Swinburne University Of Technology

Faculty of Information and Communication Technologies

ASSIGNMENT COVER SHEET

Subject Code: HIT3303/8303
Subject Title: Data Structures & Patterns
Assignment number and title: 2 - Arrays, Indexers, and Dynamic Link Libraries
Due date: **March 30, 2011, 10:30 a.m., on paper**
Lecturer: Dr. Markus Lumpe

Your name:_____

Marker's comments:

Problem	Marks	Obtained
1	99	
2	25	
3	2	
4	12	
Total	138	

Extension certification:

This assignment has been given an extension and is now due on _____

Signature of Convener:_____

Vigenere

Vigenere.h

```
#ifndef VIGENERE_H_
#define VIGENERE_H_

#include <string>
class Vigenere
{
private:
	char fCharacterMap[26][26];
	std::string fKey;
	unsigned int fKeyIndex;
	int fSourceFrequency[26];
	int fTargetFrequency[26];
	int fTotalEncoded;
public:
	Vigenere(char* aKey);
	void resetFrequencies();
	char encode(char aCharacter);
	char decode(char aCharacter);
	std::string encode(const std::string& aString);
	std::string decode(const std::string& aString);
	friend std::ostream& operator<<(std::ostream& aOStream, const Vigenere& aScrambler);
};

#endif /* VIGENERE_H_ */
```

Vigenere.cpp

```
#include "Vigenere.h"
#include <iostream>
#include <iomanip>
#include <string>
#include <string.h>

using namespace std;

Vigenere::Vigenere(char* aKey)
{
	fKeyIndex = 0;
	//Go through each character in aKey, convert to uppercase if needed, remove whitespace and non-alphabetical characters
	for(unsigned int i = 0; i < strlen(aKey); i++){
		if((aKey[i] >= 'a') && (aKey[i] <= 'z')){
			fKey += (aKey[i] - 32);
		}
		else if((aKey[i] >= 'A') && (aKey[i] <= 'Z')){
			fKey += aKey[i];
		}
	}
	//initialise character map
	for(int i = 0; i < 26; i++){
		for(int j = 0; j < 26; j++){
			int nextChar = (int)'B' + i + j;
			if((int)nextChar > (int)'Z'){
				nextChar -=  26;
			}
			fCharacterMap[i][j] = (char)nextChar;
		}
	}
	Vigenere::resetFrequencies();
}

void Vigenere::resetFrequencies()
{
	for(int i = 0; i < 26; i++){
		fSourceFrequency[i] = 0;
		fTargetFrequency[i] = 0;
	}
	fTotalEncoded = 0;
}

//Encode selected character - retains case
char Vigenere::encode(char aCharacter)
{
	int returnChar;

	if((aCharacter >= 'A') && (aCharacter <= 'Z')){
		int i = ((int)fKey[fKeyIndex] - (int)'A'); //so if character from cipher is 'A', index will be 0
		int j = ((int)aCharacter - (int)'A'); //same as above
		returnChar = fCharacterMap[i][j];

		//increment fKeyIndex - wrap back around if necessary
		fKeyIndex++;
		if(fKeyIndex >= fKey.length()){
			fKeyIndex = 0;
		}

		//increment frequencies - only done if character was alphabetical - take away 'A' to convert to index form
		fTotalEncoded++;
		fSourceFrequency[((int)aCharacter - (int)'A')]++;
```

```
                fTargetFrequency[((int)returnChar - (int)'A')]++;
        }
        else if((aCharacter >= 'a') && (aCharacter <= 'z')){
                int i = ((int)fKey[fKeyIndex] - (int)'A'); //so if character from cipher is 'A', index will be 0
                int j = ((int)aCharacter - (int)'A' - 32); //same as above - minus extra 32 to get uppercase
                returnChar = fCharacterMap[i][j] + 32; //add 32 to return to correct case

                //increment fKeyIndex - wrap back around if necessary
                fKeyIndex++;
                if(fKeyIndex >= fKey.length()){
                        fKeyIndex = 0;
                }

                //increment frequencies - only done if character was alphabetical - take away 32 to convert to lowercase, then
                'A' to convert to index form
                fTotalEncoded++;
                fSourceFrequency[((int)aCharacter - (int)'A' - 32)]++;
                fTargetFrequency[((int)returnChar - (int)'A' - 32)]++;
        }
        else{
                returnChar = (int)aCharacter;
        }
        return (char)returnChar;
}

//Decode the selected character - retains case
char Vigenere::decode(char aCharacter)
{
        int returnChar;
        if((aCharacter >= 'A') && (aCharacter <= 'Z')){
                int i = ((int)fKey[fKeyIndex] - (int)'A');
                //find character in column i and use that to infer j (the original character)
                for(int j = 0; j < 26; j++){
                        if(fCharacterMap[i][j] == aCharacter){
                                returnChar = (j + (int)'A'); //character = index + 'A' (i.e. A = 0)
                        }
                }

                //increment fKeyIndex - wrap back around if necessary
                fKeyIndex++;
                if(fKeyIndex >= fKey.length()){
                        fKeyIndex = 0;
                }
                //increment frequencies - only done if character was alphabetical - take away 'A' to convert to index form
                fTotalEncoded++;
                fSourceFrequency[((int)aCharacter - (int)'A')]++;
                fTargetFrequency[((int)returnChar - (int)'A')]++;
        }
        else if((aCharacter >= 'a') && (aCharacter <= 'z')){
                int i = ((int)fKey[fKeyIndex] - (int)'A');
                //find character in column i and use that to infer j (the original character)
                for(int j = 0; j < 26; j++){
                        if(fCharacterMap[i][j] == (aCharacter - 32)){ //-32 to make uppercase
                                returnChar = (j + (int)'A' + 32); //character = index + 'A' (i.e. A = 0) //add 32 to make
                                                                        lowercase again
                        }
                }

                //increment fKeyIndex - wrap back around if necessary
                fKeyIndex++;
                if(fKeyIndex >= fKey.length()){
                        fKeyIndex = 0;
                }
                //increment frequencies - only done if character was alphabetical - take away 32 to convert to lowercase, then
                'A' to convert to index form
                fTotalEncoded++;
```

```
                fSourceFrequency[((int)aCharacter - (int)'A' - 32)]++;
                fTargetFrequency[((int)returnChar - (int)'A' - 32)]++;
        }
        else{
                returnChar = (int)aCharacter;
        }

        return (char)returnChar;
}

std::string Vigenere::encode(const std::string& aString)
{
        string returnString = "";
        for(unsigned int i = 0; i < aString.length(); i++){
                returnString += Vigenere::encode(aString[i]);
        }
        return returnString;
}

std::string Vigenere::decode(const std::string& aString)
{
        string returnString = "";
        for(unsigned int i = 0; i < aString.length(); i++){
                returnString += Vigenere::decode(aString[i]);
        }
        return returnString;
}

std::ostream& operator<<(std::ostream& aOStream, const Vigenere& aScrambler)
{
        aOStream << "Frequency Distribution: " << endl;
        aOStream << "Char:  ";
        for(int i = 0; i < 26; i++){
                aOStream << setw(5) << setfill(' ') << (char)(i + (int)'A');
        }
        aOStream << endl << "Input: ";
        for(int i = 0; i < 26; i++){
                aOStream << setw(5) << setfill(' ') << aScrambler.fSourceFrequency[i];
        }
        aOStream << endl << "Output:";
        for(int i = 0; i < 26; i++){
                aOStream << setw(5) << setfill(' ') << aScrambler.fTargetFrequency[i];
        }

        return aOStream;
}
```

