# BASTELTRUPPE 2

BASICS!

# WHAT ARE THE BASICS?

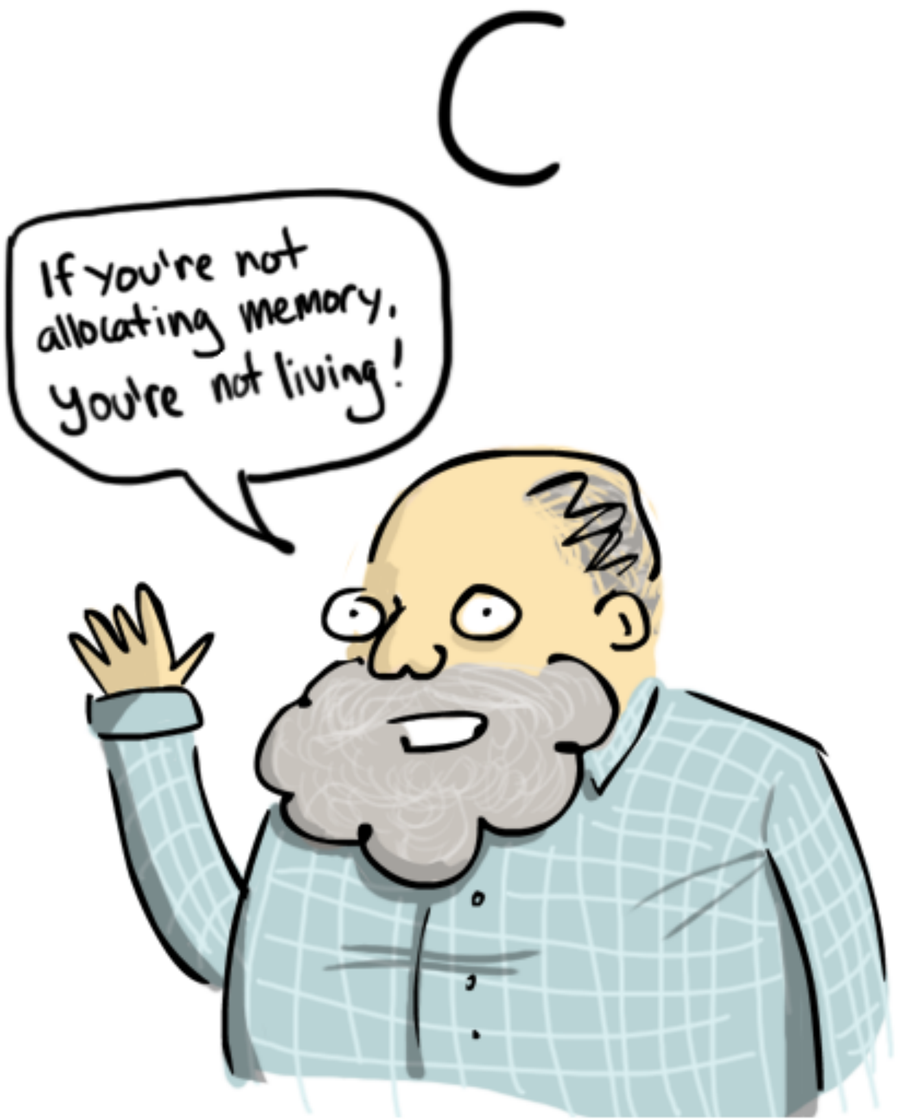
Variables

Loops

Function

It’s not as simple as you think!

# WE WILL LEARN C



# EVEN THOUGH ARDUINO USES C++

# WHY?

C++ is backwards compatible with C

- You can write C wherever you would write C++

C++ introduces a lot of flashy features

- These features are difficult to learn, hard to master

This is a beginners course, C offers plenty of features, we don’t need C++

# VARIABLES

Computers are bad at implicitly doing stuff

We want them to calculate something

They need to store some data for a short while in RAM

Store the data in a variable

(I assume you all kind of know how to program)

# PROBLEM

Programmers are bad at handling data and variables

Which data goes into which variables?

