CIS 241 System-Level Programming and Utilities
Project 1 Substitution-Based Cipher

Due: Monday, 2/19/2018 by 10:00 pm

<u>Educational Objectives:</u>

This project is designed to help students gain a better understanding of the following concepts:

1) The make utility in Unix/Linux systems
2) C-functions
3) C-arrays
4) C-array name and pointer
5) File I/O in C
6) Software design

<u>Programming Environment</u>

The project should be implemented with C program language in a Linux environment.

Section 1 A Substitution Cipher

For simplicity, an encryption algorithm is also called a cipher. In old time, most ciphers were based on substitution and permutation. A cipher based on substitution is also named as a substitution cipher. One substitution cipher takes a word as the key and encrypts the upper case 26 letters. It works as follows:

Suppose the key word is FEATHER. The first step is to remove duplicate letters from the key word, yielding string of FEATHR. Then append the other letter of the alphabet in reverse order and you obtain the substitution table as shown below:

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

With the substitution table, encryption can be described as a mapping shown below:

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

Here, the upper row represents the letters in the plain text (original text) and the lower row is the corresponding letter in the cipher text (encrypted text). With the substitution table, both encryption and decryption are a simple table look-up. In other words, encryption is to substitute each letter in the plain text with a corresponding letter in the substitution table as shown above. Decryption is to do the opposite.

In this cipher, the alphabets of plain text, cipher text, and the key word can be defined as a set with the elements if 26 capitalized letters, as shown below:

{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}

Section 2 Tasks

Simply speaking, your tasks for this project is to write a program to implement the aforementioned cipher and make sure your program works correctly.

The tasks for this project are:

1) Design your project.
2) Write a program that implements the substitution cipher as described in the previous section.
3) Write a makefile to compile your project in a Linux/Unix system.
4) Test your program and make it working as specified.

Software requirements:

1) You should divide this cipher into several small parts and implement each part with a function.
2) You should split your implementation into three files. For example:
    a. **functions.h:** where you define all of the functions and include other necessary libraries.
    b. **functions.c:** where you actually implement each function defined in **functions.h**.
    c. **p1.c:** where you have the main function to take the user input, check errors/typos in user input, encrypt and decrypt data by calling functions. Return when completed.
3) Your program should be able to encrypt a plain text that is stored in a given file. Then save the cipher text in another given file.
4) Your program should be able to decrypt a cipher text that is stored in given file. Then save the discovered plain text in another given file.
5) Your program should provide the user with two options: encryption (e), or decryption (d).

For example, assume your executable is named as **p1**. Then the following command will encrypt the contents of **inFile** with the key Key and save the encrypted cipher text in the file **outFile**.

```
p1 e Key inFile outFile
```

On the other hand, the following command will decrypt the contents of **inFile** with the key **Key** and save the discovered plain text in the file **outFile**.

```
p1 d Key inFile outFile
```

Note: You, as a good programmer, cannot assume that users do not make mistakes. Their command line input may have typos or errors. To make it less complex, you can assume that the contents of a file and the Key string are all in upper cases.

How to test your program?

Please try the following to test your program:

1) Generate a few files with the contents over the defined alphabets.
2) Encrypt them, and then decrypt the cipher texts to obtain the discovered plain text.
3) Compare the discovered plain text with the original plain text.
4) If the discovered plain text is identical to the original plain text, most likely your program is correct. Otherwise, your program is incorrect. Please pay attention to "boundary" conditions.
5) Do what you think helpful to convince yourself the correctness of your program.

Section 3 Guideline for Your Project Design

As a general guideline, you should divide your project into a number of different parts. Each part has a manageable and clearly defined task. You can design your project in a way you think is good. This section gives a general guideline for your project design.

Firstly, the mapping shown above is to give you an idea as to how encryption/decryption is done. It does not imply that you have to use two arrays in your program. Actually, one array that holds the substitution table as shown the letters in the lower row will be sufficient. If storage space is not a concern, using two separate arrays will make your program more readable.

Secondly, note that as shown above, letters in the upper row are in alphabetic order, from 'A' to 'Z' (that is, from 65 to 90 in their ASCII coding); thus, each letter's position relative to letter 'A" is the index of the corresponding cipher letter in the lower row. Since the relative position of each original letter can be calculated, an array for the upper row is not needed. Similarly, you can use one array to do decryption – find a way to create such an array in order to perform decryption. In other words, no linear search for either encryption or description is necessary. Calculating the index of a substitute letter, rather than using linear search to find it, allows your program to do the work in the most efficient way. Hence, declare two arrays as follows in your main function:

```
char encrypt[MAXNUM], decrypt[MAXNUM];
```

Also define the following functions and use them in your program. Each function represents a step in the procedure that solves this problem, except for functions removeDuplicates() and targetFound(), which are helpers for other functions.

```
// remove duplicate characters in array word and return the resulting string
char * removeDuplicates(char word []);

// search the first num characters in array charArray for character target
// return a non-zero integer if found, otherwise, return 0
int targetFound(char charArray[], int num, char target);

// initialize the encrypt array with appropriate cipher letters according to the given key
void initializeEncryptArray(char key[], char encrypt[]);

// initialize decrypt array with appropriate substitute letters based on the encrypt array
void initializeDecryptArray(char encrypt[], char decrypt[]);

// process data from the input file and write the result to the output file
// pass the encrypt array to parameter substitute if encryption is intended
// pass the decrypt array to parameter substitute if decryption is intended
void processInput(FILE * inf, FILE * outf, char substitute[]);
```

Since each function above represents a step for your program to do its work, you can develop the program in an incremental fashion; that is, define one function, test it, and if it works correctly, move on to the next function. A minor error may produce a quite different result for a program like this. Incremental development is an effective way to ensure the correctness of a program.

When you test your program, you need to run it to generate an encrypted file from a data file, run your program again with the encrypted file as input to produce a decrypted file, and then compare the decrypted file to the original data file and see if they are the same. Since such a process involves multiple command lines and you have to run them as often as needed, it would be much convenient to define a makefile and use command make to automate your testing process. Test your program twice with the given data files. Use Linux command `diff –s originalTextFile resultingTextFile` to see if they are identical. Also add this command line to your makefile so as to perform your test and compare the result automatically.

Section 4 What Do You Submit?

You can demonstrate your code working correctly with your instructor by the due time. If you are not able to demo, please submit all of your source code, including your program files and makefile by due time through Blackboard.

Grading

If your program works correctly as specified, you will receive 100 on this project. Otherwise, partial credits will be given as described below:

1) Compilation:	30 points
2) Splitting code into 3 files:	10 points
3) removeDuplicates:	10 points
4) targetFound:	10 points
5) initializeEncryptArray:	10 points
6) initializeDecryptArray:	10 points
7) processInput:	10 points
8) main:	10 points

You will receive a 0 on this project if you don't turn in your work or your program does not compile.