/

Assignment 1: **OS**
Due: **R.H.**

Purpose: Multi-Threading using pthread

Problem: A message is composed of three parts. Each part may have a different length and it is either an encrypted text or a plain text. The order of parts within the message is unknown. The number of asterisks at the beginning of each part determines the part number. For example part 2 within the message starts with two asterisks. The goal is to encrypt the plain parts and decipher the encrypted parts separately using threads.

To meet the goal, write a C++ program that creates a thread named *sifter*. Sifter asks user to enter a message (a character string made up of three parts) as input. If the message is invalid then the program gives the user only three chances to enter a valid message. (From view point of the sifter, a valid message is the one with three no-null parts. The thread sifter then creates another thread *decoder* that takes the message and divides it into its three parts. Part1, Part2, and Part3 (without their preceding asterisks) are fed to three new threads of *fence*, *hill*, and *valley,* respectively. The new threads are generated by decoder. Details of these threads are provided below.

Fence Thread

This thread uses the *rail fence algorithm* to decrypt a given encrypted text. The thread divides the text into two sections of Section_1 and Section_2. Section_1 carries all the numerical characters in front of the text and Section_2 carries the remaining characters in the text. An example of the encrypted text is given below:

43125678812ttnaAptMTSUOaodwcoIXknLypETZ

Section_1 = "43125678812"
Section_2 = "ttnaAptMTSUOaodwcoIXknLypETZ"

Section_1 is searched for the key, K, which is a set of *non-repeated valid* numerical characters. To find K, Let us assume that the first repeated numerical character that has been met for the first time in position P is repeated in position Q within Section_1. If Q = P+1 then, K includes all numerical characters from the beginning of Section_1 to the position of P-1; Otherwise, K includes all the numerical characters from the beginning of Section_1 to the position of Q-1. For the above example, the first repeated numerical character is 8 that for the first time it was met at the position P = 8 and for the second time at the position Q = 9. Since Q = P+1, K includes all the numerical characters from the beginning of the Section_1 to position P-1 =7 (i.e. K = 4312567.) (In the case that the numerical characters of 93456312, for example, is at the beginning of the message, then P = 2, Q =6, and Q ≠ P+1; Thus, K = 93456.)

To check the validity of the identified non-repeated numerical character, K, let us assume that N =Length(K). K is valid if it is composed of numerical characters from 1 to N (in any random order.) For example, K = 4312567, is valid because for K, the value of N = 7 and all numerical characters from 1 to 7 (including 1 and 7) are in K.

For the same reason K = 93456 is not valid. If K is empty or invalid then, the encrypted text is considered an erroneous one and the decryption process cannot be completed.

The text in Section$_2$ is partitioned into groups such that each group is j characters long, where j = Length(Section$_2$)/N. In the above example N = Legth(S) =7 and length of Section$_2$ is 28, thus, j = 28/7=4. (The length is always divisible by N; otherwise, it is considered an erroneous encrypted text.) As a result, Section$_2$ is partitioned such that each group has 4 characters in it:

ttna AptM TSUO aodw coIX knLy pETZ

Rearrange the groups vertically such that each group becomes one column and collectively they make a matrix with N columns.

1	2	3	4	5	6	7
t	A	T	a	c	k	p
t	P	S	o	o	n	E
n	t	U	d	I	L	T
a	M	O	w	X	y	Z

Rearrange the columns using the digits in S = 4312567 and create a new matrix:

Column 4 of the old matrix becomes column 1 of the new matrix because <u>4</u> is the <u>first</u> digit in S.
Column 3 becomes column 2 of the new matrix because <u>3</u> is the <u>second</u> digit in S.
Following the same procedure,
Column 2 becomes column 4 of the new matrix
Column 6 becomes column 6 of the new matrix
Column 1 becomes column 3 of the new matrix
Column 5 becomes column 5 of the new matrix
Column 7 becomes column 7 of the new matrix

New Matrix

1	2	3	4	5	6	7
a	T	t	A	c	k	p
o	S	t	P	o	n	E
d	U	n	t	I	l	T
w	O	a	M	X	y	Z

The rows of the new matrix collectively make the decrypted message of the above example: "attack postponed until two am xyz".

