http://www.ksi.mff.cuni.cz/~svoboda/courses/232-NSWI170/

Practical Classes

NSWI170: Computer Systems

2023/24 Summer

Martin Svoboda
martin.svoboda@matfyz.cuni.cz

Charles University, Faculty of Mathematics and Physics

Class 1: **Basics of C and C++ Languages**

Tools Used

Mattermost

- https://ulita.ms.mff.cuni.cz/mattermost/
  - .../ar2324ls/channels/nswi170-compsys-svoboda

ReCodEx

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

Online C++ Compilers

- https://www.online-cpp.com/
- https://onecompiler.com/cpp/
- …

Arduino IDE

- https://www.arduino.cc/en/software/

E11: Hello World

Implement a simple **Hello World** application

- I.e., print *Hello World* message to the standard output
- Useful hints
  - `#include <cstdio>`
  - `int main(int argc, char** argv) { ... }`
  - `int main() { ... }`
  - `printf("...");`
  - `\n`

E12: Christmas Tree

Print a **textual tree** to the standard output

- Size of the tree is determined by its height
  - Print the corresponding number of **stars** on each level
  - Align them to the center, i.e., use spaces for **indentation**
- Sample output for a tree of size 4

```
   *
  ***
 *****
*******
```

- Decompose the code appropriately into individual functions
- Additional help
  - `printf("%c", '...');`
  - `for (int i = 0; i < height; ++i) { ... }`

E13: Integer Average

Calculate the **integer average** of given natural numbers

- Assume the input in the form of a local variable
  - `int numbers[] = { 6, 10, 12, 8 };`
  - `int size = sizeof(numbers) / sizeof(numbers[0]);`
- Calculate the average value at first
- Print it to the output as the corresponding number of **stars**
  - E.g.: `*********`
- Suggested interface

```
int average(
    const int numbers[], int count
) { ... }
```

E14: Sliding Average

Calculate **sliding averages** of given natural numbers

- Assume the input in the form of a constant expression
  - `constexpr int numbers[] = { 3, 8, 5, 7, 2, 5 };`
- Sample expected output
  - For the above input numbers and window size 3

```
*****
******
****
****
```

- Help
  -
```
void averages(
  const int numbers[], int count, int window
) { ... }
```

Class 2: **Arduino: Diodes**

Arduino

Arduino platform

- **Arduino UNO**
  - Motherboard, 14 digital and 6 analog pins
  - CPU ATmega328P, 16 MHz, FLASH memory 32 kB
- **Funduino**
  - Multifunction shield
  - Diodes, buttons, segment display, …
- Documentation
  - https://docs.arduino.cc/
  - https://www.arduino.cc/reference/
  - http://kabinet.fyzika.net/dilna/ARDUINO/funduino-popis.php

Arduino

Arduino IDE

- Basic control
  - Ctrl + S = file save (extension `*.ino`)
  - Ctrl + R = program compilation
  - Ctrl + U = upload to Arduino

Program structure

- Function `void setup();`
  - Executed once at startup
  - Contains various initializations
    - E.g., setting pin modes, initial values, …
- Function `void loop();`
  - Contains the actual execution code
  - Invoked perpetually, approximately 1000× per second

Arduino: Diodes

Diodes D1 to D4

- Accessible via pins 13 to 10
  - **Pin constants** `led1_pin`, `led2_pin`, …, `led4_pin`
- We will use **logical numbers 0 to 3** to reference our diodes
  - In order to achieve a higher level of abstraction
- **Program initialization**
  - Setting pin modes
  - `void pinMode(pin, OUTPUT);`
  - Explicit turning off of all diodes
- **Diode control**
  - Writing `LOW` (turn on) / `HIGH` (turn off) to a given pin
  - `void digitalWrite(pin, value);`

E21: Diode Lighting

Light up a particular selected diode

- Header file with constants needs to be attached first
  - `#include "funshield.h"`
  - https://www.ksi.mff.cuni.cz/teaching/nswi170-web/downloads/Funshield.zip
  - File `funshield.h` must be put into the project directory
- **Translation array** from diode numbers to pin numbers
  - `constexpr int diodePins[] = { led1_pin, led2_pin, led3_pin, led4_pin };`
- Encapsulate the necessary code into the following functions
  - `void diodeInitialize(int number);`
  - `void diodeChange(int number, bool state);`
- Test everything by turning on one particular diode

E22: Diode Flashing

Flash a particular selected diode

- **Represent individual diodes using objects**
  - I.e., instances of an appropriately designed class
    - It will contain not only the necessary data members, …
    - … but also encapsulates the required functionality
  - **Instances of all diodes will be kept in a global array**
    - `Diode diodes[diodesCount];`
    - Their initialization will be performed within `setup()`
- Flash the selected diode
  - Only in a naive way for now
  - `diodes[1].change((millis() / 500) % 2 == 0);`

