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:	1 - Arrays, Indexers, and Iterators
Due date:	**March 31, 2009, 02:30 p.m., on paper**
Lecturer:	Dr. Markus Lumpe

Your name:____________________

Marker's comments:

Problem	Marks	Obtained
1	55	
2	15	
3	30	
Total	100	

Extension certification:

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

Signature of Convener:________________________

Problem Set 1: Arrays, Indexers, and Iterators

Problem 1: Problem Solving in C++ (55%)

Around 1550 Blaise de Vigenère, a French diplomat from the court of Henry III of France, developed a new scrambling technique that uses 26 alphabets to cipher a text. The *Vigenère Cipher* is a polyalphabetic substitution technique based on the following *tableau:*

Key\Letter	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	**L**	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

The Vigenère cipher uses this table together with a keyword to encode a message. For example, suppose we wish to scramble the following message:

```
TO BE OR NOT TO BE THAT IS THE QUESTION
```

using the keyword RELATIONS. We begin by writing the keyword, repeated as many times as necessary, above the message. To derive the encoded text using the tableau, for each letter in the message, one finds the intersection of the row given by the corresponding keyword letter and the column given by the message letter itself to pick out the encoded letter.

Keyword:	RE LA TI ONS RE LA TION SR ELA TIONSREL
Message:	TO BE OR NOT TO BE THAT IS THE QUESTION
Scrambled Message:	LT NF IA CCM LT NF NQPH BK YTF KDTGMATZ

Decoding of an encrypted message is equally straightforward. One writes the keyword repeatedly above the message:

Keyword:	RE LA TI ONS RE LA TION SR ELA TIONSREL
Scrambled Message:	LT NF IA CCM LT NF NQPH BK YTF KDTGMATZ
Decoded Message:	TO BE OR NOT TO BE THAT IS THE QUESTION

This time one uses the keyword letter to pick a row of the table and then traces the row to the column containing the encoded letter. The index of that column is the decoded letter.

Stage 1 (35%)

Implement a C++ dynamic link library called Vigenere (Windows file name Vigenere.dll, Unix file name libVigenere.so, and MacOS file name libVigenere.dylib). That is, define a header file Vigenere.h defining the class Vigenere and a C++ file Vigenere.cpp that implements this class.

```
class Vigenere
{
private:
  char fCharacterMap[26][26];

  void SetupTable();
  void EncodeChar( char aKey, char& aChar ) const;
  void DecodeChar( char aKey, char& aChar ) const;

public:
  Vigenere();

  std::string MakeKeyUppercase( char* aKey );

  void Encode( const std::string& aUppercaseKey, int& aKeyIndex,
               std::string& aText ) const;
  void Decode( const std::string& aUppercaseKey, int& aKeyIndex,
               std::string& aText ) const;
};
```

The methods Encode and Decode only cipher characters (or letters). All other characters remain unchanged (i.e., the cipher process ignores them). Furthermore, the Vigenère cipher uses upper case characters only. That is, both the keyword and the message have to be converted to uppercase characters strings first before applying the cipher. However, one requirement of this assignment is that the all methods of class Vigenere properly handle case-sensitive spelling. That is:

Keyword:	R elati o nsRel at ionsRel ati o nsRel
Message:	A horse! a horse! my kingdom for a horse.
Encoded Message:	S masmn! p vhjxq! ns txbzvty gia p vhjxq.

Both `Encode` and `Decode` take three reference arguments: `aUppercaseKey`, `aKeyIndex`, and `aText`. The parameter `aUppercaseKey` is a "constant reference" to the key string in the caller space. In other words, `aUppercaseKey` is a constant object that provides a read-only access to the key string in the caller space without copying. The parameters `aKeyIndex` and `aText`, on the other hand, are reference parameters that allow for the occurrences of side effects. More precisely, both `Encode` and `Decode` alter these parameters. The parameter `aKeyIndex` denotes an index into `aUppercaseKey` to locate the next key character. The cipher process increments `aKeyIndex` every time a character in `aText` has been processed. Since `aKeyIndex` is a reference parameter, the effect on `aKeyIndex` is visible also to the caller. The purpose of passing `aText` by reference is to avoid copying and to perform an in-place manipulation of the individual characters in `aText`. That is, `Encode` and `Decode`, both read and write to `aText` in order to perform their corresponding cipher function. The result is visible to the caller. Conceptually, actual parameters to `aUppercaseKey` are passed by constant reference, whereas actual parameters to `aKeyIndex` and `aText` are passed by reference.

Example:

```
string lUppercaseKey = “RELATIONS”

int lKeyIndex = 0

string lText = “A horse! A horse!”

Encode( lUppercaseKey, lKeyIndex, Text )

        -> lText  = “S masmn! P vhjxq!”

        -> lKeyIndex = 3

lText = “my kingdom for a”

Encode( lUppercaseKey, lKeyIndex, lText )

        -> lText = “ns txbzvty gia p”

        -> lKeyIndex = 7
```

The class `Vigenere` defines also an additional public method `MakeKeyUppercase` and three private methods. These are auxiliary methods that facilitate the definition of the public members. In this assignment it is required to define and use the private methods appropriately! **You must not change the class specification!**

Stage 2 (10%)

Using the dynamic link library `Vigenere` implement the C++ console application `scramble` that takes two arguments `key` and `file_name` and encodes the text file named `file_name`:

```
$> ./scramble Relations Sample.txt
```

generates the file `Sample.txt.secure.txt`, the encoded version of `Sample.txt`.