Hill Thread

This thread uses the *hill encryption algorithm* to encrypt a given plain text. The thread breaks the plain text into three sections of Section$_1$, Section$_2$, and Section$_3$. Section$_1$ carries the first token of the plain text. Section$_2$ carries the second token of the plain text, and section$_3$ carries the rest of the tokens in the plain text. If any of the following conditions exists, then input is erroneous and a proper error message is printed in lieu of encryption of the plain text:

a. Section$_1$ is longer than one character.
b. The character in Section$_1$ is not either '2' or '3'.
c. Section$_2$ includes characters other than alphabet characters.

d. The length of the token in $Section_2$ is not a multiplicative of 2 or 3 when the character in $Section_1$ is '2' or '3', respectively.
e. Each token in $section_3$ includes characters other than numerical characters
f. The number of tokens in $section_3$ is less than 4 when the character in $Section_1$ is '2'
g. The number of tokens in $section_3$ is less than 9 when the character in $Section_1$ is '3'

Example: The input text arrived at the hill thread is:
"3 paymoremoney 17 17 5 21 18 21 2 2 19 0 15"

$Section_1$ is the first token of input: "3",
$Section_2$ is the second token of input: "paymoremoney". This is the plaintext that will be encrypted. As a general rule, spaces between words of the plaintext are ignored and the length of the token is 12 which is a multiplicative of 3 (the token in $Section_1$.)
$Section_3$ is the remaining 11 tokens of input: "17 17 5 21 18 21 2 2 19 0 15".

The thread encrypts the plain text in $Section_2$ three letters at a time. The number of letters that can be encrypted at a time is given by $Section_1$. The encryption is done by use of a key, K, which is a square matrix of 3 X 3. Again the number of rows and columns for K are indicated by the digit in $Section_1$. This matrix is field by the first 9 tokens in $Section_3$. (*Reminder*: In the case that digit in $Section_1$ is 2, the encryption takes place 2 characters at a time, K is a 2 X 2 matrix, and the first 4 tokens of $Section_3$ make the K matrix.)

The thread creates a matrix 1 X 3, named P, out of the positions of the first 3 characters of $section_2$ in the alphabet letters. (Letter A has the position of zero and 'z' has the position of 25 in the alphabet letters.)
For our example, the first 3 characters in $section_1$ are "pay" and **P**= [15 0 24]. Thread also creates the key. For our example, the key matrix is: $\mathbf{K} = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$

The first three characters are encrypted and replaced by new three characters that their positions in the alphabet letters are calculated using the following formula:

$$\mathbf{C} = (\mathbf{PK}) \bmod (26)$$

$$C=[15\ \ 0\ \ 24]\begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix} mod(26)= [303\ \ 303\ \ 531]\ mod(26)= [17\ \ 17\ \ 11]$$

The three letters of "pay" are replaced by the three letters of "RRL". The same process continues for every three characters of $Section_1$ until Section1 is exhausted. The matrix K will not change but P will change. the ciphertext for the entire plaintext is RRLMWBKASPDH.

Valley Thread

This thread uses the reverse process that took place in Hill thread to decrypt a ciphered text. The thread behaves exactly like the hill thread in breaking the part 3 of the input message into three sections of $Section_1$, $Section_2$, and $Section_3$. In Valley thread the $Section_1$ and $Section_3$ play the same role as they played in the hill thread. $Section_2$ carries the ciphered text that needs to be deciphered. The erroneous input are identified by the same rules presented in the hill thread.

