Charles University

Computer Systems

Labs - summer 2025/26

Tomáš Faltín, https://fan1x.github.io/

Lab 06

27/04/2026

18/05/2026

Lab Tests

2 hours long, 2+1 attempts, terms are in SIS

In the lab rooms, using local PCs (no laptops allowed)

- Access to Internet resources will be restricted

Completing (fixing) home assignments before the test is recommended

- You may re-use code from your home assignments in the test

Beware ALL unforgivable curses!

Practice

- AD&D dice example task (follow a link on the Grading page)
- Try the extra tasks introduced on the slides
- Demo test (in SIS)

Ask if anything is unclear

Homework Feedback

Be aware of deadline (Tuesday 21:59)

Request a review after fixing the problems

Test locally first (10 attempts only)

Homework Discussion

Take **your colleague's** code, discuss and code-review it together

- ☺ What do you like and **why**? What did you do **differently?**
- ☹ What do you dislike and **why**? Discuss a potential fix
- What was the **hardest**?

Pointers

type ***p =** *<pointer expression>*

- **`nullptr`** – pointer to nowhere (special value)
- Taken as address of existing value (**`&`**)
- Computed from another pointer (pointer arithmetic)
- Memory allocation function/operator
  - Not needed on Arduino, use only static array allocation, e.g., **`int a[4]`**

```
int x = 42;
```

Create a space for one `int`

```
int *p1 = &x;
```

address-of operator
Not a reference

Declare pointer

```
int *p2 = p1 + 5;
```

Arrays

```
int buffer[8];
```

Array ~ pointer at the beginning.

```
int *p1 = buffer;
```

Increment/decrement

```
int *p2 = ++p1;
```

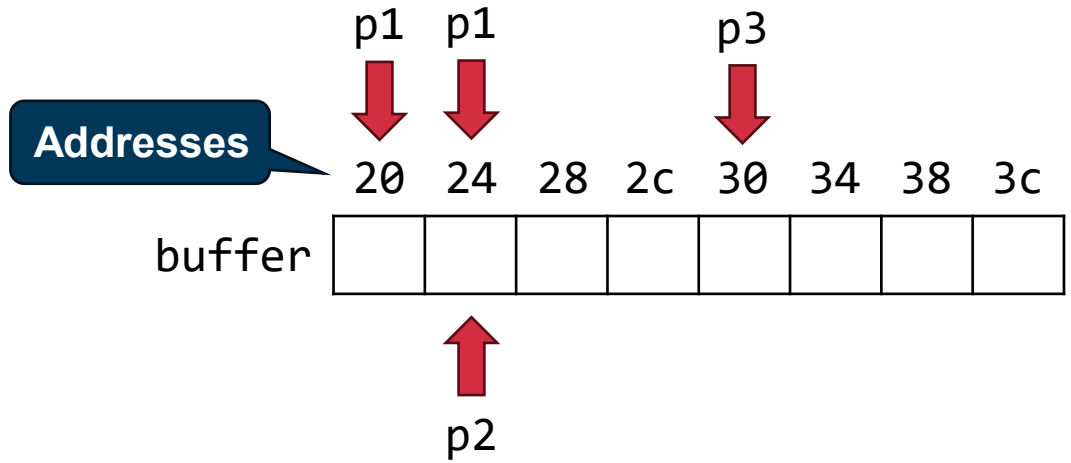
Number of bytes deduced from the type, i.e., `sizeof(int)`

Arithmetic

```
int *p3 = p1 + 3;
```

Prefer brackets when working with arrays

`buffer[1]` ~ `*(buffer + 1)`

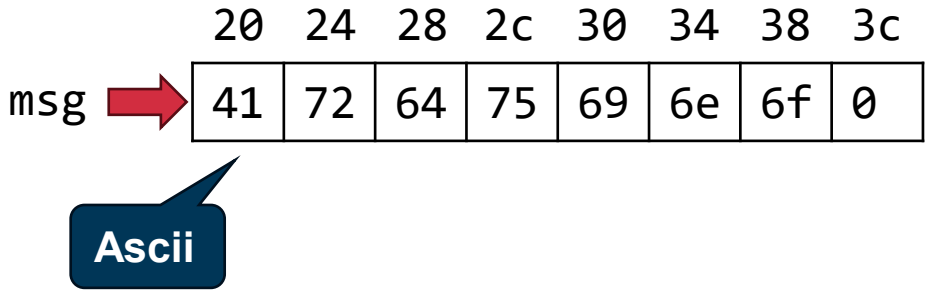


Strings

Array of chars, terminated by zero char ‘\0’

```
const char *msg = “Arduino”;
```

Read-only



Pitfalls

Uninitialized pointers

```
int *p;
```

May hold a random address

Use `nullptr`

Reaching beyond arrays

```
int buf[5];
int *p = buf + 5;
*p = 42;
```

Writing right after the array. What data is there?

Strings in fix-sized buffers (like `char buf[32]`)

```
char msg[] = {'A','r','d','u','i','n','o','\0'};
```

- Possible, but dangerous (must not overflow!)
- Do not forget to make space for trailing zero!

```
char msg1[] = {
  'A','r','d','u','i','n','o','\0'};
char msg2[] = "Arduino";

char msg2_copy[7];
int msg2_copy_size =
  sizeof(msg2_copy)/sizeof(msg2_copy[0]);
for (int i = 0; i < msg2_copy_size; ++i) {
  msg2_copy[i] = msg2[i];
}
printf("%s", msg2_copy);
```

What is the output?

Task 1: Personal strlen

Create a C function that counts a given string size

Use pointers for implementation

Use, e.g., Coliru: https://coliru.stacked-crooked.com for testing

Task 2: Running Text

Create a running text message on LED display

- Download the prepared starter pack and open it
- Message is sent over via serial link
  - Use **`SerialInputHandler`** class
- Display multiplexing works like in the last assignment
- Text scrolls (4 chars are visible at a time)
  - Advance position by 1 to right every 300ms
  - Loop the scrolling (once it scrolls our, start over again)
- Use pointers in your code!

Questions?

Ping me if anything unclear

Fill the survey!

Good luck ☺

Lab 05

13/04/2026

04/05/2026

Homework Feedback

Be aware of (new) deadline - Tuesday 21:59

Request a review after fixing the problems

Test locally first (10 attempts only)

Lab test demo

Time Multiplexing

Lab4: 7segment display

- The counter shows a number 0-9999, a single digit at a time
- Buttons 1 & 2: perform increment & decrement
- Button 3: change the visible digit

Take **your colleague's** code, discuss and code-review it together

- ☺ What do you like and why? What did you do differently?
- ☹ What do you dislike and why? Discuss a potential fix

Beeper

Task 0: Beeper

Display Experiments (with your colleague's code)

Task 1: Show all numbers - naïve approach

- Button 1 & 2: increment/decrement by 1
- Display all 4 numbers in every loop()

Do you see any anomaly?