The encoding of a text file has to be implemented in a while loop:

```
while ( getline( lReader, lLine ) )
{
  lScrambler.Encode( lUppercaseKey, lKeyIndex, lLine );
  lOutput << lLine << endl;
}
```

Remember to set the system-specific environment to locate the `Vigenere` shared library (Windows: `PATH`, Linux: `LD_LIBRARY_PATH`, and MacOS: `DYLD_LIBRARY_PATH`).

Stage 3 (10%)

Using the dynamic link library `Vigenere` implement the C++ console application `unscramble` that takes two arguments `key` and `file_name` and decodes the text file named `file_name`:

```
$> ./unscramble Relations Sample.txt.secure.txt
```

produces the file `Sample.txt.secure.txt.public.txt`, the decoded version of `Sample.txt.secure.txt`.

The decoding of a text file has to be implemented in a while loop:

```
while ( getline( lReader, lLine ) )
{
  lScrambler.Decode( lUppercaseKey, lKeyIndex, lLine );
  lOutput << lLine << endl;
}
```

Problem 2: Indexer (15%)

Define a Vigenère indexer adhering to the following class specification:

```
class VigenereIndexer
{
private:
  Cipher fCipher;
  std::string fUppercaseKey;
  bool fMode;
  int fKeyIndex;

public:
  VigenereIndexer( char* aKey, bool aMode );

  char operator[]( const char aChar );
};
```

A Vigenère indexer is an object that is initialized with a code word `aKey` and an encryption modus `aMode` (i.e., `aMode == true` for encoding, `aMode == false` for decoding). The indexer defines an on-the-fly encryption mechanism. Each consecutive use of the `[]` operator will yield the corresponding encoded or decoded character.

Build a program using the Vigenère indexer and change the main function to contain a while loop as follows (`lScrambler` is the corresponding indexer):

```
char lChar;
while ( (lChar = lReader.get()) != EOF )
{
      lOutput << lScrambler[lChar];
}
```

Problem 3: Iterators (30%)

Stage 1 – Output Iterator (15%)

Define a write-only Vigenère output iterator:

```
class VigenereOutputIterator
{
private:
  Cipher fCipher;
  std::string fUppercaseKey;
  int fKeyIndex;
  std::ofstream& fOutStream;

public:
  VigenereOutputIterator( char* aKey, std::ofstream& aOutStream );

  // Iterator behavior
  VigenereOutputIterator& operator*();
  VigenereOutputIterator& operator=( const char aChar );
  VigenereOutputIterator& operator++();
  VigenereOutputIterator& operator++(int);
};
```

The output iterator performs the encoding process. We initialize the output iterator with the output file stream `aOutStream` of type `ofstream`. We pass this stream by reference to the constructor of the output iterator.

An output iterator is like a "black hole". It only provides a write operation. No read is supported. However, each operation supported by the output iterator has to return the very same iterator object. For this reason each defined operator has as return type `VigenereOutputIterator&`. Interestingly, only the assignment operator alters the state of the output iterator. All other operators, though required to build a program, just return `*this`.

See also http://www.cplusplus.com/reference/std/iterator/ostream_iterator.html

Build a program using the Vigenère output iterator and change the main function to contain a while loop as follows (`lKey` stands for the keyword argument and `lOutput` for the output file stream):

```
VigenereOutputIterator lScrambler( lKey, lOutput );
char lChar;
while ( (lChar = lReader.get()) != EOF )
{
      *lScrambler++ = lChar;
}
```

Stage 2 – Forward Iterator (15%)

Define a read-only Vigenère forward iterator:

```
class VigenereForwardIterator
{
private:
  Cipher fCipher;
  std::string fUppercaseKey;
  int fKeyIndex;
  std::ifstream& fInStream;
  char fCurrentChar;

public:
      VigenereForwardIterator( char* aKey, std::ifstream& aInStream );

      char operator*() const;
      VigenereForwardIterator& operator++();
      VigenereForwardIterator operator++(int);

      bool eof() const;
};
```

The forward iterator performs the decoding process. We initialize the forward iterator with the input file stream `aInStream` of type `ifstream`. We pass this stream by reference to the constructor of the forward iterator. In addition, the constructor initializes `fCurrentChar` using one of the increment operators.

A forward iterator provides a read operation only. No write is supported. The magic happens in the increment operators. Here, we read the next character from the input stream and perform on-the-fly decoding. The result of this process is stored in `fCurrentChar`. The next read (i.e., an application of the dereference operator) will yield the decoded character. If we do not increment the iterator between dereference calls, then the very same decoded character is returned.

The method `eof()` is a special auxiliary function to test for the end of the forward iterator.

Build a program using the Vigenère forward iterator and change the main function to contain a while loop as follows (`lKey` stands for the keyword argument and `lReader` for the input file stream):

```
  for ( VigenereForwardIterator iter( lKey, lReader ); !iter.eof(); iter++ )
  {
        lOutput << *iter;
  }
```

Submission deadline: Tuesday, March 31, 2009, 2:30 p.m.

Submission procedure: on paper.