

Problem Set #6: Mastermind

Goals

- Learn to work with the *Arduino prototyping board*

!Warning

This assignment must be submitted by **2022-05-29 23:59:59** . Use the slack channel for discussion [#arduino](#).

!Warning

If you have agreed with your trainer, you can also create the assignment on the [Tinkercad](#) server .

Logik a.k.a. Mastermind

[Mastermind](#) is a game for two players, which in our region is better known under the name *Logik* . I assume you've played it before. However, if you belong to the group of players who *do not play Quake 3* , I recommend that you play a few games of this game before entering this entry. For example, the online version [here](#) or install the version for [Android OS](#) .



Game rules

- At the beginning, one of the players places a combination of colored pegs under the canopy (see image above). Their number is usually 4, but depending on the version of the game and its difficulty, there may be more of them. Depending on the agreement of the players, the colors of the pins may be repeated.
- The goal of the game is to guess the color combination of the other player, including the order of the individual colors. The first player evaluates each of his attempts as follows:
 - for each game peg that the guessing player has placed in the correct color in the correct place, he adds a black peg
 - for each game peg that the guessing player placed in the correct color but in the wrong place, he adds a white peg
- The guessing player wins if they guess the entire combination. This means that he guesses the colors of all the pegs in the correct positions (his move will be scored with four black pegs).
- The guessing player loses if he fails to guess the color combination in the allowed number of attempts.

The players then switch roles and play again. The game is won by the player who manages to guess the opponent's combination in the least number of moves.

Task

Yes - your task is to create an implementation of the *Logik game in the C* language . But to make it not so easy, here are some changes:

- *Your solution will be built on an Arduino* prototyping board .
- The role of the player who generated the secret color combination will always be represented by *Arduino* .
- Instead of colored pins, it will be necessary to guess a numerical combination consisting of four digits in the range from 0 to 9 .
- The display will be used for communication with the player.
- To enter the combination, the player will use buttons - one for each digit of the code, so a total of 4 buttons. After pressing it, the value of the digit will change gradually from the value 0 to the value 9 . After reaching the value 9 and pressing the button again, the value of the digit on the display changes again to the value 0 , then to 1 , etc. But be careful - when you press the button, the number changes by 1 . So if I want to change the initial value from 0 to 5 , I have to press the button 5 times .
- To end the move, it is necessary to confirm the move by pressing the *ENTER button* - the fifth button.
- At the end of the turn, the *Arduino* evaluates the player's move using four RGB lights:
 - if the player guesses the number and guesses its position at the same time, it lights up red
 - if the player guesses the number, but does not guess its position, the blue color lights up
- If the player guesses the whole combination, you can represent this state with different effects depending on the parts used.
- If the player presses the first button (**BTN_1_PIN**) and then presses the second (**BTN_2_PIN**) or third (**BTN_3_PIN**) button, he will be able to move forward (when pressing the third button) and backward (when pressing the second button) in the history of previous attempts.

Demo game

For a better idea of how one party can look like, you can look at this video on [YouTube](#) . Or the following statement can help you:

```
9347
- - - - -
1234 0A2B
5678 0A1B
9012 1A0B
9340 3A0B
9345 3A0B
9346 3A0B
9347 4A0B
```

Subsequent communication within the two-line LCD display can look like this, for example:

- Initial screen after power on:

```
Welcome to MasterMind
Your goal is to guess my secret combination.
```

- Subsequent launch of the game and prompt for the player:

```
I am thinking a number:
Your guess: 0000
```

- When pressing the buttons that are assigned to the pins `BTN_1_PIN`until `BTN_2_PIN`the values of the individual displayed digits gradually change, the player tip:

```
I am thinking a number:
Your guess: 1234
```

- After pressing the button connected to the pin `BTN_ENTER_PIN`, the player's move will be evaluated (on the display and at the same time by the corresponding combination of RGB diodes) and the next prompt will be displayed:


```
01: 1234 0A2B
Your guess: 0000
```

- After entering the next tip and confirming it, it will be evaluated:

```
02: 5678 0A1B
Your guess: 0000
```

- In this case, if the player starts scrolling back through the history (by holding down the button in combination with the button), the previous move will be displayed in the history line and the RGB LEDs will also light up accordingly. However, the line with the currently entered tip will not change in any way:

```
01: 1234 0A2B
Your guess: 0000
```

Note

It is possible to move backward through the history only after the first attempt and forward only after the last attempt.

