

## 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, Design Pattern, and Iterators

**Due date:** November 17, 2023, 12:00

**Lecturer:** Dr. James Jackson

**Your name:** Tran Hoang Hai Anh **Your student ID:** 104177513

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Tues 16:30	Wed 08:30	Wed 10:30	Wed 12:30	Thu 08:00
											<input checked="" type="checkbox"/>

Marker's comments:

Problem	Marks	Obtained
1	68	
2	120	
3	56	
4	70	
Total	314	

```

1  #include "KeyProvider.h"
2
3  // Constructor
4  KeyProvider::KeyProvider(const std::string& aKeyword) : fKeyword(new char[aKeyword.length()]), fSize(aKeyword.length()), fIndex(0)
5  {
6      // Initialize the KeyProvider object with the provided keyword
7      initialize(aKeyword);
8  }
9
10 // Destructor
11 KeyProvider::~KeyProvider()
12 {
13     // Release the allocated memory for the keyword
14     delete[] fKeyword;
15 }
16
17 // Initialize method to set up the keyword
18 void KeyProvider::initialize(const std::string& aKeyword)
19 {
20     // Calculate the length of the provided keyword
21     size_t aKeywordLength = aKeyword.length();
22
23     // Release any previously allocated memory for the keyword
24     delete[] fKeyword;
25
26     // Reset index and size, and allocate memory for the keyword
27     fIndex = 0;
28     fSize = aKeywordLength;
29     fKeyword = new char[fSize];
30
31     // Convert each character in the keyword to uppercase
32     for (size_t i = 0; i < fSize; i++)
33     {
34         fKeyword[i] = toupper(aKeyword[i]);
35     }
36 }
37
38 // Overloaded dereference operator (*) to get the current character
39 char KeyProvider::operator*( ) const
40 {
41     return fKeyword[fIndex];
42 }
43
44 // Overloaded left shift operator (<<) to add a key character
45 KeyProvider& KeyProvider::operator<<(char aKeyCharacter)
46 {
47     // Convert the key character to uppercase and assign it to the current index
48     fKeyword[fIndex] = toupper(aKeyCharacter);
49
50     // Move to the next index
51     fIndex++;
52
53     // Reset the index to 0 if it exceeds the size
54     if (fIndex >= fSize)
55     {
56         fIndex = 0;
57     }
58 }

```

```

1  #include "KeyProvider.h"
2  #include "Vigenere.h"
3
4  // Initialize Vigenere encryption/decryption table
5  void Vigenere::initializeTable()
6  {
7      for (char row = 0; row < CHARACTERS; row++)
8      {
9          char lChar = 'B' + row;
10
11          for (char column = 0; column < CHARACTERS; column++)
12          {
13              if (lChar > 'Z')
14                  lChar = 'A';
15
16              fMappingTable[row][column] = lChar++;
17          }
18      }
19  }
20
21 // Constructor
22 Vigenere::Vigenere(const std::string& aKeyword) : fKeyword(aKeyword), fKeywordProvider(KeyProvider(aKeyword))
23 {
24     // Initialize the Vigenere table
25     initializeTable();
26 }
27
28 // Get the current keyword based on the Vigenere cipher
29 std::string Vigenere::getCurrentKeyword()
30 {
31     std::string theCurrentKeyword;
32
33     for (size_t i = 0; i < fKeyword.length(); i++)
34     {
35         // Append the current character from the keyword
36         theCurrentKeyword += *fKeywordProvider;
37
38         // Update the keyword for the next character
39         fKeywordProvider << *fKeywordProvider;
40     }
41
42     return theCurrentKeyword;
43 }
44
45 // Reset the Vigenere cipher to its initial state
46 void Vigenere::reset()
47 {
48     fKeywordProvider.initialize(fKeyword);
49 }
50
51 // Encode a character using the Vigenere cipher
52 char Vigenere::encode(char aCharacter)
53 {
54     char encodedChar = aCharacter;
55     bool isAlphabet = isAlpha(encodedChar);
56 }

