ECE2036	**FALL Semester, 2018**
Lab 4 – Cryptic Connections	
Assigned: *Oct. 10, 2018*	Due: Oct. 19, 2018

Warning: This lab is for educational purposes only. There are countries where it is illegal to send encrypted text and email messages, so please be careful.

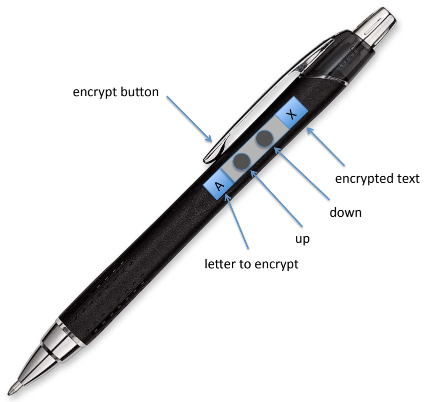
Directions: To complete this lab, you will need to demo your working *mbed* projects to the TAs during their office hours. Once you get all your signoffs from the TA, please take a picture (or scan this page) and upload the image (or pdf) to canvas on assignment 4.

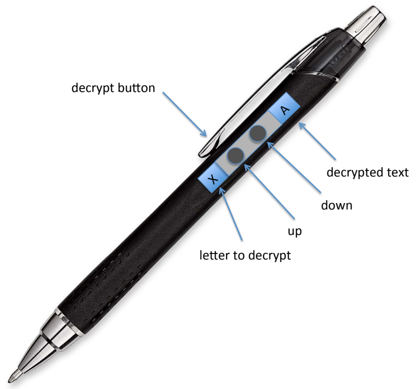
Student Name: ______________________________ Final Turn-In Date: _______________

Item	TA Signoff
Part 1. (20%) Create One-Time Pad	
Part 2. (30%) Create Encrypting Pen	
Part 3. (30%) Create Decrypting Pen	
(20%) Student Used Object Oriented Design (OOD)	
All Checkoffs Completed on or before Oct 16 (+3% extra)	
All Checkoffs Completed on Oct 17 (+2% extra)	
All Checkoffs Completed on Oct 18 (+1% extra)	
All Checkoffs Completed on Oct 19 (Final Due Date!)	
Late Checkoffs 20% off per day (Sat &Sun count as one day)	

In this lab, you will continue to explore an object oriented design (OOD) philosophy. In the last lab, I briefly walked you through the design process and told you the objects to create. In this lab, you will apply what you have learned to explore OOD on your own. When you demonstrate to the TAs you will show them the C++ classes that you created.

The goal of this mbed lab is to create a portable system that can be used to encrypt and decrypt text messages between two people. The hypothetical idea is that the final design of your system would be incorporated into a set of special pens – let's call them cipher pens -- that look like the following pictures below. One pen would encrypt a letter and the second pen would decrypt the letter as seen in the figures below. The encryption algorithm that you will use is based upon one-time pad encryption.





Encryption Pen

Decryption Pen

You will build and test the prototype for these pens using your mbed system that you created in an earlier lab.

Encryption Algorithm

"One-time pad" encryption is unbreakable assuming that you use it correctly. To do so, the message sender and the receiver must have the same sequence of randomly selected upper case letters. Spies would carry around a pad of random encryption key letters that they could use to encrypt messages to send back to their headquarters. In this lab, the idea will be that these pads have been stored on a microSD card inside a set of cipher pen. Each encryption pen will have a corresponding decryption pen with the *same* one-time pad stored internally.

For this encryption, each letter is assigned a ***number code*** from 0 to 25 (i.e. A=0; B=1; … Z=25). Worry ONLY about upper case letters in your encryption scheme with NO SPACES or special characters.

For "one-time pad" encryption you need to generate a random sequence of letters that are the **encryption key** (see row 3 in below table). First, each **encryption letter key number code** (row 4) is added to the **message letter number code** (row 2). The resulting addition is then divided by 26 and the remainder of this division corresponds to the number code (as seen in row 5) for the **cipher text** (as seen in row 6).

E	N	C	R	Y	P	T	T	H	I	S	M	E	S	S	A	G	E	←message
4	13	2	17	24	15	19	19	7	8	18	12	4	18	18	0	6	4	←num code
L	R	F	K	Q	Y	U	Q	F	J	K	X	Y	Q	V	N	R	T	←rand key
11	17	5	10	16	24	20	16	5	9	10	23	24	16	21	13	17	19	←num code
15	4	7	1	14	13	13	9	12	17	2	9	2	8	13	13	23	23	←key + message modulus 26
P	E	H	B	O	N	N	J	M	R	C	J	C	I	N	N	X	X	cipher Text

Decryption Algorithm

To decrypt the message you basically subtract the number code of the SAME cipher key from the number code of the encrypted text (with some additional algorithmic features that are mentioned below). This is both the strength and the weakness of this encryption scheme. The sender and receiver *must* have the same list of random cipher keys. In this way your cipher pens must be completely matched with the same list of random cipher keys. You must also keep track of where you are in the cipher key list!

