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 & Iterators
Due date:	April 26, 2024, 10:30
Lecturer:	Dr. Markus Lumpe

Your name: H M Asfaq Ahmed Shihab **Your student ID:** 103849248

Marker's comments:

Problem	Marks	Obtained
1	106	
2	194	
Total	300	

```cpp
//
//  KeyProvider.cpp
//  midSem
//
//  Created by H M Asfaq Ahmed Shihab on 25/4/2024.
//
#include "KeyProvider.hpp"
#include <cctype>
#include <cassert>

std::string KeyProvider::preprocessString(const std::string& aString)
 noexcept {
    std::string result;
    for (char ch : aString) {
        if (std::isalpha(ch)) {
            result += std::toupper(ch);
        }
    }
    return result;
}

KeyProvider::KeyProvider(const std::string& aKeyword, const
 std::string& aSource) noexcept {
    std::string processedKeyword = preprocessString(aKeyword);
    std::string processedSource = preprocessString(aSource);

    while (fKeys.length() < processedSource.length()) {
        fKeys += processedKeyword;
    }

    fKeys = fKeys.substr(0, processedSource.length());
    fIndex = 0;
}

char KeyProvider::operator*() const noexcept {
    return fKeys[fIndex];
}

KeyProvider& KeyProvider::operator++() noexcept {
    if (fIndex < fKeys.size()) fIndex++;
    return *this;
}

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

bool KeyProvider::operator==(const KeyProvider& aOther) const
 noexcept {
```

```
    return fIndex == aOther.fIndex && fKeys == aOther.fKeys;
}

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

KeyProvider KeyProvider::begin() const noexcept {
    KeyProvider temp(*this);
    temp.fIndex = 0;
    return temp;
}

KeyProvider KeyProvider::end() const noexcept {
    KeyProvider temp(*this);
    temp.fIndex = fKeys.size();
    return temp;
}
```

```cpp
//
//  VigenereForwardIterator.cpp
//  midSem
//
//  Created by H M Asfaq Ahmed Shihab on 25/4/2024.
//
#include "VigenereForwardIterator.hpp"
//Constractor
VigenereForwardIterator::VigenereForwardIterator(const std::string&
 keyword, const std::string& source, EVigenereMode mode) noexcept
    : fKeys(keyword, source), fSource(source), fMode(mode), fIndex(0)
     {
    initializeTable();
    if (!fSource.empty()) {
        if (fMode == EVigenereMode::Encode) {
            encodeCurrentChar();
        } else {
            decodeCurrentChar();
        }
    }
}
// Encode the current character based on the Vigenere cipher
void VigenereForwardIterator::encodeCurrentChar() noexcept {
    if (std::isalpha(fSource[fIndex])) {
        int keyCharIndex = std::toupper(static_cast<unsigned
         char>(fKeys.operator*())) - 'A';
        int sourceCharIndex = std::toupper(static_cast<unsigned
         char>(fSource[fIndex])) - 'A';
        fCurrentChar = fMappingTable[keyCharIndex][sourceCharIndex];
        fCurrentChar = std::isupper(fSource[fIndex]) ? fCurrentChar :
         std::tolower(fCurrentChar);
        ++fKeys;
    } else {
        fCurrentChar = fSource[fIndex];
    }
}

void VigenereForwardIterator::decodeCurrentChar() noexcept {
    if (std::isalpha(fSource[fIndex])) {
        char keyChar = std::toupper(static_cast<unsigned
         char>(fKeys.operator*()));
        char encodedChar = std::toupper(static_cast<unsigned
         char>(fSource[fIndex]));
        int keyCharIndex = keyChar - 'A';

        int decodedCharIndex = 0;
        for (decodedCharIndex = 0; decodedCharIndex < CHARACTERS;
         ++decodedCharIndex) {
            if (fMappingTable[keyCharIndex][decodedCharIndex] ==
             encodedChar) {
                break;
```

```
            }
        }

        for (decodedCharIndex = 0; decodedCharIndex < CHARACTERS;
         ++decodedCharIndex) {
            if (fMappingTable[keyCharIndex][decodedCharIndex] ==
             encodedChar) {
                break;
            }
        }
        char decodedChar = 'A' + decodedCharIndex;
        if (!std::isupper(fSource[fIndex])) {
            decodedChar = std::tolower(decodedChar);
        }
        fCurrentChar = decodedChar;
        ++fKeys;

    } else {
        fCurrentChar = fSource[fIndex];
    }
}

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

VigenereForwardIterator& VigenereForwardIterator::operator++()
 noexcept {
    ++fIndex;
    if (fIndex < fSource.length()) {
        if (fMode == EVigenereMode::Encode) {
            encodeCurrentChar();
        } else {
            decodeCurrentChar();
        }
    }
    return *this;
}

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

bool VigenereForwardIterator::operator==(const
 VigenereForwardIterator& other) const noexcept {
    return fIndex == other.fIndex && fSource == other.fSource;
}
```

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

VigenereForwardIterator VigenereForwardIterator::begin() const
 noexcept {
    VigenereForwardIterator it(*this);
    it.fIndex = 0;
    return it;
}

// Get the iterator pointing to the end of the source string
VigenereForwardIterator VigenereForwardIterator::end() const noexcept
 {
    VigenereForwardIterator it(*this);
    it.fIndex = fSource.length();
    return it;
}
```