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: April 27, 2022, 23:59
Lecturer: Dr. Markus Lumpe

Your name: Nguyen Thi Thanh Minh **Your student ID:** 104169617

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	Wed 14:30
			X	X							

Marker's comments:

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

```cpp

// COS30008, Midterm, Problem 1, 2024

#include "KeyProvider.h"

#include <cctype>

KeyProvider::KeyProvider(const std::string& aKeyword) :
    // Member initializers
    fKeyword(nullptr),
    fSize(0),
    fIndex(0)
{
    initialize(aKeyword);
}

KeyProvider::~KeyProvider()
{
    // Release resources
    delete[] fKeyword;
}

void KeyProvider::initialize(const std::string& aKeyword)
{
    // Delete existing keyword
    if (fKeyword != nullptr)
    {
        delete[] fKeyword;
    }

    // Initialize or reset keyword
    fSize = aKeyword.length();
    fKeyword = new char[fSize];
    fIndex = 0;

    for (size_t i = 0; i < fSize; i++)
    {
        // Use operator<< to push new keyword character
        *this << aKeyword[i];
    }
}

char KeyProvider::operator*() const
{
    // Return current keyword character
    return fKeyword[fIndex];
}

KeyProvider& KeyProvider::operator<<(char aKeyCharacter)
```

```
50  {
51      // Push new keyword character if aKeyCharacter is a letter
52      if (isalpha(aKeyCharacter))
53      {
54          // Replace current keyword character
55          fKeyword[fIndex] = toupper(aKeyCharacter);
56          // Advance to next keyword character (circular)
57          fIndex = (fIndex + 1) % fSize;
58      }
59
60      return *this;
61  }
```

```cpp

// COS30008, Midterm, Problem 2, 2024

#include "Vigenere.h"

#include <cctype>

using namespace std;

Vigenere::Vigenere(const string& aKeyword) :
    // Member initializers
    fKeyword(aKeyword),
    fKeywordProvider(aKeyword)
{
    initializeTable();
}

string Vigenere::getCurrentKeyword()
{
    string lResult;

    // Go through the keyword provider
    // and copy keyword characters into result string
    for (size_t i = 0; i < fKeyword.length(); i++)
    {
        char lChar = *fKeywordProvider;
        lResult += lChar;
        // Advance to next keyword character
        // while keeping the keyword provider unchanged
        fKeywordProvider << lChar;
    }

    return lResult;
}

void Vigenere::reset()
{
    fKeywordProvider.initialize(fKeyword);
}

char Vigenere::encode(char aCharacter)
{
    char lResult = aCharacter;

    // Only encode letters
    if (isalpha(aCharacter))
    {
        // Get encoded character from mapping table
        // Please ignore the warning about reading invalid data
```

```
50          // because we know that fKeywordProvider only contains letters
51          lResult = fMappingTable[*fKeywordProvider - 'A'][toupper
              (aCharacter) - 'A'];
52          // Advance to next keyword character
53          // and update keyword provider
54          fKeywordProvider << aCharacter;
55
56          // Keep the case of the original character
57          if (islower(aCharacter))
58          {
59              lResult = tolower(lResult);
60          }
61      }
62
63      return lResult;
64  }
65
66  char Vigenere::decode(char aCharacter)
67  {
68      char lResult = aCharacter;
69
70      if (isalpha(aCharacter))
71      {
72          char lRow = *fKeywordProvider - 'A';
73
74          // Find the column in the mapping table
75          for (size_t i = 0; i < CHARACTERS; i++)
76          {
77              if (fMappingTable[lRow][i] == toupper(aCharacter))
78              {
79                  lResult = 'A' + i;
80                  break;
81              }
82          }
83
84          // Advance to next keyword character
85          fKeywordProvider << lResult;
86
87          // Keep the case of the original character
88          if (islower(aCharacter))
89          {
90              lResult = tolower(lResult);
91          }
92      }
93
94      return lResult;
95  }
```

```cpp

// COS30008, Midterm, Problem 3, 2024

#include "iVigenereStream.h"

using namespace std;

iVigenereStream::iVigenereStream(Cipher aCipher, const string& aKeyword,
  const char* aFileName) :
    fCipher(aCipher),
    fCipherProvider(aKeyword)
{
    open(aFileName);
}

iVigenereStream::~iVigenereStream()
{
    // Close the file stream
    close();
}

void iVigenereStream::open(const char* aFileName)
{
    // Open file in binary mode
    fIStream.open(aFileName, ios_base::binary);
}

void iVigenereStream::close()
{
    fIStream.close();
}

void iVigenereStream::reset()
{
    // Reset the cipher provider
    fCipherProvider.reset();
    // Reset the file stream
    seekstart();
}

bool iVigenereStream::good() const
{
    return fIStream.good();
}

bool iVigenereStream::is_open() const
{
    return fIStream.is_open();
}
```

```
49
50  bool iVigenereStream::eof() const
51  {
52      return fIStream.eof();
53  }
54
55  iVigenereStream& iVigenereStream::operator>>(char& aCharacter)
56  {
57      if (fIStream.good())
58      {
59          // Read a character using get() method of ifstream
60          // so as not to skip whitespace characters
61          fIStream.get(aCharacter);
62
63          if (fIStream.good())
64          {
65              // Call the cipher function to encode/decode the character
66              aCharacter = fCipher(fCipherProvider, aCharacter);
67          }
68      }
69
70      return *this;
71  }
```

```cpp

// COS30008, Midterm, Problem 4, 2024

#include "VigenereForwardIterator.h"

VigenereForwardIterator::VigenereForwardIterator(iVigenereStream& aIStream) :
    fIStream(aIStream),
    fCurrentChar(0),
    fEOF(false)
{
    // Read first character
    fIStream >> fCurrentChar;
}

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

VigenereForwardIterator& VigenereForwardIterator::operator++()
{
    // Read next character
    fIStream >> fCurrentChar;

    // Check for EOF
    if (!fIStream)
    {
        fEOF = true;
    }

    return *this;
}

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

bool VigenereForwardIterator::operator==(const VigenereForwardIterator& aOther) const
{
    // Check if both iterators point to the same stream
    // and have the same EOF status
    return (fIStream == aOther.fIStream)
        && (fEOF == aOther.fEOF);
}
```

```
48
49  bool VigenereForwardIterator::operator!=(const VigenereForwardIterator&
      aOther) const
50  {
51      return !(*this == aOther);
52  }
53
54  VigenereForwardIterator VigenereForwardIterator::begin() const
55  {
56      VigenereForwardIterator temp = *this;
57      // Reset the stream
58      temp.fIStream.reset();
59      return temp;
60  }
61
62  VigenereForwardIterator VigenereForwardIterator::end() const
63  {
64      VigenereForwardIterator temp = *this;
65      // Set EOF status to true
66      temp.fEOF = true;
67      return temp;
68  }
```