Python => (Almost) no worries

C => You need to be careful

# TYPE INFERENCE

In Python you have type inference

A = „Hello World“

B = 52.5

C = my_cool_method()

Python evaluates what kind of data you want to store and automatically does magic so that you can store your data

# STATIC TYPING

THE RIGHT WAY

In a static type system you need to declare what kind of variables you create

String s = String(„Hello World“);

float f = 52.5f;

ImageConverter i = my_cool_method();

# WHAT IDIOT WOULD USE STATIC TYPING?

Languages such as Python/JavaScript are generally not compiled but interpreted

- Programs are run in an interpreter
- Each type etc. is evaluated at runtime
- Huge performance cost

Languages such as C/C++/Java/Go/Rust are compiled

- A compiled binary is executed close to the hardware
- All types are known at runtime
- Extremely efficient, but more difficult to write

**Table 4.** Normalized global results for Energy, Time, and Memory

| Total | | | | | |
|---|---|---|---|---|---|
| | Energy | | Time | | Mb |
| (c) C | 1.00 | (c) C | 1.00 | (c) Pascal | 1.00 |
| (c) Rust | 1.03 | (c) Rust | 1.04 | (c) Go | 1.05 |
| (c) C++ | 1.34 | (c) C++ | 1.56 | (c) C | 1.17 |
| (c) Ada | 1.70 | (c) Ada | 1.85 | (c) Fortran | 1.24 |
| (v) Java | 1.98 | (v) Java | 1.89 | (c) C++ | 1.34 |
| (c) Pascal | 2.14 | (c) Chapel | 2.14 | (c) Ada | 1.47 |
| (c) Chapel | 2.18 | (c) Go | 2.83 | (c) Rust | 1.54 |
| (v) Lisp | 2.27 | (c) Pascal | 3.02 | (v) Lisp | 1.92 |
| (c) Ocaml | 2.40 | (c) Ocaml | 3.09 | (c) Haskell | 2.45 |
| (c) Fortran | 2.52 | (v) C# | 3.14 | (i) PHP | 2.57 |
| (c) Swift | 2.79 | (v) Lisp | 3.40 | (c) Swift | 2.71 |
| (c) Haskell | 3.10 | (c) Haskell | 3.55 | (i) Python | 2.80 |
| (v) C# | 3.14 | (c) Swift | 4.20 | (c) Ocaml | 2.82 |
| (c) Go | 3.23 | (c) Fortran | 4.20 | (v) C# | 2.85 |
| (i) Dart | 3.83 | (v) F# | 6.30 | (i) Hack | 3.34 |
| (v) F# | 4.13 | (i) JavaScript | 6.52 | (v) Racket | 3.52 |
| (i) JavaScript | 4.45 | (i) Dart | 6.67 | (i) Ruby | 3.97 |
| (v) Racket | 7.91 | (v) Racket | 11.27 | (c) Chapel | 4.00 |
| (i) TypeScript | 21.50 | (i) Hack | 26.99 | (v) F# | 4.25 |
| (i) Hack | 24.02 | (i) PHP | 27.64 | (i) JavaScript | 4.59 |
| (i) PHP | 29.30 | (v) Erlang | 36.71 | (i) TypeScript | 4.69 |
| (v) Erlang | 42.23 | (i) Jruby | 43.44 | (v) Java | 6.01 |
| (i) Lua | 45.98 | (i) TypeScript | 46.20 | (i) Perl | 6.62 |
| (i) Jruby | 46.54 | (i) Ruby | 59.34 | (i) Lua | 6.72 |
| (i) Ruby | 69.91 | (i) Perl | 65.79 | (v) Erlang | 7.20 |
| (i) Python | 75.88 | (i) Python | 71.90 | (i) Dart | 8.64 |
| (i) Perl | 79.58 | (i) Lua | 82.91 | (i) Jruby | 19.84 |

# BACK TO THE BASICS

## INTEGERS

There are two ways of learning integers in C++

I will entirely skip the conventional method and teach it right to you

The two important things about integers you need to know are

Are they signed?

How big numbers can you store in them?

# SIGNED?

If you have 8 bits, you have 256 slots

Either start with 0 and go up to 255

Store unsigned numbers

Use 7 bits for positive numbers and 7 bits for negative numbers

(Use the 8th bit to indicate which one it is)

-128 <-> 127

Question: Why not -128 to 128?

# HOW BIG?


What you store needs to be big enough to store your numbers

Data type too small?

- Errors happen

Data type too big?