If not, try to put `delay(2)` at the end of the loop

Display Experiments (with your colleague's code)

Task 2: Show all numbers – multiplexing

- Display a single digit at a time
- 1 loop ~ 1 digit
- Each loop takes the same amount of time

Why?

Separate a display control from the rest of your code

Display Experiments (with your colleague's code)

Task 3: Improve appearance

Do not display leading zeros

Include a decimal dot

- E.g., a function that sets its location

A good code decomposition should require minimal changes when adding a new feature

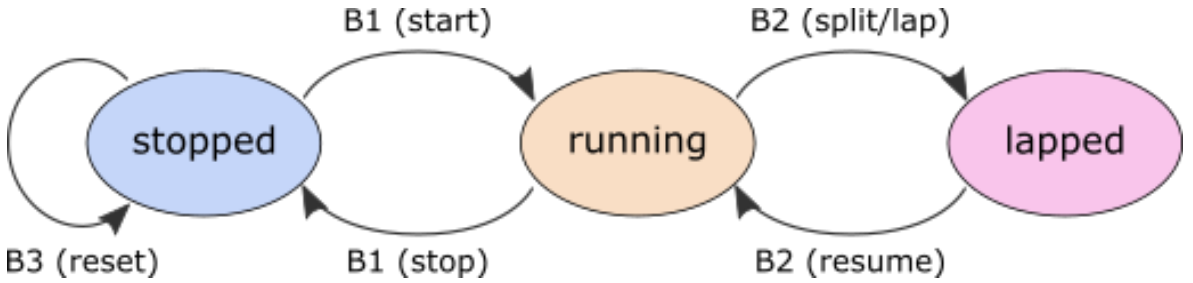
Glyph combination?

Use bit & (and)

Why & and not | (or)?

Display Experiments (with your own code)

Task 4 (Recodex): Create a stopwatch

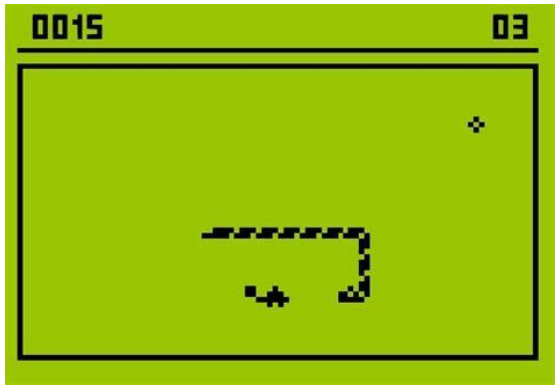


State automaton

- The displayed precision is 100ms (i.e., 0.1s)
  - **But internal precision should be at least 1ms** (use `millis()` for correct timing)
  - Use the decimal dot to separate the last digit
  - Always display at least 2 digits (zero is 0.0 – Czech locale notation)
- *Button #1* starts/stops the stopwatch
- *Button #2* splits the time (i.e., freezes the display, but the watch keeps running)
- *Button #3* resets the stopwatch to 0 (only when stopped, no effect when counting)
- Be precise (correct) with the timing! (tested in ReCodEx)

Other Display Experiments (with your own code)

- Game: Snake
- Game: Tetris
- Graphical equalizer
- ….



UC5: Encapsulation

```
bool alarmActive = false;
time_t alarmTime, snoozedUntil;
bool setBtnState, resetBtnState, snoozeBtnState;
```

Alarm's (internal) state

Button's (internal) state

```
bool processButton(int pin, bool &state) { … }
```

Button's logic

Alarm's logic

```
loop() {
  if (alarmActive) {
    if (alarmTime < now
        && (snoozedUntil == not_set || snoozedUntil < now)){
      enable_sound_output();
    } else {
      disable_sound_output();
    }

    if (processButton(reset_btn_pin, resetBtnState)) {
      alarmActive = false;
    }
    if (processButton(snooze_btn_pin, snoozeBtnState) &&
      alarmTime < now &&
      (snoozedUntil == not_set || snoozedUntil < now)) {
        snoozedUntil = get_current_time() + snooze_delay;
      }
  }

  if (processButton(set_btn_pin, setBtnState)) {
    alarmActive = true;
    alarmTime = get_input_time();
    snoozedUntil = not_set;
  }
}
```

UC5: Encapsulation

```
class AlarmClock {
private:
  bool active;
  time_t alarmTime;
  time_t snoozedUntil;

public:
  void setAlarm(time_t at) {
    active = true;
    alarmTime = at;
    snoozedUntil = not_set;
  }
  bool isRinging() {
    time_t now = get_current_time();
    return active && alarmTime < now &&
      (snoozedUntil == not_set || snoozedUntil < now);
  }
  void snooze() {
    if (isRinging()) {
      snoozedUntil = get_current_time() + snooze_delay;
    }
  }
  void reset() { active = false; }
};
```

(Private) internals

(Public) API

Semicolon

☹ Global (Arduino ☹)

```
Button setBtn, resetBtn, snoozeBtn;
// ...
AlarmClock alarm;

void setup() { ... }
// ...
void loop() {
  if (alarm.isRinging()) {
    enable_sound_output();
  } else {
    disable_sound_output();
  }

  if (setBtn.isPressed())
    alarm.setAlarm(get_input_time());
  if (resetBtn.isPressed())
    alarm.reset();
  if (snoozeBtn.isPressed())
    alarm.snooze();
}
```

Instances ~ variables

Using API only

UC5: Encapsulation

Encapsulation (sufficient OOP design)

- Each class has a clear explicit purpose (e.g., Button, Display, …)
- A well-defined set of functions/methods allows the programmer to exploit this purpose
  - But the internal implementation is hidden (maybe changed in the future)

Hints

- Interface first (imagine how it is used), implementation after
- Imagine the implementation changes over time (e.g., button behavior)
- Make member variables private (so you are not tempted to access them)
  - Do not create simple getters/setters unless you really need them and re-think them twice (it may be a sign you are exporting behavior out of a class)

Lab 04

30/03/2026

20/04/2026

Homework Feedback

Be aware of deadline

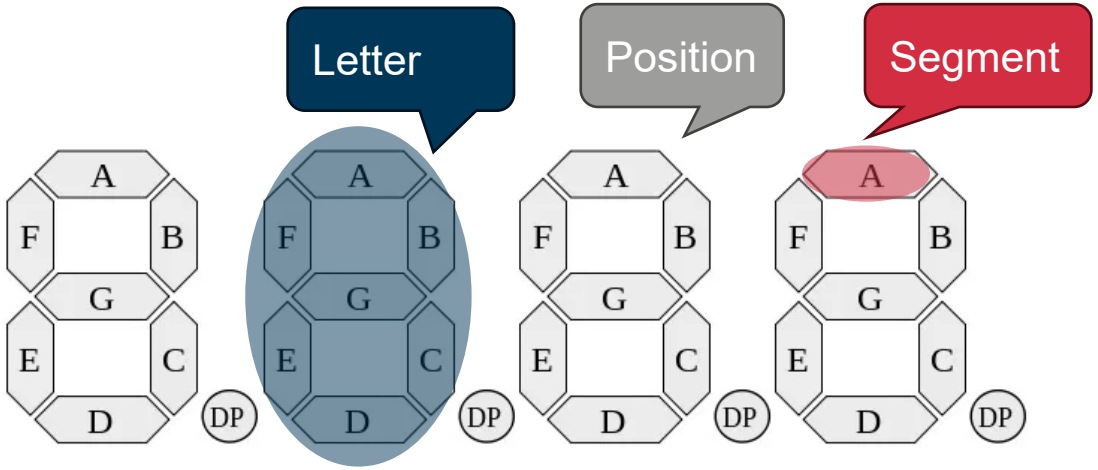
Request a review after fixing the problems

