Interactive Automata
Initialising Arrays

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

```c
int leds[7];
void setup() {
  leds[0] = 13;
  leds[1] = 12;
  leds[2] = 11;
  leds[3] = 10;
  leds[4] = 9;
  leds[5] = 8;
  leds[6] = 7;
}
```

While-Loops

```c
i = 1;
while (i < 5) {
  Serial.println(i);
  i = i + 1;
}
Serial.println("END");
```

Executed Instructions
```
i = 1;
Serial.println(i);
i = i + 1; // 2
Serial.println(i);
i = i + 1; // 3
Serial.println(i);
i = i + 1; // 4
Serial.println(i);
i = i + 1; // 5
Serial.println("END");
```
Today’s Topics

- Strings
- Interaction
- Automata
A Purchase

Select Product → Insert Coins → Dispense Product
Selecting the Product

- Show a selection (menu) of available products
- The user navigates the products with buttons
- In each case, the current price is displayed
A Purchase

Select Product → Insert Coins → Dispense Product

Select Product

Insert Coins

Dispense Product
Inserting Coins

• Recognize coins (0,10€–2,00€)
• Deduct from the amount after insertion
  
  Insert
  Coins

• Show pending amount
• Repeat until pending amount = 0,00€
A Purchase

Select Product → Insert Coins → Dispense Product
Dispensing the Product

- In our case: turn on LED
Interaction
LCD Display

- For interaction with customers:
  - Show price
  - Show pending amount
Plan

- We connect an LCD Display
- We connect buttons…
  - …for selecting the product
  - …as sensors for coin insertion
- We lead the buyer through the purchase
Connecting the LCD
Connecting the LCD
LCD Library

• A library is a collection of functions for some common goals

• The LiquidCrystal library enables the user to communicate with a connected LCD

• To use the library, it must first be included into our program

```cpp
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```
Setting up the LCD

- This code sets up an LCD object, whose function we can then use

```c
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
```

- 0x27 is the I2C Address of the LCD Module
- The two other parameters represent the number of characters of the LCD (16x2)
LCD Output

• lcd.print() prints a text or a value on the LCD

```cpp
void setup() {
    lcd.init();
    lcd.backlight();
    lcd.print("Hello, world!");
}
```
Cursor

- The position of the cursor determines where text will be printed next
- Starting at top left; moved with every output
- Similar: cursor in word processing
Moving the Cursor

- The function `lcd.setCursor(x, y)` moves the cursor to column x, row y.
- The top left position is (0, 0).
void setup() {
  lcd.init();
  lcd.backlight();
  lcd.clear(); // clear the screen
  lcd.print("Hello, world!");
  lcd.setCursor(7, 0);
  lcd.print("class");
}
void setup() {
  lcd.init();
  lcd.backlight();
  lcd.clear(); // clear the screen
  lcd.print("Hello, world!");
  lcd.setCursor(7, 0);
  lcd.print("class");
}
Printing the Time

```cpp
void setup() {
  lcd.init();
  lcd.backlight();
  lcd.print("Hello, world!");
}

void loop() {
  lcd.setCursor(0, 1);
  lcd.print(millis() / 1000);
}
```
Characters in C

• A single character in C is written enclosed between two single quotes:

```c
char c = 'a';
Serial.println(c);
```

• The most important use is as an array of characters (a string)

• Strings end with a special “null character”, written as ‘\0’
String

char s[] = { 'H', 'e', 'l', 'l', 'o', '!', '\0'};

or shorter

char s[] = "Hello!";

What is s[0]?
String Functions

• C offers multiple functions to handle strings:
  • strcpy() – copy a string
  • strcat() – concatenate strings
  • strlen() – determine length
  • strcmp() – compare strings
Copying Strings

- `strcpy(target, source)` copies source to target
- `target` must be large enough for source to fit

```c
char name[20];
strcpy(name, "Test");
```
## Concatenating Strings

- `strcat(target, source)` appends `source` to `target`
- `target` must be large enough

