CSCE 236 Embedded Systems, Spring 2021
Homework 4

Due: Tuesday, Mar 16th, 2021 (start of class)

Instructions: This homework is an individual assignment, collaboration is not allowed. Show your work and describe your reasoning to get partial credit if your solution is incorrect. Unless otherwise specified, assume problems refer to the Arduino board we are using. This assignment is out of 100 points, but is equally weighted with other homework assignments.

Name: _____

Problem 1. *(5 pts) (To be completed at end of assignment) Approximately how much time did the total assignment take? Which problem took longest and how much time did it take? List any resources you used to complete this assignment (e.g. websites, datasheets, office hours, discussions with others, etc.). Be specific and if you did not use any other resources include the statement "I did not use outside resources for this assignment."*

Problem 2. Debugging: *For this problem you should download the code from the course website for this assignment. This code is from a developer who was trying to keep his job safe by writing very hard to understand code. Unfortunately, the author died in a hang gliding accident (which he as doing while he should have been at work!). Now we are left with code that is nearly impossible to understand, but is critical to our company. Your job is to use the debugging techniques we have learned about in class to figure out how to use this code.* ***You do not need to turn in your code online for this part of the assignment.***

For this problem you will have to view your assembly code. To view the assembly code you first need to find the directory where the compiler places the compiled code. To do this, configure the verbose compile output in the Arduino Sketch setup, which will cause it to print the full path when compiling. Then open a terminal/command prompt and go to that directory.
Run the command:

```
avr-objdump -d -t -h -S file.cpp.elf
```

but you will need to replace file.cpp.elf with the right filename for your project. The command ***avr-objdump*** *is located in different places depending on your system and you probably need to specify the full path to execute it. On OS X it is most likely located at:*

/Applications/Arduino.app/Contents/Java/hardware/tools/avr/bin/avr-objdump

in GNU/Linux it should already be in your path so you can just type ***avr-objdump****. In Windows it is probably located at:*

C:\\Program Files\\Arduino\\hardware\\tools\\avr/bin/avr-objcopy

There are also more details about how to do this for windows here:

http://rcarduino.blogspot.com/2012/09/how-to-view-arduino-assembly.html

and other google searches will give you results for other platforms.

a). *(5 pts) Before modifying any of the code, you should compile the program as is and determine how much memory and flash the program is using the* **`avr-objdump -d -t -h -S file.cpp.elf`** *(replace file.cpp.elf with the proper filename). How much RAM and flash are being used? Make sure to explain how you got this and include relevant lines from your objdump.*

b). *(10 pts) In order to start the program the functions* ***startOne(), startTwo(), startThree(), and startFour()*** *must all be called. Unfortunately, these must be called in a particular order and the proper order is unknown (and is not one, two, three, four). Further, it seems that calling them in the wrong order will cause the Arduino to freeze* ***and*** *some of these function calls mess up the serial printing. Use the debugging techniques discussed in class to figure out the proper order to call these functions. What is the order and* ***describe*** *how you figured it out.*

c). *(10 pts) There is another function* ***setMem(char i)*** *that takes a character and writes it to a particular location in memory. After much examination, we were able to determine that the value passed to* ***setMem*** *is being stored at a memory location at or greater than 0x40000800. Determine which memory locations the first character of your first name and first character of your last name are stored. Give the memory location (in hex), the initials you used (case sensitive), and briefly describe how you figured this out.* ***Hint: setMem(char i)*** *stores it as an offset in an integer array starting at location 0x40000800, so you can access it as *((volatile int *)0x40000800).*

d). *(10 pts) There is a final function,* ***runLoop()****, that should be called from the main loop function. This performs the key computation for our system whenever the button is pressed (attached to Arduino pin 5[1]). Time how long this function takes when the button is pressed and when it is not pressed. Answer in milliseconds and briefly describe how you figured this out.*

Problem 3. Functions – "Cryptography: *You'll practice your programming skills by writing some functions. You'll need to use both the call-by-value and call-by-reference techniques to pass arguments to your functions.*

You must also turn in your code on Canvas. *Failing to electronically turn in your code will result in a 10 point penalty on this assignment. Points may also be deducted for no header, coding errors, poor style, or poor commenting.*

The Basic Idea

You'll write a program that decrypts an encrypted message using a simple encryption technique. To achieve A-level functionality, you'll have to decrypt a message without knowledge of its key.

A simple, yet effective encryption technique is to XOR a piece of information with a key and send the result. The receiver must have the key in order to decrypt the message - which is accomplished simply by XOR ing the encrypted data with the key. Let's say I wanted to send the binary byte 0b01100011 and my key was 0b11001010 . To encrypt, I XOR the two - the resulting byte is 0b10101001 . To decrypt, I XOR it with the key again - the resulting byte is 0b01100011 - the same as the original byte!

An encrypted message of arbitrary length is stored in ROM. Your job is to decrypt it, given a key - which is also stored in ROM. The contents of the message are ASCII characters - each character is encoded in a single byte. You can tell how long the message is by counting the number of bytes.

You must write three functions:

- The job of the first is to decrypt an individual piece of information. It should use the pass-by-value technique and take in the encrypted value and the key and pass out the decrypted value.
- The job of the second is to leverage the first function to decrypt the entire message. It should use the pass-by-reference technique to take in the address of the beginning of the message, the address of the key, and the address in RAM where the decrypted message will be placed. It should use the pass-by-value technique to take in the length of the message. It will pass the encrypted message byte-by-byte to the first function, then store the decrypted results in RAM.

