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: Tran Quoc Dung **Your student ID:** 103803891

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

Marker's comments:

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

Mid Term 2022

Problem 1:

File: KeyProvider.h

```
#pragma once
#include <string>

using namespace std;

class KeyProvider
{
private:
   char* fKeyword;    // keyword
   size_t fSize;       // length of keyword
   size_t fIndex;      // index to current keyword character

public:

   // Initialize key provider. [10]
   // aKeyword is a string of letters.
   KeyProvider(const std::string& aKeyword);

   // Destructor, release resources. [4]
   ~KeyProvider();

   // Initialize (or reset) keyword [30]
   void initialize(const string& aKeyword);

   // Dereference, returns current keyword character. [4]
   char operator*() const;

   // 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& operator<<(char aKeyCharacter);
};
```

File: KeyProvider.cpp

```
#pragma once
#include "KeyProvider.h"
#include <string>

using namespace std;

KeyProvider::KeyProvider(const string& aKeyword) : //components of KeyProvider class
        fSize(aKeyword.length()),
        fIndex(0),
        fKeyword(new char[aKeyword.length()])
{
        initialize(aKeyword);
}
```

```
KeyProvider::~KeyProvider()
{
	delete[] fKeyword; //release memory
}

void KeyProvider::initialize(const string& aKeyword)
{
	fSize = aKeyword.length();
	fKeyword = new char[fSize];
	for (size_t i = 0; i < fSize; i++)
	{
		fKeyword[i] = static_cast<char>(toupper(aKeyword[i])); //similar to constructor (converting
string to char)
	}
	fIndex = 0;
}

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

KeyProvider& KeyProvider::operator<<(char aKeyCharacter)
{
	if (isalpha(aKeyCharacter)) //checking alphabet letters
	{
		fKeyword[fIndex] = static_cast<char>(toupper(aKeyCharacter)); //replacing with
AKEYCHARACTER
	}

	fIndex++; //KeyProvider advances to next keyword character

	if (fIndex >= fSize)
	{
		fIndex = 0; //checking to reset keyword index
	}

	return *this;
}
```

Problem 2:

File: Vigenere.h

```
#pragma once
#include "KeyProvider.h"
#define CHARACTERS 26
class Vigenere
{
private:
	char fMappingTable[CHARACTERS][CHARACTERS]; // 26x26 board of alphabets
	const string fKeyword;
	KeyProvider fKeywordProvider;
	// Initialize the mapping table
```

```
	// Row 1: B - A
	// Row 26: A - Z
	void initializeTable();

public:

	// Initialize Vigenere scrambler [8]
	Vigenere(const string& aKeyword);

	// Return the current keyword. [22]
	// This method scans the keyword provider and copies the keyword characters
	// into a result string.
	string getCurrentKeyword();

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

	// Encode a character using the current keyword character and update keyword. [36]
	char encode(char aCharacter);

	// Decode a character using the current keyword character and update keyword. [46]
	char decode(char aCharacter);
};
```

File: Vigenere.cpp

```
#pragma once
#include "Vigenere.h"

using namespace std;

void Vigenere::initializeTable()
{
	for (char row = 0; row < CHARACTERS; row++)
	{
		char lChar = 'B' + row;

		for (char column = 0; column < CHARACTERS; column++)
		{
			if (lChar > 'Z')
				lChar = 'A';

			fMappingTable[row][column] = lChar++;
		}
	}
}

Vigenere::Vigenere(const string& aKeyword) : fKeyword(aKeyword),
fKeywordProvider(KeyProvider(aKeyword))
{
	initializeTable();
}

string Vigenere::getCurrentKeyword()
{
```

```
	string result;

	for (size_t i = 0; i < fKeyword.length(); i++) {
		result += *fKeywordProvider;
		fKeywordProvider << *fKeywordProvider;
		//copying keyword characters into result string
	}

	return result;
}

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

char Vigenere::encode(char aCharacter)
{
	if (isalpha(aCharacter))
	{
		bool isLower = islower(aCharacter); //checking aCharacter uppercase or lowercase

		char encoded = fMappingTable [*fKeywordProvider - 'A'] [toupper(aCharacter) - 'A'];

		fKeywordProvider << aCharacter;

		return isLower ? static_cast<char>(tolower(encoded)) : (encoded); //updating current
keyword character as part of the autokey cipher process
	}
	else
	{
		return aCharacter; //non-alphabetical character
	}
}

char Vigenere::decode(char aCharacter)
{
	if (isalpha(aCharacter))
	{
		bool isLower = islower(aCharacter); //checking aCharacter uppercase or lowercase

		char encoded = static_cast<char>(toupper(aCharacter));

		char decoded = 0;

		for (char column = 0; column < CHARACTERS; column++)
		{
			if (fMappingTable [*fKeywordProvider - 'A'] [column] == encoded)
			{
				decoded = static_cast<char>(column + 'A');
				break;
			}
		}
		fKeywordProvider << decoded;

		return isLower ? static_cast<char>(tolower(decoded)) : (decoded); //updating current
keyword character as part of the autokey cipher process
```

```
        }
        else
        {
                return aCharacter; //non-alphabetical character
        }
}
```

