CS 105 – Lab #10 – One-time pad

A one-time pad is similar to a Vigenere cipher. But the important feature is that the key is long and random, and each time we use part of the key, we need to throw it away so it's not used again. This is what we will practice in lab today.

Log into the computer as usual. It's a good idea to have two terminal windows in the screen so that you can see them side by side. One window can be devoted to editing a program, and the other window can be used for running programs or viewing other files.

Create a new folder called lab10 to hold today's work. Make sure that your current working directory is lab10 before continuing. You can enter the command `pwd` to verify this. From the class Web site, find the lab10 folder, and download the file `masterkey.txt` into your lab10 folder. This file will be essential for our one-time pad encipherment today.

Overview of the lab:

We pretend that Alice wishes to send an encrypted message to a friend Bob. Both Alice and Bob will use the same one-time pad. The original copy of the one-time pad is the file `masterkey.txt`. This file contains 5000 random capital letters, arranged in 50 lines of 100 letters per line.

Throughout this exercise, `masterkey.txt` will not change. Instead, Alice and Bob will each maintain their own versions of the one-time pad, which over time will be consumed. To begin, Alice will copy `masterkey.txt` into her local copy `key1.txt`. And Bob will copy `masterkey.txt` into his local copy `key2.txt`.

Let's assume that every message that is sent contains no more than 100 letters. Therefore, it suffices to use just 1 line of the key in order to encrypt or decrypt each message. Each time we encrypt or decrypt, we need to delete this (first) line from the key so ensure it is never used again. Therefore: Alice will use the first line of `key1.txt` to encrypt her message. Then, Alice will copy `key1.txt` into a new version `key3.txt`, which omits the entire line that has just been used during the encipherment. For example, if `key1.txt` has 50 lines, then `key3.txt` will contain the last 49 lines only. Later, if Alice wants to send another message, then she needs to copy `key3.txt` into `key1.txt` before running her encryption program again.

Meanwhile, Bob will use `key2.txt` to decrypt the ciphertext message he received from Alice. He will use just the first line of `key2.txt` as the key. After deciphering the message, he'll copy `key2.txt` into `key4.txt`, omitting the line that has just been used as the key. For example, if `key3.txt` has 50 lines, then `key4.txt` will contain only the last 49 lines. Later,

if Bob receives another message from Alice, he'll need to copy `key4.txt` into `key2.txt` before beginning the decryption program again.

So, to sum up, here is how to distinguish among the various "key" files:

- masterkey.txt – The original 5000 random letters of the one-time pad. We will not change this file, so that later we can refer back to it in case we need to start the experiment over.
- key1.txt – Alice's copy of the one-time pad before starting the encryption
- key2.txt – Bob's copy of the one-time pad before starting the decryption
- key3.txt – Alice's copy of the one-time pad AFTER the encryption is finished
- key4.txt – Bob's copy of the one-time pad AFTER the decryption is finished

To conduct this experiment with one-time pads, you will need to write two programs. Alice's program will be called `encipher.pas`, and Bob's program is called `decipher.pas`. Let's do the encipherment first. Using nano, let's create a new Pascal source file called `encipher.pas`. Here is the source code, with some details you need to fill in. The missing steps are numbered 1-5. For your convenience, all of the variables you need have been declared.

```
(* encipher.pas - Encrypt using a one-time pad. *)
program encipher;

const
  asciioffset = 64;

var
  infile, outfile : textfile;
  key, plaintext, ciphertext, line : string;
  i, plainvalue, keyvalue, ciphervalue : integer;
  plainletter, keyletter, cipherletter : char;

begin
  (* 1. Have infile point to the file key1.txt for input. *)


  (* The key for the Vigenere cipher will be just the first line
from key1.txt. *)
  (* 2. Read a line from infile; put it in variable line. *)


  (* Ask user for plaintext message. *)
```

```
  writeln('Please enter plaintext message.  No spaces or
punctuation, please.');
  readln(plaintext);
  plaintext := upcase(plaintext);

  (* For each letter of plaintext, "add" next letter of key. *)
  ciphertext := '';
  for i := 1 to length(plaintext) do
    begin
      plainletter := plaintext[i];
      plainvalue := ord(plainletter) - asciioffset;
      keyletter := key[i];
      keyvalue := ord(keyletter) - asciioffset;
      ciphervalue := plainvalue + keyvalue;
      if ciphervalue > 26 then
        ciphervalue := ciphervalue - 26;
      cipherletter := chr(ciphervalue + asciioffset);
      ciphertext := ciphertext + cipherletter;
    end;

  writeln('Ciphertext:  ', ciphertext);

  (* Finally, we modify the key.  Omit the first line, which we
already read. *)
  (* 3. Associate outfile with key3.txt.  Open for writing. *)


  while not eof(infile) do
    begin
      (* 4. Read a line from the infile, put into variable line,
       *    and write the variable line into the output file.
       *)
    end;

  (* 5.  Close both files. *)
end.
```

Save, compile, and run the program `encipher.pas`. Enter this plaintext message: EGGBASKET. Be sure not use any spaces or punctuation. What is the ciphertext?

Next, examine the files `key1.txt` and `key3.txt`. A quick way you can do this is with the command `wc key?.txt`. The first number in the output indicates the number of lines a file has. How can you tell that the encipherment program correctly deleted the first line of the one-time pad?

Next, let's compose Bob's program: `decipher.pas`. The code for this program should be analogous to `encipher.pas`, with these important exceptions:

- We use files `key2.txt` and `key4.txt`, rather than `key1.txt` and `key3.txt`.
- We should ask for the ciphertext, not plaintext, from the user.
- Similarly, the loop should be traversing the ciphertext instead of the plaintext. The loop should be subtracting the key from the ciphertext, instead of adding the key to the plaintext. Many variable names should be changed in the loop.
- Finally, the output should be introduced as the recovered plaintext.

Save, compile, and run `decipher.pas`. Use the ciphertext that you obtained above as input. Does the program give you the correct plaintext original?

How many lines does `key4.txt` have now? _____

Suppose you want Alice to send another message to Bob. What specifically must you do to the key text files before you run `encipher.pas` and `decipher.pas` again?

Run `encipher.pas` and `decipher.pas` on a second message from Alice to Bob. How many lines do `key3.txt` and `key4.txt` have?