My Project

Doxygen 1.9.4

Chapter 1

1.1

.

Chapter 2

2.1

.

Chapter 3

3.1

.

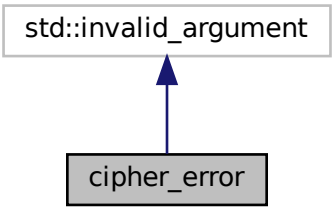
Chapter 4

4.1 cipher_error

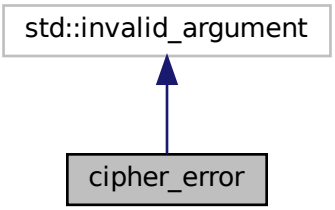
overview of the cipher_error class designed for

```
#include <modAlphaCipher.h>
```

:cipher_error:



cipher_error:



- **cipher_error** (const std::string &what_arg)
- **cipher_error** (const char ∗what_arg)

4.1.1

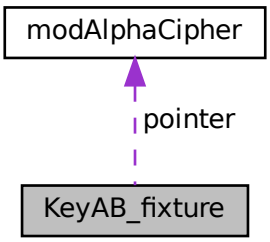
overview of the cipher_error class designed for

:

- modAlphaCipher.h

4.2 KeyAB_fixture

KeyAB_fixture:



- modAlphaCipher ∗ **pointer**

:

- main.cpp

4.3 modAlphaCipher

Class that implements the Gronsveld encryption method.

`#include <modAlphaCipher.h>`

- **modAlphaCipher** ()=delete

  Forbidden constructor without parameters.
- modAlphaCipher (const std::wstring &skey)

  Constructor for the key.
- std::wstring encrypt (const std::wstring &open_text)

  Method for encryption.
- std::wstring decrypt (const std::wstring &cipher_text)

  Method designed to decrypt.

- std::vector< int > convert (const std::wstring &s)

  Converting a string to a vector.
- std::wstring convert (const std::vector< int > &v)

  Converting a vector to a string.
- std::wstring getValidKey (const std::wstring &s)

  Key validation.
- std::wstring getValidText (const std::wstring &s)

  information about Validation preparation and placement techniques

- std::map< char, int > alphaNum

  The alphabet used in order for messages that are encrypted by the "Gronsveld"method std::wstring numAlpha = L"ABVGDEEZHZIYKLMNOPRSTUFHTSCHSHYYYY";.
- ::vector< int > **key**

  Attribute that stores the key for std encryption and decryption.

4.3.1

Class that implements the Gronsveld encryption method.

Only works with Russian-language messages

4.3.2 ()

4.3.2.1 modAlphaCipher()

```
modAlphaCipher::modAlphaCipher (
            const std::wstring & skey )
```

Constructor for the key.

The for loop is built according to the alphabetical string and adds it to the associative array at each step the symbol and its number.

```
for (unsigned i=0; i<numAlpha.size(); i++) {
alphaNum[numAlpha[i]]=i;
}
```

std::wstring	- the key in the form of a string

4.3.3

4.3.3.1 convert() [1/2]

```
std::wstring modAlphaCipher::convert (
            const std::vector< int > & v )  [inline], [private]
```

Converting a vector to a string.

A string is written to a variable of type "wstring" named "result" according to the indexes of each letter of the alphabet "numAlpha". Indexes are stored in an "int" type vector that was entered.

```
wstring result;
for(auto i:v) {
result.push_back(numAlpha[i]);
}
```

a string of text like "wstring"

4.3.3.2 convert() [2/2]

```
std::vector< int > modAlphaCipher::convert (
            const std::wstring & s )  [inline], [private]
```

Converting a string to a vector.

Numbers are written to an "int" vector named "result", which are the indexes of the alphabet "numAlpha" used for the string, which was received at the entrance.

```
vector<int> result;
for(auto c:s) {
result.push_back(alphaNum[c]);
}
```

std::vector <int>, which stores the letter indexes of the message from the alphabet "numAlpha"

4.3.3.3 decrypt()

```
std::wstring modAlphaCipher::decrypt (
            const std::wstring & cipher_text )
```

Method designed to decrypt.

Here, the work vector is first formed from the string of the ciphertext using the method convert(). The ciphertext is also checked for errors using the getValidAlphabetText() method.

```
vector<int> work = convert(getValidAlphabetText(cipher_text));
```

If we added the key value when encrypting, then we need to subtract the key value when decrypting. And in order not to ∗ get negative values, the value of the module is also added, since such an addition does not affect the result of the module.

```
for(unsigned i=0; i < work.size(); i++) {
work[i] = (work[i] + alphaNum.size() - key[i % key.size()]) % alphaNum.size();
}
```

std::wstring	cipher_text - the message that needs to be decrypted

cipher_error,if	the line that was entered is empty or contains invalid characters

a string of decrypted text of the type "wstring"

4.3.3.4 encrypt()

```
std::wstring modAlphaCipher::encrypt (
            const std::wstring & open_text )
```

Method for encryption.

