CSC112 Fall 2010
Fundamentals of Computer Science
Lab 8

In this lab, we will be working on our first use of C++ classes.

Create a directory named Lab8. Keep all of your source and compiled programs in the directory Lab8.

In this lab, we will try our hand at a simple substitution cipher to encode and decode a text file. You will write two programs: one to encode the text, and one to decode it. Your program must be implemented using a C++ class that you will write. The class definition is:

```
const int ALPHABET_SIZE=26 ;  // Global constant

class cipher {
   private:
      int encode_map[ALPHABET_SIZE] ;
      int decode_map[ALPHABET_SIZE] ;
   public:
      cipher() ;   // Constructor
      ~cipher() ;  // Destructor
      void encode_text( char * input_file, char * output_file ) ;
      void decode_text( char * input_file, char * output_file ) ;
} ;
```

How does the encoding work ?

The array named `encode_map` is intended to contain a permutation of the integers $[0, 1, 2, ..., 25]$. The permutation defines the substitution scheme. Suppose we let the letter 'a' correspond to position 0 in the array `encode_map`, let the letter 'b' correspond to position 1, and so forth through the alphabet to letter 'z' which corresponds to position 25.

Suppose our input letter is 'a', and in position 0 of the `encode_map` we find the number 10. The 10^{th} letter of the alphabet (starting with 'a' as the 0^{th} letter) is 'k', so we output the letter 'k' instead of 'a'.

Similarly, if our input letter is 'd', and in position 3 (the position corresponding to 'd', again counting from zero) of the `encode_map`, we find the number 2. The 2^{nd} letter of the alphabet (starting from zero) is 'c', so we output the letter 'c' instead of the letter 'd'.

Ok, what about capital letters ?

For capital letters, use the same encoding scheme and output the corresponding capital letter. For our examples above, 'A' would translate to 'K' and 'D' would translate to 'C'.

What permutation do we use ?

There are many ways to generate a permutation. An algorithm to do this is given below:

```
Pick any number between 1 and ALPHABET_SIZE-1 inclusive as the
starting point; we will call this number s.
```

```
Pick any number between 3 and 23 that is relatively prime to 26
as our increment ;  we will call this number t.

for (k = 0 ; k < ALPHABET_SIZE ; k++ ) {
      encode_array[ (s % ALPHABET_SIZE) ] = k ;
      s += t ;
}
```

What ?

The algorithm above creates a permutation of the numbers $[0, 1, 2, ..., 25]$ because of a mathematical theorem which we will discuss in class. If we consider the multiples of t namely $[0, t, 2t, 3t, 4t, ..., 25t]$, we can prove that if t is relatively prime to 26, each of these numbers are distinct modulo 26. If you take a course called "Modern Algebra" in the Mathematics department, you will call the number t a **group generator**. Starting with s does not change this fact. When starting with s we are generating the sequence:

$$(s + \ell t) \text{ modulo } 26 \quad \text{where } \ell = 0, 1, 2, ...25$$

Each of the numbers generated in this sequence are distinct.

There is nothing magical about 26 here. You can generate permutations in this way with any number n in place of 26, provided that t is relatively prime to n.

So they are distinct, how do I know they are all there ?

If you have a list of 26 numbers in the range 0 through 25, and there are no duplicates on the list, then all of the numbers 0 through 25 must be on the list. This line of logical thinking is sometimes called the "pigeon hole principle".

How do the arrays `encode_map` and `decode_map` get set up correctly ?

The two maps are computed by the class constructor. When the cipher object comes into scope, the constructor will automatically be called, and the maps will be set up correctly. If you need any additional functions to implement the maps, make them private functions.

You must define a destructor, but it does not have to do anything.

How does the member function `encode_text` work ?

The member function `encode_text` accepts two file names: an input file and an output file. Open the input file and read characters one at a time until end-of-file. Open the output file, and for each character read, use the array `encode_map` to find a substitute letter, and output the substitute letter to the output file. Lower case letters are mapped to substitute lower case letters, and upper case letters are mapped to substitute upper case letters. To do this, you will need to distinguish between upper and lower case letters. Fortunately, there are some C++ library functions so you can do this easily. An example using `isupper()` and `islower()` is shown below. (See next page.)

If you encounter an input character that is neither an upper case letter, nor a lower case letter, then write it to the output file unchanged.

```
#include <iostream>
#include <cctype>
using namespace std ;

int main()
{
   char ch ;

   cin >> ch ;

   if ( isupper(ch)) {
      cout << "Upper case" << endl ;
   }
   else if (islower(ch)) {
      cout << "Lower case" << endl ;
   }
   else {
      cout << "Input letter is not A-Z, nor a-z" << endl ;
   }
}
```

So what about the array named `decode_map` ?

The array `decode_map` holds the **inverse permutation** of the array `encode_map`. After generating the array `encode_map`, you need to compute the inverse permutation and store it in the array named `decode_map`. Inverse permutations, and how to compute them will be discussed in class.

And the two main programs ? You will write one program named `encode.cc`, and another named `decode.cc`. Both programs will use the same cipher class.

The program `encode.cc` will:

- Declare an object of class `cipher`
- Read from a file named "plain_text"
- Output to a file named "cipher_text"

The program `decode.cc` will:

- Declare an object of class `cipher`
- Read from a file named "cipher_text"
- Output to a file named "decoded_text"

You can download a test file named "plain_text" from the web page:

`http://menehune.opt.wfu.edu/CSC112`

or you can create a test file of your choice. Please include at least 2 lines of text and at least 10 words (total) of input text. Include both upper and lower case letters, and at least one character that is not a letter.

Test the encoding program with the input text. Run your decode program and verify that you recover the original text.

Turn in: Change to the directory containing the sub-directory "Lab8". Create a file named "lab8.tar" using the command:

```
tar cf lab8.tar Lab8
```

Upload the file "lab8.tar" to your account on telesto.