```c
char name[20];
strcpy(name, "Test");
strcat(name, "ing");
```

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>e</th>
<th>s</th>
<th>t</th>
<th>\0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concatenating Strings

- `strcat(target, source)` appends source to target
- target must be large enough

```c
char name[20];
strcpy(name, "Test");
strcat(name, "ing");
```
Determining Length

- `strlen(s)` returns the length of `s`

```c
char name[20];
strcpy(name, "Test");
n = strlen(name);  // n = 4
```
Comparing Strings

- Strings cannot be compared with ==, != etc.
- `strcmp(s, t)` compares `s` and `t`
- Return value:
  - 0 – if contents are equal
  - <0 – if `s` is alphabetically smaller than `t`
  - >0 – if `s` is alphabetically larger than `t`

```c
int u = strcmp("Anton", "Anton");
int v = strcmp("Anton", "Berta");
```
Strings as Parameters

- A string name is declared as follows as a parameter:

```c
char name[] (often also: char *name)
```

```c
void print_ten_times(char text[]) {
    lcd.print(text);
    // print nine more times
}
```
An implementation of `strcpy()` could look like this:

```c
void strcpy(char target[], char source[]) {
    int i = 0;
    while (source[i] != '\0') {
        target[i] = source[i];
        i++;
    }
    target[i] = '\0';
}
```
strcpy()

- Alternative, shorter implementation

```c
void strcpy(char target[], char source[]) {
    int i = 0;
    int j = 0;
    while (target[i++] = source[j++]) {
    }
}
```

True C experts can make this even shorter
strcat()

• An implementation of strcat() could look like this:

```c
void strcat(char target[], char source[]) {
    int i = strlen(target);
    int j = 0;
    while (source[j] != '\0') {
        target[i] = source[j];
        i++; j++;
    }
    target[i] = '\0';
}
```
strcat(target, “ing”)

```c
void strcat(char target[], char source[]) {
    int i = strlen(target);
    int j = 0;
    while (source[j] != '\0') {
        target[i] = source[j];
        i++; j++;
    }
    target[i] = '\0';
}
```

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>e</td>
<td>s</td>
<td>t</td>
<td>\0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
`strcat(target, “ing”)`

```c
void strcat(char target[], char source[]) {
    int i = strlen(target);
    int j = 0;
    while (source[j] != '\0') {
        target[i] = source[j];
        i++; j++;
    }
    target[i] = '\0';
}
```

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>T</td>
<td>e</td>
<td>s</td>
<td>t</td>
<td>i</td>
<td>n</td>
<td>g</td>
<td>\0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An implementation of `strlen()` could look like this:

```c
int strlen(char s[]) {
    int i = 0;
    while (s[i] != '\0') {
        i++;
    }
    return i;
}
```

Type of the return value: `int`

Returned value: `i`

```c
int n = strlen(target);
```
Selecting the Product

- Show a selection (menu) of available products
- The user navigates the products with buttons
- In each case, the current price is displayed
Selecting the Product

- Show a selection (menu) of available products
- The user navigates the products with buttons
- In each case, the current price is displayed
Plan

• Show product in the top line
Showing the Products

```c
int DRINKS = 3;
char *drink_name[] = { "Water", "Soda", "Beer" };

void print_drinks() {
    int pos = 0;

    for (int i = 0; i < DRINKS; i++) {
        lcd.setCursor(pos, 0);
        lcd.print(drink_name[i]);
        pos += strlen(drink_name[i]) + 1;
    }
}
```
Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Select Product

- Show a selection (menu) of available products
- The user navigates the products with buttons
- In each case, the current price is displayed
Select Product

- Show a selection (menu) of available products
- The user navigates the products with buttons
- In each case, the current price is displayed
Plan

- Show prices in bottom line
- Under each product name
- Problem: Represent prices as strings
Characters to Numbers

- The function atoi(s) transforms the prefix of the string s into an integer number
- Leading white spaces are ignored
- s remains unmodified
- No error detection