[1] Fortunately, the developer left a properly wired board behind.

- The job of the third is to print the final decrypted results to the serial port. It should use the pass-by-reference technique to take in the address of the decrypted of the message. It should use the pass-by-value technique to take in the length of the message.

Almost all of the work of your program will be performed in your two functions. Your main program after setup should just be a function call to **decryptMessage().**

Sometimes the key isn't conveniently the same length as the unit of information you're trying to decrypt. To achieve B functionality, you'll have to adjust your implementation to handle arbitrary length keys.

Be sure to maintain good program structure and programming discipline when adjusting your program for B Functionality.

To achieve unbreakable encryption, the key and the message must be the same length. For long messages, this is often impractical and a key substantially shorter than the message is used. Thus, the key must be applied repeatedly to decrypt the message. You can exploit this repetition to crack the message. This is even easier if you have knowledge of the contents of the message (ASCII text, for instance). To achieve A Functionality, you'll have to use this technique to decrypt a message without knowledge of the key.

a). Required Functionality (30 points)

- The encrypted and decrypted message will be in memory locations. The encrypted message and key will be stored in ROM - any location in ROM is acceptable. The message will be of arbitrary length, but the key will be one byte long. The decrypted message will be stored in RAM. Labels shall be used to refer to the location of the encrypted message, decrypted message, and key.
- The key and encrypted message will be given to you. You can tell how long the message is by counting the bytes.
- Good coding standards, in accordance with the Lab guidelines, must be used throughout.

Encrypted Message:

```
0xef, 0xc3, 0xc2, 0xcb, 0xde, 0xcd, 0xd8, 0xd9, 0xc0, 0xcd, 0xd8, 0xc5, 0xc3,
0xc2, 0xdf, 0x8d, 0x8c, 0x8c, 0xf5, 0xc3, 0xd9, 0x8c, 0xc8, 0xc9, 0xcf, 0xde,
0xd5, 0xdc, 0xd8, 0xc9, 0xc8, 0x8c, 0xd8, 0xc4, 0xc9, 0x8c, 0xef, 0xff, 0xef,
0xe9, 0x9e, 0x9f, 0x9a, 0x8c, 0xc4, 0xc5, 0xc8, 0xc8, 0xc9, 0xc2, 0x8c, 0xc1,
0xc9, 0xdf, 0xdf, 0xcd, 0xcB, 0xc9, 0x8c, 0xcd, 0xc2, 0xc8, 0x8c, 0xcd, 0xCf,
0xc4, 0xc5, 0xc9, 0xda, 0xc9, 0xc8, 0x8c, 0xde, 0xc9, 0xdd, 0xd9, 0xc5, 0xde,
0xc9, 0xc8, 0x8c, 0xca, 0xd9, 0xc2, 0xcf, 0xd8, 0xc5, 0xc3, 0xc2, 0xcd, 0xc0,
0xc5, 0xd8, 0xd5, 0x8f
```

Key: `0xac`

Decrypted Message:

b). B Functionality (15 points)

In addition to the Required Functionality, your program must decrypt messages with arbitrarily long keys. The keys are arbitrarily long series of bytes.

The length of the key should be a parameter passed into your function. You know the length of the key in advance.

Encrypted Message:

```
0xf8, 0xb7, 0x46, 0x8c, 0xb2, 0x46, 0xdf, 0xac, 0x42, 0xcb, 0xba, 0x03, 0xc7,
0xba, 0x5a, 0x8c, 0xb3, 0x46, 0xc2, 0xb8, 0x57, 0xc4, 0xff, 0x4a, 0xdf, 0xff,
0x12, 0x9a, 0xff, 0x41, 0xc5, 0xab, 0x50, 0x82, 0xff, 0x03, 0xe5, 0xab, 0x03,
0xc3, 0xb1, 0x4f, 0xd5, 0xff, 0x40, 0xc3, 0xb1, 0x57, 0xcd, 0xb6, 0x4d, 0xdf,
0xff, 0x4f, 0xc9, 0xab, 0x57, 0xc9, 0xad, 0x50, 0x80, 0xff, 0x53, 0xc9, 0xad,
0x4a, 0xc3, 0xbb, 0x50, 0x80, 0xff, 0x42, 0xc2, 0xbb, 0x03, 0xdf, 0xaf, 0x42,
0xcf, 0xba, 0x50
```

Key: `0xacdf23`

Decrypted Message:

c). A Functionality (15 points)

In addition to B Functionality, you must decrypt the following message without knowledge of its key. In addition to the decrypted message you need to give the key you used to solve it too.

There are many ways to attack this problem. Some techniques require substantially more CPU time than others. Some techniques can be done by hand. Take the time to think through your approach before you begin coding.

Encrypted Message:

```
0x35, 0xdf, 0x00, 0xca, 0x5d, 0x9e, 0x3d, 0xdb, 0x12, 0xca, 0x5d, 0x9e, 0x32,
0xc8, 0x16, 0xcc, 0x12, 0xd9, 0x16, 0x90, 0x53, 0xf8, 0x01, 0xd7, 0x16, 0xd0,
0x17, 0xd2, 0x0a, 0x90, 0x53, 0xf9, 0x1c, 0xd1, 0x17, 0x90, 0x53, 0xf9, 0x1c,
0xd1, 0x17, 0x9e
```

Decrypted Message:

Key (in hex): _____

[i]

[i] Do not forget to fill in the amount of time you spent on this assignment and resources you used in Question 1.