- After entering the last hint leading to the correct solution of the game, a suitable text will appear on the display and the RGB diodes can emit different colors:

```
07: 9347 4A0B
Well done! You win in 7 guesses!
```

- If the player exhausts all attempts, a sad message will be displayed:

```
10: 9348 3A0B
Looser! My secret combination was 9347.
```

Note

Since you will definitely ask if the communication with the player should look exactly like this, notice the text "*can look like this*" at the beginning of the listing : no, your implementation can look different and can display other texts.

Equipment

To create the game you will need:

- 1x Arduino UNO
- 1x breadboard
- 1x display s i2c moduom
- 4x two-color or RGB LEDs
- 5x button
- 8x resistor 220 Ohm
- 5x resistor 10 kOhm
- connecting cables (including USB cable)

Development environment

The development environment (*IDE*) for *Arduino* can be downloaded directly from www.arduino.cc .
However, if you are using a *Linux* distribution, you can install this tool directly from the repository:

- In case you are using *Fedora* (*RedHat*):

```
sudo dnf install arduino
```

- In case you are using *Ubuntu* (*Debian*):

```
sudo apt-get install
```

Note

It is very likely that the *Arduino IDE* version in your distribution's repositories will be older than the one available on the official site. Therefore, instead of installing the *Arduino IDE* from the repositories, we recommend that you install the latest version directly from www.arduino.cc .

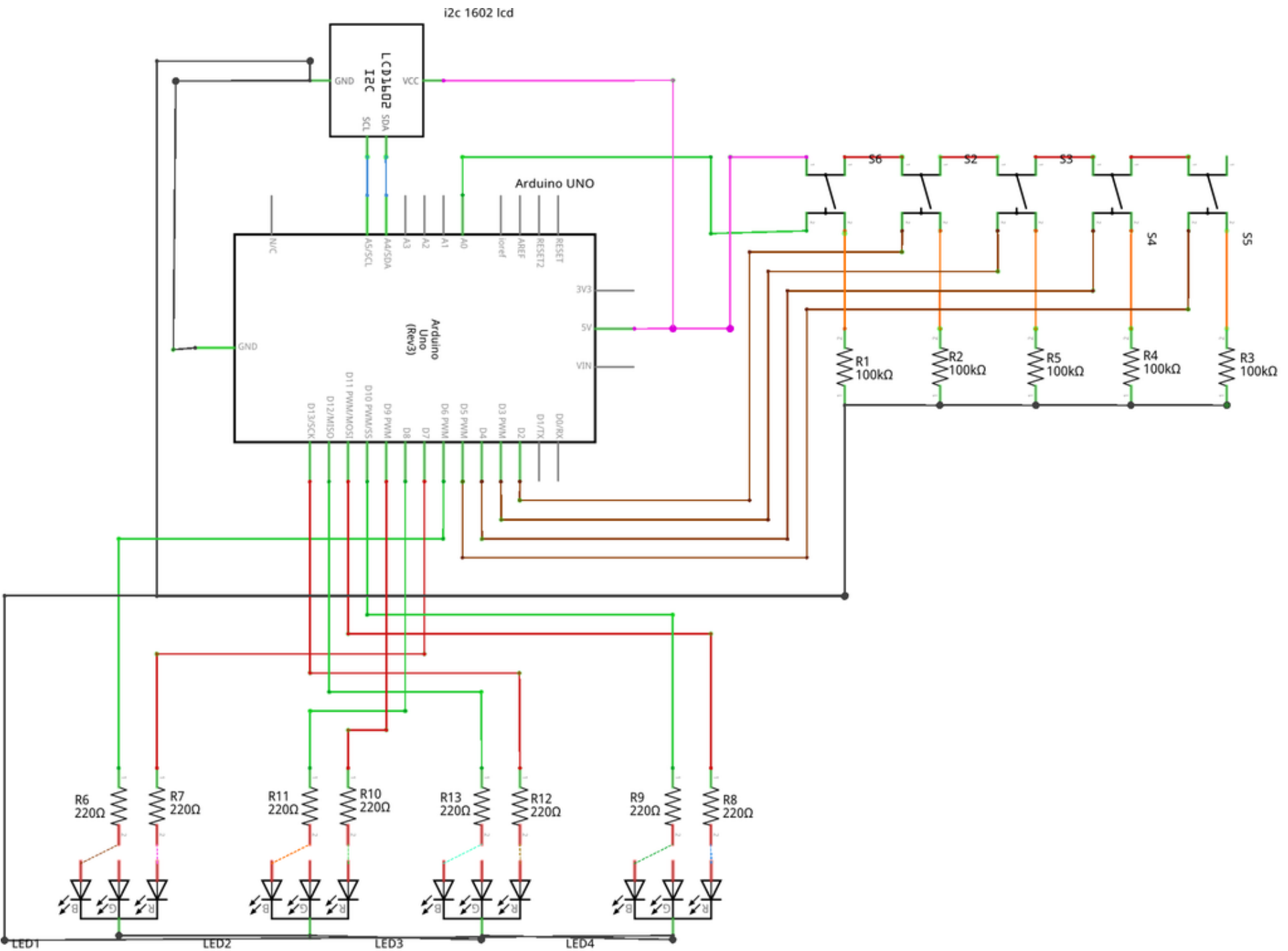
Instead of the *Arduino IDE* , you can also use the [Fritzing](#) tool , which offers more options than just a development environment for programming a microcontroller. You can use it, for example, to draw a circuit diagram of your solution.