Test locally first (10 attempts only)

Homework Feedback

7Segment Display

Controlling individual segments

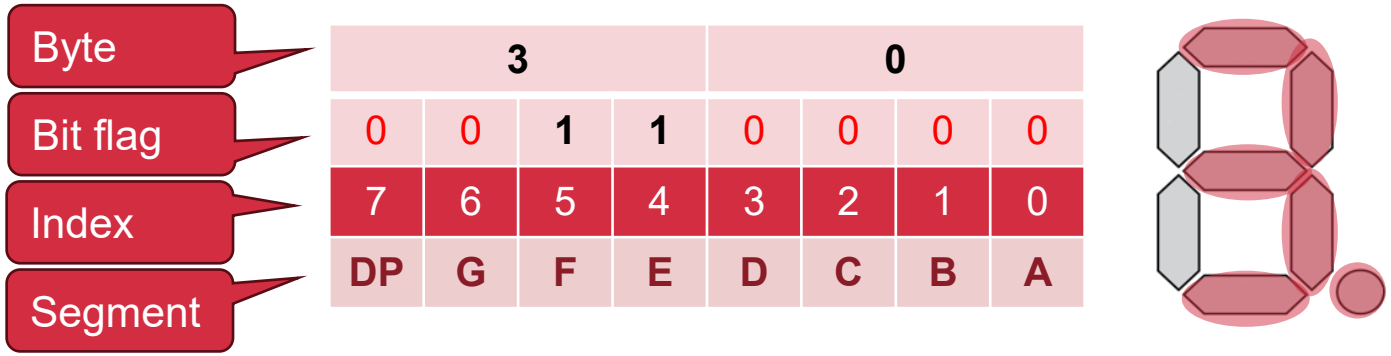


Glyph

- A “picture” in a char (a set of segments)

Glyph Encoding

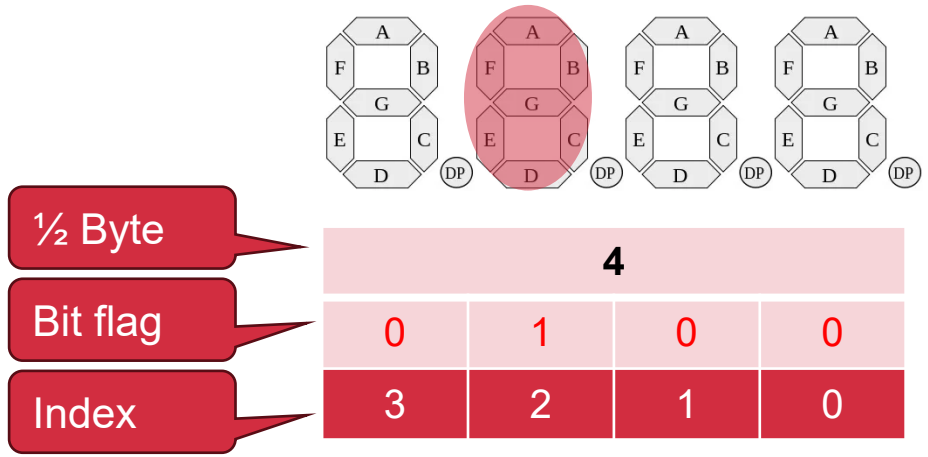
- bits in a single byte
- Inversion logic! (1 ~ off, 0 ~ on)



'3.' ≈ 0b00110000 = 0x30 = 48

Glyph index

- Encoded with bits
- Uses only a half of the byte



7Segment Display – Communication

Using a shift register

Use 3 pins: latch, clock, data

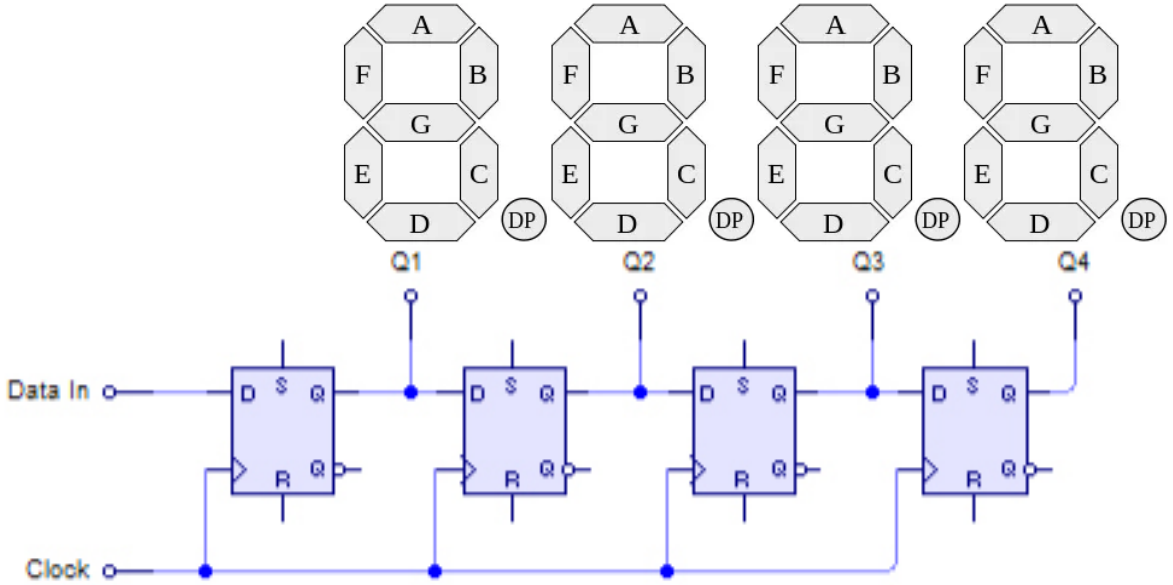
Initialize! (`pinMode`)

Write a glyph:

1. Close latch
2. Send a glyph
3. Send a glyph index
4. Open latch

```
shiftOut(
  data_pin,
  clock_pin,
  bitOrder,
  value);
```

```
digitalWrite(SEG7_LATCH_PIN, LOW);
shiftOut(SEG7_DATA_PIN, SEG7_CLOCK_PIN,
  MSBFIRST, glyph);
shiftOut(SEG7_DATA_PIN, SEG7_CLOCK_PIN,
  MSBFIRST, position);
digitalWrite(SEG7_LATCH_PIN, HIGH);
```