Here, the work vector is first formed from a plaintext string using the method convert(). And the text is also checked for errors using the getValidAlphabetText() method.

```
vector<int> work = convert(getValidAlphabetText(open_text));
```

Then, in the loop, a key element is added to each element of the vector modulo the size the alphabet. Since the key may be shorter than the text, when indexing the key , the $*$ operation is performed modulo the key size. This allows the key to be used cyclically on long messages.

```
for(unsigned i=0; i < work.size(); i++) {
work[i] = (work[i] + key[i % key.size()]) % alphaNum.size();
}
```

Next, when the value is returned, the work vector is converted back to a string.

std::wstring	open_text - the message that needs to be encrypted

cipher_error,if	the line that was entered is empty or contains invalid characters

a string of encrypted text of type "wstring"

4.3.3.5 getValidKey()

```
std::wstring modAlphaCipher::getValidKey (
            const std::wstring & s )  [inline], [private]
```

Key validation.

First, the entered key is checked for emptiness using the usual condition. If the key is not empty, then it is checked for invalid characters.

Lowercase letters of the alphabet are converted to uppercase.

std::wstring	s is the key that needs to be checked for errors, in the form of a string

cipher_error	if the key is empty or contains invalid characters

Key in the form of a string of type "wstring", which has been successfully validated

4.3.3.6 getValidText()

```
wstring modAlphaCipher::getValidText (
            const std::wstring & s )  [inline], [private]
```

information about Validation preparation and placement techniques

related to the introduction of the new method are checked for simplicity with the help of ordinary education. If the text is not empty, then it is checked for invalid characters.

, Alawite string letters are being translated into written ones.

std::wstring	s is a string technology for tuning or arranging that moves a longer distance.

cipher_error,if	the text is a result or contains invalid characters

the text as a string of type "wstring", which you have successfully executed.

4.3.4

4.3.4.1 alphaNum

```
std::map<char,int> modAlphaCipher::alphaNum  [private]
```

The alphabet used in order for messages that are encrypted by the "Gronsveld"method std::wstring numAlpha = L"ABVGDEEZHZIYKLMNOPRSTUFHTSCHSHYYYY";.

Associative array "number by symbol"

:

- modAlphaCipher.h
- modAlphaCipher.cpp

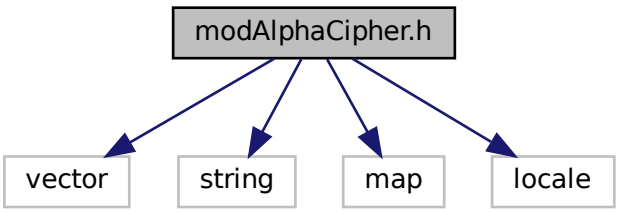
Chapter 5

5.1 modAlphaCipher.h

Description of the modAlphaCipher class.

```
#include <vector>
#include <string>
#include <map>
#include <locale>
```

modAlphaCipher.h:



- class modAlphaCipher

  Class that implements the Gronsveld encryption method.
- class cipher_error

  overview of the cipher_error class designed for

5.1.1

Description of the modAlphaCipher class.

Ivashkin A.G.

1.0

22.01.2023

IBT PSU

5.2 modAlphaCipher.h

```
..
1
9 #pragma once
10 #include <vector>
11 #include <string>
12 #include <map>
13 #include <locale>
14 using namespace std;
18 class modAlphaCipher
19 {
20 private:
23 std::map <char,int> alphaNum; //associative array "number by symbol"
25 ::vector <int> key; //key
37 std::vector<int> convert(const std::wstring& s);//string-vector conversion
49 std::wstring convert(const std::vector<int>& v);//vector-string conversion
58     std::wstring getValidKey(const std::wstring & s);
67     std::wstring getValidText(const std::wstring & s);
68 public:
70  modAlphaCipher()=delete;//disable the constructor without parameters
81    modAlphaCipher(const std::wstring& skey); //constructor for setting the key
102 std::wstring encrypt(const std::wstring& open_text); //encryption
123     std::wstring decrypt(const std::wstring& cipher_text);
124 };
126 class cipher_error:  public std::invalid_argument
127 {
128 public:
129     explicit cipher_error (const std::string& what_arg):
130         std::invalid_argument(what_arg) {}
131     explicit cipher_error (const char* what_arg):
132         std::invalid_argument(what_arg) {}
133 };
```