- You waste precious resources

# {u}int_{8/16/32/64}_t

```c
#include <stdint.h>

int8_t first = 127;
uint8_t second = 255;
int16_t third = 32767;
uint16_t fourth = 65535;
int32_t fifth = 2147483647;
uint32_t sixth = 4294967295;
int64_t seventh = 9223372036854775807;
uint64_t eigth = 18446744073709551615;
```

# DOING MATH

```
uint8_t seven = 7;
uint8_t fifteen = 15;
uint8_t twenty = 20;
uint8_t twenty_two = seven + fifteen;
uint8_t eight = fifteen - seven;
```

# DOING MATH?

```
uint8_t seven = 7;
uint8_t fifteen = 15;
uint8_t twenty = 20;
uint8_t twenty_two = seven + fifteen;
uint8_t eight = fifteen - seven;
uint8_t what = seven - twenty;
uint8_t is = twenty / seven;
uint8_t going = twenty * fifteen;
int8_t on = seven - eight;
```

# FLOATING POINT NUMBERS

FLOAT OR DOUBLE

IEEE Floating Point standard

Either 32 or 64 bit precision

Why precision?

It is only an approximation

Use them as such!

```
float zero = 0.0f;
while (zero != 30.0f){
    zero += 0.01f;
        Serial.println(zero);
}
Serial.println("Done!");
```

# BOOLEAN

TRUE OR FALSE?

Store either a true or a false value

(Technically this is C++, as C doesn’t natively know boolean)

Useful for many applications

boolean wahr = true;

boolean falsch = false;

# LOOPS

Do something multiple times

while(condition){ }

while(button_not_pressed()) { delay(200); }

for(initialisation; condition; what to do after a loop){ }

for(uint8_t i = 0; i < 20; i++){ }

# FUNCTIONS

A function receives some input, calculates something with the variables in its scope and may return something

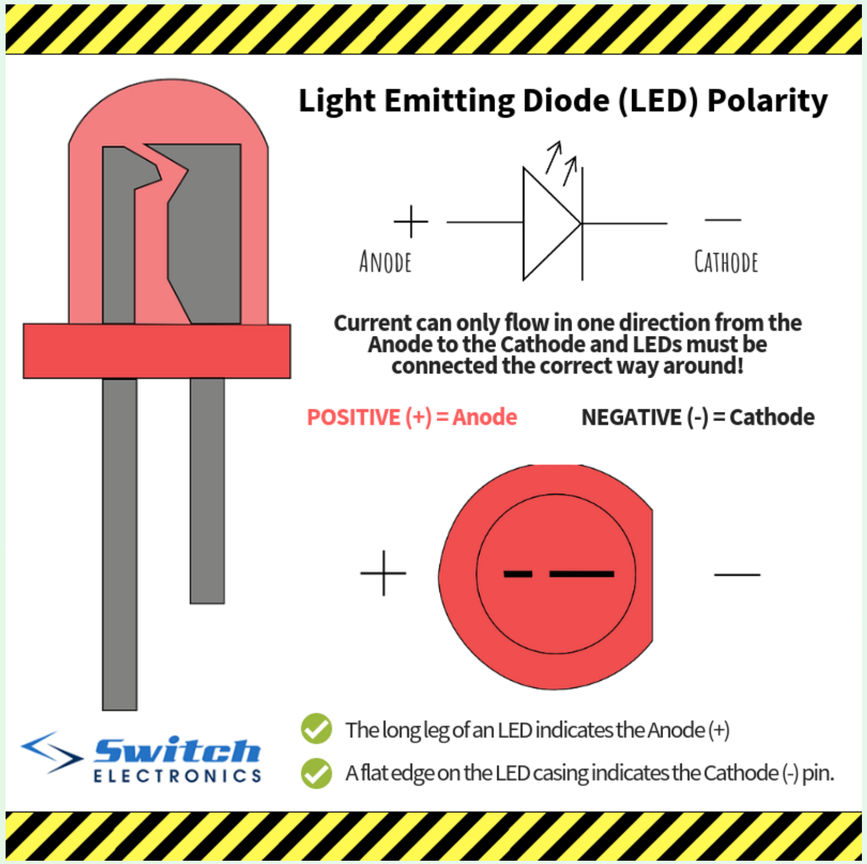
```
uint8_t i = 5;

uint16_t do_stuff(uint8_t blablub){

   uint8_t temp = 5;

   return temp + blablub + i;

}
```

# LAMPS!

LIGHT EMITTING DIODE!