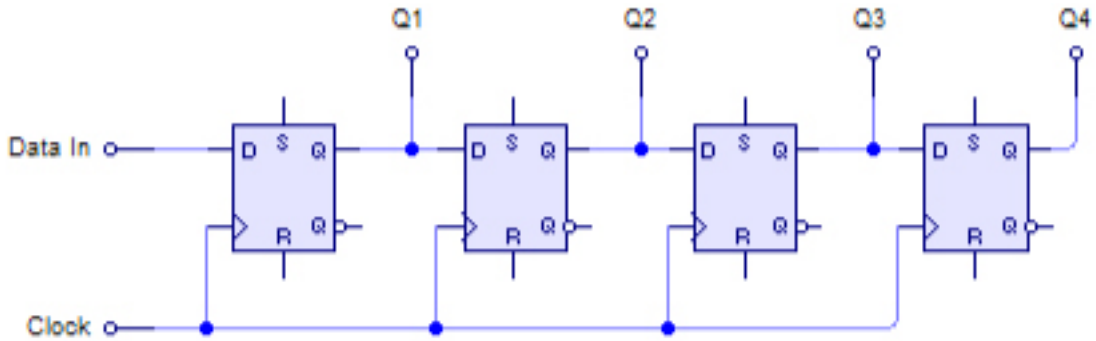
Shift Register

74HC959D Shift Registers

Serial input (sequence of bits), parallel output

Input is controlled by a clock signal

See https://en.wikipedia.org/wiki/Shift_register



Display a Digit

Task0: Write any number on any position

Task1: Write a function writing a digit on a given position

Task2: Change bit ordering from `MSBFIRST` to **`LSBFIRST`** and write numbers 1, 2, 3, 4

Task3: Use code that controls buttons and shows numbers from 0-9 only
Other digits stay blank
Use the function from Task1

Task4*: Create a snake animation that goes round the whole display
A single LED is on at any given time
Speed is given by a constant, e.g., 1s

Task5: Show All Digits (Simplified) - Recodex

Modify the counter to run modulo 10,000

- Formatted as a 4-digit number **with** leading zeros
- Make sure you handle the modular arithmetic correctly!

One digit (position) is shown at a time

- Rightmost digit (units) at the beginning
- Button 3 changes the digit/order in a cyclic fashion (1, 10, 100, 1000, 1, …)

Buttons 1 and 2 increment/decrement “the selected digit”

- I.e., button 1 performs +1, +10, +100, or +1000, based on which order is selected (displayed)
  - Button 2 analogically (-1, -10, …)

Use a simple arithmetic/logical operations, no `pow`, …

Unforgivable Curse #4 – No Global Variables

Restricted access to global variables

- Global variables are bad, but we have no choice…

Rules

- Do not use global variables instead of local variables
- Do not use global variables just to pass values to/from functions
- If you could modify the value of a global variable in between `loop()` calls and it would not matter … then it should not be a global variable

To be on the safe side…

- Access global variables only from `loop()` and `setup()`

Constants can be used globally (to make it easier for you)

Writing Better Code…

Your code should handle 3 different things

- Button control (when a button triggers an action)
- Display control
- Application logic (handling the counter modifications, triggering display update…)

Make sure each part is duly encapsulated and reusable (esp. button and display)

- Use function(s)/method(s), structures/classes, …

For instance, it is not appropriate when…

- Button handling routine updates the counter directly
- Button handling triggers display update
- Display update tries to read a button state

Lab 03

16/03/2026

25/03/2026

Homework Feedback

Be aware of deadline

Request a review after fixing the problems

Test locally first (10 attempts only)

Homework Feedback

Avoid confusing naming

What does this function do?

```
int minimum(const int temperatures[], int len) {
  int min = 0;
  for (int i = 0; i < len; i++) {
    if (temperatures[i] != NO_VALUE && min > temperatures[i]) {
      min = temperatures[i];
    }
  }
  return min;
}
```

→ If I see my function only without the context, is the name descriptive?

→ Function should ideally be without any side-effects

- Prefer parameters/return value over side-effects
- Prefer descriptive name over a comment
- Comment the side-effect

Homework Feedback

Avoid using magic numbers

```
int cl = 0;
if (cl == 3 || cl == 0) {
  d = -d;
}
```

Homework Feedback

Avoid using magic numbers

```
int current_led = 0;
if (current_led == 3 || current_led == 0) {
  direction = -direction;
}

…

if (delta >= 300)

…

constexpr int LedIndex[4] = {
  LED1_PIN, LED2_PIN,
  LED3_PIN, LED4_PIN
};
```

The only change if adding more LEDs

```
constexpr int LED_COUNT = 4;
constexpr int FIRST_LED_INDEX = 0;
constexpr int LAST_LED_INDEX = LED_COUNT - 1;
constexpr int INITIAL_LED = FIRST_LED_INDEX;

constexpr int STEP_INTERVAL_MS = 300;

constexpr int LedIndex[] = {
  LED1_PIN, LED2_PIN,
  LED3_PIN, LED4_PIN
};
```

Let the compiler fill in the correct constant

→ **Most numbers (other than 0, 1) should be constants**

Homework Feedback

Do Not Repeat Yourself (DRY)

```
void one_on() {
  all_off(); digitalWrite(LED1_PIN, ON);
}
void two_on() {
  all_off();
  digitalWrite(LED2_PIN, ON);
}
void three_on() {
  all_off();
  digitalWrite(LED3_PIN, ON);
}
void four_on() {
  all_off();
  digitalWrite(LED4_PIN, ON);
}

void show_active_led() {
  if (active_Led == 0) one_on();
  else if (active_Led == 1) two_on();
  else if (active_Led == 2) three_on();
  else four_on();
}
```

Visually similar

What if we have N LEDs?

```
void turn_on(int led_id) {
  digitalWrite(LEDS_PIN[led_id], ON);
}

void turn_off(int led_id) {
  digitalWrite(LEDS_PIN[led_id], OFF);
}

void show_active_led() {
  turn_off(active_Led - 1);
  turn_on(active_Led);
}
```

→ Wrap visually similar things into functions

→ Would my code scale with minimal changes

- 4 LEDS → N LEDs, 3 buttons → N buttons
- Minimal ~ a change of a constant, not C&P

Homework Feedback

Noneffective LED OFF

```
void clear_leds() {
  for (int i = 0; i < n_leds; i++) {
    digitalWrite(pins[i], OFF);
  }
}
```

```
void show_active_led(int old_led, int active_led) {
  turn_off(old_led);
  turn_on(active_led);
}
```

→ Is your code efficient with a large N?

