CS262 Programming Assignment 1

The Omega I Invariant Cipher

Introduction

Nearly 80 years ago, the world was at war. Germany had invaded Poland, annexed Austria, and France, and was in an air battle against England (The Battle of Britain). During this period, in the English countryside, a manor house called "Bletchley Park" was the central site for British code breakers. There, they worked on cracking the German ciphers known as Enigma and Lorenz. Among the many famous mathematicians who worked there was a man named Alan Turing. Turing later became known as the father of theoretical computer science and artificial intelligence.

For this project, you are going to write a program to encode and decode messages using the Omega I Invariant Cipher (OIIC), a cipher similar to one used by the Enigma machines. This cipher is a method of encrypting alphabetic text using a keyword along with a pseudo-random number generator and polyalphabetic substitution.

Background

A simple polyalphabetic cipher uses a matrix known as a *Tabula Recta* to encode and decode messages. An example matrix can be seen here. This matrix is used in conjunction with a *keyword*, and the letters of this keyword are paired with the letters of the message to be encoded (known as the *plaintext*).

As an example, suppose we have a plaintext message such as "GMU COMPUTER SCIENCE" and we wish to encrypt it. Using the keyword MASON and the given tabula recta, we would first find the column containing the first letter (G) of our message. We would then find the row with the first letter of our keyword (M) and note the letter contained at that row, column intersection (which in this case is S). This is the first character of our encrypted text. Continuing with the next letter of the plaintext (M) and of the keyword (A), we get the next letter of our encrypted text (M). Continuing through the rest of the message and the keyword, we eventually finish the characters of the keyword. At this point, we return to the beginning of the keyword and continue encrypting the plaintext:

Plaintext - GMU COMPUTER SCIENCE
Keyword - MAS ONMASONM ASONMAS
Encrypted text - SMM QBYPMHRD SUWRZCW

So, the resulting encryption of our message is "SMM QBYPMHRD SUWRZCW." The spaces shown are for clarity. Generally, spaces are not used in the encrypted text.

To decrypt the given message, we use the reverse process using the same tabula recta. So, we take the first letter of our keyword (M) and go to the corresponding row. Then, we look in that row for the first letter of our encrypted text, and we see that it is in column G. Therefore, the first letter of our decrypted text is letter G.

The Omega I Invariant Cipher

The OIIC uses a similar method of encrypting and decrypting messages. However, there are a few differences. First, the tabula recta is generated based on a pseudo-random number sequence. The initial alphabet is shuffled, and is used for the first row (A) in the tabula recta. The following rows are then shifted by one character, similar to the shifting of the alphabet in the example tabula recta. Therefore, in addition to the keyword, the person decrypting the message must also know the random number seed in order to generate the proper tabula recta. The second difference between the example and the OIIC is that other symbols (such as punctuation) can be used. This means that punctuation can also be part of the plaintext, and that the encrypted text may also have punctuation symbols (although not in any meaningful position in the text).

For example, suppose we have the following set of symbols that we will use for creating messages:

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .,;!-'?

We can take a message such as:

KERNIGHAN AND RITCHIE DEVELOPED THE C PROGRAMMING LANGUAGE.

and a keyword such as:

GMU COMP SCI

We can generate a tabula recta with the set of symbols by shuffling the order of the symbols (using a random number seed, and calls to a random number generator), and using that as the first row. We then shift the letters in each successive row by one column to make the tabula recta given here. Encrypting the message in the manner described earlier (using the non-shuffled tabula recta), we get the following encrypted text:

,,;!CL1B!CSUHK;RU,1QEC!G5,T'.YB90I'R KWH,L4BKPCUGKTNBL?BXJ2

Notice that there are no spaces to delineate the separate words in the plaintext, but there are space symbols in the encrypted text. The space characters (and the period) from the plaintext have been encoded within the encrypted text.

Specifications

You are to write a menu driven program to encrypt and decrypt messages. The menu will contain the following options:

1. Enter a random number seed to generate the Tabula Recta
2. Enter the Keyword
3. Enter a message to encrypt
4. Enter a message to decrypt
5. Exit the program

User input to the menu will consist of the numbers 1-5. Do not use any additional or different input values. Descriptions of the functionality of each input is as follows:

1. When this option is selected, prompt the user to enter a random number seed. Use this value to seed the random number generator with a call to srandom() (do NOT use srand()). Then use the shuffle routine given in the provided code to randomize the string, and generate the Tabula Recta

