

```
1
2 #include "VignereForwardIterator.h"
3 #include <cassert>
4 /*
5  * @brief The decodeCurrentChar function will be use for handling      ↗
6  *   decryptions from decoded letter to original letter
7  * based on the created mapping table given in the fMappingTable. It will ↗
8  * choose which original letter would suit based on both
9  * the encoded letter and the keyword letter. For example, fKeys = 'A', ↗
10  * fCurrentChar (encoded) = 'D' -> fCurrentChar (decrypted) = 'C'
11  * @param None
12  * @return void
13  */
14 void VignereForwardIterator::decodeCurrentChar() noexcept {
15     //Check if the current character is alphabetic
16     if (std::isalpha(fCurrentChar)) {
17         //Get the row based on the current keyword letter
18         size_t row = *fKeys - 'A';
19         for (size_t i = 0; i < CHARACTERS; ++i) {
20             //Iterating through the table
21             //Check if the character at the row and the column matches ↗
22             with the current character
23             if (fMappingTable[row][i] == std::toupper(fCurrentChar)) {
24                 //Check if the character is uppercase
25                 if (std::isupper(fCurrentChar))
26                     //If then set the founded original character in ↗
27                     uppercase
28                     fCurrentChar = 'A' + i;
29                 else
30                     //Else then set the founded original character in ↗
31                     lowercase
32                     fCurrentChar = 'a' + i;
33                 //Must break otherwise it will continuously running the ↗
34                 loop -> cause error
35                 break;
36             }
37         }
38         //Increment the keyword letter to one index
39         fKeys++;
40     }
41 }
42 /*
43  * @brief The encodeCurrentChar function will be use for handling      ↗
44  *   encryptions from original letter to encoded letter
45  * based on the created mapping table given in the fMappingTable. It will ↗
46  * choose which letter would suit based on both
47  * the source letter and the keyword letter. For example, fKeys = 'A', ↗
48  * fCurrentChar (decrypted) = 'C' -> fCurrentChar (encoded) = 'D'
49  * @param None
```

```
40 * @return void
41 */
42 void VignereForwardIterator::encodeCurrentChar() noexcept {
43     //Check if the current character is alphabetic
44     if (std::isalpha(fCurrentChar)) {
45         //Get the index value of the keyword letter
46         size_t row = *fKeys - 'A';
47         //Get the index value of the source letter
48         size_t col = std::toupper(fCurrentChar) - 'A';
49         //Get the encrypted keyword based on the index of source and keyword letter
50         char temp = fMappingTable[row][col];
51         //Check that if the current char is upper-case
52         if (std::isupper(fCurrentChar))
53             //If then assign the encrypted keyword
54             fCurrentChar = temp;
55         else
56             //Else then assign the encrypted keyword in lower case
57             fCurrentChar = std::tolower(temp);
58         //Increment the keyword letter to one index
59         fKeys++;
60     }
61 }
62
63 /*
64 * @brief Constructor of VignereForwardIterator class, accepts the keyword, the phrase that wanted to be encrypted/decrypted and which mode to run
65 * @param aKeyword (const std::string&): The keyword
66 * @param aSource (const std::string&): The phrase that needs to encode/decode
67 * @param aMode (EVignereMode): Vignere Mode (Encode/Decode)
68 * @return None
69 */
70 VignereForwardIterator::VignereForwardIterator(const std::string& aKeyword, const std::string& aSource, EVignereMode aMode) noexcept
71 : fMode(aMode), fKeys(aKeyword, aSource), fSource(aSource), fIndex(0)
72 {
73     //Initialize the table
74     initializeTable();
75     //Check if the encode/decode string is empty and the first character is empty
76     if (!fSource.empty() && std::isalpha(fSource.at(0)))
77     {
78         //If not then set the current char to be the first char of the phrase
79         fCurrentChar = fSource[fIndex];
80         if (fMode == EVignereMode::Decode)
81             //If EVignereMode is decode then call decodeCurrentChar()
```

```
82         decodeCurrentChar();
83     else
84         //Else call encodeCurrentChar()
85         encodeCurrentChar();
86     }
87 }
88
89 /*
90 * Get the keyword character where the iterator is pointed on
91 * @param None
92 * @return char: The keyword character where the iterator is pointed on
93 */
94 char VignereForwardIterator::operator*() const noexcept {
95     return fCurrentChar;
96 }
97
98 /*
99 * Advance the iterator to one index and return the updated iterator
100 * @param None
101 * @return VignereForwardIterator&: the updated iterator
102 */
103 VignereForwardIterator& VignereForwardIterator::operator++() noexcept {
104     //Ensurt that fIndex must not exceed the source phrase length
105     assert(fIndex++ < fSource.size());
106     //set the current char to be the character at the advanced index
107     fCurrentChar = fSource[fIndex];
108     //Must ensure that current char is alphabetic
109     if (std::isalpha(fCurrentChar))
110     {
111         if (fMode == EVignereMode::Decode)
112             //If EVignereMode is decode then call decodeCurrentChar()
113             decodeCurrentChar();
114         else
115             //Else call encodeCurrentChar()
116             encodeCurrentChar();
117     }
118     //Return the VignereForwardIterator instance
119     return *this;
120 }
121
122 /*
123 * Advance the iterator to one index and return the old iterator
124 * @param None
125 * @return VignereForwardIterator&: the old iterator
126 */
127 VignereForwardIterator VignereForwardIterator::operator++(int) noexcept ↗
128 {
129     VignereForwardIterator fTemp = *this;
130     ++(*this);
```

```
130     return fTemp;
131 }
132
133 /*
134 * Return the equality of the underlying collection and the position of
135 * this
136 * object and other VignereForwardIterator object
137 * @param aOther (const VignereForwardIterator&): Other
138 * VignereForwardIterator object
139 * @return bool: The equality between objects
140 */
141 bool VignereForwardIterator::operator==(const VignereForwardIterator&
142     aOther) const noexcept {
143     return fIndex == aOther.fIndex && fSource == aOther.fSource;
144 }
145
146 /*
147 * Return the in-equality of the underlying collection and the position of
148 * this
149 * object and other VignereForwardIterator object
150 * @param aOther (const VignereForwardIterator&): Other
151 * VignereForwardIterator object
152 * @return bool: The in-equality between objects
153 */
154 bool VignereForwardIterator::operator!=(const VignereForwardIterator&
155     aOther) const noexcept {
156     return !(*this == aOther);
157 }
158
159 /*
160 * Return a copy of 'this' iterator object positioned in the first index of
161 * the string
162 * @param None
163 * @return VignereForwardIterator: A copy of 'this' iterator at the first
164 * index
165 */
166 VignereForwardIterator VignereForwardIterator::begin() const noexcept {
167     VignereForwardIterator fTemp = *this;
168     fTemp.fIndex = 0;
169     return fTemp;
170 }
171
172 /*
173 * Return a copy of 'this' iterator object positioned in the last index of
174 * the string
175 * @param None
176 * @return VignereForwardIterator: A copy of 'this' iterator at the last
177 * index
178 */
```

---

```
169 VignereForwardIterator VignereForwardIterator::end() const noexcept {
170     VignereForwardIterator fTemp = *this;
171     fTemp.fIndex = fSource.size();
172     return fTemp;
173 }
174
175
```