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: Rox Nguyen **Your student ID:** 103500095

Marker's comments:

Problem	Marks	Obtained
1	106	
2	194	
Total	300	

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

std::string KeyProvider::preprocessString(const std::string& aString) noexcept
{
    std::string lResult;
    for (char ch : aString)
    {
        if (std::isalpha(static_cast<unsigned char>(ch))) // static_cast<unsigned char>() to prevent out of scope later
        {
            lResult += std::toupper(static_cast<unsigned char>(ch));
        }
    }
    return lResult;
}

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

    std::string lProcessedSource = preprocessString(aSource);
    std::string lProcessedKeyword = preprocessString(aKeyword);
    fKeys = "";

    while (fKeys.length() < lProcessedSource.length())  // to make sure that the fkey >= source
    {
        fKeys += lProcessedKeyword;
    }
    fKeys = fKeys.substr(0, lProcessedSource.length());  // then trim to make key = source length
    assert(fKeys.length() == lProcessedSource.length());
}

char KeyProvider::operator*() const noexcept
{
    if (fIndex < fKeys.size())
    {
        return fKeys[fIndex];
    }
    // if index is out of range -> reset
    return fKeys[fIndex % fKeys.size()];
}

// prefix
KeyProvider& KeyProvider::operator++() noexcept
{
    ++fIndex;
    fIndex %= fKeys.size();  // to double ensure it reset
    return *this;
}

// postfix
```

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

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

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 "VigenereForwardIterator.h"
#include <cctype>
#include <cassert>
#include <iostream> //for debug purpose

VigenereForwardIterator::VigenereForwardIterator(const std::string& aKeyword, 
  const std::string& aSource, EVigenereMode aMode) noexcept
    : fKeys(KeyProvider(aKeyword, aSource)), fSource(aSource), fMode(aMode), 
      fIndex(0)
{
    initializeTable();
    if (!fSource.empty())
    {
        //fMode switch
        if (fMode == EVigenereMode::Encode)
        {
            encodeCurrentChar();
        }
        else if (fMode == EVigenereMode::Decode)
        {
            decodeCurrentChar();
        }
    }
}

//the mode part
void VigenereForwardIterator::encodeCurrentChar() noexcept
{
    // if non-alpha -> not changed
    if (fIndex >= fSource.length() || !isalpha(fSource[fIndex]))
    {
        fCurrentChar = fSource[fIndex];
    }
    else
    {
        char keyChar = toupper(static_cast<unsigned char>(*fKeys));
        char sourceChar = fSource[fIndex];
        int row = keyChar - 'A'; // the row index from the key character
        int column = toupper(static_cast<unsigned char>(sourceChar)) - 
          'A';  //the col index from the source character

        fCurrentChar = isupper(sourceChar) ? fMappingTable[row][column] : 
          tolower(fMappingTable[row][column]);

        assert(row >= 0 && row < CHARACTERS); //for debug purpose
        assert(column >= 0 && column < CHARACTERS);
    }
}

void VigenereForwardIterator::decodeCurrentChar() noexcept
{
    if (fIndex >= fSource.length() || !isalpha(fSource[fIndex]))
    {
        fCurrentChar = fSource[fIndex];
    }
    else
    {
        char keyChar = toupper(static_cast<unsigned char>(*fKeys));
        char sourceChar = fSource[fIndex];
        int row = keyChar - 'A';

        // iterates over the column in the specified row
        int column = 0;
        while (column < CHARACTERS && fMappingTable[row][column] != toupper(static_cast<unsigned char>(sourceChar)))
        {
            ++column;
        }

        //compare to source char -> upper || lower
        fCurrentChar = isupper(sourceChar) ? ('A' + column) : tolower('A' + column);
    }
}

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

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

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

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

```cpp

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

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

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