

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 <cctype>
3 #include <cassert>
4
5 std::string KeyProvider::preprocessString(const std::string& aString) ➤
    noexcept {
6     std::string result;
7     for (char c : aString)
8     {
9         if (isalpha(c))
10        {
11            result += toupper(c);
12        }
13    }
14    return result;
15 }
16
17 KeyProvider::KeyProvider(const std::string& aKeyword, const std::string& ➤
    aSource) noexcept {
18     std::string lKeyword = preprocessString(aKeyword);
19     std::string lSource = preprocessString(aSource);
20     for (size_t i = 0; i < lSource.length(); i++)
21     {
22         fKeys += lKeyword[i % lKeyword.length()];
23     }
24     fIndex = 0;
25     assert(fKeys.length() == lSource.length());
26 }
27
28 char KeyProvider::operator*() const noexcept {
29     return fKeys[fIndex];
30 }
31
32 KeyProvider& KeyProvider::operator++() noexcept {
33     fIndex++;
34     return *this;
35 }
36
37 KeyProvider KeyProvider::operator++(int) noexcept {
38     KeyProvider old = *this;
39     ++(*this);
40     return old;
41 }
42
43 bool KeyProvider::operator==(const KeyProvider& aOther) const noexcept {
44     return fIndex == aOther.fIndex && fKeys == aOther.fKeys;
45 }
46
47 bool KeyProvider::operator!=(const KeyProvider& aOther) const noexcept {
48     return !(*this == aOther);
49 }
50
51 KeyProvider KeyProvider::begin() const noexcept {
```

```
52     KeyProvider temp = *this;
53     temp.fIndex = 0;
54     return temp;
55 }
56
57 KeyProvider KeyProvider::end() const noexcept {
58     KeyProvider temp = *this;
59     temp.fIndex = fKeys.size();
60     return temp;
61 }
62
```

```
1 #include <cctype>
2 #include "VigenereForwardIterator.h"
3
4 VigenereForwardIterator::VigenereForwardIterator(const std::string& aKeyword, const std::string& aSource, EVigenereMode aMode) noexcept :
5     fMode(aMode),
6     fKeys(aKeyword, aSource),
7     fSource(aSource),
8     fIndex(-1),
9     fCurrentChar('\0')
10 {
11     initializeTable();
12 }
13
14 void VigenereForwardIterator::encodeCurrentChar() noexcept {
15     char sourceChar = fSource[fIndex];
16     if (std::isalpha(sourceChar))
17     {
18         char keywordChar = std::toupper(*fKeys);
19         size_t row = keywordChar - 'A';
20         size_t col = std::toupper(sourceChar) - 'A';
21         char encodedChar = fMappingTable[row][col];
22         if (std::islower(sourceChar)) {
23             fCurrentChar = std::tolower(encodedChar);
24         }
25         else {
26             fCurrentChar = encodedChar;
27         }
28         fKeys++;
29     }
30     else {
31         fCurrentChar = sourceChar;
32     }
33 }
34
35 void VigenereForwardIterator::decodeCurrentChar() noexcept {
36     char sourceChar = fSource[fIndex];
37     if (std::isalpha(sourceChar))
38     {
39         char keywordChar = std::toupper(*fKeys);
40         size_t row = keywordChar - 'A';
41         for (size_t i = 0; i < CHARACTERS; ++i) {
42             if (fMappingTable[row][i] == std::toupper(sourceChar)) {
43                 char decodedChar = 'A' + i;
44                 if (std::islower(sourceChar)) {
45                     fCurrentChar = std::tolower(decodedChar);
46                 }
47                 else {
48                     fCurrentChar = decodedChar;
49                 }
50                 break;
51             }
52         }
53     }
```

```
53     fKeys++;
54 }
55 else {
56     fCurrentChar = sourceChar;
57 }
58 }
59
60
61 char VigenereForwardIterator::operator*() const noexcept {
62     return fCurrentChar;
63 }
64
65 VigenereForwardIterator& VigenereForwardIterator::operator++() noexcept ↗
66 {
67     fIndex++;
68     if (fIndex < fSource.size())
69     {
70         if (fMode == EVigenereMode::Encode) {
71             encodeCurrentChar();
72         }
73         else {
74             decodeCurrentChar();
75         }
76     }
77     return *this;
78 }
79 VigenereForwardIterator VigenereForwardIterator::operator++(int) ↗
80     noexcept {
81     VigenereForwardIterator old = *this;
82     ++(*this);
83     return old;
84 }
85 bool VigenereForwardIterator::operator==(const VigenereForwardIterator& ↗
86     aOther) const noexcept {
87     return fIndex == aOther.fIndex && fSource == aOther.fSource;
88 }
89 bool VigenereForwardIterator::operator!=(const VigenereForwardIterator& ↗
90     aOther) const noexcept {
91     return !(*this == aOther);
92 }
93 VigenereForwardIterator VigenereForwardIterator::begin() const noexcept ↗
94 {
95     VigenereForwardIterator result = *this;
96     if (result.fIndex < result.fSource.size())
97     {
98         if (result.fMode == EVigenereMode::Encode) {
99             result.encodeCurrentChar();
100         }
101         else {
```

```
101         result.decodeCurrentChar();
102     }
103 }
104 return result;
105 }
106
107 VigenereForwardIterator VigenereForwardIterator::end() const noexcept {
108     VigenereForwardIterator result = *this;
109     result.fIndex = fSource.size();
110     return result;
111 }
112
```