Problem 3:

File: IVigenereStream.h

```
#pragma once
#include <fstream>
#include <functional>

#include "Vigenere.h"

using Cipher = function<char(Vigenere& aCipherProvider, char aCharacter)>;

class iVigenereStream
{
private:
        ifstream fIStream;
        Vigenere fCipherProvider;
        Cipher fCipher;

public:

        iVigenereStream(Cipher aCipher, const string& aKeyword, const char* aFileName = nullptr); // [8]

        ~iVigenereStream(); // [2]

        void open(const char* aFileName); // [8]

        void close(); // [2]

        void reset(); // [10]

        // conversion operator to bool
        operator bool() { return !eof(); }

        // stream positioning
        uint64_t position() { return fIStream.tellg(); }

        void seekstart() { fIStream.clear(); fIStream.seekg(0, ios_base::beg); }

        bool good() const; // [3]

        bool is_open() const; // [3]

        bool eof() const; // [3]

        iVigenereStream& operator>>(char& aCharacter); // [17]
};
```

File: IVigenereStream.cpp

```
#pragma once
#include "IVigenereStream.h"

using namespace std;

iVigenereStream::iVigenereStream(Cipher aCipher, const string& aKeyword, const char* aFileName) :
	fIStream(ifstream()),
	fCipherProvider(Vigenere(aKeyword)),
	fCipher(move(aCipher))
{
	if (aFileName != nullptr)
		open(aFileName);

}

iVigenereStream::~iVigenereStream()
{
	close(); //releasing memory
}

void iVigenereStream::open(const char* aFileName)
{
	fIStream.open(aFileName, ios::binary); //opening file
}

void iVigenereStream::close()
{
	fIStream.close(); //closing file
}

void iVigenereStream::reset() //restart an iVigenereStream stream
{
	fCipherProvider.reset();
	seekstart();
}

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

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

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

iVigenereStream& iVigenereStream::operator>>(char& aCharacter)
{
	aCharacter = fCipher(fCipherProvider, static_cast<char>(fIStream.get()));
	return *this; }
```

Problem 4:

File: VigenereForwardIterator.h

```
#pragma once
#include "IVigenereStream.h"
class VigenereForwardIterator
{
private:
	iVigenereStream& fIStream;
	char fCurrentChar;
	bool fEOF;

public:
	VigenereForwardIterator(iVigenereStream& aIStream); // [10]

	// forward iterator interface
	char operator*() const; // [2]

	VigenereForwardIterator& operator++(); // prefix increment [10]

	VigenereForwardIterator operator++(int); // postfix increment [10]

	bool operator==(const VigenereForwardIterator& aOther) const; // [8]

	bool operator!=(const VigenereForwardIterator& aOther) const; // [4]

	VigenereForwardIterator begin() const; // [16]

	VigenereForwardIterator end() const; // [10]
};
```

File: VigenereForwardIterator.cpp

```
#pragma once
#include "VigenereForwardIterator.h"

using namespace std;

VigenereForwardIterator::VigenereForwardIterator(iVigenereStream& aIStream) :
   fIStream(aIStream),
   fCurrentChar(0),
   fEOF(aIStream.eof())
{
   if (!fEOF)
      fIStream >> fCurrentChar;   //checking whether the underlying stream has reached end-of-file
}

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

VigenereForwardIterator& VigenereForwardIterator::operator++()
{
```

```
   fIStream >> fCurrentChar;
   fEOF = fIStream.eof();
   return *this;
}

VigenereForwardIterator VigenereForwardIterator::operator++(int)
{
   VigenereForwardIterator tmp = *this;

   ++(*this);  //advancing the iterator

   return tmp;
}

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

bool VigenereForwardIterator::operator!=(const VigenereForwardIterator& aOther) const
{
   return !(*this == aOther);
}

VigenereForwardIterator VigenereForwardIterator::begin() const
{
   VigenereForwardIterator lResult = *this;
   lResult.fIStream.reset();
   lResult.fEOF = lResult.fIStream.eof();
   if (!lResult.fEOF)
      lResult.fIStream >> lResult.fCurrentChar;
   return lResult;
}

VigenereForwardIterator VigenereForwardIterator::end() const
{
   VigenereForwardIterator lResult = *this;
   lResult.fEOF = true;

   return lResult;
}
```

