NAME(S) ____

CS 341 – Lab 3
Computer Architecture and Organization
Non-volatile Storage for Configuration Data (EEPROM)

Equipment: Arduino UNO microcomputer, PC with Arduino IDE installed, and a USB cable.

Configuration data may be information about the product such as manufacturer's ID, part number, and even the customer service phone number or web/email address. The device may be programmed to display this information upon user request. Configuration data may also be calibration information used to properly calculate the values read from or written to analog ports or other data needed for the product to operate that may differ from device to device. Configuration data must be preserved through power outages and/or processor resets, but it usually not part of the compiled program code because it may need to have different values for different individual devices. It may be stored in Electrically Erasable PROM (EEPROM).

The contents of an EEPROM are preserved across power outages so your code can save data while your device is running. After it is powered off and back on again and/or reset, your code can retrieve it during your initialization sequence. However, you can never tell if your code is executing on an embedded system where your code has previously run and left valid configuration data in the EEPROM or not. So the format of your configuration data in the EEPROM must have some unique characteristics that your initialization code can use to verify its validity. If it appears to be valid, your code can go ahead and use it for configuring this new run. If it appears to be invalid, your initialization code should not rely on the contents of the EEPROM. It should either initialize it to default values or go through an interaction with the operator or another system (e.g. a network server) to obtain and store new configuration data in EEPROM.

Two validation mechanisms commonly used to indicate and verify the validity of the contents of EEPROM are magic numbers and checksums (or both).

> A magic number is a specific location in the EEPROM as part of the configuration data that is of a specific size and contains a specific value. A magic number should contain enough bits to make it unlikely that it will be found accidently in an uninitialized memory. With the Arduino compiler, an int data type is 16 bits and an int magic number provides a 1 in 65536 chance of an accidental match. A long data type is 32 bits and a long magic number is better with only a 1 in 65536^2 chance of an accidental match. The magic number must be a value that is unlikely to appear in an uninitialized EEPROM. A value of all ones or all zeros is usually a bad choice because the memory might be shipped after manufacturing test with all locations set to one of those values. (I have observed that our Arduino boards start out with an all ones value in all EEPROM locations.) How can you choose a suitable value for your magic number? You pick one at random. See – it's magic! In embedded systems, commonly used magic number values consist of alternation of bit states, e.g. 0xA5A5 or 0xA5A5A5A5.

> A checksum is generated by operating on the contents of the configuration data with an algorithm that produces a checksum value that includes all values in the range of memory being protected. That value is then stored in a specific location in the EEPROM along with the configuration data. Again, the number of bits in the checksum should be large enough to avoid accidental matches and the algorithm should not typically produce a checksum value that will match when run on an uninitialized EEPROM containing all zeros or all ones.

However, recognize that although it may be unlikely that the chosen magic number and/or checksum is accidently determined to be valid, it is not impossible. Your code should also include some sanity checks on the values or combinations of values for parameters in the configuration data and escape to error handling code in case an accidental match has occurred.

NOTE: EEPROM memories have a limited lifetime in terms of the number of times that they can be written/read. In the Arduino UNO, this limit is specified as 100,000 read/write cycles. That value is specified conservatively. In practice, the memory will be good for more cycles than that, but keep this limitation in mind when you are planning use of an EEPROM memory. You don’t want your design to excessively read/write an EEPROM memory, e.g. by frequently storing/retrieving transient data in it.

In the Arduino UNO, the EEPROM is mapped to a memory space that is separate from program memory and RAM memory. The EEPROM memory is organized by bytes and its map has addresses ranging from 0 to the constant E2END. You should have learned the value of E2END in Lab 2. For reference, copy that value here: 0x______________.

In the Arduino IDE window, use the Help >> Reference menu, select the Libraries, and look at the EEPROM library. To access EEPROM, you need to include the header file `<EEPROM.h>.` The function `EEPROM.read(int address)` returns a char value from the indicated EEPROM address. The function `EEPROM.write(int address, char value)` stores the char value at the indicated EEPROM address.

In the setup function in your initial sketch, write a loop to display the existing values in the EEPROM from the beginning to the end of its memory space. What values do you find? Describe them in your report.

Now, change your setup function code to simulate storing, validating, and using configuration data in the EEPROM. Make it read and display a small number of bytes in the EEPROM. Then, run a bitwise exclusive-or on all bytes except the last one and ones-complement the result. Compare that value with the value stored in the last location in the EEPROM area that you are using.

> If it is incorrect (which it should be the first time you run this sketch on your board), write some specific values into those locations except for the last one. You can use the letters of one of your team member’s name. These values represent your default configuration data. Then, run a char bitwise exclusive-or on all locations except the last one, ones compliment the result, and store it as a checksum in the last location. Print a message that "EEPROM has been reinitialized".

If it is correct, just print a message that "EEPROM checksum is OK".

Unplug the Arduino board from the PC to cause a power off. Plug it back in and run your code again to check the values in those EEPROM locations. Push the reset button on the Arduino board and check the values in those EEPROM locations again. Describe the results in your report.

Lastly, swap Arduino boards with another team that has also gotten this far in the assignment and run your code on that board. Check the values in the EEPROM initially and again after a power up/reset cycle. Explain what has happened.

Submit your report to the TA in your next lab session with a copy of the code for your final sketch.

____ / 10