Scramble

main.cpp

```
#include <Vigenere.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>

using namespace std;

int main(int argc, char* argv[])
{
	ifstream lReader;
	ofstream lWriter;
	string lLine;

	//check correct number of arguments
	if(argc != 3){
		cout << "Incorrect number of arguments!" << endl;
		cout << "Program execution: 'scramble' 'key' fileToScramble.txt" << endl;
		exit(1);
	}

	Vigenere* lScrambler = new Vigenere(argv[1]);

	//create and open input and output files
	lReader.open(argv[2]);
	char temp[strlen(argv[2]) + 11]; //create temp character array 11 characters longer than file name (to allow for appending)
	strcpy(temp, argv[2]);
	lWriter.open(strcat(temp, ".secure.txt")); //append original file with '.secure.txt' and open for output

	//check files opened correctly
	if(!lReader.is_open()){
		cout << "Error opening specified file!" << endl;
		exit(1);
	}
	else if(!lWriter.is_open()){
		cout << "Error opening file to save scrambled text!" << endl;
		exit(1);
	}

	//Read each line, scramble, and write to the output file
	while(getline(lReader, lLine)){
		lWriter << lScrambler->encode(lLine);
	}

	//Display character frequencies
	cout << "Scrambling '" << argv[2] << "' using key: " << argv[1] << endl;
	cout << *lScrambler;

	lReader.close();
	lWriter.close();
	delete lScrambler;

	return 0;
}
```

