Electronics II
exercise 7

Deadline: Thu 19.3.2026 14:00

Data storage

In code, memory is often presented as a 1D array of memory addresses regardless of its physical layout. Addresses can contain data in various sizes, depending on the device, usually in bytes or 16-/32-bit words. The physical layout of random access memory (RAM) is usually not relevant as all addresses take the same time to access. In addition to other memories, the Arduino has 512 bytes of electrically erasable programmable read-only[1] memory (EEPROM). EEPROM is a non-volatile memory type, so it can be used to store data even if the Arduino is powered off.

With the EEPROM (or memory in general) there is no set way of storing data, other than the size of each memory address. This means that you will have to find a suitable storage scheme or come up with your own. Usual contenders are fixed-width data, storing start and end addresses of entries at a known location, and using delimiters. Each has their own pros and cons, but sometimes the storage type is decided by the data you're handling. For example, fixed-width data is easy to access, as each byte of any entry can be found with simple arithmetic, but it wastes space if your entries can be anything from a single byte to dozens of bytes.

Fixed-width data is easily accessed with arithmetic operations. Imagine and draw (or look down) a chunk of memory with a length of 24 addresses (bytes). These 24 addresses consist of four entries, each with a length of six bytes. This can be thought of as a 2D array of four rows and six columns, with unique addresses from 0 to 23. For any given address (test yourself):

- row = address//rowLength (true division, i.e., division without remainder)
  - $15//6 = 2$
- column = address%rowlength (modulo operator)
  - $15\%6 = 3$
- address = row·rowLength+column
  - $2 \cdot 6 + 3 = 15$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Player 1						Player 2						Player 3						Player 4					
Initials			Score																				

Table 1: 1D representation with entry separators for clarity.

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

Table 2: 2D representation of four fixed-width data entries (rows). Each of the entries is 6 addresses long. For example, the entries could be player high scores. The first three addresses of each row could be used to store the initials of a player, while the last three addresses could be used to store the score.

The course "Tietokoneen toiminta" explains memory accessing and allocation in more depth.

NB: EEPROM has a finite number of write cycles (about 100k). This sounds like a lot, but you can easily wear out sections of the memory by making a very fast loop and taking a coffee break. Use delays in your code when you're still testing things out. You can then remove the delays when you know that your code is working. Reading has no effect on lifetime, although you may have to refresh the data once a century or so (onsemi.com).

What is the real lifetime of EEPROM? (stackexchange.com)

Wear leveling on a microcontroller's EEPROM (stackexchange.com)

[1] Why can you write into a read-only memory (thousands of times)? In the memory hierarchy, EEPROM is considered to be too slow to be written into during runtime (unlike RAM), so EEPROM is practically read-only when a program is running. Keep in mind that computer and human time frames are vastly different (mooc.fi) and it takes about 50k clock cycles to write a single byte. EEPROM is somewhere between the fridge and the Moon. In addition, EEPROMs live for too few write cycles to not be considered "read-only".

Task 1 – Setting up the game (0.5+0.5+1p) [Arduino]

a) Display the position of the joystick as a dot on the dot matrix display. Use the joystick button to change the brightness of the dot. Take a moment to appreciate the simplicity of the joystick.

b) Read the RFID tag's ID.

c) Use the Arduino as a primitive "USB stick". Write a string of text from the computer to the EEPROM, unplug the Arduino, plug it back in, and read the text back to computer. Have two strings prepared for the demonstration, so you can swap them around.

Task 2 – Snake game (2p) [Arduino]

Make a snake game (or come up with your own minigame) using the joystick and the dot matrix. Before the game starts display "Show RFID" (on the serial monitor or on an LCD). If it's an unknown ID assign a new player number to it and store the ID on the EEPROM. Then display the current high score and the player number of the record holder. When the game ends, if it's a new high score congratulate the player and store it on the EEPROM.

Task 3 – Accelerometer (2p)

Extract data from the accelerometer and make an orientation detector. The detector is calibrated to the current orientation when a button is pressed. Use an RGB LED to indicate if the orientation is within a certain angle from the calibrated orientation. Include also a way to change the angle value which is registered as "tilted".

An application note on tilt calculation Tilt Sensing Using a Three-Axis Accelerometer (AN3461) (nxp.com)

This data sheet is for a different device, but Section 5.6 may be of use: BMA220 datasheet (dfrobot.com)