To illustrate this algorithm consider the above message now decrypted in the table below. Basically the cipher key is subtracted from the encoded text, 26 is added to the difference, and modulus 26 is taken of the resulting sum.

P	E	H	B	O	N	N	J	M	R	C	J	C	I	N	N	X	X	←message
15	4	7	1	14	13	13	9	12	17	2	9	2	8	13	13	23	23	←num code
L	R	F	K	Q	Y	U	Q	F	J	K	X	Y	Q	V	N	R	T	←rand key
11	17	5	10	16	24	20	16	5	9	10	23	24	16	21	13	17	19	←num code
4	13	2	17	24	15	19	19	7	8	18	12	4	18	18	0	6	4	← (message – key +26) modulus 26
E	N	C	R	Y	P	T	T	H	I	S	M	E	S	S	A	G	E	Oringal Text

Note that each part of this lab is *a separate program* that you will need to run on your mBED system.

Part 1: Create One-Time Pad

You need to create a one-time pad with 1000 random cipher characters. You will use the temperature sensor to add extra randomness to the pseudo-random number generator in C++. I would like for you read the temperature two times in a row. Subtract the two readings and multiply the difference by 1000. Type cast the result to an integer, and let this be a "thermal random number" based on fluctuations of the temperature reading in the room. You could even touch the sensor to add a little more fluctuation. This "thermal random number" will be ADDED to the pseudo-random number generator, `rand()`, to produce the final random number that you can use to select a random character for the cipher key between 0 and 25. Please seed the `srand()` function based on the current time as shown in class. Remember only seed once at the beginning of your main function once using this time function.

Also, call your file with the random cipher keys OTP.txt. This file should contain ASCII upper case letters. You will need to convert number codes to ASCII encodings.

Create two more text files for your system with this program. These files will store a single integer that keeps track of how many of the cipher keys have been used. Call these files positionCipherSender.txt and positionCipherReceiver.txt. Both will be initialized with 1 (or zero depending on your implementation choices). This will be needed to correctly keep track of what parts of the one-time pad have been used in each decoding session.

This program should generally operate as follows:

1. When this program runs on your mbed system, it will generate an internal character array that holds all the cipher key characters. I would suggest putting a null character at the end so that you can use fprintf(fp, "%s", cipherText).
2. Your program will then prompt the user to download the cipher key to the microSD card. Please use two of the push buttons to interact with the system. Maybe one button is "yes" and the other is "no."
3. The system will then ask if you want to copy the code to another microSD card. You can just test this with your same card inserted in the microSD card holder. Again use the push buttons to get user input during this phase.
4. You need to create at least one additional C++ class in this design. Please put the temperature and any other classes that you create in separate files. For the header files use .h extension and for the implementation files use .cpp extension for this to work properly with the cloud compiler.
5. Use a state machine implementation to manage how the system responses to button presses.

You will submit your code on canvas for assignment 4 according to the following instructions:

1. I would like for you to submit the code that you created to make the one-time pad on canvas as well. Please call the source code mainPart1.cpp
2. Submit your header and implementation files for other classes you create as well on canvas.

Part 2: Create encrypting pen!

For this part you will assume that the OTP.txt has been created AND a positionCipherSender.txt file also exists. You will then start to encrypt the following short message.

```
ECEROCKSTHEWORLD
```

Use three of your push buttons in the following way:

1. Two of the push buttons will scroll through the letters A through Z. Make sure that you put the text on the LCD display. I suggest using the following member functions with the LCD object

```
uLCD.locate(0,0);
uLCD.text_width(5);
uLCD.text_height(5);
uLCD.printf("%c",currLetter);
```

2. The third button will be the encryption button. Once pressed, your system should show the encrypted key for the current letter on the lower part of your LCD screen. Note that you will need to keep track of where you are in your encoding process. You must immediately update the file positionCipherSender.txt each time in case the system is turned off! Remember this file holds the position for where you are in the cipher key sequence. I give you some code to help you do this below.

3. Make sure that you use at least one additional C++ class for this section.

4. I would like for you to read in the entire set of cipher keys from the OTP.txt file for internal storage and manipulation. You will need to use the fscanf function. Your code might look something like:

```
FILE *fp = fopen("/sd/mydir/OTP.txt", "r");
 if(fp == NULL) {
    uLCD.printf("Open Error!!!\n"); }
 else
 {
    fscanf(fp, "%s",arrayCipher);
    fclose(fp);
 }
```

For reading in the current position in the cipher you might have:

```
FILE *fp2 = fopen("/sd/mydir/positionCipher.txt", "r");
   if(fp2 == NULL) {
     uLCD.printf("Open Error!!!\n"); }
  else
  {
     fscanf(fp2, "%i",&position);
     fclose(fp2);
  }
```

To change and update the file you might use code like the following:

```
fp2 = fopen("/sd/mydir/positionCipherSender.txt", "w");

if(fp2 == NULL)
{
   uLCD.printf("Error Open \n");
}
else
{
       fprintf(fp2, "%i",position);
       fclose(fp2);
 }
```

When you are done encoding, you can simply turn off the system.