Similar to the *Arduino IDE* , you can find *Fritzing* in the repositories of your distribution or you can download it directly from the [project](#) home page .

However, the biggest geeks will definitely use the *Vim* text editor for development . [In that case, for example, this article](#) will certainly help them .

Wiring diagram

The diagram below shows the entire connection. In this case, it is very important that your connected inputs and outputs match the specified PINs.



fritzing

mastermind - schematic

ModuleLCD

This module concerns your LCD display. It will not be controlled by the person himself, because its functionality depends on the type of your LCD display. Individual functions and their list can be found in the header file `lcd_wrapper.h`.

The function `lcd_init()` that serves to initialize the display is important. Call this function as needed from the function `setup()`. If the use of your library also requires some global variable, create or write the corresponding call in the file `lcd_wrapper.cpp`, because the entire work and all display-a calls will be located only in this file (module).

ModuleMastermind

This module contains the very logic of the entire game.

Task #1: Function `generate_code()`

This function is used to generate a secret code to be guessed. It returns the generated code as a reference to a character string that is composed only of numeric characters.

The function has the following positional parameters:

- `bool repeat`- If it is set to the value `true`, individual digits can be repeated in the resulting code. Otherwise, they cannot be repeated.
- `int length`- Determines the length of the generated code (the number of digits that the resulting code should consist of). If the length is less than `1`, the function returns `NULL`.

The function returns a reference to the generated string or `NULL`, if the string could not be generated. The function also returns `NULL` if the parameter `repeat` has a value `false` and the length of the string is greater than `10`.

Example of use

```
char* code = NULL;

// code will be generated as sequence not repeating digits of length 4
code = generate_code(false, 4);
// code = "1234";
free(code)

// code will be generated as sequence of repeating digits of length 5
code = generate_code(true, 5);
// code = "65656";
free(code);

// no code will be generated
code = generate_code(true, -10);
// code = NULL;
```

Task #2: Function `get_score()`

Using this function, the current attempt will be evaluated. This will result in setting correct rolls for the number of guessed digits in the correct position, as well as for the number of guessed digits that are not in the correct position.

The function has the following positional parameters:

- `char* secret`- Reference to the number combination to be guessed. Input parameter.
- `char* guess`- A reference to the number combination that represents the player's current tip. Input parameter.
- `int* peg_a`- A reference to a number that represents the number of guessed digits found in the correct position. Output parameter.
- `int* peg_b`- A reference to a number that represents the number of guessed digits that are not in the correct position. Output parameter.

The function does not return any value.

Example of use

```

int peg_a;
int peg_b;

get_score("9347", "1234", &peg_a, &peg_b);
assert(peg_a == 0 && peg_b == 2);

get_score("9347", "9348", &peg_a, &peg_b);
assert(peg_a == 3 && peg_b == 0);

get_score("9347", "1256", &peg_a, &peg_b);
assert(peg_a == 0 && peg_b == 0);

get_score("9347", "9436", &peg_a, &peg_b);
assert(peg_a == 1 && peg_b == 2);

```

Task #3: Functions `turn_off_leds()` `render_leds()`

Both of these functions are used to work with RGB diodes. The function has no parameter and when it is called, all LEDs turn off. `turn_off_leds()`

The function `render_leds()`, on the other hand, lights up the corresponding RGB diodes based on how many digits the player has guessed - for each guessed digit in the position, one RGB diode lights up red, and for each guessed digit that is not in the correct position, one RGB diode lights up blue. `render_leds()`

Note

No diode is associated with any digit position. This means that the player must not know that if the RGB diode no. 1 that he guessed the digit in the first position.