```c
n = atoi("25");
n = atoi(" 25");
n = atoi(" 25 years");
n = atoi("25years");
```
Numbers to Characters

- The function `sprintf(s, format, values...)` fills `s` with values as specified in `format`:

  %d – **decimal number**

  ```c
  char buf[128];
  sprintf(buf, "%d", 25);  // buf[] == "25"
  ```

  %s – **String**

  ```c
  sprintf(buf, "%s", "Hugo");  // buf[] == "Hugo"
  ```
Numbers to Characters

- The sprintf() format can contain more text, which will be copied as well:

```c
char buf[128];
int n = 25;
sprintf(buf, "Buy %d furs", n);
// buf[] == "Buy 25 furs"
```
Numbers to Characters

- Outputting multiple parameters is possible as well:

```c
char buf[128];
int n = 25;
int p = 600;
sprintf(buf, "%d monkeys at %d Euro", n, p);
// buf[] == "25 monkeys at 600 Euro"
```
Maximum Length

• Numbers before the format specifier determine the length of the text to be printed

```c
char buf[128];
int n = 25;

sprintf(buf, "Quantity:%5d", n);
// buf[] == "Quantity: 25"  5 Characters

sprintf(buf, "Quantity:%7d", n);
// buf[] == "Quantity: 25"  7 Characters
```
Leading Zeros

- If the length specifier starts with 0, the output is padded with zeros instead of white spaces

```c
char buf[128];
int n = 25;

sprintf(buf, "Quantity:%05d", n);
// buf[] == "Quantity:00025"

sprintf(buf, "Quantity: %03d", n);
// buf[] == "Quantity: 025"
```
Menu with Price

```c
int DRINKS = 3;
char *drink_name[] = { "Water", "Soda", "Beer" };
int drink_price[] = { 100, 150, 250 };

void print_prices() {
    int x = 0;
    for (int i = 0; i < DRINKS; i++) {
        char buffer[100];

        lcd.setCursor(x, 1);
        sprintf(buffer, "%d.%02d",
                drink_price[i] / 100,
                drink_price[i] % 100);
        lcd.print(buffer);
        x += strlen(drink_name[i]) + 1;
    }
}
```
# Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selecting the Product

- Show a selection (menu) of available products
- The user navigates the products with buttons
- In each case, the current price is displayed
Selecting the Product

- Show a selection (menu) of available products
- The user navigates the products with buttons
- In each case, the current price is displayed
Plan

- Mark currently selected product by blinking
- Selection with buttons (left, right, OK)
Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>✅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
</tbody>
</table>
# Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
</tbody>
</table>

- Use the arrow buttons to navigate through the drink options.
- Press the OK button to select your choice.
Drink Menu

0 Water  Soda  Beer
1 1.00   1.50  2.00
Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Use the arrow keys to select a drink, then press 'OK' to confirm your choice.
Drink Menu

0 Water 1 Soda 2 Beer
1 1.00 1.50 2.00
Drink Menu

- **0**: Water
- **1**: Soda
- **2**: Beer

Prices:
- **0**: $1.00
- **1**: $1.50
- **2**: $2.00
Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
</tbody>
</table>
# Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Use the arrow keys to navigate and the OK button to select your choice.
# Drink Menu

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Soda</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Use the arrow keys to select your choice and press 'OK' to confirm.

- Use the left arrow to go back.
- Use the right arrow to move forward.
- Press 'OK' to select.
Drink Menu

0. Water
1. Soda
2. Beer

0. $1.00
1. $1.50
2. $2.00

→ ← OK
Visible Cursors

- With `lcd.cursor()` we can make the cursor visible as an underscore (_)
- `lcd.blink()` makes the cursor blink
- `lcd.noCursor()`, `lcd.noBlink()` turns this off again
Showing the Selection

// Position the cursor on drink name
// drink = 0: first drink,
// drink = 1: second drink, etc.
void show_selection(int drink) {
    int x = 0;

    for (int i = 0; i < drink; i++)
    {
        x += strlen(drink_name[i]);
        x += strlen(" ");
    }
    lcd.setCursor(x, 0);
    lcd.blink();
}
Connecting the Buttons
// choose a drink and return its number
int choose_drink() {
    int current_selection = 0;
    unsigned long last_select = millis();

    show_selection(current_selection);
    while (1) {
        if (millis() - last_select > 20) {
            // right
            if (digitalRead(PIN_RIGHT) == LOW) {
                if (current_selection < DRINKS - 1) {
                    current_selection++;
                }
            }
            show_selection(current_selection);
            while (digitalRead(PIN_RIGHT) == LOW) {}
            last_select = millis();
        }
    }
}
if (digitalRead(PIN_RIGHT) == LOW) {} 
last_select = millis();
}

// left
if (digitalRead(PIN_LEFT) == LOW) {
  if (current_selection > 0) {
    current_selection--;
  }
  show_selection(current_selection);
  while (digitalRead(PIN_LEFT) == LOW) {} 
  last_select = millis();
}

// select
if (digitalRead(PIN_OK) == LOW) {
  lcd.noBlink();
  while (digitalRead(PIN_OK) == LOW) {} 
  return current_selection;
}

}
/choose a drink and return its number