2. When this option is selected, prompt the user to enter a Keyword that will be used to encrypt and decrypt messages. Be sure to remove the trailing linefeed character ('\n') that will be found at the end of the user entry. Also, convert all letters in the keyword to capital letters (you may use the C function toupper() - read the manpage).
3. When this option is selected, check to ensure that a Tabula Recta and a Keyword have been entered. If neither (or both) have not been entered during the current run of the program, write an error message to the screen and return to the main menu. Otherwise, prompt the user to enter a plaintext message. Be sure to remove the trailing linefeed character, and convert all letters in the message to capital letters. Once this is done, use the Tabula Recta and the Keyword to encrypt the message in the manner described above. Print the result of this encryption (the encrypted message) to the screen.
4. When this option is selected, check to ensure that a Tabula Recta and a Keyword have been entered. If neither (or both) have not been entered during the current run of the program, write an error message to the screen and return to the main menu. Otherwise, prompt the user to enter an encrypted message. Be sure to remove the trailing linefeed character, and convert all letters in the message to capital letters. Once this is done, use the Tabula Recta and the Keyword to decrypt the message in the manner described above. Print the result of this decryption (the plaintext message) to the screen.
5. When this option is selected, print an appropriate closing message, and exit the program.

Do not prompt for additional inputs for each of the menu choice descriptions. As part of the grading, your code will be run by a script (sample scripts will be provided), and any additional or missing prompts will cause your program to fail to run correctly.

Other Specifications and Additional Information

- The character set of ASCII symbols that can be used for your messages is:
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .,;!-'?
  - Note that there is a space between the 9 and the period.
  - The symbols must be used IN THIS ORDER.
  - The number of characters in the set is 44.

- You may NOT use global variables. All data must be passed to functions using function parameters. However, you MAY use a global constant to represent the character set that is allowed for your messages:
  - const char CHARSET[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .,;!-'?";

- You should create cstring variables to hold your Tabula Recta, Keyword, and messages.
  - The maximum size for a keyword is 20 characters (not including the NULL terminator). Note that you will have to handle the linefeed character on input, so plan the size of your cstring, depending upon how it will be used for input.
  - The size of the Tabula Recta will be 44 rows by 44 columns
  - The maximum size of a message that will be encoded or decoded will be 250 characters (not including the NULL terminator). Note that you will have to handle the linefeed character on input, so plan the size of your cstring, depending upon how it will be used for input.
  - You may assume that input during the grading of your program will not exceed the maximum lengths, and that no unknown characters will be entered (other than lower case letters).
  - Use #define to set the lengths of your cstring variables. Do not declare cstrings using numerical constants (i.e. char Keyword[20];).

- You must use random() and srandom(). Using rand() and srand() will likely cause different results.

- During encoding, be sure to use the original order of the character set to find the column and row indexes in the Tabula Recta. In other words, do not search for the character in the first row of the generated Tabula Recta for the proper index. Use the original order in the CHARSET array.

- You will find the use of the C String library function strchr() to be very helpful for this project. Read the manual page for this function.

- You must include a Makefile to compile your project.
- It is encouraged once you get your program to work correctly, that you post encrypted messages (along with the random number seed and Keyword used to encrypt them) to Piazza so that your fellow students can test their code and see if they can decode your messages.

Some Helpful Code and Other Hints:
A function that can be used to find the position of a specific character within a character string is:

```
/*
 *  Function: GetPosition
 *
 *  Description: Finds the position of the first instance of a given character within a given cstring.
 *
 *  Parameters: const char *str   - Pointer to the first element of the cstring in which to search
 *              const char c      - The character to find within the input cstring
 *
 *  Returns: Non-negative position value of the first instance of c in str
 *           If c is not found in str, returns -1
 */
int GetPosition(const char *str, const char c)
{
  int retVal = -1;
  char *pos = strchr(str, c);

  if (pos != NULL)
  {
    retVal = pos - str;
  }

  return retVal;
}
```

This function can be used as follows:

```
char str[] = "Hello there!";
int X = GetPosition(myString, 'o');
```

X should be equal to 4, since the first occurrence of the character 'o' can be found at str[4].

The shuffle routine:
Use the following algorithm (the Fisher-Yates shuffle) to shuffle the initial line of your Tabla Recta. You may use whatever variable names fit with your code.
/*

```
 * charSet is a character array initialized with the characters in the set
 */
for (i = SizeOfCharacterSet - 1; i > 0; i--)
{
    char temp;
    idx = random()%(i+1);
    temp = charSet[idx];
    charSet[idx] = charSet[i];
    charSet[i] = temp;
}
/*
 * The character set is now in a random order in the charSet array
 */
```

You may find it helpful to print the generated Tabla Recta during debugging to ensure that it looks correct. You don't need to print it in your submitted program.

Read the Wikipedia page for the Fisher-Yates shuffle (click the link above) so that you understand what is going on. Get in the habit of understanding all code that you have in your program, even if it is code that you have not written.

Testing and Submitting:
Test your program on Zeus to make sure it compiles and runs properly in various types of cases. Perform appropriate error checking.

Create a tarfile that contains your source code and Makefile, and all other files used for your project.

Submit this tarfile to Blackboard no later than 11:59 p.m., March 3, 2018.