Example: The input text arrived at the Valley thread is:

3 RRLMWBKASPDH 17 17 5 21 18 21 2 2 19 3 8

$Section_1$ is the first token of input: "3",

$Section_2$ is the second token of input: "RRLMWBKASPDH". This is the ciphered text that will be decrypted. The length of the token is 12 which is a multiplicative of 3 (the token in $Section_1$.)

$Section_3$ is the remaining 11 tokens of input: "17 17 5 21 18 21 2 2 19 0 15".

The thread decrypts the ciphered text in $Section_2$ three letters at a time. The number of letters that can be decrypted at a time is given by $Section_1$. The decryption is done by use of a key, K, which is a square matrix of 3 X 3. Again the number of rows and columns for K are indicated by the digit in $Section_1$. This matrix is field by the first 9 tokens in $Section_3$. (*Reminder*: In the case that digit in $Section_1$ is 2, the decryption takes place 2 characters at a time, K is a 2 X 2 matrix, and the first 4 tokens of $Section_3$ make the K matrix.)

For the above example, key matrix is $\mathbf{K} = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$

The Valley thread builds the inverse of K (i.e. K^{-1}) and decrypts the ciphered text of $Section_2$ three characters at a time by using the following formula:

$P = (CK^{-1}) \bmod (26)$

Where, C is a 1 X 3 matrix and it is created out of the positions of the first 3 characters of $Section_2$ in the alphabet letters. For our example, the first 3 characters are "RRL" and $C = [17\ \ 17\ \ 11]$. The inverse matrix is $K^{-1} = \begin{bmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{bmatrix}$. (See the attachment-A for building the inverse matrix of K mod m.)

If the inverse matrix cannot be built

$$P = [17\ \ 17\ \ 11] \begin{bmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{bmatrix} \text{mod } 26 = [587\ \ 442\ \ 544]\ mod\ 26 = \ [15\ \ 0\ \ 24]$$

The three letters of "RRL" are replaced by the three letters of "PAY". The process is repeated for the next three letters in the ciphered text until the text is exhausted.

Notes:

1. Attachment-B includes codes for creating a thread.
2. For this project following libraries are needed:
   fstream.h, iostream.h, stdio.h, pthread.h, stdlib.h, and unistd.h
3. To create a thread, the pthread_create(. . .) is used. The details of this system call are attached.
4. When a thread is finished, for the control to go back to the parent thread, the system call of pthread_join(. . .) is used. The details of this system call are attached.
5. Both abovementioned system calls return integer value of zero for a successful execution.
6. Use the following commands to compile and link your program:
   ```
   g++ -D_REENTRANT sourcefileName.cpp –o sourcefileName –lpthread
   ```
7. Use the following command to run your program:
   ```
   ./sourcefileName
   ```
8. Turn in a hardcopy of your program along with the output.
9. This assignment requires a demonstration of your program behavior. During the demonstration, your program will ask the user to enter a message and then takes proper actions on the three parts of the valid message output. Your program cannot end as soon as the first input message is processed. It must ask for the next input message from the user. Your program must continue until the user wants to quit by entering a specific input string. *If your program does not support this choice you lose 50% of score for assignment.*

