* Your program can't Quit
Caesar cipher Part 1

Characters * + Alpha letters Only +#'s

Assignment 1: **OS**
Due: 10/04/16 **R.H.**

Purpose: Multi-Threading using pthread

Problem: A message has been broken into three parts. The length of each part may be different from the rest. Each part has been encrypted by a different method of encryption. The three parts along with the necessary information have been concatenated again to make one long ciphered message. The order of the parts within the ciphered 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 decipher each part separately using threads. To meet the goal, write a C++ program that creates a thread named *sifter*. Sifter asks user to enter a ciphered message (a character string) 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 in which the three no-null parts exists. The thread sifter then creates another thread *decoder* that takes the message and divides it into its three parts. Each part (without its preceding asterisks) is fed to three new threads generated by decoder that are named *substitute*, *hill*, and p*innacol*. Details of the three threads are provided below.

* **Substitute Thread**

This thread uses the *replacement algorithm* for deciphering the message. The details of the replacement algorithm are as follows: The first character of the message delivered to the thread must be an alphabet letter and its position among the alphabet letters is used in the following formula to calculate N:

N = (The position of the letter among the alphabet letters) mod (10) +2.

(We assume that letter "a" (or "A") is in position zero and letter z (or Z) is in position 25. And also, we assume that alphabet letters make a circle. That is the letter after Z is A.)

After the calculation of N, the first token of the message is removed and the rest of the message is considered as the actual ciphered message. The deciphering of the message starts with replacing every letter in the ciphered message by a letter that is sitting N characters before the ciphered letter in the alphabet. For example, if the ciphered letter is G and N = 3 then, it is replaced by D.

If either of the following conditions arises, then a proper error message is printed in lieu of the deciphered message:

- The first character of the message used to calculate the value of N is not an alphabet letter.
- The ciphered message does not include non-alphabet character.

As an example , a typical string of input to the substitute thread is: J RZZOWFNY

The first character is "J" and its position among the alphabet characters is 9 and it delivers the value of 11 for N. (9 %10 + 2= 11.) The actual ciphered message is RZZOWFNV which is deciphered into GOODLUCK.

Hill Thread

This thread breaks its message into two sections: section 1 and section2. Section1 carries the first token of the message and section 2 carries the rest of the tokens. If either of the following conditions arises, then a proper error message is printed in lieu of the deciphered message:

.Section 1 includes characters other than alphabet characters

- The number of characters in the section 1 is not even
- Either the number of tokens in section 2 is not 4 or they are not made up of only digits.

The thread creates a matrix 2 X 1, named C, out of the positions of the first 2 characters of section 1 in the alphabet letters. It also creates a square matrix of 2 X 2, named **S**, out of the tokens in section 2. The two characters will be deciphered by new two characters that their positions in the alphabet letters are calculated using the following formula:

$\mathbf{C'} = (\mathbf{SC}) \bmod (26)$

Example: The message arrived at the hill thread is:
"PQLG 4 3 2 7". The message will be divided into section1 = PQLG and section2 = 4 3 2 7. Both sections are valid. The first two characters of section1 are PQ that generates the 2 X 1 matrix of $\mathbf{C} = \begin{bmatrix} 15 \\ 16 \end{bmatrix}$ and 2 X 2 matrix of $\mathbf{S} = \begin{bmatrix} 4 & 3 \\ 2 & 7 \end{bmatrix}$.

The matrix $\mathbf{C'} = \mathbf{SC} \bmod 26 = \begin{bmatrix} 4 & 3 \\ 2 & 7 \end{bmatrix} \begin{bmatrix} 15 \\ 16 \end{bmatrix} \bmod 26 = \begin{bmatrix} 108 \\ 142 \end{bmatrix} \bmod 26 = \begin{bmatrix} 4 \\ 12 \end{bmatrix}$

Therefore, PQ are translated into E and M (the letters located in positions 4 and 12 among the alphabet letters.)

The above process is repeated for the next two characters in section1 (i.e. LG). Matrix **C** changes but matrix **S** remains the same. Since the matrix S is a 2 X 2 matrix then, the section1 is always deciphered 2 characters at a time.

Pinnacol Thread

This thread acts exactly the same as *hill* thread, except for the fact that it decipherers the message 3 characters at a time. That is, the message is broken into two sections: section 1 and section2. Section1 carries the first token of the message and section 2 carries the rest of the tokens. If either of the following conditions arises, then a proper error message is printed in lieu of the deciphered message:

.Section 1 includes characters other than alphabet characters

use long int

- The number of characters in the section 1 is not divisible to 3
- Either the number of tokens in section 2 is not 9 or they are not made up of only digits.