Homework Feedback

“Using int variable instead of bool”

```
void spad() {
  if (!r) {
    if (p == L) {
      r = !r;
      p -= 1;
    } else {
      p += 1;
    }
  } else {
    if (p == F) {
      r = !r;
      p += 1;
    } else {
      p -= 1;
    }
  }
}
```

Homework Feedback

“Using int variable instead of bool”

```
void setPinAndDirection() {
  if (!reversed) {
    if (pin == LAST) {
      reversed = !reversed;
      pin -= 1;
    } else {
      pin += 1;
    }
  } else {
    if (pin == FIRST) {
      reversed = !reversed;
      pin += 1;
    } else {
      pin -= 1;
    }
  }
}
```

Visually similar

```
void setPinAndDirection() {
  if (pin == FIRST) {
    direction = 1;
  } else if (pin == LAST) {
    direction = -1;
  }

  pin += direction
 }
```

Homework Feedback

Useful approach - adding one more indirection

```
constexpr int LEDS[] = { LED1, LED2, LED3, LED4 };
constexpr int LED_PATTERN[] = { 0, 1, 2, 3, 2, 1 }; // Indexes to LEDS array

void turn_pattern_next(int pattern[], int pattern_length, int &curr_idx) {
  turn_off(LED_PATTERN[curr_idx++ % pattern_length]);
  turn_on(LED_PATTERN[curr_idx % pattern_length]);
}
```

Doesn't scale with number of LEDs

What is this?

What is this?

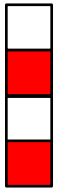
→ Can I simplify my code by putting things into an array or/and a loop?

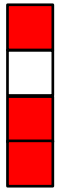
Bit Encoder

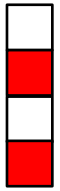
Task 1: Bit-wise LED Encoder

Write a function that takes an integer and changes the LED states according to its lower 4 bits

- 0 ~ LED is off, 1 ~ LED is on

5 = **0101**b = 

11 = **1011**b = 

21 = 1**0101**b = 

Useful to remember

- Bitwise operators: `&` (AND), `|` (OR), `~` (negation)
- Bit shifts `<<`, `>>`

Test it (try to display some constant values)

Continuous Counting

Task 2: Continuous Counting

Create a counter that continuously increments itself (modulo 16)

- With reasonable interval (e.g., 1s)
- Use the same approach as in Lab02 (avoid `delay()` or other attempts to block the `loop()`)

Display the value of the counter on LEDs

- Using the function from Task 1

Buttons

Setup

```
pinMode(button1_pin, INPUT)
```

- Actually, this should be the default, but do it anyway (makes it clearer)

Get button state

```
digitalRead(button1_pin)
```

- Inverse (pullup) logic (`false` ~ button is pressed)

Need to be checked periodically in the main loop

- Note (no need to address this): buttons not perfect (creates a short oscillation effect - as if the button is pressed and released in quick succession a few times).

Buttons Controlling Counter

Task 3: Increment when the button is pressed

Update the code so the counter is incremented only when button #1 is in a pressed state (is being held down)

The speed of the counter updates should remain the same

Small improvement

The counter is incremented once at the exact moment the button is pressed (changes its state from up to down) and then it continues in regular intervals

- The beginning of the interval starts when the button goes down

Buttons Controlling Counter

Task 4:

Create an alternate behavior for the button

- But don’t throw your code away yet

Stop/remove the continuous increment of the counter

The counter should increment only when button #1 is pressed (once)

- I.e., when the button state changes from not pressed to pressed
- Keep a copy of the last known state of the button in the global variable to find out when the state is changing

Buttons Controlling Counter

Task 5: Decrement

Make the second button work similarly to the first one,
but it will decrement the counter

Note: How do you decrement in modulo arithmetic nicely?

- `x = (x + 15) % 16`

Side task

Make the third button add +5 to the counter

Do you see the pattern (all three buttons are doing almost the same)?

- If your code seems redundant, simplify it!

Smart Buttons (ReCodex)

Task 6: Making buttons smarter
Buttons will still increment/decrement the counter when pressed

If the button is held for a while it starts to increment/decrement the counter continuously

- There should be 2 delay constants – how long before the continuous update starts and how fast it proceeds
  (e.g., it starts after 1s and increments every 300ms)
- Do not use `delay()`!

Remember there is more than 1 button

- You need to time the operations of each button independently

Think about reusability!

- Button handling might be required in future assignments

Debugging

Debugging embedded devices is tough

We can use the serial link for dumping (printing) debug info

```
void setup() {
  Serial.begin(9600);
}


void loop() {
  // ...
  Serial.println("Hail the computer...");
  Serial.println(analogValue, HEX);
}
```

Unforgivable Curse #3 - DRY

Make the buttons implementation re-usable (remember DRY)
Might prove handy for subsequent assignments

Hints
Do not copy-paste your code

- …unless you do it only once and perform major changes in the copy
- …you just need to keep a copy so you can experiment/debug the original

Use functions for the same logical pieces

Use loops

- Possibly in combination with (constant) arrays

Some cases cannot be avoided (without knowing more complex C++ features)

- At least try to minimize the duplicated parts

More Practice

Extension 1: Alternating behavior

- Make the third button switch between the three implemented strategies of updating the counter (i.e., of the behavior of the first two buttons)
  - Represented by tasks 3, 4, and 6

Extension 2: Larger counter

- Allow the counter to grow from 0 to 256
- LEDs will alternate to show the lower 4 bits and higher 4 bits of the counter in an interval (if no button is pressed)
  - E.g., alternating every 1 second or when button #3 is pressed

Extension 3: Create a piano

Lab 02

02/03/2026+09/03/2026

Homework Feedback

No Plagiarisms

- Read University Ethical Guideline
- Okay to cooperate/discuss and then write it on your own

Searching min wrongly

- Initiate to `MAX_INT` or other `INVALID` value
- Or use a `bool`

```
int last_valid_temperature = 0;
int minimal_temperature = 0;
```

Homework Feedback

DRY: put repeating code into a function

```
for (int u = 0; u < -temperature; ++u) {
  printf("*");
}

…

for (int u = 0; u < min-temperature; ++u) {
  printf("*");
}
```

Homework Feedback

Split the logic into functions

- Don’t put everything into the main

Homework Feedback

Non-descriptive naming

```
int lowest(int count) { ... }
```

Homework Feedback

Non-effective use of strings

```
std::string result = "";
for (int i = 0; i < buffer; i++){
  result += ' ‘;
}
std::cout << result;
```

Homework Feedback

Questions?

