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: Nov 10, 2023, 23:59
Lecturer: Dr. Van Dai PHAM

Your name: Nguyen Quoc Bao Huynh **Your student ID:** 103804535

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Thursday 10:00 (Innovation Lab)
							✓

Marker's comments:

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

KeyProvider.cpp:


```cpp
#include "KeyProvider.h"

// Initialize key provider. [10]
// aKeyword is a string of letters.
KeyProvider::KeyProvider(const std::string& aKeyword)
{
    //call the initialize function, initializes the KeyProvider object using aKeyword argument.
    initialize(aKeyword);
}

// Destructor, release resources. [4]
KeyProvider::~KeyProvider()
{}

// Initialize (or reset) keyword [30]
void KeyProvider::initialize(const std::string& aKeyword)
{
    //Assign the length of the Keyword to variable fSize.
    fSize = aKeyword.length();
    //Creates an array of character with a size of fSize and assign to fKeyword.
    fKeyword = new char[fSize];
    //Initialize the Index to 0. This variable is used to track the current keyword index.
    fIndex = 0;

    //Loop that iterates over each character in the aKeyword.
    for (size_t i = 0; i < fSize; i++)
    {
        //converts the character to uppercase.
        fKeyword[i] = toupper(aKeyword[i]);
    }
}

// Dereference, returns current keyword character. [4]
char KeyProvider::operator*() const
{
    //Returns the character in the current index in the fKeyword array.
    return fKeyword[fIndex];
}

// Push new keyword character. [18]
// aKeyCharacter is a letter (isalpha() is true).
// aKeyCharacter replaces current keyword character.
// Key provider advances to next keyword character.
KeyProvider& KeyProvider::operator<<(char aKeyCharacter)
{
    //check if aKeyCharacter is an alphabet char.
```

```
47      if (isalpha(aKeyCharacter))
48      {
49          //Converts aKeyCharacter to uppercase.
50          fKeyword[fIndex] = toupper(aKeyCharacter);
51          //Updates the fIndex value to the next index.
52          //The modulo wraps the array to the first index if the index
              reaches the fSize value.
53          fIndex = (fIndex + 1) % fSize;
54      }
55      //Returns a reference to the current KeyProvider object.
56      return *this;
57  }
```

Vigenere.cpp:

...\data struct\Visual Studio lab\Test\Test\Vigenere.cpp 1

```cpp
#include "Vigenere.h"

// Initialize Vigenere scrambler [8]
Vigenere::Vigenere(const std::string& aKeyword) : fKeyword(aKeyword), fKeywordProvider(aKeyword) {
    //calls the initializeTable function.
    initializeTable();
}

// Return the current keyword. [22]
// This method scans the keyword provider and copies the keyword characters
// into a result string.
std::string Vigenere::getCurrentKeyword() {
    //initialize an empty string that holds the result characters.
    std::string result;
    //create a temprary keyProvider object.
    KeyProvider tempProvider(fKeyword);
    //Loop that iterates through the characters of the keyword.
    for (size_t i = 0; i < fKeyword.size(); i++) {
        //Result = empty result + current keyword character.
        result += *tempProvider;
        //Advances the tempProvider object to the next character in the keyword.
        tempProvider << *tempProvider;
    }
    //returns the complete keyword
    return result;
}

// Reset Vigenere scrambler. [6]
// This method has to initialize the keyword provider.
void Vigenere::reset() {
    fKeywordProvider.initialize(fKeyword);
}

// Encode a character using the current keyword character and update keyword. [36]
char Vigenere::encode(char aCharacter)
{
    //Initialize encoding holder variable
    char result = aCharacter;
    //check if result is an alphabet char.
    if (isalpha(result))
    {
        //Converts aCharacter to uppercase.
        char aCharacter_upper = toupper(aCharacter);
        //retrieves the encoded character from the fMappingTable.
        //% 'A' operation is used to map the character's index to the corresponding position in the mapping table.
```

```
46          result = fMappingTable[*fKeywordProvider % 'A'][aCharacter_upper % 'A'];
47          //Advances the fKeywordProvider object to the next character in the keyword.
48          fKeywordProvider << aCharacter_upper;
49          //check and convert to lowercase.
50          if (islower(aCharacter)) result = tolower(result);
51      }
52      //returns the complete encoding.
53      return result;
54  }
55
56  // Decode a character using the current keyword character and update keyword. [46]
57  char Vigenere::decode(char aCharacter)
58  {
59      char result = aCharacter;
60      if (isalpha(result))
61      {
62          char aCharacter_upper = toupper(aCharacter);
63          //This line retrieves the decoded character from the fMappingTable.
64          //(CHARACTERS - 2) - *fKeywordProvider % 'A' map the character's index to the corresponding position in the mapping table (remove header and footer).
65          result = fMappingTable[(CHARACTERS - 2) - *fKeywordProvider % 'A'][aCharacter_upper % 'A'];
66          //Advances the fKeywordProvider object to the next character in the keyword.
67          fKeywordProvider << result;
68          //check and convert to lowercase.
69          if (islower(aCharacter)) result = tolower(result);
70      }
71      return result;
72  }
73
74  // Initialize the mapping table
75  // Row 1: B - A
76  // Row 26: A - Z
77  void Vigenere::initializeTable()
78  {
79      for (char row = 0; row < CHARACTERS; row++)
80      {
81          char lChar = 'B' + row;
82          for (char column = 0; column < CHARACTERS; column++)
83          {
84              if (lChar > 'Z')
85              {
86                  lChar = 'A';
87              }
```

```
88              fMappingTable[row][column] = lChar++;
89          }
90      }
91  }
```