We show the behavior of the pinnacol thread which is similar with the behavior of the hill thread through an example.
Let us assume the received message is: ABCDEF 1 2 3 5 2 1 0 2 4. The two sections of the message are:
Section1: ABCDEF
Section2: 1 2 3 5 2 1 0 2 4

In the first iteration we pick the first three characters of ABC for which the matrix

$$\mathbf{C} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \text{ and 3 X 3 matrix of } \mathbf{S} = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 2 & 1 \\ 0 & 2 & 4 \end{bmatrix}$$

$$\text{The matrix } \mathbf{C'} = \mathbf{SC} \bmod 26 = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 2 & 1 \\ 0 & 2 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \bmod 26 = \begin{bmatrix} 8 \\ 4 \\ 10 \end{bmatrix}$$

Therefore, ABC are translated into IEK (the letters located in positions 8, 4 and 10 among the alphabet letters.)

The above process is repeated for the next three characters in section1 (i.e. DEF). Matrix **C** changes but matrix **S** remains the same. Since the matrix S is a 3 X 3 matrix then, the section1 is deciphered 3 characters at a time.

Notes:

1. For this project following libraries are needed:
   fstream.h, iostream.h, stdio.h, pthread.h, stdlib.h, and unistd.h
2. To create a thread, the pthread_create(. . .) is used. The details of this system call are attached.
3. 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.
4. Both abovementioned system calls return integer value of zero for a successful execution.
5. Use Astro for this project. You may use any editor available on Astro. Use the following commands to compile and link your program:
   g++ -D_REENTRANT sourcefileName.cpp –o sourcefileName –lpthread
6. Use the following command to run your program:
   ./sourcefileName
7. Turn in a hardcopy of your program along with the output.
8. This assignment requires a demonstration of your program behavior. During the demonstration, your program will ask the user to enter a ciphered message and generates the deciphered version of the message as output. Your program cannot end as soon as the

first input string 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.

Samples of input character strings (or messages)

```
**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 strings are not limited to the above ones.

NAME

pthread_create - create a new thread

SYNOPSIS

```
#include <pthread.h>

int pthread_create(pthread_t * thread, pthread_attr_t *
attr, void * (*start_routine)(void *), void * arg);
```

DESCRIPTION

pthread_create creates a new thread of control that executes concurrently with the calling thread. The new thread applies the function start_routine passing it arg as first argument. The new thread terminates either explicitly, by calling pthread_exit(3), or implicitly, by returning from the start_routine function. The latter case is equivalent to calling pthread_exit(3) with the result returned by start_routine as exit code.

The attr argument specifies thread attributes to be applied to the new thread. See pthread_attr_init(3) for a complete list of thread attributes. The attr argument can also be NULL, in which case default attributes are used: the created thread is joinable (not detached) and has default (non real-time) scheduling policy.

RETURN VALUE

On success, the identifier of the newly created thread is stored in the location pointed by the thread argument, and a 0 is returned. On error, a non-zero error code is returned.

PTHREAD_JOIN(3)

NAME

pthread_join - wait for termination of another thread

SYNOPSIS

```
#include <pthread.h>

int pthread_join(pthread_t th, void **thread_return);
```

DESCRIPTION

pthread_join suspends the execution of the calling thread until the thread identified by th terminates, either by calling pthread_exit(3) or by being cancelled.

If thread_return is not NULL, the return value of th is stored in the location pointed to by thread_return. The return value of th is either the argument it gave to pthread_exit(3), or PTHREAD_CANCELED if th was cancelled.

The joined thread th must be in the joinable state: it must not have been detached using pthread_detach(3) or the PTHREAD_CREATE_DETACHED attribute to pthread_create(3).

When a joinable thread terminates, its memory resources (thread descriptor and stack) are not deallocated until another thread performs pthread_join on it. Therefore, pthread_join must be called once for each joinable thread created to avoid memory leaks.

At most one thread can wait for the termination of a given thread. Calling pthread_join on a thread th on which another thread is already waiting for termination returns an error.

CANCELLATION

pthread_join is a cancellation point. If a thread is canceled while suspended in pthread_join, the thread execution resumes immediately and the cancellation is executed without waiting for the th thread to terminate. If cancellation occurs during pthread_join, the th thread remains not joined.