Arduino

Open-source HW and SW project

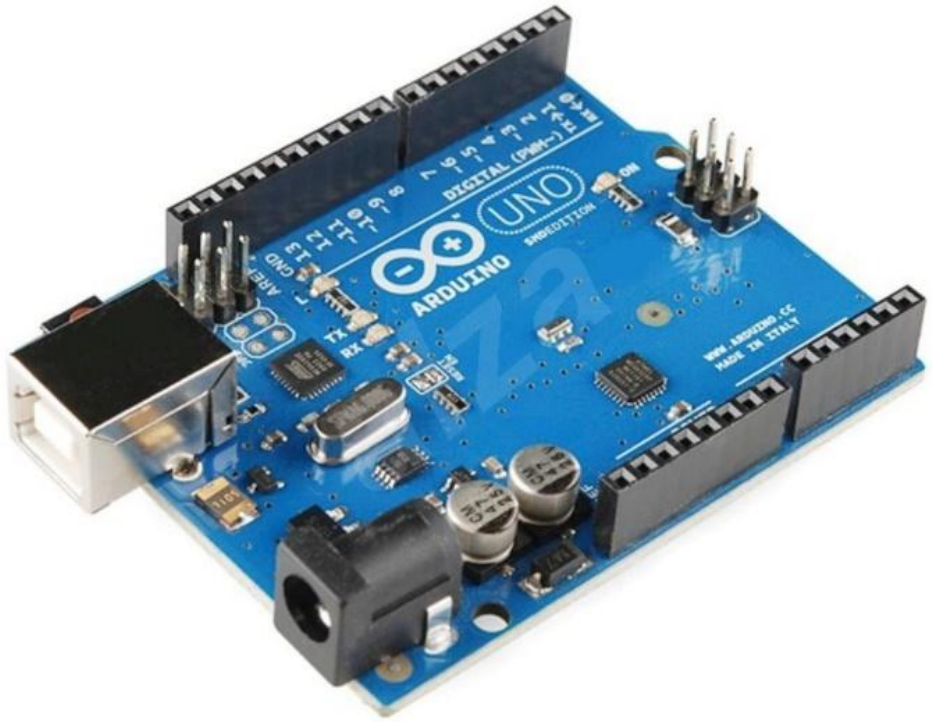
Arduino board
Base board with sets of digital and analog pins

Expansion board (shield)
Connected by pins providing various functionality

Arduino IDE (not Arduino Web Editor!)
https://www.arduino.cc/en/main/software

Arduino Uno

- CPU ATmega328P
- 14 digital I/O pins
- Of which 6 can be used as PWM outputs
- 6 analog inputs
- Clock speed 16 MHz
- FLASH memory 32 KB
- SRAM 2 KB
- EEPROM 1 KB
- USB
- DC power



Arduino IDE

Sketch
An application

Library
Shared interface and implementation

Two important functions
`void setup() { ... }`

- Called once at the start of a sketch

`void loop() { ... }`

- Called repeatedly ~1000-times per sec

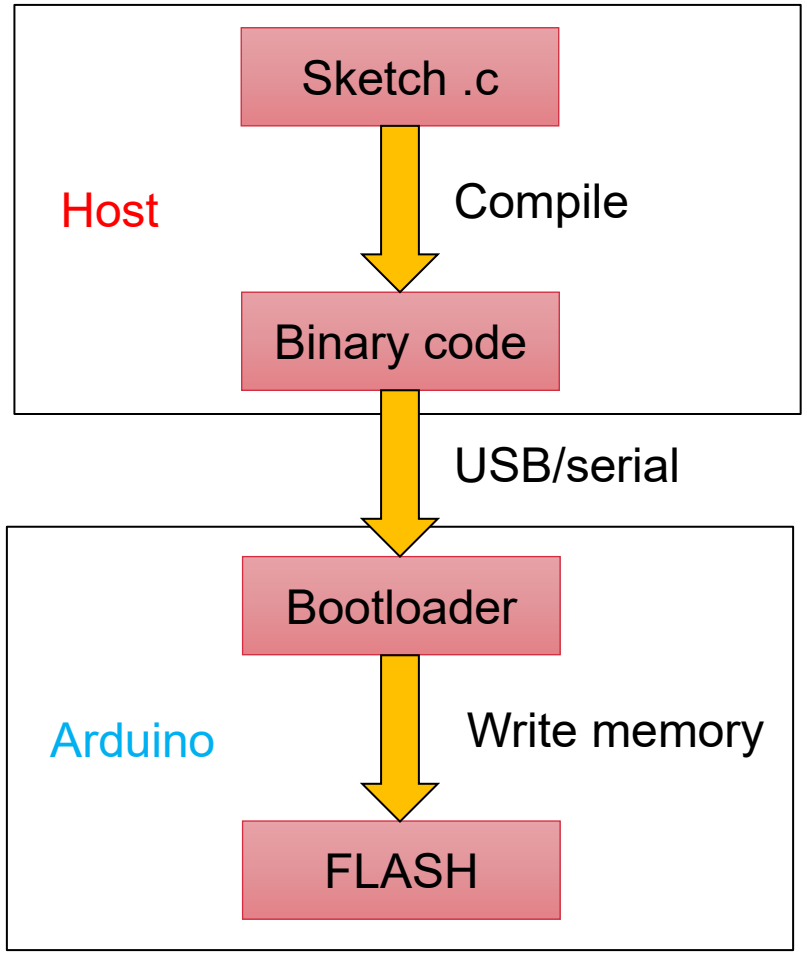
The `main()` is supplied by the framework itself

```
// Arduino main.cpp
int main(void){
    init();
    ...

    setup();

    for (;;) {
        loop();
        if (serialEventRun)
            serialEventRun();
    }
    return 0;
}
```

Executing Sketch on Arduino

Compilation and Execution

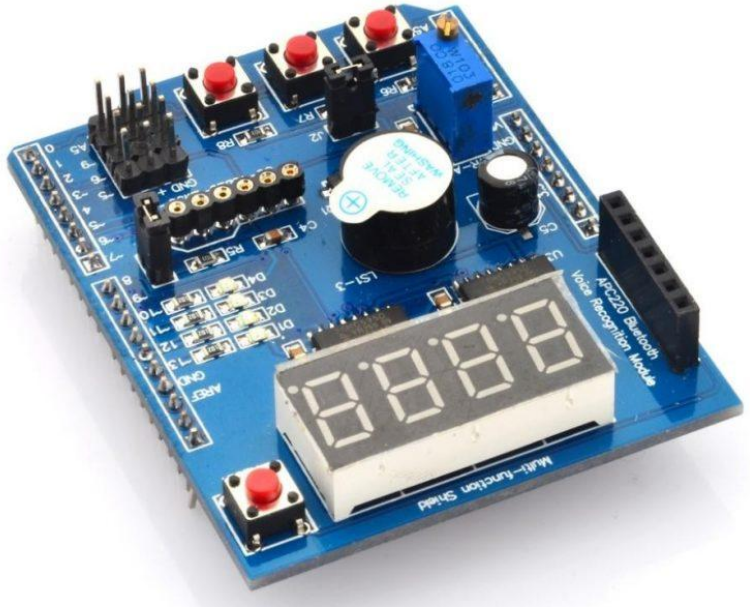
- The sketch is compiled by a compiler in Arduino IDE for the target CPU
- The resulting binary code is uploaded via USB/serial to an Arduino board and the board is reset
- EEPROM on the board contains a boot loader, which is executed after the board reset
- If there is some USB/serial payload after the reset, it is read by the boot loader, stored in the FLASH memory on the board, and executed
- If there is no payload, the boot loader executes the already stored sketch in the FLASH memory



Funshield

Simple “interactive” shield

- 3 buttons
- 4 LEDs
- 4-digit display
- 1 buzzer – will not be used, too noisy
- 1 potentiometer – will not be used, too small
- More info – follow the URL in the “Links” tab



Funshield library

- Download it from the lecture web (Labs)
- Sketch > Include Library > Add .ZIP library… (once)
- Sketch > Include Library > Funshield (every project)

Work: Blink a LED

Initialize a LED pin

- `pinMode(pin, OUTPUT/INPUT)`
- Pin constants are in `funshield.h`
- Use constants instead of literal numbers!