```c
int choose_drink() {
    int current_selection = 0;
    unsigned long last_select = millis();

    show_selection(current_selection);
    while (1) {
        if (millis() - last_select > 20) {

            // right
            if (digitalRead(PIN_RIGHT) == LOW) {
                if (current_selection < DRINKS - 1) {
                    current_selection++;
                }
                show_selection(current_selection);
                while (digitalRead(PIN_RIGHT) == LOW) {} 
                last_select = millis();
            }

            // left
            if (digitalRead(PIN_LEFT) == LOW) {
                if (current_selection > 0) {
                    current_selection--;
                }
                show_selection(current_selection);
                while (digitalRead(PIN_LEFT) == LOW) {} 
                last_select = millis();
            }

            // select
            if (digitalRead(PIN_OK) == LOW) {
                lcd.noBlink();
                while (digitalRead(PIN_OK) == LOW) {} 
                return current_selection;
            }
        }
    }
}
```
Confirming the Selection

```java
void print_selection(int selection) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("You bought:");
    lcd.setCursor(0, 1);
    lcd.print(drink_name[selection]);
    // blink for 3 seconds
    long m = millis();
    while (millis() - m < 3000) {
        if (millis() / 200 % 3)
            lcd.backlight();
        else
            lcd.noBacklight();
    }
    lcd.backlight();
}
```
All in One

void loop() {
    lcd.clear();
    print_drinks();
    print_prices();
    int selection = choose_drink();
    print_selection(selection);

    // further functions, e.g.:
    // pay_for_drink(selection);
    // dispense_drink(selection);
}
A Purchase

Select Product → Insert Coins → Dispense Product
A Purchase

- Select Product
- Insert Coins
- Dispense Product

Done
A Purchase

Select Product → Insert Coins → Dispense Product

Done

In Exercise
A Purchase

Select Product → Insert Coins → Dispense Product

Done

In Exercise

In Exercise
Setting up the LCD

- This code sets up an LCD object, whose function we can then use

```c
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

- 0x27 is the I2C Address of the LCD Module
- The two other parameters represent the number of characters of the LCD (16x2)

Characters in C

- A single character in C is written enclosed between two single quotes:

  ```c
  char c = 'a';
  Serial.println(c);
  ```

- The most important use is as an array of characters (a string)
- Strings end with a special “null character”, written as `\0`

Menu with Price

```c
int DRINKS = 3;
char *drink_name[] = { "Water", "Soda", "Beer" };
int drink_price[] = { 100, 150, 250 };

void print_prices() {
  int x = 0;
  for (int i = 0; i < DRINKS; i++) {
    char buffer[100];
    lcd.setCursor(x, 1);
    sprintf(buffer, "%d,%d",
            drink_price[i] / 100, 
            drink_price[i] % 100);
    lcd.print(buffer);
    x += strlen(drink_name[i]) + 1;
  }
}
```