The function has the following positional parameters: `render_leds()`

- `const int peg_a`- The number of guessed digits in the correct position.
- `const int peg_b`- The number of guessed digits in the wrong position.

Functions do not return any value.

Example of use

```

turn_off_leds();

render_leds(2, 1);
// two leds will have red color, one will have blue and last one will be off

```

Task #4: Function `render_history()`

The task of this function is to write on the display the relevant record from the history along with its evaluation.

The function has the following positional parameters: `render_history()`

- `char* secret`- Reference to the number combination to be guessed. Input parameter.
- `char** history`- Reference to trial history.
- `const int entry`- Index of the position in the history to be written.

The function does not return any value.

Example of use

```
char* history[3][4];

history[0][0] = "1234";
history[1][0] = "5678";
history[2][0] = "9012";

render_history("9347", *history, 2);
// renders and evaluates "9012"
```

Task #5: Function `play_game()`

A feature that represents the game itself from launch to completion.

This function is specific due to the fact that it will contain the game loop in itself and will not use the function offered by *Arduino*. With this approach, we can avoid excessive use of global variables, which most examples/tutorials/books dedicated to *Arduino* directly encourage us to do. `loop()`

The function has one positional parameter:

- `char* secret`- A reference to a string that represents the secret code to be guessed in that game.

The function does not return anything when finished.

Example of use

```
void loop(){
    char* code = generate_code(false, 4);
    play_game(code);
    free(code);
}
```

Handing over the project

The assignment is submitted via the [Git version control system on the git.kpi.fei.tuke.sk](https://git.kpi.fei.tuke.sk) server. You will submit the solution to this task as part of your project.

The structure of your project will look like this:

```
.
├── ps6
│   ├── ps6.ino
│   ├── lcd_wrapper.cpp
│   ├── lcd_wrapper.h
│   ├── mastermind.cpp
│   └── mastermind.h
└── README
```

where the meaning of the files in the folder `ps6/` is as follows:

- `mastermind.cpp`, `mastermind.h`- Source code and library header file for the game *Mastermind* (*Logik*).
- `lcd_wrapper.cpp`, `lcd_wrapper.h`- Source code and module header file for working with LCD display.
- `ps6.ino`- Source code (*sketch*) containing the function `.main()`
- `README`- A file that will indicate the designation of your group that you attend for exercises in the form:

GROUP : A1

If you are a **repeat** student, put in **README** the group in the form:

GROUP : O<P>

where <P> you replace with the letter of the parallel which, according to the schedule, corresponds to your study program (e.g. A for computer scientists).

!Warning

If you do not have the necessary hardware components, use the [Tinkercad](#) service to complete the assignment . In that case, however, you will be limited both by the available components and by the non-modular approach to programming - you will write the entire program in one file.

In this case, submit your project with the following structure:

```
.
├── ps6
│   └── ps6.ino
└── README
```

!Warning

It is important that your files maintain the structure listed. If any of the files are in the repository, but in a different folder, it will be considered an error and such a project will not be considered correct! If, on the other hand, there are extra files or folders in your project, these will not be considered an error.

!Warning

Folder, file and file content names are **README case** -sensitive!

Project staff

This time, the project skeleton is represented by just one header file, which you can download from this [link](#) .

Evaluation and testing

This assignment will **be submitted and graded in person!** An assignment that was successfully graded at least 51% is considered a successful submission ! Therefore, it is not enough to just upload the entry to the *Git* version management system by the submission deadline , but it **must also be evaluated by then !**

During the evaluation, the following will be evaluated:

- The structure of your project (whether it contains all the necessary files).
- The presence of global variables in your code.
- The functionality of your implementation.
- Communication with the solver and his ability to respond to the questions asked, or his programming experience.

!Warning

If the examiner gets the impression that you do not understand the submitted project, your assignment will not be graded. This also means that you will not get credit for the subject, even though you will have a sufficient number of points from the previous assignments!

Additional resources

- [Language Reference](#)
- Arduino: [6. Press the button a second time](#) , or press to turn on, press to turn off

Programming

The course covers advanced topics in the *C language and Arduino* microcontroller programming .



Except, where otherwise noted, content on this site is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International \(CC BY-NC-SA 4.0\)](#).