Set LED pin state

- `digitalWrite(pin, HIGH/LOW)`
- Actually `LOW` value means the LED is on (see `ON`/`OFF` in funshield)

Make 1st LED blink

- The main loop is too fast, use `delay(ms)`

Work: Blinking without Blocking

Avoid using **`delay()`** – replace with timing

`unsigned long millis()`

- number of milliseconds passed from the start

You can use it to monitor the passage of time

Put into a function

- Save actual ms time
- Perform a check in every loop on how much time has passed, once it exceeds the desired threshold (delay)…
- More than 1 ms may pass between two loops or the same value may be returned by two subsequent calls to **`millis()`** – no telling for sure (depends on the amount of work being done inside **`loop()`**)

How long does it take for the counter to overflow?

- And how can we deal with it (if we measure only the diff between consecutive loops)?

Work: Beyond Blinking

Work: Blink all LEDs together

- Do not forget to initialize all four LEDs in the setup
- Instead of just one LED, turn them all ON/OFF (based on your timing from the previous step)

Keep your code DRY

- DRY = Do not Repeat Yourselves
- Do not copy/paste code
- When the same code is executed multiple times use for-loops (and arrays when necessary)
    - Remember previous labs!

Beyond Blinking

Work: Create animated pattern

- Blink LEDs sequentially (snake pattern)
  - First to last, exactly 1 LED is always on
- Alternatively, create a different pattern (e.g., ping-pong)

Work Extension: Create a longer snake

- E.g. for length 3, the pattern, and the sequence for LEDs turned on will be:

  ____ ___o __oo _ooo ooo_ oo__ o___ ____

Work Extension: Anti-aliased snake using PWM (Pulse-Width Modulation)

- Last LED is dimming whilst the following LED is lightening up…

Submitting in ReCodEx

Moccarduino: Arduino (and Funshield) API-based emulator

- Emulates API IDE at the function level, not the device itself
- Designed for offline batch testing (like in ReCodEx)
- Some output variations are tricky to test, stick to the specification!

Minor differences in compilation and execution

- We keep a list (on the GitHub readme page)

Use the funshield.h library!

- Do not create arbitrary constants, and do not rely on low-level features

If your code works on Arduino and you are 100% sure it is correct, but it does not pass in ReCodEx, contact us

Rule 1: Code Decomposition

Structure your code well, divide it into functions

- When a block of code is used multiple times
- When a name needs to be assigned to a block of code
- A piece of code is too long and hard to read
- You wish to encapsulate local variables/arguments to avoid naming collisions

Indicators that you are not doing it right…

- The code is much longer and there is no apparent benefit
- Created function(s) have way too many arguments (or use global variables extensively)
- You have a hard time coming up with the name for a function

A good function has only one purpose and is self-contained

- Pure functions – no side effects, dependent only on argument

Rule 2: Use Constants

Use constants instead of value literals

- A constant bears a name (more readable)
- It is easier to change the value (if necessary)
  - Especially, when used multiple times in code

Do not over-do it

- Do not create constants for obvious values without meaning

```
constexpr int zero = 0;
```

- If you do not have a name and the value will never change

```
float fahrenheit_to_celsius(float f) {
    return (f - 32.0f) * 5.0f / 9.0f;
}
```

Questions?

Lab 1

16/02/2026

Communication

Be proactive

Web

https://fan1x.github.io/teaching/2526-pocsys

https://www.ksi.mff.cuni.cz/teaching/nswi170-web/

Mattermost

nswi170-compsys-faltin

Mail

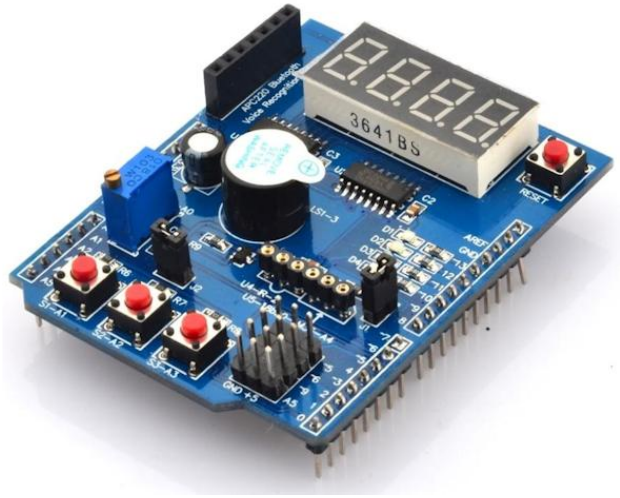
tomas.faltin@matfyz.cuni.cz

Labs

Motivation

Hands on hardware and computer systems

See a different language, other than Python



Lectures

C language basics

Operating systems, compilers, …

Labs (1/14 days)

Programming in C/C++

Programming Arduino

Labs and lectures are independent but complement each other

Labs Credit

Homeworks

Recodex: https://recodex.mff.cuni.cz

Submitted **100%** correct solution **in time**

1 week ~ solution + 1 week ~ reviews

1+N points (where N=1..6)

Homeworks build on top of each other

Labs exam

A final lab credit exam

A variation on homeworks

Credit: Correct homeworks + >= 13 points + lab exam

Programming Support

Language - C/C++
Close to HW, statically typed, compiled,

Editors
coliru.stacked-crooked.com (online web editor+compiler)
https://godbolt.org (online web editor+compiler)
VSCode, VS Studio, … (editors)
GCC, Clang, … (compilers)

Next week
Arduino IDE - www.arduino.cc/en/main/software

Master your editor – compilation + debugging + …

Playtime ☺

Hello World

```
#include <stdio.h>

void greet() {
    printf("Hello world\n");
}

int main() {
    greet();
}
```

Includes the library containing the printf

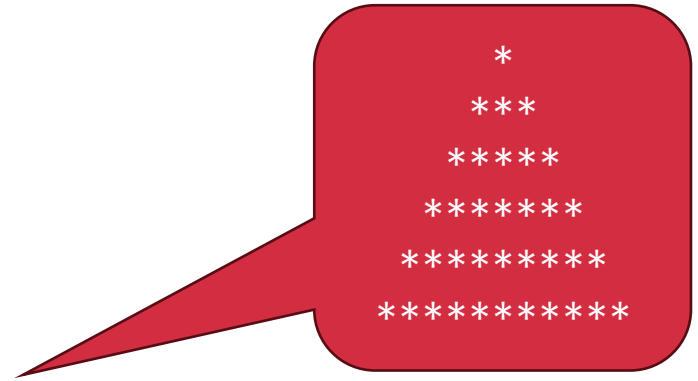
A return value type

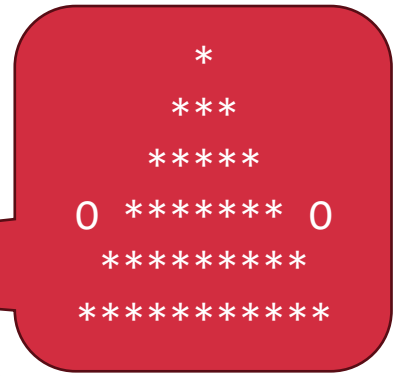
Semicolon

Program's entry point – the first function to be called

Exercise 1

1. **Hello World**
2. **Draw a rectangle of size MxN**
3. **Draw a tree of size N**
4. (3) + allow decoration (e.g., every N)
5. (4) + allow step variation (e.g., every N)