Samples of input character strings (or messages)

```
***3 rrlmwbkaspdh 17 17 5 21 18 21 2 2 19 12 *123555eu5eotsya**3 GoodMorningJohn 3 2 1 20 15 4 10 22 3
**2 Global Warming 12 4 5 23 18 13 4 *** 2 RFRWYQ  5  8  17 3 4 9*314277810AOSRSDKYPWIKSRO
**ACDUJF  1   4   6   12***MNKLHY 1 2 3 4 5 6 7 8 9*M FGCV
**   KMLN  12  3  4  17 *** GHL 9 9 8 8 7 7 6   a6    5 * P TRBKHYT
*B  PHHW PH DIVHU WKH WRJD SDUWB
*2  MNKK LLSGQW
**GG  CI   MOQN   9   2   1    15 ***ST  TNJVQLAIVDIG  21  20   24   21  24   9  -5  21  21
** XNPD   5   8   17  3 ***KL  LNSHDLEWMTRW 4  9  15  15  17  6  24  0  17*D   GJFZNKZQDITSJ
**GGMLN  12  3  4  b7 *** GHL 9 9 8 8 7 7 6   16    5 * P GRBKHYT
**   BBLN  12  3  4  17 *** GHL 9 9 8 8 7 7 6   6    5 * CRKLMNBKHYT
**     MWUULNGXAP 12 15  7  6***      GHYTRKL MNP 1  0  0  -1  2  3   12*CDM UYKOOKPIRQQN
*zg ZHCHUUHOPXKPYAF***GHEDQHQQ 9  11  4  5 ** ABHJUTVOP 5  -2  3  4  12  32  1  0  3
***ABC 5  6   7  8  10  1  4  8 11*K SAAPXGOW**MKPW 10  2  12  1*J  RZZOWFNV
```

During your demonstration the input messages are not limited to the above ones and I will give you a set of new inputs.

ATTACHMENT-A

Background(1): **Modulus operation**

b mod n = r such that b = nq +r

12 mod 5= 2 // mod result must be positive

-b mod n = -r Since mod result must be positive; The final mod result is –r+n

-12 mod 5 = -2 -2 + 5 = 3 Therefore -12 mod 5 = 3

Background(2): **modular multiplicative inverse**

b^{-1} mod *n* denotes the modular multiplicative inverse of b.

A modular multiplicative inverse of an integer b is an integer x such that the bx mod n =1. The integer x exist if b and n are **relatively prime**. The integer b and integer n are relatively prime, if they have no common positive factors beside 1.

Example:

23^{-1} mod 27 //This is not the same as (1/23) mod 27.

We are looking for an integer number like x such that 23(x) mod 27 =1

Before finding x, let us conduct the relatively prime checking.

Factors for 23 are 1, 23 Factors for 27 are 1, 3, 9, 27

They have only one common factor and it is 1. Therefore, 23 and 27 are relatively prime.

Let us find the value for x. To do so:

27 = 23(?) + ?
27 = 23(1) + 4
23 = 4(?) + ?
23 = 4(5) + 3
4 = 3(?) + ?
4 = 3(1) + 1

Let us rewrite the above equations and from bottom to top

4 = 3(1) + 1
23 = 4(5) + 3
27 = 23(1) + 4

---

4 - 3(1) = 1	→	(a)	4 + 3(-1) =1
23 = 4(5) + 3	→	(b)	23 + 4(-5) =3
27 = 23(1) + 4	→	(c)	27 + 23(-1) =4

Replace the left side of (b) for 3 in (a)

(d) 4 + [23 +4(-5)](-1) = 1
(e) 4 -23 +4(+5) = 1
(f) 4(+6) – 23 = 1

Replace the left side of (c) for 4 in (f)

(g) [27+23(-1)](+6) - 23 = 1
(h) 162 + 23(-6) -23 = 1
(i) 162 +23(-6) +23(-1) =1
(j) 162 + 23(-7) =1

(k) 162 mod 27 = 0

(l) 23(-7) mod 27 = 1

We need to change – 7 to a number between 0 and 26 because we deal with the mod 27.

-7 +27 = 20

(k) 23(20) mod 27 =1

Therefore **x = 20.**

To check the correctness of value of x, the 23*20 mod 27 must be 1.

23*20 = 560 and 560 mod 27 is equal to 1.

As a practice complete 23^{-1} mod 26. *The answer is 17.*

Calculation of the K^{-1}mod n.

A matrix is given, **K** = $\begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$ Calculate the K^{-1}mod 26.

