

Swinburne University of Technology*Faculty of Science, Engineering and Technology***MIDTERM COVER SHEET**

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: Midterm: Solution Design & Iterators
Due date: April 26, 2024, 10:30
Lecturer: Dr. Markus Lumpe

Your name: _____ **Your student ID:** _____

Marker's comments:

Problem	Marks	Obtained
1	106	
2	194	
Total	300	

```
1  #include "KeyProvider.h"
2  #include <cassert>
3  #include <cctype>
4
5  std::string KeyProvider::preprocessString(const std::string& aString) noexcept
6  {
7      std::string lResult;
8      for (char ch : aString)
9      {
10         if (std::isalpha(static_cast<unsigned char>(ch))) //
11             static_cast<unsigned char>() to prevent out of scope later
12             {
13                 lResult += std::toupper(static_cast<unsigned char>(ch));
14             }
15     }
16     return lResult;
17 }
18 KeyProvider::KeyProvider(const std::string& aKeyword, const std::string&
19     aSource) noexcept
20 {
21     fIndex = 0;
22     std::string lProcessedSource = preprocessString(aSource);
23     std::string lProcessedKeyword = preprocessString(aKeyword);
24     fKeys = "";
25
26     while (fKeys.length() < lProcessedSource.length()) // to make sure that
27         the fkey >= source
28     {
29         fKeys += lProcessedKeyword;
30     }
31     fKeys = fKeys.substr(0, lProcessedSource.length()); // then trim to make
32     key = source length
33     assert(fKeys.length() == lProcessedSource.length());
34 }
35 char KeyProvider::operator*() const noexcept
36 {
37     if (fIndex < fKeys.size())
38     {
39         return fKeys[fIndex];
40     }
41     // if index is out of range -> reset
42     return fKeys[fIndex % fKeys.size()];
43 }
44 // prefix
45 KeyProvider& KeyProvider::operator++() noexcept
46 {
47     ++fIndex;
48     fIndex %= fKeys.size(); // to double ensure it reset
49     return *this;
50 }
51
52 // postfix
```

```
53 KeyProvider KeyProvider::operator++(int) noexcept
54 {
55     KeyProvider temp = *this;
56     ++(*this);
57     return temp;
58 }
59
60 bool KeyProvider::operator==(const KeyProvider& aOther) const noexcept
61 {
62     return fKeys == aOther.fKeys && fIndex == aOther.fIndex;
63 }
64
65 bool KeyProvider::operator!=(const KeyProvider& aOther) const noexcept
66 {
67     return !(*this == aOther);
68 }
69
70 KeyProvider KeyProvider::begin() const noexcept
71 {
72     KeyProvider temp(*this);
73     temp.fIndex = 0;
74     return temp;
75 }
76
77 KeyProvider KeyProvider::end() const noexcept
78 {
79     KeyProvider temp(*this);
80     temp.fIndex = fKeys.size();
81     return temp;
82 }
83
```

```
1 #include "VigenereForwardIterator.h"
2 #include <cctype>
3 #include <cassert>
4 #include <iostream> //for debug purpose
5
6 VigenereForwardIterator::VigenereForwardIterator(const std::string& aKeyword,
7 const std::string& aSource, EVigenereMode aMode) noexcept
8 : fKeys(KeyProvider(aKeyword, aSource)), fSource(aSource), fMode(aMode),
9 fIndex(0)
10 {
11     initializeTable();
12     if (!fSource.empty())
13     {
14         //fMode switch
15         if (fMode == EVigenereMode::Encode)
16         {
17             encodeCurrentChar();
18         }
19         else if (fMode == EVigenereMode::Decode)
20         {
21             decodeCurrentChar();
22         }
23     }
24 }
25
26 //the mode part
27 void VigenereForwardIterator::encodeCurrentChar() noexcept
28 {
29     // if non-alpha -> not changed
30     if (fIndex >= fSource.length() || !isalpha(fSource[fIndex]))
31     {
32         fCurrentChar = fSource[fIndex];
33     }
34     else
35     {
36         char keyChar = toupper(static_cast<unsigned char>(*fKeys));
37         char sourceChar = fSource[fIndex];
38         int row = keyChar - 'A'; // the row index from the key character
39         int column = toupper(static_cast<unsigned char>(sourceChar)) -
40 'A'; //the col index from the source character
41
42         fCurrentChar = isupper(sourceChar) ? fMappingTable[row][column] :
43 tolower(fMappingTable[row][column]);
44
45         assert(row >= 0 && row < CHARACTERS); //for debug purpose
46         assert(column >= 0 && column < CHARACTERS);
47     }
48 }
49
50 void VigenereForwardIterator::decodeCurrentChar() noexcept
51 {
52     if (fIndex >= fSource.length() || !isalpha(fSource[fIndex]))
53     {
54         fCurrentChar = fSource[fIndex];
55     }
56     else
57     {
58         char keyChar = toupper(static_cast<unsigned char>(*fKeys));
59         char sourceChar = fSource[fIndex];
60         int row = keyChar - 'A'; // the row index from the key character
61         int column = toupper(static_cast<unsigned char>(sourceChar)) -
62 'A'; //the col index from the source character
63
64         fCurrentChar = isupper(sourceChar) ? fMappingTable[row][column] :
65 tolower(fMappingTable[row][column]);
66
67         assert(row >= 0 && row < CHARACTERS); //for debug purpose
68         assert(column >= 0 && column < CHARACTERS);
69     }
70 }
```

```
53     {
54         char keyChar = toupper(static_cast<unsigned char>(*fKeys));
55         char sourceChar = fSource[fIndex];
56         int row = keyChar - 'A';
57
58         // iterates over the column in the specified row
59         int column = 0;
60         while (column < CHARACTERS && fMappingTable[row][column] != toupper    ↗
61             (static_cast<unsigned char>(sourceChar)))
62         {
63             ++column;
64         }
65
66         //compare to source char -> upper || lower
67         fCurrentChar = isupper(sourceChar) ? ('A' + column) : tolower('A' +    ↗
68             column);
69     }
70 char VigenereForwardIterator::operator*() const noexcept
71 {
72     return fCurrentChar;
73 }
74
75 //prefix
76 VigenereForwardIterator& VigenereForwardIterator::operator++() noexcept
77 {
78     ++fIndex;
79     ++fKeys;
80     if (fIndex < fSource.size())
81     {
82         if (fMode == EVigenereMode::Encode)
83         {
84             encodeCurrentChar();
85         }
86         else
87         {
88             decodeCurrentChar();
89         }
90     }
91     return *this;
92 }
93
94 //postfix
95 VigenereForwardIterator VigenereForwardIterator::operator++(int) noexcept
96 {
97     VigenereForwardIterator temp = *this;
98     ++(*this);
99     return temp;
100 }
101
102 bool VigenereForwardIterator::operator==(const VigenereForwardIterator&    ↗
103     aOther) const noexcept
104 {
105     return fIndex == aOther.fIndex && fSource == aOther.fSource;
```

```
106
107 // Inequality comparison operator
108 bool VigenereForwardIterator::operator!=(const VigenereForwardIterator& aOther) const noexcept 7
109 {
110     return !(*this == aOther);
111 }
112
113 VigenereForwardIterator VigenereForwardIterator::begin() const noexcept
114 {
115     return VigenereForwardIterator(*this);
116 }
117
118 VigenereForwardIterator VigenereForwardIterator::end() const noexcept
119 {
120     VigenereForwardIterator temp = *this;
121     temp.fIndex = fSource.size();
122     return temp;
123 }
```