E22: Diode Flashing

Pattern for a diode representation class

```
class Diode {
private:
  int diodeNumber_;
  bool currentState_;
public:
  void initialize(int diodeNumber) {
    ...
  }
  void change(bool newState) {
    ...
  }
  void change() {
    ...
  }
};
```

E23: Timing Control

Flash a particular selected diode (cont'd)

- Use an appropriate **timing control mechanism** this time
  - `unsigned long currentTime = millis();`
    - Returns the current **system time** in milliseconds
  - Basic idea of detecting the moment of the next event
    - `if (currentTime - previousTime >= periodLength) { ... }`
  - We actually also need to check for **time value overflows**
    - They occur after approximately 50 days
    - Trick for finding the maximal value: `~(unsigned long)0`
- Encapsulate the whole mechanism into a `Timer` **class**
  - Remember the previous event time
- Test the code again

E24: Railway Traffic Lights

Implement the railway traffic lights

- I.e., alternately light up pairs of adjacent diodes

Class 3: **Arduino: Buttons**

Arduino: Buttons

Buttons B1 to B3

- Pins `button1_pin`, `button2_pin`, and `button3_pin`
  - We want to work at a higher level of abstraction again
  - And so we will use **logical numbers 0 to 2** for buttons
- **Button initialization**
  - `void pinMode(pin, INPUT);`
- **Press detection**
  - Reading `LOW` (pressed) / `HIGH` (released) on a given pin
  - `int digitalRead(pin);`

E31: Button Pressing

Signal the button state by lighting up the corresponding diode

- **Translation array** for button pin numbers
  - `constexpr int buttonPins[] = { button1_pin, button2_pin, button3_pin };`
- Entire functionality will be encapsulated into our own class
  - Similarly as in the case of diodes

E32: Diodes Control

Change the diode state by pressing the corresponding button

- I.e., turn on / off the given diode
  - It does not matter for how long the button will be pressed

E33: Button Bouncing

Solve the problem with bad detection of button pressing

- It is caused by mechanical features of buttons
  - They can generate **short bounces**
- We therefore simply **filter out** very short state changes
  - I.e., we ignore them entirely
  - In particular, let us assume an interval of, e.g., 10 ms
- We also refactor the existing code
  - Function for **detection of press / release event occurrences** will be detached and separated from **queries on such events**
    - That will enable even repeated queries within just one execution of the main `loop`

E34: Binary Decomposition

Display the value of an incremented counter using diodes

- Counter starts at 0 and increments by 1 every 1 second
  - Permitted counter values are only within the interval 0 to 15
    - On overflow, we reset it back to 0
- Always **show the lowest 4 bits of the current number**
  - Bit 1 turns a given diode on, bit 0 turns it off
  - E.g., for a number $5_{10} = 101_2$, display `0101`
    - I.e., diode number 0 does not light, 1 yes, 2 no, 3 yes
  - Little help
    - Bitwise conjunction `x & 1`, bitwise shift `x << 1`
- Encapsulate the entire counter into a separate class
- Pressing button B1 manually resets the current value to 0

Class 4: **Arduino: Display I**

Arduino: Serial Line

Serial line

- **Initialization** of bidirectional connection
  - Our program: function `setup`
    - `Serial.begin(9600);`
  - Arduino IDE: Tools > Serial Monitor
    - Set the same speed
- **Sending text**
  - Function `Serial.print(…)` or `println(…)`
    - Different variants for numbers, symbols or whole strings

E41: Simple Timer

Print the elapsed time using the serial line

- I.e., send its value regularly from Arduino to the computer
  - Do that every second
  - Truncate the value to whole seconds

Arduino: Display

Segment display

- **Pins** `latch_pin`, `data_pin`, and `clock_pin`
  - Initialize them in mode `OUTPUT`
- Process of **displaying a specific glyph**
  - Close the latch
    - `digitalWrite(latch_pin, LOW);`
  - Send the **glyph mask**
    - `shiftOut(data_pin, clock_pin, MSBFIRST, glyphMask);`
  - Send the **position mask**
    - `shiftOut(data_pin, clock_pin, MSBFIRST, positionMask);`
  - Open the latch
    - `digitalWrite(latch_pin, HIGH);`

Arduino: Display

Segment display (cont'd)

- **Glyph representation**
  - `byte glyphMask = 0bHGFEDCBA;`
    - State of each segment needs to be described
    - Bit `0` (turn on), bit `1` (turn off)
    - Mapping of segments: from the upper one (`A`) in a clockwise direction, then the middle bar (`G`), finally the decimal point (`H`)
- **Position representation**
  - `byte positionMask = 0b0000LKJI;`
    - Positions are assigned numbers 0 (`L`) to 3 (`I`) from right to left
    - Bit `0` (inactive), bit `1` (active)
    - Multiple positions can in fact be activated at a time
- **Display clearing** (during the initialization)
  - Glyph with a mask `0b11111111` at positions `0b00001111`

