CSE 232 Spring 2016

Programming Project 05

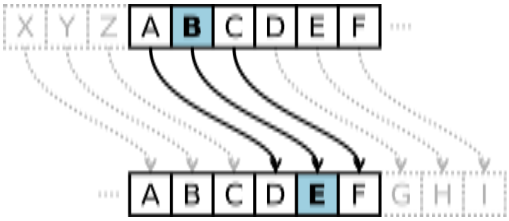
Assignment Overview

This assignment is worth 40 points (4.0% of the course grade) and must be completed and turned in before 11:59pm on Monday, February 29th , 2016. That's two weeks because of the midterm on Thur, February 18th.

Background

Caesar cipher

The Caesar cipher is named after Julius Caesar who used this type of encryption to keep his military communications secret. A Caesar cipher replaces each plain-text letter with one that is a fixed number of places down the alphabet. The fixed number is called the ***shift***. The ***plain-text*** is your original message; the ***cipher-text*** is the encrypted message. The example shown below uses a shift of three so that "B" in the plain-text becomes "E" in the cipher-text, a "C" becomes "E", and so on. The mapping wraps around so that "X" maps to "A" and so on.



Here is the complete mapping for a shift of three:

Plain-text: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher-text: DEFGHIJKLMNOPQRSTUVWXYZABC

To encrypt a message simply substitute the plain-text letters with the corresponding cipher-text letter. For example, here is an encryption of "the quick brown fox jumps over the lazy dog" using our shift-three cipher (case is ignored and spaces are preserved):

Plaintext: the quick brown fox jumps over the lazy dog
Ciphertext: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ

To decrypt the message simply reverse the process.

A more complicated cipher, round 1

It turns out the Caesar cypher can be cracked pretty easily just by using stats about English (or any language's) letters. An improved version of a Caesar is called a **homophonic** cipher (https://en.wikipedia.org/wiki/Substitution_cipher#Homophonic_substitution) which has multiple candidate substitutions for each letter, making the statistics more difficult. Consider a fairly simple such approach.

We map each letter of the alphabet to an integer. It's easiest to do this by considering a string of the form `"abcdefghijklmnopqrstuvwxyz"` where the index of `'a'` is 0 and of `'z'` is 25, with the string's length being 26. Thus we can map each letter to a number, its index. However, we need multiple numbers for each letter for a homophonic cipher.

To do so we genearte random number integers as multipliers. We encode the letter using the following formulat: `encoded = 26 * random_num + letter_index`. Consider the example below:

letter	'a'	'b'	'c'	'a'	'b'	'c'
index	0	1	2	0	1	2
random int	5	10	6	25	80	3
encoded	130	261	158	650	2081	54

Using the formula, we can have an infinite set of numbers for each letter. To decode, you simply take `encoded % 26` which gives you the proper index back.

A more complicated cipher, round 2
That's still a little weak. Most cryptographers would be struck by the multiples of 26 that show up in the encoded stream, making it a little obvious. We can add one more element, a `shift_key`, to make the encoding better.

The idea is pretty simple. We read in a vector of integers to use as values to add to the encoding. Every time we generate the encoded value, we add one of the `shift_key` values to the encoded value. If the `shift_key` is shorter than the length of the encoded stream, we reuse the `shift_key` in order; that is if we reach the last value of the `shift_key`, the next value will be the first value. Let's assume that we read in a vector that has the numbers such that:
shift_key = {123, -554}

We change the enhance the encoding process as follows:

letter	'a'	'b'	'c'	'a'	'b'	'c'
index	0	1	2	0	1	2
random int	5	10	6	25	80	3
encoded1	130	261	158	650	2081	54
shift_key	123	-554	123	-554	123	-554
final encoded	253	-293	281	96	2204	-500

To decode, you must reverse the process. You must know the shift_key (the secret code if you will), reverse the shift_key addition, then take the modulo value.

`(final_encoded - shift_key[index]) % 26`, where `index` is determined by order of addition during the encoding round.

ASCII
As a note, you don't need a string to turn a letter into an index number. The index order of an ascii letter can be found by subtracting the character `'a'` from any other lower-case letter. Thus the letter `'f'` is index 5, found by `'f' - 'a'` . You did this in lab last week.

Program Specifications
As before, we provide the header and the main program, you provide the functions listed below.

function `filter_string`: returns indicating if the string argument is all lower case letters.
```
string filter_string(string s)
```
- returns a string containing all the alphabetic character in lower case. Non alphabetic characters are removed (numbers, punctuation, spaces, etc.)
- if s contains no alphabetic characters, returns the null string.

function `read_key`: no return.
```
void read_key(ifstream &in_file, vector<long> &shift_key)
```
- reads ***a single line*** from the provided `ifstream` called `in_file`
- reads all the integers in the gathered line from `in_file` into `shift_key`

function `encode`: returns the encoded string from `to_encode` using the vector `shifts` and random numbers generated from the provided random engine and distribution
```
string encode(string to_encode, vector<long>& shifts,
              mt19937_64 &reng,
              uniform_int_distribution<long>& dist)
```
- the `to_encode` string should have been processed by `filter_string`

function `decode`: returns the decoded string from `to_decode` using the vector `shifts`
```
string decode(string to_decode, vector<long>& shifts)
```
returns the decoded string.

Deliverables
You must use handin to turn in a file called `functions-05.cpp` . Do not provide the `main-05.cpp` or the `functions-05.h` that we already gave you. Only `functions-05.cpp`. your code. Please be sure to use the specified file name, and save a copy of your `functions-05.cpp` file to your H drive as a backup.

Assignment Notes
1. All input will be from a file called `"test.txt"`. See the example.
2. The `test.txt` file format is as follows:
    a. First line is a space separated list of the `shift_key` values
    b. Every other line starts with one of two characters:
        i. `e` – encode what follows
        ii. `d` – decode what follows
    c. the remainder of the line is the message (to encode or decode depending)

3. You should not change the main.cpp file, but you can make your declarations easier with a using statement, especially when the declaration is long. For example
   ```
   using r_eng=mt19973_64;
   using distribution = uniform_int_distribution<long>;
   ```
   In which case you can use your own shortcut for a declaration
   ```
   r_eng my_reng(32);
   distribution dist(1,100);
   ```
   It's the kind of thing that can save you from fat-fingered typing. Up to you
4. You may write other functions as well. They can be in `functions-05.cpp` to be called by other functions in the same file. They ***do not go*** in `functions-05.h` as they are not going to be used in `main.cpp`. In some sense those local functions are private.