,11,19,9,18,21,24,2,12,8,23,20,13,3,17,7,26,14,16,25,6,22,15,5};
00038         turning_notch = 4;
00039     }
00040     else if (type == "III"){
00041         notches = {2,4,6,8,10,12,3,16,18,20,24,22,26,14,25,5,9,23,7,1,11,13,21,19,17,15};
00042         turning_notch = 21;
00043     }
00044     else if (type == "IV"){
00045         notches = {5,19,15,22,16,26,10,1,25,17,21,9,18,8,24,12,14,6,20,7,11,4,3,13,23,2};
00046         turning_notch = 9;
00047     }
00048     else {
00049         notches = {22,26,2,18,7,9,20,25,21,16,19,4,14,8,12,24,1,23,13,10,17,15,6,5,3,11};
00050         turning_notch = 25;
00051     }
00052
00053     if (rotorConfig.ring_config != 0){
00054         for(auto i = notches.begin(); i != notches.end(); ++i) {
00055             *i = *i + rotorConfig.ring_config;
00056             if(*i > ALPHABET_LENGTH) *i = *i - (ALPHABET_LENGTH);
00057         }
00058     }
00059     else;
00060 }
00061
00062 //SETTERS
00063 void Rotor::reset(){
00064     num_rotations = rotorConfig.initial_pos;
00065 }
```

```
00066
00067 void Rotor::rotate(){
00068     num_rotations++;
00069     if (num_rotations == ALPHABET_LENGTH) num_rotations = 0;
00070 }
00071
00072 //GETTERS
00073 const int Rotor::stepPreReflector(int pos){
00074     pos = pos + num_rotations - rotorConfig.ring_config;
00075     pos = standarizationValue(pos);
00076     pos = notches[pos-1] - num_rotations;
00077     pos = standarizationValue(pos);
00078     return pos;
00079 }
00080
00081 const int Rotor::stepPastReflector(int letter){
00082     vector<int>::iterator pos;
00083     letter = letter + num_rotations;
00084     letter = standarizationValue(letter);
00085     pos = find(notches.begin(), notches.end(), letter);
00086     letter = pos - notches.begin() - num_rotations + rotorConfig.ring_config + 1;
00087     letter = standarizationValue(letter);
00088     return letter;
00089 }
00090
00091 const bool Rotor::rotateNotchPos(){
00092     return num_rotations == turning_notch;
00093 }
00094
00095 const ConfigData Rotor::getRotorConfig(){
00096     return rotorConfig;
00097 }
00098
00099 //PRIVATE METHOD
00100 int Rotor::standarizationValue(int value){
00101     if (value > ALPHABET_LENGTH) value = value - ALPHABET_LENGTH;
00102     else if(value <= 0) value = value + (ALPHABET_LENGTH);
00103     else;
00104     return value;
00105 }
00106
```

Index