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: Nur E Siam **Your student ID:** 103842784

Marker's comments:

Problem	Marks	Obtained
1	106	
2	194	
Total	300	

```cpp
// Cos3008-Mid
//Created By NUR E SIAM


#include <stdio.h>
#include "KeyProvider.h"
#include <cctype>
#include <cassert>

KeyProvider::KeyProvider(const std::string& aKeyword, const std::string&
  aSource) noexcept
{
    fIndex = 0;
    std::string Ckeyword = preprocessString(aKeyword);
    std::string Csource = preprocessString(aSource);

    // goes through the length of preprocessed aSource, and for each
      letter, applies a preprocessed aKeyword letter
    for (size_t i = 0; i < Csource.length(); i++)
    {
        fKeys += Ckeyword[i % Ckeyword.length()];
    }

    assert(fKeys.length() == Csource.length());
}

std::string KeyProvider::preprocessString(const std::string& aString)
  noexcept
{
    // For each loop that goes through each letter of aString
    // Checks that it complies to isalpha()
    // make it uppercase with toupper()
    // return string

    std::string pString;
    for (char c : aString)
    {
        if (isalpha(c))
        {
            pString += toupper(c);
        }
    }
    return pString;
}

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

```cpp

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

KeyProvider KeyProvider::operator++(int) noexcept
{
    // Return's a copy before advancing
    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); // Creates a copy
    temp.fIndex = 0;
    return temp; // Return's the old copy
}

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

```

```
// Cos3008-Mid
//Created By NUR E SIAM


#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();
}

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

VigenereForwardIterator& VigenereForwardIterator::operator++() noexcept
{
    ++fIndex;
    if (fIndex >= fSource.length())
    {
        return *this;
    }

    if (std::isalpha(fSource[fIndex]))
    {
        if (fMode == EVigenereMode::Encode)
            encodeCurrentChar();
        else
            decodeCurrentChar();

        ++fKeys;
    }
    else
    {
        fCurrentChar = fSource[fIndex]; // Non-alphabetic characters
          remain unchanged
    }

    return *this;
}

```

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

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

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

VigenereForwardIterator VigenereForwardIterator::begin() const noexcept
{
    return *this;
}

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

void VigenereForwardIterator::encodeCurrentChar() noexcept
{
    char sourceCharacter = fSource[fIndex];
    char keyCharacter = std::toupper(*fKeys);

    if (std::isalpha(sourceCharacter))
    {
        char encodedCharacter = fMappingTable[std::toupper(keyCharacter) -
          'A'][std::toupper(sourceCharacter) - 'A'];
        if (std::islower(sourceCharacter))
        {
            encodedCharacter = std::tolower(encodedCharacter);
        }
        fCurrentChar = encodedCharacter;
    }
    else
    {
        fCurrentChar = sourceCharacter;
    }
}

void VigenereForwardIterator::decodeCurrentChar() noexcept
{
    char sourceCharacter = fSource[fIndex];
    char keyCharacter = std::toupper(*fKeys);

    if (std::isalpha(sourceCharacter))
    {
        char decodedCharacter = 'A';
        if (std::isupper(sourceCharacter))
        {
            for (int i = 0; i < CHARACTERS; ++i)
            {
                if (fMappingTable[keyCharacter - 'A'][i] == std::toupper(sourceCharacter))
                {
                    decodedCharacter = 'A' + i;
                    break;
                }
            }
        }
        else if (std::islower(sourceCharacter))
        {
            for (int i = 0; i < CHARACTERS; ++i)
            {
                if (fMappingTable[keyCharacter - 'A'][i] == std::toupper(sourceCharacter))
                {
                    decodedCharacter = 'a' + i;
                    break;
                }
            }
        }
        fCurrentChar = decodedCharacter;
    }
    else
    {
        fCurrentChar = sourceCharacter;
    }
}
```