```

```

56 bool isAlphabet = isalpha(encodedChar);
57 if (isAlphabet)
58 {
59     char upperCharacter = toupper(aCharacter);
60     char fMappingTableRow = *fKeywordProvider - 'A';
61     char fMappingTableColumn = upperCharacter - 'A';
62     // Perform the Vigenere encryption
63     encodedChar = fMappingTable[fMappingTableRow][fMappingTableColumn];
64     // Update the keyword for the next character
65     fKeywordProvider << aCharacter;
66     // Convert to lowercase if the original character was lowercase
67     bool isLowered = islower(aCharacter);
68     if (isLowered)
69     {
70         encodedChar = tolower(encodedChar);
71     }
72 }
73 return encodedChar;
74 }
75 // Decode a character using the Vigenere cipher
76 char Vigenere::decode(char aCharacter)
77 {
78     char decodedChar = aCharacter;
79     bool isAlphabet = isalpha(decodedChar);
80     if (isAlphabet)
81     {
82         char upperCharacter = toupper(aCharacter);
83         char fMappingTableRow = *fKeywordProvider - 'A';
84         // Find the corresponding column in the Vigenere table for decryption
85         for (char i = 0; i < 26; ++i)
86         {
87             if (fMappingTable[fMappingTableRow][i] == upperCharacter)
88             {
89                 char fMappingTableColumn = i + 'A';
90                 decodedChar = fMappingTableColumn;
91                 // Update the keyword for the next character
92                 fKeywordProvider << fMappingTableColumn;
93                 break;
94             }
95         }
96         // Convert to lowercase if the original character was lowercase
97         bool isLowered = islower(aCharacter);
98         if (isLowered)
99         {
100             decodedChar = tolower(decodedChar);
101         }
102     }
103 }

```

```

1 #include "VigenereStream.h"
2
3 // Constructor
4 iVigenereStream::iVigenereStream(Cipher aCipher, const std::string& aKeyword, const char* aFileName) : fCipherProvider(aKeyword), fCipher(aCipher)
5 {
6     // Open the file if a filename is provided
7     if (aFileName != nullptr)
8     {
9         open(aFileName);
10    }
11 }
12
13 // Destructor
14 iVigenereStream::~iVigenereStream()
15 {
16     // Close the file when the object is destroyed
17     close();
18 }
19
20 // Open a file for reading
21 void iVigenereStream::open(const char* aFileName)
22 {
23     // Close any previously opened file
24     close();
25     // Open the specified file
26     fIStream.open(aFileName);
27 }
28
29 // Close the currently opened file
30 void iVigenereStream::close()
31 {
32     // Check if the file stream is open before attempting to close it
33     bool isOpenedfIStream = fIStream.is_open();
34     if (isOpenedfIStream)
35     {
36         fIStream.close();
37     }
38 }
39
40 // Reset the file stream to the beginning
41 void iVigenereStream::reset()
42 {
43     // Seek to the start of the file
44     seekstart();
45 }
46
47 // Check if the file stream is in a good state
48 bool iVigenereStream::good() const
49 {
50     return fIStream.good();
51 }
52
53 // Check if the file stream is open
54 bool iVigenereStream::is_open() const
55 {
56     return fIStream.is_open();
57 }

```

```

25 // Open the specified file
26 fstream.open(fileName);
27
28 // Close the currently opened file
29 void ivigenerestream::close()
30 {
31     // Check if the file stream is open before attempting to close it
32     bool isOpenedFStream = fstream.is_open();
33     if (isOpenedFStream)
34     {
35         fstream.close();
36     }
37 }
38
39 // Reset the file stream to the beginning
40 void ivigenerestream::reset()
41 {
42     // Seek to the start of the file
43     seekstart();
44 }
45
46 // Check if the file stream is in a good state
47 bool ivigenerestream::good() const
48 {
49     return fstream.good();
50 }
51
52 // Check if the file stream is open
53 bool ivigenerestream::is_open() const
54 {
55     return fstream.is_open();
56 }
57
58 // Check if the end of the file has been reached
59 bool ivigenerestream::eof() const
60 {
61     return fstream.eof();
62 }
63
64 // Read and process a character from the file using the Vigenere cipher
65 ivigenerestream& ivigenerestream::operator>>(char& aCharacter)
66 {
67     // Read a character from the file
68     char getChar = fstream.get();
69
70     // Process the character using the Vigenere cipher
71     char processedChar = fCipher(fCipherProvider, getChar);
72
73     // Assign the processed character to the output parameter
74     aCharacter = processedChar;
75
76     // Return the updated ivigenerestream object
77     return *this;
78 }