Use online IDEs (or locally configured IDE)
https://coliru.stacked-crooked.com
https://godbolt.org

```
#include <stdio.h>

void greet() {
    printf("Hello world\n");
}

int main(){
    greet();
}
```

Code Guidelines

Assume we are in a **big team** working on a **big project**
→ Working program is not enough
→ Code should be maintainable (=readable)

Six unforgivable courses
https://teaching.ms.mff.cuni.cz/nswi170-web/pages/labs/coding/

NPRG043: Best practices in programming
NPRG024: Design patterns

Rule 0: Consistency

Use English naming
☹ `cas, tlacitko, currCas`
☺ `time, button, currentButton`

Consistency with your/team's code
`ledState vs. Led_state vs. LEDState`

Use descriptive names
☹ `data, stuff, var1`
☺ `buttonState, sensor_value`

No abbreviations
☹ `mv, tmpx, dly`
(☺ `idx, i (in for loops)`)

Self-check
☑ English
☑ Same naming style
☑ My colleague immediately understands my code

Rule 0: Consistency - Example


```
constexpr  int ButtonAr[3] = {button1_pin ,button2_pin ,button3_pin };
constexpr  int PocetButt = sizeof(ButtonAr) / sizeof(ButtonAr[0]);
```

Mysterious Function 1 - Arrays

```
#include <stdio.h>

int fn1(int array[], int length) {
    int res = 0;
    for(int i = 0; i < length; ++i) {
        res += array[i];
    }
    return res;
}

int main()
{
    int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int res = fn1(array, 10);
    printf("Result: %d", res);
}
```

Array of Integers

Size of the array

Semicolon

Size is deduced automatically

Any problems with the code?

Mysterious Function 1 - Arrays

```
#include <stdio.h>

int fn1(int array[], int length) {
    int res = 0;
    for(int i = 0; i < length; ++i) {
        res += array[i];
    }
    return res;
}

int main()
{
    int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int res = fn1(array, sizeof(array)/sizeof(array[0]));
    printf("Result: %d", res);
}
```

Hello Tomas

```c
#include <stdio.h>

void greet(const char name[]) {
    printf("Hello %s\n", name);
}

int main(){
    greet("Tomas");
}
```

Mysterious Function 2 - Arrays

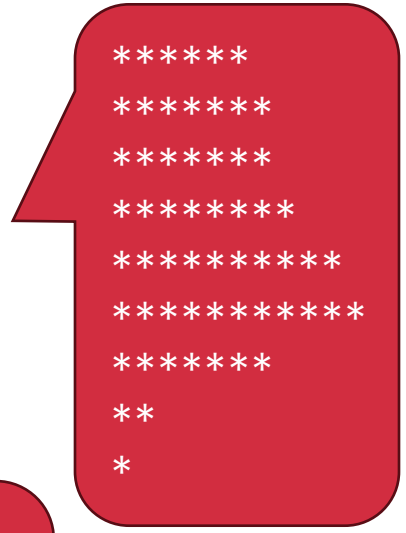
```
int fn3(int array[], int length) {
    int j = 0;
    int k = 0;
    for(int i = 0; i < length; ++i) {
        if (array[i] % 2 == 0) {
            ++j;
        } else {
            ++k;
        }
    }

    if (j > k) {
        return j;
    } else if (k > j) {
        return -k;
    } else {
        return 0;
    }
}
```

Mysterious Function 2 – Refactored*

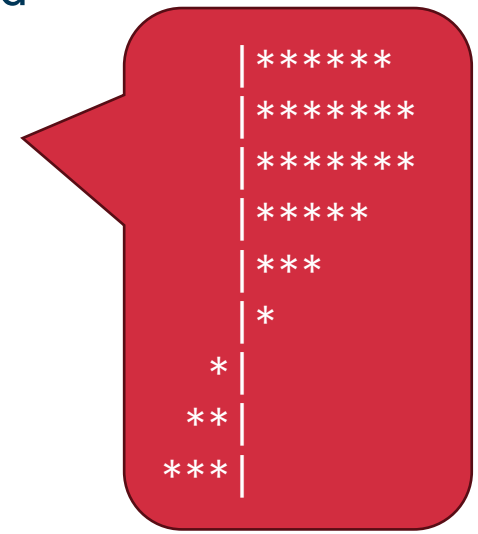
```
int count_and_compare_odd_even(int array[], int length) {
    int even_count = 0;
    int odd_count = 0;

    for(int i = 0; i < length; ++i) {
        if (array[i] % 2 == 0) {
            ++even_count;
        } else {
            ++odd_count;
        }
    }

    if (even_count > odd_count) {
        return even_count;
    } else if (odd_count > even_count) {
        return -odd_count;
    } else {
        return 0;
    }
}
```

Exercises 2

1. Print an average of an array
2. Draw a graph of positive values in an array
3. Draw negative values in an array
4. Negative values + indentation
5. Some values can be invalid



→ Celmomether (in Recodex)



Problem decomposition: Solve the problem iteratively

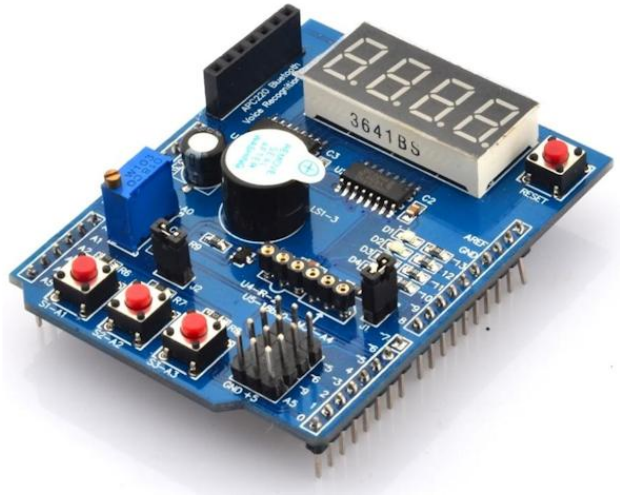
Homeworks

Homework in ReCodEx

https://recodex.mff.cuni.cz

(join our lab exercise - `SIS integration/Join groups/…`

→ Deadline: Next Tuesday 3:59



Arduino

Get one in library at Malá Strana

Install Arduino IDE: www.arduino.cc

Discussion