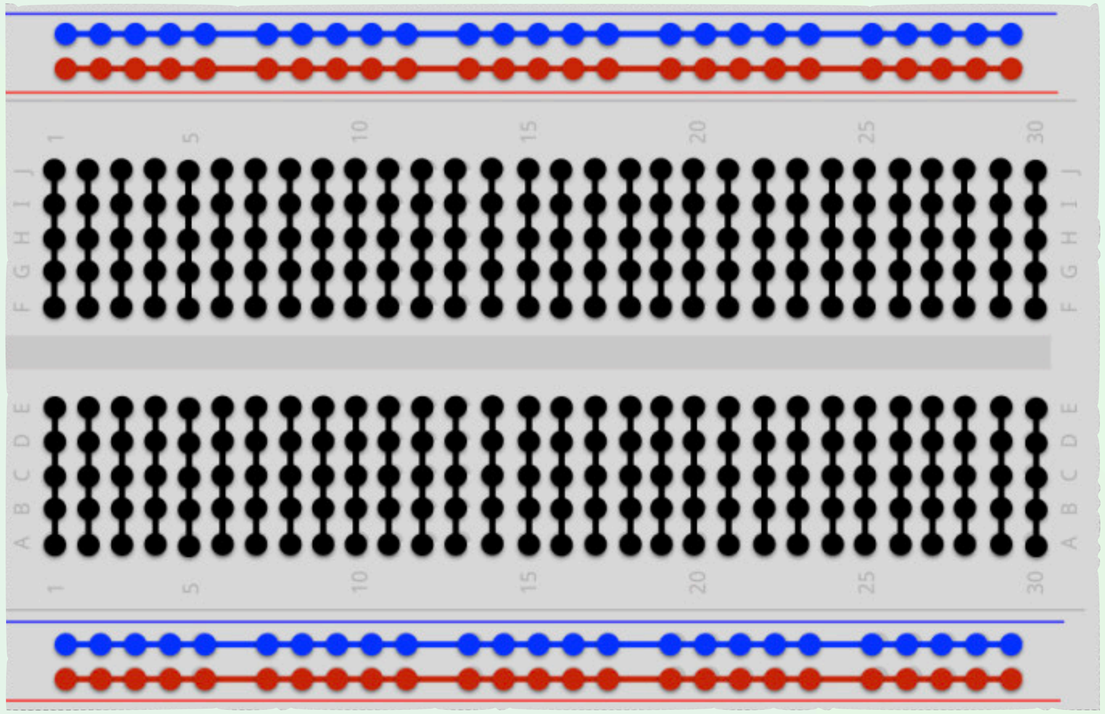
Through a ✨*magic optoelectric effect*✨ if you put electricity to the

Anode (+) of an LED, and connect the Cathode (-) to ground, the LED will begin to emit light!

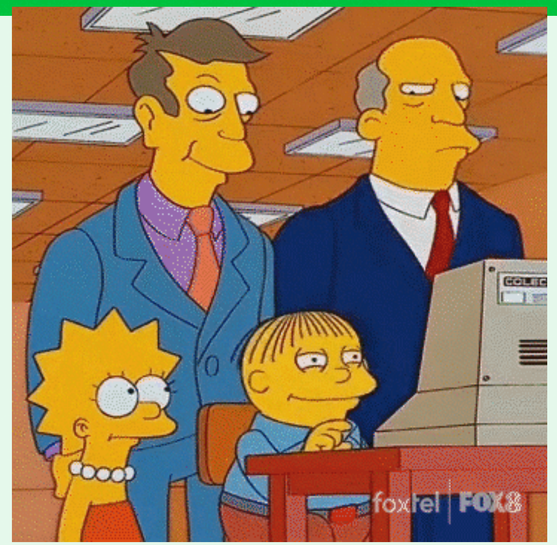



Add a resistor to resist too much electricity to pass through the LED!

# YOUR BREADBOARD!



# EXERCISES!




1) Wire up a LED on the Breadboard and to the Arduino to let it blink!

   (Ask Peter for help)

2) Write two functions. One should turn the LED on for 200 ms, one should turn it off for 200 ms

3) Write a for loop to turn the LEDs on and off

4) Use the variable of your for loop as input for your functions!

   (Hint, void turn_on(uint8_t time); )

5) Are you able to dim your LED?