Step a) calculate determinant of matrix K = $\begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$

$(\det K) = 17\begin{pmatrix} 18 & 21 \\ 2 & 19 \end{pmatrix} - 17\begin{pmatrix} 21 & 21 \\ 2 & 19 \end{pmatrix} + 5\begin{pmatrix} 21 & 18 \\ 2 & 2 \end{pmatrix}$

=17(18*19 -21*2) -17(21*19 -21*2) +5(21*2 -18*2) = -939

for n = 27,

z = (det K) mod 27 = -21 z = -21 +27 = 6.

for n = 26 //as another example

z = (det K) mode 26 = -3 g = -3 + 26 = 23

Step b) Calculate $(z)^{-1}$ mod 27 where z = 6

//z= 6 was obtained in Step a and for modulus n = 27.

$(6)^{-1}$ mod 27

The integer values of 6 and 27 are not relatively prime because they have the common factor of 3. Therefore, the modular multiplicative inverse of 6 does not exist and K^{-1} mode 27 cannot be created

---

Let us use z =23 as another example. // obtained in Step a and for modulus n = 26.

$(23)^{-1}$ mod 26

The integer values of 23 and 26 are relatively prime.
Therefore we can find the value of x.

Following the same process in section 2 delivers the value of 17 for x
Thus, h = $(\det K)^{-1}$ mod 26 = 17

Step c) The element in row i and column j in K^{-1} is calculated using the following formula:

$[K^{-1}]_{i,j} = h(-1)^{i+j} (D_{j,i}) \bmod m$

where, $(D_{j,i})$is the sub-determinant formed by deleting the jth row and the ith column of K and h is an integer value obtained in Step b. m is the modulus.

row i = 1, column j= 1, h =17, m =26, and $K = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$

$[K^{-1}]_{1,1} = 17\ (-1)^{1+1} \begin{bmatrix} 18 & 21 \\ 2 & 19 \end{bmatrix} = 17(18*19 - 21*2) \bmod 26 = 5100 \bmod 26 = 4$

i = 1, j= 2

$[K^{-1}]_{1,2} = 17\ (-1)^{1+2} \begin{bmatrix} 17 & 5 \\ 2 & 19 \end{bmatrix} = -17(17*19 - 5*2) \bmod 26 = -5321 \bmod 26 = -17+26 = 9$

i = 1, j= 3

$[K^{-1}]_{1,3} = 17\ (-1)^{1+3} \begin{bmatrix} 17 & 5 \\ 18 & 21 \end{bmatrix} = 17(17*21 - 5*18) \bmod 26 = 4539 \bmod 26 = 15$

i = 2, j= 1

$[K^{-1}]_{2,1} = 17\ (-1)^{2+1} \begin{bmatrix} 21 & 21 \\ 2 & 19 \end{bmatrix} = -17(21*19 - 21*2) \bmod 26 = -6069 \bmod 26 = 15$

i = 2, j= 2

$[K^{-1}]_{2,2} = 17\ (-1)^{2+2} \begin{bmatrix} 17 & 5 \\ 2 & 19 \end{bmatrix} = 17(17*19 - 5*2) \bmod 26 = 5321 \bmod 26 = 17$

Following the same procedure, the $K^{-1} = \begin{bmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{bmatrix}$

ATTACHMENT-B

```
#include <pthread.h>
void  *part1 (char *section1);
..
..

int main( ) {
            pthread_t  thread_id
            void*     threadPointer
            int result;
            char * message = "today is a good day";
            result = pthread_create(&thread_id, null, part1, (void*) message);
                        // &thread_id :     the pointer to the thread ID
                        // null:            Thread attributes
                        // part1:           Name of the thread
                        // (void*)message the message that will be sent to the thread.
            If result = 0 then everything is OK;

            result = pyhread_join(thread_id, &threadPointer);
                        // &thread_id :    the pointer to the thread ID;
                        // &threadPointer:  The return value of thread1 is stored in the location pointed to by
                                            threadpointer
                        // &threadPointer: can also be null

.
.
}

void *part1(char * section1)
{
. .
. .
pthread_exit("test");
```