File: Main.cpp

```
#include <iostream>
#include <stdexcept>
#include <fstream>
#include <cctype>

using namespace std;

#define P1
#define P2
#define P3
#define P4

#ifdef P1

#include "KeyProvider.h"

int runP1(string argv[2])
{
	cout << "Testing KeyProvider with \"" << argv[0]
		<< "\" and \"" << argv[1] << "\"" << endl;
	KeyProvider lKeyWord("Relations");
	string& lMessage = argv[1];
	for (char c : lMessage)
	{
		if (isalpha(c))
		{
			cout << *lKeyWord;
			lKeyWord << c;
		}
		else
		{
			cout << " ";
		}
	}
	cout << "\n";
	for (char c : lMessage)
	{
		cout << (isalpha(c) ? static_cast<char>(toupper(c)) : c);
	}
	cout << "\nCompleted" << endl;
	return 0;
}

#endif

#ifdef P2

#include "Vigenere.h"

int runP2(string argv[2])
{
   string lMessage = argv[1];
   Vigenere lSrambler(argv[0]);
```

```
    // Test encoding

    cout << "Encoding \"" << lMessage
        << "\" using \"" << lSrambler.getCurrentKeyword() << "\"" << endl;

    for (char c : lMessage)
    {
        cout << (isalpha(c) ? static_cast<char>(toupper(c)) : c);
    }

    cout << "\n";

    string lEncodedMessage;

    for (char c : lMessage)
    {
        lEncodedMessage += lSrambler.encode(c);
    }

    cout << lEncodedMessage << "\nCompleted" << endl;

    // Test decoding
    lSrambler.reset();

    cout << "Decoding \"" << lEncodedMessage
        << "\" using \"" << lSrambler.getCurrentKeyword() << "\"" << endl;

    for (char c : lEncodedMessage)
    {
        cout << (isalpha(c) ? static_cast<char>(toupper(c)) : c);
    }

    cout << "\n";

    string lDecodedMessage;

    for (char c : lEncodedMessage)
    {
        lDecodedMessage += lSrambler.decode(c);
    }

    cout << lDecodedMessage << "\nCompleted" << endl;

    return 0;
}

#endif

#ifdef P3

#include "iVigenereStream.h"

int runP3(string argv[2])
{
    iVigenereStream lInput([](Vigenere& aCipherProvider, char aCharacter)
        {
            return aCipherProvider.decode(aCharacter);
```

```
        }, argv[0], argv[1].c_str());

    if (!lInput.good())
    {
        cerr << "Cannot open input file: " << argv[1] << endl;

        return 2;
    }

    cout << "Decoding \"" << argv[1] << "\" using \"" << argv[0] << "\"." << endl;

    char lCharacter;

    while (lInput >> lCharacter)
    {
        cout << lCharacter;
    }

    lInput.close();

    cout << "Completed." << endl;

    return 0;
}

#endif

#ifdef P4

#include "VigenereForwardIterator.h"

int runP4(string argv[2])
{
    iVigenereStream lInput([](Vigenere& aCipherProvider, char aCharacter)
        {
            return aCipherProvider.encode(aCharacter);
        }, argv[0], argv[1].c_str());

    if (!lInput.good())
    {
        cerr << "Cannot open input file: " << argv[1] << endl;

        return 2;
    }

    cout << "Forward Iterator Encoding \"" << argv[1] << "' using \"" << argv[0] << "\"." << endl;

    for (char c : VigenereForwardIterator(lInput))
    {
        cout << c;
    }

    lInput.close();

    cout << "Completed." << endl;

    return 0;
```

```
}

#endif

int main()
{
    #ifdef P1
        string message1[] = { "Relations", "To be, or not to be: that is the question:" };

        return runP1(message1);

    #endif

    #ifdef P2
        string message2[] = { "Relations", "To be, or not to be: that is the question:" };

        return runP2(message2);
    #endif

    #ifdef P3
        string message3[] = { "Relations", "sample_3.txt" };

        return runP3(message3);
    #endif

    #ifdef P4
        string message4[] = { "Relations", "sample_4.txt" };

        return runP4(message4);
    #endif

    return 0;
}
```