You will submit your code on canvas for assignment 4 according to the following instructions:

1. I would like for you to submit the file that you created to encode the text. Please call this file mainPart2.cpp.
2. Submit your header and implementation files for other classes you create as well on canvas.

Part 3: Create decrypting pen!

For this part you will assume that the OTP.txt has been created AND a positionCipherReceiver.txt file also exists. You will then decrypt the message that you encrypted in Part 2.

Use three of your push buttons in the following way

1. Two of the push buttons will scroll through the letters A through Z that you are trying to decrypt.

2. The third button will be the decryption button. Once pressed your system should show the decrypted character on your screen. Note that you will need to keep track of where you are in your decoding. You must immediately update the file `positionCipherReceiver.txt` each time in case the system is turned off! Remember this file holds the position for where you are in the cipher key sequence.

3. Make sure that you use at least one additional C++ class for this section.

When you are done encoding you can simply turn off the system.

You will submit the code on canvas, please call the file `mainPart3.cpp`. Submit your header and implementation files for other classes you create as well on canvas.

Appendix A: Pre-Lab Exercise: State Machine Design with mBED

Instructions: This is related to your lab 4 – I would like for you to use a state machine design in the control of your system.

New C++ Element: Notice the "enumerated" types declared globally in the following code with keyword enum. This can be a great way to make your code more readable (i.e. self-documenting).

```
#include "mbed.h"
#include "Speaker.h"
#include "PinDetect.h"


//declare objects for pins used with pushbuttons
PinDetect pb1(p15);
PinDetect pb2(p16);
PinDetect pb3(p18);

//declare a speaker object
Speaker mySpeaker(p21);

enum InputType {FWD, BACK,STAY};
enum StateType {Q0, Q1, Q2, Q3};

InputType input = STAY;
StateType state = Q0;

// Callback routine is interrupt activated by a debounced pb3 hit
void pb3_hit_callback (void)
{
// ADD CODE HERE THAT YOU WHAT TO RUN WHEN INTERUPT IS GENERATED
input = STAY;
}

// Callback routine is interrupt activated by a debounced pb1 hit
void pb1_hit_callback (void)
{
// ADD CODE HERE THAT YOU WHAT TO RUN WHEN INTERUPT IS GENERATED
input = FWD;
}

// Callback routine is interrupt activated by a debounced pb2 hit
void pb2_hit_callback (void)
{
input = BACK;
}


int main() {


  pb1.mode(PullUp);
  pb2.mode(PullUp);
  pb3.mode(PullUp);

  // Delay for initial pullup to take effect
  wait(.01);

  // Setup Interrupt callback functions for a pb hit
```

```
pb1.attach_deasserted(&pb1_hit_callback);
pb2.attach_deasserted(&pb2_hit_callback);
pb3.attach_deasserted(&pb3_hit_callback);

// Start sampling pb inputs using interrupts
pb1.setSampleFrequency();  //default is 20KHz sampling
pb2.setSampleFrequency();
pb3.setSampleFrequency();
// pushbuttons now setup and running


while(1) {

   switch(state)
   {

   case(Q0):
      //Produce output for this state
      mySpeaker.PlayNote(200.0,0.5,0.05);
      //calculate next state
      if (input == FWD)
         state = Q1;
      else if (input == BACK)
         state = Q3;
      else //input should be stay
         state = Q0;
      break;

    case (Q1):
    //Produce output for this state
      mySpeaker.PlayNote(300.0,0.5,0.05);
      //calculate next state
      if (input == FWD)
         state = Q2;
      else if (input == BACK)
         state = Q0;
      else //input should be stay
         state = Q1;

      break;

    case (Q2):
    //Produce output for this state
      mySpeaker.PlayNote(400.0,0.5,0.05);
      //calculate next state
      if (input == FWD)
         state = Q3;
      else if (input == BACK)
         state = Q1;
      else //input should be stay
         state = Q2;
      break;

    case (Q3):
    //Produce output for this state
      mySpeaker.PlayNote(500.0,0.5,0.05);
      //calculate next state
      if (input == FWD)
         state = Q0;
      else if (input == BACK)
         state = Q2;
      else //input should be stay
```

```
            state = Q3;
          break;

        }//end switch
        wait (0.5);
    }
}
```