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: Dao Khanh Nga Thi **Your student ID:** 104177393

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							

Marker's comments:

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

Problem 1: KeyProvider.cpp

```
#include "KeyProvider.h"


// Constructor
KeyProvider::KeyProvider(const std::string& keyword) : fSize(keyword.length()),
fIndex(0)
{
    fKeyword = new char[fSize];
    initialize(keyword);
}


// Destructor
KeyProvider::~KeyProvider()
{
    // Clean up the dynamically allocated memory for the keyword
    delete[] fKeyword;
}

void KeyProvider::initialize(const std::string& keyword)
{
    delete[] fKeyword;
    fSize = keyword.length();
    fKeyword = new char[fSize];
    size_t i = 0;

    while (i < fSize)
    {
       // Convert each character in the keyword to uppercase
       fKeyword[i] = std::toupper(keyword[i]);
       ++i;
    }

    fIndex = 0;
}

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


// Assign a new character to the keyword
KeyProvider& KeyProvider::operator<<(char keyCharacter)
{
    fKeyword[fIndex] = std::toupper(keyCharacter);

    if (fSize - 1 == fIndex)
    {
        fIndex = 0;
    }
    else
    {
        ++fIndex;
    }
```

```
    return *this;
}
```

Problem 2: VigenereMT.cpp

```
#include "Vigenere.h"

void Vigenere::initializeTable()
{
	char row = 0;

	// Fill the Vigenere mapping table with characters in a circular pattern
	while (row < CHARACTERS)
	{
		char currentChar = 'B' + row;
		char col = 0;

		while (col < CHARACTERS)
		{
			// Wrap around to 'A' if the character exceeds 'Z'
			if (currentChar > 'Z')
				currentChar = 'A';
			fMappingTable[row][col] = currentChar++;
			++col;
		}
		++row;
	}
}

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

// Get the current keyword by extracting characters from the keyword provider
std::string Vigenere::getCurrentKeyword()
{
	std::string current_keyword;
	size_t i = 0;

	while ( i < fKeyword.length() )
	{
		current_keyword += *fKeywordProvider;
		fKeywordProvider << *fKeywordProvider;
		++i;
	}

	return current_keyword;
}

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

```
}

char Vigenere::encode(char aCharacter)
{
	if (!isalpha(aCharacter))
	{
		return aCharacter;
	}

	bool isLowerCase = std::islower(aCharacter);
	char encoded = fMappingTable[*fKeywordProvider -
'A'][std::toupper(aCharacter) - 'A'];

	// Update the keyword provider with the current character
	fKeywordProvider << aCharacter;

	if (isLowerCase)
	{
		return static_cast<char>(std::tolower(encoded));
	}
	return encoded;

}

char Vigenere::decode(char aCharacter)
{
	if (!isalpha((aCharacter)))
	{
		return aCharacter;
	}

	bool isLowerCase = std::islower(aCharacter);
	char encoded = static_cast<char>(toupper(aCharacter));
	char decoded = 0;
	char col = 0;

	while (col < CHARACTERS)
	{
		// Search for the matching decoded character in the mapping table
		if (fMappingTable[*fKeywordProvider - 'A'][col] == encoded)
		{
			decoded = static_cast<char>(col + 'A');
			break;
		}
		++col;
		}

		fKeywordProvider << decoded;

		// Convert the decoded character back to lowercase if necessary
		if (isLowerCase)
		{
			return static_cast<char>(std::tolower(decoded));
		}
		return decoded;
}
```

Problem 3: iVigenereStream.cpp

```
#include "iVigenereStream.h"

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

// Destructor
iVigenereStream::~iVigenereStream()
{
	close();
}

// Open a file for reading
void iVigenereStream::open(const char* aFileName)
{
	fIStream.open(aFileName, std::ios::binary);
}

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

// Reset the stream and cipher provider to the initial state
void iVigenereStream::reset()
{

	fCipherProvider.reset();
	seekstart();
}

// Check if the stream is in good condition
bool iVigenereStream::good() const
{
	return fIStream.good();
}

// Check if the stream is open
bool iVigenereStream::is_open() const
{
	return fIStream.is_open();
}

// Check if end-of-file has been reached
bool iVigenereStream::eof() const
{
	return fIStream.eof();
```

```
}

// Read characters from the file and apply the Vigenere cipher
iVigenereStream& iVigenereStream::operator>>(char& aCharacter)
{
	char streamCharacter = static_cast<char>(fIStream.get());

	// if read was successful, apply the Vigenere cipher on the read character
	if (fIStream) {
		aCharacter = fCipher(fCipherProvider, streamCharacter);
	}
	else {
		aCharacter = '\n';
	}

	return *this;
}
```

Problem 4: VigenereForwardIterator.cpp

```
#include "VigenereForwardIterator.h"

// Constructor
VigenereForwardIterator::VigenereForwardIterator(iVigenereStream& aIStream) :
fIStream(aIStream), fCurrentChar(0), fEOF(aIStream.eof())
{

    // Iterate until the end of file is reached or a character is successfully read
    while (!fEOF) {
        fIStream >> fCurrentChar;
        fEOF = fIStream.eof();

        // Exit the loop if a character is successfully read
        if (!fEOF) {
            break;
        }
    }
}

// Dereference operator: Return the current character
char VigenereForwardIterator::operator*() const
{
    return fCurrentChar;
}

// Pre-increment operator: Increment the iterator to the next character
VigenereForwardIterator& VigenereForwardIterator::operator++()
{
    fIStream >> fCurrentChar;
    fEOF = fIStream.eof();
    return *this;
}

// Post-increment operator: Increment the iterator and return the previous state
VigenereForwardIterator VigenereForwardIterator::operator++(int)
{
    VigenereForwardIterator temp = *this;
```

```
    while (!fEOF) {
        ++(*this);

        if (!fEOF) {
            break;
        }
    }

    return temp;
}

// Equality operator: Check if two iterators are equal
bool VigenereForwardIterator::operator==(const VigenereForwardIterator& aOther)
const
{
    return (&fIStream == &aOther.fIStream) && (fEOF == aOther.fEOF);
}

// Inequality operator: Check if two iterators are not equal
bool VigenereForwardIterator::operator!=(const VigenereForwardIterator& aOther)
const
{
    return !(*this == aOther);
}

// Begin function: Return an iterator pointing to the beginning of the stream
VigenereForwardIterator VigenereForwardIterator::begin() const {
    VigenereForwardIterator result = *this;
    result.fIStream.reset();
    result.fEOF = result.fIStream.eof();

    while (!result.fEOF) {
        result.fIStream >> result.fCurrentChar;
        if (!result.fEOF) {
            break; // Break out of the loop after reading the first character
        }
    }

    return result;
}

// End function: Return an iterator pointing to the end of the stream
VigenereForwardIterator VigenereForwardIterator::end() const
{
    VigenereForwardIterator lResult = *this;

    for (; !lResult.fEOF; lResult.fEOF = true)
    {
    }

    return lResult;
}
```