

Write a LINUX pthreads program to decrypt several messages. The messages will be of the following form:

```
Code Key <return>
Message Text<return>
```

The code key consists of a sequence of digits from 0 through 9. Each one of these corresponds to an encryption algorithm. Your program will have eleven threads, ten decryption threads and one master thread. The master thread will read a file of messages, and farm them out to various threads. The first action taken by the master thread will be to read ALL (that is ALL) messages and place them in its own first-in first-out queue. (Every thread will have a first-in first-out queue.) The individual messages will be stored in data structures of the form:

```
class Message
{
public:
    Message * Next;
    char * Key;
    char * Data;
};
```

A message key of 1232197, for example, means that the message was first encrypted using algorithm 1 then algorithm 2, then algorithm 3, and so forth. It must be decrypted in *REVERSE ORDER* from the way it was encrypted. The master thread will go through its queue of messages and pass them to the thread that corresponds to the LAST DIGIT in the key string.

To keep things simple, we will assume that each algorithm is a letter-substitution algorithm with a different letter pattern. Here are the patterns. The first line is before encryption, the second line is after. Letters numbers spaces and other characters are passed through unchanged. It's OK for a letter to get encrypted into itself

0. abcdefghijklmnopqrstuvwxyz  
kwxjfhaurmepyv tcsobgnlizdq
1. abcdefghijklmnopqrstuvwxyz  
oziwacplqdunsymrgtvkfxehjb
2. abcdefghijklmnopqrstuvwxyz  
cjkvdazuphqewgfslyionmbrxt
3. abcdefghijklmnopqrstuvwxyz  
zhxdugsvqimyp lkaofenrcbjwt
4. abcdefghijklmnopqrstuvwxyz  
yzhnxoitrcwcegp bqjmdkvalsfu

5. abcdefghijklmnopqrstuvwxyz  
ikrapnjqvhozguwtclxbsdmezf
6. abcdefghijklmnopqrstuvwxyz  
buclqrmigatedfpkxnvhwzjosy
7. abcdefghijklmnopqrstuvwxyz  
cyrqkguoldxmfsjvbwzwpinahte
8. abcdefghijklmnopqrstuvwxyz  
davhoumpwtcyrngbzjkeqfsixl
9. abcdefghijklmnopqrstuvwxyz  
iusefmnxjlkzpwcbbygahovdrtq

Use a first-in first-out queue for each thread encryption thread and another first-in first-out queue for the master thread. The queues will contain Message objects. When a thread is not busy, it will wait on a condition variable. When a decryption thread runs, it will decrypt the message, and delete the **LAST** character from the encryption key. It will then place the message in the queue for the master thread. Message queues must be protected with mutex variables. When a message is placed in a queue it is necessary to check to see if the associated thread is waiting on a condition variable. If it is, then it must be awakened. All threads wait on condition variables when their queues are empty. When a message is completely decrypted (the key must now be the null string) it is placed in the output queue. When all messages are placed in the output queue, the master thread will print them, and all threads will terminate.

Turn in a **PRINTOUT** of all code.

**WARNING:** Don't get clever and find some new way to decrypt the messages. **Do it the way I've described here.**

The file of messages is here:

<http://cs.ecs.baylor.edu/~maurer/Messages.txt>

The decrypted messages are in this file:

<http://cs.ecs.baylor.edu/~maurer/Answers.txt>