IVigenereStream.cpp:

```
...tructVisual Studio labTestTestiVigenereStream.cpp                    1
 1  #include "IVigenereStream.h"
 2
 3  iVigenereStream::iVigenereStream(Cipher aCipher, const std::string&
      aKeyword, const char* aFileName)
 4      : fCipher(aCipher), fCipherProvider(aKeyword)
 5  {
 6      if (aFileName != nullptr)
 7          open(aFileName);
 8  }
 9
10  iVigenereStream::~iVigenereStream()
11  {
12      close();
13  }
14
15  void iVigenereStream::open(const char* aFileName)
16  {
17      close();
18      fIStream.open(aFileName);
19  }
20
21  void iVigenereStream::close()
22  {
23      if (fIStream.is_open())
24          fIStream.close();
25  }
26
27  void iVigenereStream::reset()
28  {
29      seekstart();
30  }
31
32  bool iVigenereStream::good() const
33  {
34      return fIStream.good();
35  }
36
37  bool iVigenereStream::is_open() const
38  {
39      return fIStream.is_open();
40  }
41
42  bool iVigenereStream::eof() const
43  {
44      return fIStream.eof();
45  }
46
47  iVigenereStream& iVigenereStream::operator>>(char& aCharacter)
48  {
```

```
49      if (fIStream)
50      {
51          char ch;
52          fIStream.get(ch);
53          if (fIStream)
54          {
55              auto lCallable = [](Vigenere& aCipherProvider, char aCharacter)
56              {
57                  return aCipherProvider.decode(aCharacter);
58              };
59              aCharacter = fCipher(fCipherProvider, ch);
60          }
61      }
62      return *this;
63  }
```

VigenereForwardIterator.cpp:


```cpp
#include "VigenereForwardIterator.h"

VigenereForwardIterator::VigenereForwardIterator(iVigenereStream& aIStream)
    : fIStream(aIStream), fCurrentChar('\0'), fEOF(false)
{
    if (fIStream)
    {
        fIStream >> fCurrentChar;
        if (fIStream.eof())
        {
            fEOF = true;
        }
    }
}

char VigenereForwardIterator::operator*() const
{
    return fCurrentChar;
}

VigenereForwardIterator& VigenereForwardIterator::operator++()
{
    if (fIStream)
    {
        fIStream >> fCurrentChar;
        if (fIStream.eof())
        {
            fEOF = true;
        }
    }
    return *this;
}

VigenereForwardIterator VigenereForwardIterator::operator++(int)
{
    VigenereForwardIterator temp = *this;
    ++(*this);
    return temp;
}

bool VigenereForwardIterator::operator==(const VigenereForwardIterator&
  aOther) const
{
    return fEOF == aOther.fEOF;
}

bool VigenereForwardIterator::operator!=(const VigenereForwardIterator&
  aOther) const
{
```

```
48      return !(*this == aOther);
49  }
50
51  VigenereForwardIterator VigenereForwardIterator::begin() const
52  {
53      return *this;
54  }
55
56  VigenereForwardIterator VigenereForwardIterator::end() const
57  {
58      VigenereForwardIterator temp = *this;
59      temp.fEOF = true;
60      return temp;
61  }
```