E42: Display Control

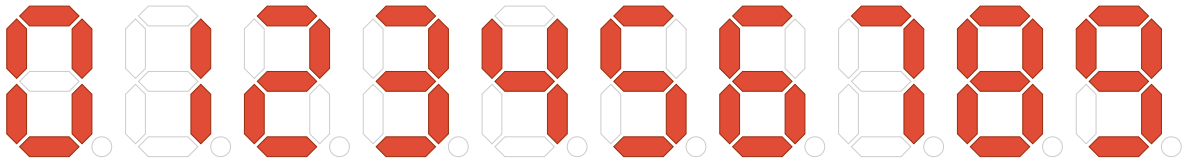
Display a given glyph at a particular display position

- Glyph itself will be specified by its mask
- Position by its logical number

E43: Displaying Digits

Display a given digit at a particular display position

- Construct glyph masks for individual digits first
  - ```
    constexpr byte digitGlyphs[] = {
      0b11000000, // 0
      ...
    };
    ```
  - Put them into a **translation array** from digits to masks



- Test everything experimentally
  - On a selected position, display a digit corresponding to the lowest order of the current time in seconds
    ```

E44: Single-Digit Counter

Display the value of a single-digit keystroke counter

- It can therefore only hold values from 0 to 9
  - Show the current value at one selected position
    - It will be position 0 at the beginning
- Counter is controlled by buttons as follows
  - Button B1: counter **incrementation**
  - Button B3: cyclic **position change** (moving it by 1 to the left)
  - Only simple presses without repetitions are assumed

Class 5: **Arduino: Display II**

E51: Displaying Numbers

Implement a display extension for displaying whole numbers

- Non-negative integers from 0 to 9999 are assumed
  - Displayed number will be aligned to the right
  - For now, we will also preserve leading zeros
    - E.g.: `0025` for number 25
- Use the idea of **time multiplexing**
  - We activate only one position in each loop iteration
- Implement the extended display using the **inheritance**
  - `class NumericDisplay : public Display { ... }`
- Chain the call of the basic display initialization function
  - `Display::initialize();`

E52: Negative Numbers

Extend our numeric display to support also negative numbers

- I.e., we will now consider numbers from -999 to 9999
  - Symbol − is shown immediately before the first significant digit
- We also stop displaying **unnecessary leading zeros**

E53: Simple Timer

Display the current time on the display

- Show this time in seconds with accuracy to 1 decimal place
  - E.g.: `0.0` or `12.3`
  - Number of the required decimal places will be configurable
    - None or decimal dot at positions 0 to 3
- Displaying **decimal dots**
  - Extend our existing function for displaying digits
  - Multiple masks can mutually be combined using a bitwise `&`

E54: Extended Counter

Show the current value of an improved counter on the display

- Counter can hold valid values from -999 to 999
  - In the event of an overflow, the counter stops at the specified min / max value and will no longer decrease / increase
- Counter will be controlled using buttons
  - Buttons B1 and B2: counter **incrementation** / **decrementation**
  - Button B3: cyclic **position change**
- Change of value always takes place by +/- 1 in a given order
  - I.e., +/- 1, 10 or 100 depending on the currently active position
- **Active position** will be marked using the decimal dot

Class 6: **Arduino: Display III**

E61: Displaying Characters

Extend our display to support displaying selected characters

- Specifically, we want to work with the following characters
  - **Letters** of the English alphabet (case-insensitive)
    - Glyph masks are in the assignment starter pack in ReCodEx
  - **Digits** `0` to `9`
  - **Space** `_` for any white character
  - Some special distinct glyph for all other unknown characters
- Let us assume, e.g., the following interface
  - `void showChar(char symbol, int position)`
- Useful functions and tricks
  - `isAlpha`, `isDigit`, `isSpace`, `isUpperCase`
  - `symbol - 'A'` and similarly to calculate glyph indices
- Experimentally test the newly added functionality

E62: Displaying Text

Extend our display to support displaying text strings

- We assume strings of (maximal) length 4
  - Strings will be aligned to the left
    - Spaces will hence be added on the right if necessary
  - Longer strings will be truncated, excessive characters ignored
- Use the inheritance again
  - `class TextDisplay : public Display { ... }`
- Tricks for working with strings
  - `char* p` vs. `const char* p`
  - `*p != '\0'`
  - `*p++`
- Use the idea of time multiplexing again
- Experimentally test the newly added functionality

E63: Running Text

Implement a mechanism for displaying running text messages

- Let us assume only a fixed text string for now
  - Its length can be arbitrary, even zero
  - We always show a window of its 4 current characters
  - We start with just the first symbol located on the very right
  - We then move the window to the left at regular intervals
  - 4 separating spaces will be added beyond the string end
  - Having finished, we terminate and wait for another string
- Provide the following public interface
  - `void setText(const char* string);`
  - `bool finished();`
- Experimentally test the newly added functionality

E64: Running Messages

Extend the previous mechanism for displaying multiple messages

- These messages will be defined using a constant array for now
  - ```
    constexpr char* inputMessages[] = {
      "Hello World",
      ...
    };
    ```
- Display them in a cyclical manner, one after the other
    ```