Assignment 1
CS-452
Spring 2020

Due Date: 4/9/2021 at 11:59 pm (No extensions)
You may work in groups of 6

Goals:

1. To better comprehend the classical ciphers covered in class.
2. To successfully implement the 5 classical ciphers covered in class.
3. To understand the challenges of implementing ciphers.
4. To preapare for implementing modern ciphers.

Overview

In this assignment you shall implement the following classical ciphers:

- Playfair
- Row Transposition
- Railfence
- Vigenre
- Caesar

Your program shall be called `cipher` and shall be executed as:

```
./cipher <CIPHER NAME> <KEY> <ENC/DEC> <INPUTFILE> <OUTPUT FILE>
```

where

- `CIPHER NAME:` Is the name of the cipher. Valid names are:
  - `PLF:` Playfair
  - `RTS:` Row Transposition
  - `RFC:` Railfence
  - `VIG:` Vigenre
  - `CES:` Caesar
  - `MAC:` Monoalphabetic Cipher (graduate students only)
- `KEY:` the encryption key to use.

- `ENC/DEC:` whether to encrypt or decrypt, respectively.
- `INPUT FILE:` the file from which to read the input.
- `OUTPUT FILE:` the file to which the output shall be written.

The program shall then read the input file, encrypt/decrypt the file contents using the specified key and cipher, and write the encrypted/decrypted contents to the specified file. (**Your program shall be tested on very large as well as very small files**). For example: `./cipher VIG security ENC in.txt out.txt` will read the contents of file in.txt, encrypt them using the Vigenre cipher and key `security`, and write the encrypted contents to `out.txt`. ***You may assume that the input file does not contain punctuation or spaces.***

THE CLASS FILES

You may use C++, Java, or Python. Your code must be runnable on the Tuffix VM. Sample files have been provided for C++. Each cipher shall be implemented as a separate class. All classes shall implement a common interface defined in `CipherInterface.h` (see sample codes and description below). This class **shall not** be modified. You shall then write a driver program, called `cipher`, that shall use your classes. `CipherInterface` class defines the following abstract methods:

- `CipherInterface():` the default constructor.
- `virtual bool setKey(const string& key):` Sets the key to use for encryption/decryption.
- `virtual string encrypt(const string& plaintext):` encrypts a string of plaintext and returns the cipher text string.
- `virtual string decrypt(const string& ciphertext):` decrypts a string of ciphertext and returns the decrypted plaintext.

You shall then write a separate class for **each** cipher. The class must implement the `CipherInterface` interface. In order to accomplish this, you shall use *inheritance* (you must if you are using C++. If you are using Java, you may, if you want, use Java's interface construct. See http://docs.oracle.com/javase/tutorial/java/concepts/interface.html for details. If you are using Python, please see https://docs.python.org/2/tutorial/classes.html for details):

```
class PlayFair: public CipherInterface
{
        PlayFair() { }

        virtual bool setKey(const string& key)
        {
                ... Playfair implementation ...
        }

        .
        .
```

```
        .
}
```

You are strongly encouraged to implement each cipher class in it's own `.h` and `.cpp` files e.g (or `.java` for Java and `.py` for Python). For Playfair cipher, use `Playfair.h` and `Playfair.h`.

Your driver program, `cipher.cpp`, shall then have the structure similar to the following:

```
...HEADER FILES....
...CODE FOR PARSING THE COMMAND LINE PARAMETERS...
...CODE FOR READING THE FILE...
.
.
.
/* An interface class */
CipherInterface* cipher = NULL;

if(the cipher is Playfair)
{
   /* Assuming Playfair is implemented using class Playfair */
   cipher = new Playfair();
   cipher->setKey(the key);
   .
   .
   .
}
else if(the cipher is Vigen re)
{
      /* Assuming Vigenere is implemented using class Vigenere */
      cipher = new Vigenere();
      cipher->setKey(the key)
      .
      .
      .
}
.
.
.
// Handle the other ciphers
```

The `sample` folder contains files `Playfair.h` and `Playfair.cpp` which declare and define the skeleton for implementing the Playfair cipher, respectively. A minimal driver program file (in C++) has been provided for you: `cipher.cpp`. The driver program can be compiled as described in the next part. You must modify this file to implement the aforementioned functionality.

COMPILING YOUR PROGRAM

Again, Your program must be implemented using C++, Java, or Python and must be runnable on the Tuffix VM. You must also include a `Makefile` which compiles all of your

code when the user types `make` at the command line. A sample `Makefile` has been provided. Simply type `make` at the terminal in order to compile the program.
If you are not sure how to write or modify the `makefile` please check the following resources.

- C++ make files: http://www.delorie.com/djgpp/doc/ug/larger/makefiles.html
- Java make files: http://www.cs.swarthmore.edu/ newhall/unixhelp/javamakefiles.html
- Google "writing Makefiles"
- Ask the instructor (don't be afraid!)

EXTRA CREDIT:

1. Research and learn **Hill Cipher** (not covered in class). Implement the Hill Cipher using the `CipherInterface` interface.

2. Implement the 3-rotor Enigma encryption.

Please note: implementing either (1) or (2) will result in the half of the extra points.

SUBMISSION GUIDELINES:

- This assignment may be completed using C++, Java, Python, or a another language approved by the instructor.
- Please hand in your source code electronically (do not submit `.o`, `.class`, or `.pyc` or executable code) through **TITANIUM**. You must make sure that this code compiles and runs correctly.
- **Only one person within each group should submit.**
- Write a README file (text file, do not submit a .doc file) which contains
  - Names and email addresses of all partners.
  - The programming language you use (e.g. C++ or Java)
  - How to execute your program.
  - Whether you implemented the extra credit.
  - Anything special about your submission that we should take note of.
- Place all your files under one directory with a unique name (such as `p1-[userid]` for assignment 1, e.g. `p1-mgofman1`).
- Tar the contents of this directory using the following command. `tar cvf [directory_name].tar [directory_name]` E.g. `tar -cvf p1-mgofman1.tar p1-mgofman1/`

- Use TITANIUM to upload the tared file you created above.

Grading guideline:

- Program compiles: 5'
- Correct Playfair cipher: 20'
- Correct Vigenre cipher: 20'
- Correct Row Transposition cipher: 20'
- Correct Railfence cipher: 20'
- Correct Caesar cipher: 10'
- README file included: 5'
- BONUS: 8 points
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.