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: Ruffin Remad **Your student ID:** 103840173

Marker's comments:

Problem	Marks	Obtained
1	106	
2	194	
Total	300	

```cpp
#include "KeyProvider.h"
#include <cctype>
#include <cassert>

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

KeyProvider::KeyProvider(const std::string& aKeyword, const std::string& aSource) noexcept {
    std::string lKeyword = preprocessString(aKeyword);
    std::string lSource = preprocessString(aSource);
    for (size_t i = 0; i < lSource.length(); i++)
    {
        fKeys += lKeyword[i % lKeyword.length()];
    }
    fIndex = 0;
    assert(fKeys.length() == lSource.length());
}

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

KeyProvider& KeyProvider::operator++() noexcept {
    fIndex++;
    return *this;
}

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

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
#include <cctype>
#include "VigenereForwardIterator.h"

VigenereForwardIterator::VigenereForwardIterator(const std::string& aKeyword, const std::string& aSource, EVigenereMode aMode) noexcept :
    fMode(aMode),
    fKeys(aKeyword, aSource),
    fSource(aSource),
    fIndex(-1),
    fCurrentChar('\0')
{
    initializeTable();
}

void VigenereForwardIterator::encodeCurrentChar() noexcept {
    char sourceChar = fSource[fIndex];
    if (std::isalpha(sourceChar))
    {
        char keywordChar = std::toupper(*fKeys);
        size_t row = keywordChar - 'A';
        size_t col = std::toupper(sourceChar) - 'A';
        char encodedChar = fMappingTable[row][col];
        if (std::islower(sourceChar)) {
            fCurrentChar = std::tolower(encodedChar);
        }
        else {
            fCurrentChar = encodedChar;
        }
        fKeys++;
    }
    else {
        fCurrentChar = sourceChar;
    }
}

void VigenereForwardIterator::decodeCurrentChar() noexcept {
    char sourceChar = fSource[fIndex];
    if (std::isalpha(sourceChar))
    {
        char keywordChar = std::toupper(*fKeys);
        size_t row = keywordChar - 'A';
        for (size_t i = 0; i < CHARACTERS; ++i) {
            if (fMappingTable[row][i] == std::toupper(sourceChar)) {
                char decodedChar = 'A' + i;
                if (std::islower(sourceChar)) {
                    fCurrentChar = std::tolower(decodedChar);
                }
                else {
                    fCurrentChar = decodedChar;
                }
                break;
            }
        }
        fKeys++;
    }
    else {
        fCurrentChar = sourceChar;
    }
}


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

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

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

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

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

VigenereForwardIterator VigenereForwardIterator::begin() const noexcept {
    VigenereForwardIterator result = *this;
    if (result.fIndex < result.fSource.size())
    {
        if (result.fMode == EVigenereMode::Encode) {
            result.encodeCurrentChar();
        }
        else {
            result.decodeCurrentChar();
        }
    }
    return result;
}

VigenereForwardIterator VigenereForwardIterator::end() const noexcept {
    VigenereForwardIterator result = *this;
    result.fIndex = fSource.size();
    return result;
}

```