```cpp
//
//  VigenereIterator.cpp
//  midterm
//
//  Created by Xinzhe Yu on 14/4/2025.
//
#include <iostream>
#include <string>
#include <cctype>
#include "VigenereIterator.h"


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

    if(!fSource.empty()){
        if (fMode == EVigenereMode::Encode){
            encodeCurrentChar();
        }else{
            decodeCurrentChar();
        }
    }
}

void VigenereIterator::encodeCurrentChar() noexcept{
    char currentChar = fSource[fIndex];

    if (std::isalpha(currentChar)){
        char currentCharCap = std::toupper(currentChar);
        char keyChar = *fKeys;
        ++fKeys;
        int row = keyChar - 'A';
        int col = currentCharCap - 'A';
        char charAfterEncoded = fMappingTable[row][col];
        fCurrentChar = std::isupper(currentChar) ? charAfterEncoded :
         std::tolower(charAfterEncoded);
        fKeys += currentCharCap;
    }else{
        fCurrentChar = currentChar;
    }
}

void VigenereIterator::decodeCurrentChar() noexcept{
    char currentChar = fSource[fIndex];

    if (std::isalpha(currentChar)){
        char currentCharCap = std::toupper(currentChar);
        char keyChar = *fKeys;
```

```
        ++fKeys;
        int row = keyChar - 'A';
        int col = 0;
        while(currentCharCap != fMappingTable[row][col]){
            col+=1;
        }
        char charAfterEncoded = col + 'A';
        fCurrentChar = std::isupper(currentChar) ? charAfterEncoded :
         std::tolower(charAfterEncoded);
        fKeys += charAfterEncoded;
    }else{
        fCurrentChar = currentChar;
    }

}

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

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


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

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


VigenereIterator VigenereIterator::begin() const noexcept{
    VigenereIterator another = *this;
    another.fIndex = 0;
    another.fKeys.reset();
    another.fMode = fMode;
    if(!fSource.empty()){
        if (another.fMode == EVigenereMode::Encode){
            another.encodeCurrentChar();
        }else{
            another.decodeCurrentChar();
```

```
        }
    }
    return another;
}

VigenereIterator VigenereIterator::end() const noexcept{
    VigenereIterator another = *this;
    another.fIndex = fSource.length();

    return another;
}
```