Unscramble

main.cpp

```
#include <Vigenere.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>

using namespace std;

int main(int argc, char* argv[])
{
	ifstream lReader;
	ofstream lWriter;
	string lLine;

	//check correct number of arguments
	if(argc != 3){
		cout << "Incorrect number of arguments!" << endl;
		cout << "Program execution: 'scramble' 'key' fileToUnscramble.txt" << endl;
		exit(1);
	}

	Vigenere* lScrambler = new Vigenere(argv[1]);

	//create and open input and output files
	lReader.open(argv[2]);
	char temp[strlen(argv[2]) + 11]; //create temp character array 11 characters longer than file name (to allow for appending)
	strcpy(temp, argv[2]);
	lWriter.open(strcat(temp, ".public.txt")); //append original file with '.public.txt' and open for output

	//check files opened correctly
	if(!lReader.is_open()){
		cout << "Error opening specified file!" << endl;
		exit(1);
	}
	else if(!lWriter.is_open()){
		cout << "Error opening file to save decoded text!" << endl;
		exit(1);
	}

	//Read each line, scramble, and write to the output file
	while(getline(lReader, lLine)){
		lWriter << lScrambler->decode(lLine);
	}

	//Print character frequencies
	cout << "Unscrambling '" << argv[2] << "' using key: " << argv[1] << endl;
	cout << *lScrambler;

	lReader.close();
	lWriter.close();
	delete lScrambler;

	return 0;
}
```

VigenereIndexer

VigenereIndexer.h

```
#ifndef VIGENEREINDEXER_H_
#define VIGENEREINDEXER_H_

#include "Vigenere.h"

class VigenereIndexer
{
private:
	Vigenere fCipher;
	bool fMode;

public:
	VigenereIndexer(char* aKey, bool aMode);
	char operator[](const char aChar);

	friend std::ostream& operator<<(std::ostream& aOStream, const VigenereIndexer& aIndexer);
};


#endif /* VIGENEREINDEXER_H_ */
```

VigenereIndexer.cpp

```
#include "VigenereIndexer.h"

VigenereIndexer::VigenereIndexer(char* aKey, bool aMode): fCipher(aKey)
{
	fMode = aMode; //true for encoding, false for decoding
}

char VigenereIndexer::operator[](const char aChar)
{
	if(fMode){
		//Encoding
		return fCipher.encode(aChar);
	}
	else{
		//Decoding
		return fCipher.decode(aChar);
	}
}

//Prints character frequencies
std::ostream& operator<<(std::ostream& aOStream, const VigenereIndexer& aIndexer)
{
	aOStream << aIndexer.fCipher;
	return aOStream;
}
```

main.cpp

```
#include "VigenereIndexer.h"
#include <iostream>
#include <fstream>
#include <string.h>
#include <stdlib.h>

using namespace std;

int main(int argc, char* argv[])
{
	ifstream lReader;
	ofstream lWriter;
	string strMode; //hold string representation of mode - char* causes issues
	bool lMode;

	//check arguments
	if(argc != 4){
		cout << "Invalid arguments!" << endl;
		cout << "Execution: VigenereIndexer encode/decode key fileToUse.txt" << endl;
	}
	//find mode
	strMode = argv[1];
	if(strMode == "encode"){
		lMode = true;
	}
	else if(strMode == "decode"){
		lMode = false;
	}
	else{
		cout << "Invalid mode! Mode must be 'encode' or 'decode'" << endl;
		exit(1);
	}

	//create and open files
	lReader.open(argv[3]);
	char temp[(strlen(argv[3]) + 11)];//create temp for file name, allow for appending
	strcpy(temp, argv[3]);
	if(lMode){
		//create file for encode
		lWriter.open(strcat(temp, ".encode.txt"));
	}
	else{
		//create file for decode
		lWriter.open(strcat(temp, ".decode.txt"));
	}
	//check files opened correctly
	if(!lReader.is_open()){
		cout << "Error opening file to be worked on!" << endl;
		exit(1);
	}
	else if(!lWriter.is_open()){
		cout << "Error opening file to save to!" << endl;
		exit(1);
	}

	//create indexer
	VigenereIndexer lScrambler(argv[2], lMode);

	//encode or decode specified file
	char lChar;
	while((lChar = lReader.get()) != EOF){
		lWriter << lScrambler[lChar];
	}
```

```
	//Print frequencies
	if(lMode){
		cout << "Encoding '";
	}
	else{
		cout << "Decoding '";
	}
	cout << argv[3] << "' using key: " << argv[2] << endl;
	cout << lScrambler;

	lReader.close();
	lWriter.close();

	return 0;
}
```