```

```

1 #include "Vigenerereforwarditerator.h"
2
3 // Constructor
4 Vigenerereforwarditerator::Vigenerereforwarditerator(ivigenerestream& aIStream) : fIStream(aIStream), fCurrentChar(0), fEOF(false)
5 {
6     // Check if the input stream is in a valid state
7     bool validInput = fIStream;
8     if (validInput)
9     {
10         // Read the first character from the input stream
11         fIStream >> fCurrentChar;
12
13         // Check if the end of the file is reached
14         bool isEndOfFile = fIStream.eof();
15         if (isEndOfFile)
16         {
17             fEOF = true;
18         }
19     }
20 }
21
22 // Dereference operator to get the current character
23 char Vigenerereforwarditerator::operator*() const
24 {
25     return fCurrentChar;
26 }
27
28 // Pre-increment operator to advance the iterator to the next character
29 Vigenerereforwarditerator& Vigenerereforwarditerator::operator++()
30 {
31     // Check if the input stream is in a valid state
32     bool validInput = fIStream;
33     if (validInput)
34     {
35         // Read the next character from the input stream
36         fIStream >> fCurrentChar;
37
38         // Check if the end of the file is reached
39         bool isEndOfFile = fIStream.eof();
40         if (isEndOfFile)
41         {
42             fEOF = true;
43         }
44     }
45     return *this;
46 }
47
48 // Post-increment operator to advance the iterator to the next character
49 Vigenerereforwarditerator Vigenerereforwarditerator::operator++(int)
50 {
51     // Create a copy of the current iterator
52     Vigenerereforwarditerator _this = *this;
53
54     // Advance the iterator to the next character
55     (_this)++;
56 }

```

```

35     // Read the next character from the input stream
36     fistream >> fCurrentChar;
37
38     // Check if the end of the file is reached
39     bool isEndOfFile = fistream.eof();
40     if (isEndOfFile)
41     {
42         fEOF = true;
43     }
44     }
45     return *this;
46 }
47
48 // Post-increment operator to advance the iterator to the next character
49 VigenerForwardIterator VigenerForwardIterator::operator++(int)
50 {
51     // Create a copy of the current iterator
52     VigenerForwardIterator _this = *this;
53
54     // Advance the iterator to the next character
55     (_this)++;
56
57     // Return the copy of the iterator before the increment
58     return _this;
59 }
60
61 // Equality operator to compare with another iterator
62 bool VigenerForwardIterator::operator==(const VigenerForwardIterator& aOther) const
63 {
64     // Compare the end-of-file status
65     bool result = (fEOF == aOther.fEOF);
66     return result;
67 }
68
69 // Inequality operator to compare with another iterator
70 bool VigenerForwardIterator::operator!=(const VigenerForwardIterator& aOther) const
71 {
72     // Invert the result of the equality comparison
73     bool result = !(*this == aOther);
74     return result;
75 }
76
77 // Get the iterator pointing to the beginning
78 VigenerForwardIterator VigenerForwardIterator::begin() const
79 {
80     return *this;
81 }
82
83 // Get the iterator pointing to the end
84 VigenerForwardIterator VigenerForwardIterator::end() const
85 {
86     // Create a copy of the current iterator and mark it as end-of-file
87     VigenerForwardIterator _this = *this;
88     _this.fEOF = true;
89     return _this;
90 }

```