

# **Arduino programming**

Microcontroller, C and C++, datatypes

**Hoe ging het vorige week?**

# TODAY

- History and context of C
- Binary counting
- Memory
- Variables
- Arrays/Pointers

**Programming?**

*The art of making a  
computer do what you  
want.*

# Levels of programming language

Low



High

Machine code

```
00101010100  
010010110
```

Human readable  
machine code

```
MOV AX, 0x2B  
MUL AX, BX
```

Human readable  
code

```
printf("%s is een  
blub", naam);
```

# Compiling = Converting



```
printf("%s is een  
blub", naam);
```

gcc.exe



```
001010101  
000100101  
10
```

# The C programming language

- What happened to A and B?
  - BCPL (1966) “Hello World!” (1967)
  - B (1969)
  - C (1972)
  - C++ (1983) 
  - C# (2001)  Different language!
- Updates:
- C++98 (1998)
  - C++01 (2001)
  - C++11 (2011)

# ARDUINO C++

- Lots of libraries and built in functions
  - <http://arduino.cc/en/Reference/HomePage>
  - Bookmark this!
  - No, really
- Program structure: setup() en loop()
- Syntactic sugar<sup>TM</sup>



# ENTRY POINT

Where does your program start?

- **Java**

- `public static void main(String[] args)`

- **C/C++**

- `int main(int argc, char *argv[])`

- **Arduino:**

- `void setup()?`

- `void loop()?`

# START THE PROGRAM

- Arduino powered on
  - program starts
  - never stops
- Reset Button
  - Restarts main program
- `1 × void setup()`
- `∞ × void loop()`



# BUT...BEHIND THE SCENES...

hardware\arduino\cores\arduino\main.cpp

```
#include <Arduino.h>
```

```
int main(void) {
```

```
    init();
```

```
    #if defined(USBCON)
```

```
    USB.attach();
```

```
    #endif
```

```
    setup();
```

```
    for (;;) {
```

```
        loop();
```

```
        if (serialEventRun) serialEventRun();
```

```
    }
```

```
    return 0;
```

```
}
```

# DATATYPES

What are datatypes?

# DATATYPES

- Remember:
  - Computers store everything in groups of 1s and 0s
  - Every placeholder for a 1 or 0 is called a bit (b(inary) (dig)it)
  - Combinations of bits make up numbers

# COMPUTER MEMORY

```
11101101101001010101101011001100111110011001
11111011011000101100111011011010010101011010
11001100111110011001111110110110001011001110
11011010010101011010110011001111100110011111
10110110001011001110110110100101010110101100
11001111100110011111101101100010110011101101
10100101010110101100110011111001100111111011
01100010110011101101101001010101101011001100
11111001100111111011011000101100111011011010
01010101101011001100111110011001111110110110
00101100111011011010010101011010110011001111
10011001111110110110001011001110110110100101
01011010110011001111100110011111101101100010
```

[illegible]



There are only 10 types of people in the world:  
those who understand binary,  
and those who don't.



# DECIMAL BASE SYMBOLS

0  
1  
2  
3  
4  
5  
6  
7  
8  
9



# DECIMAL BASE COUNTING

0	10	20	30	40	50	60	70
1	11	21	31	41	51	61	71
2	12	22	32	42	52	62	72
3	13	23	33	43	53	63	73
4	14	24	34	44	54	64	74
5	15	25	35	45	55	65	75
6	16	26	36	46	56	66	76
7	17	27	37	47	57	67	77
8	18	28	38	48	58	68	78
9	19	29	39	49	59	69	79

# DECIMAL BASE COUNTING

00	10	20	30	40	50	60	70
01	11	21	31	41	51	61	71
02	12	22	32	42	52	62	72
03	13	23	33	43	53	63	73
04	14	24	34	44	54	64	74
05	15	25	35	45	55	65	75
06	16	26	36	46	56	66	76
07	17	27	37	47	57	67	77
08	18	28	38	48	58	68	78
09	19	29	39	49	59	69	79

# BINARY BASE SYMBOLS

0

1

# BINARY BASE COUNTING

00

01

# BINARY BASE COUNTING

00

10

01

11

# BINARY BASE COUNTING

0	10	100	1000	10000
1	11	101	1001	10001
		110	1010	10010
		111	1011	10011
			1100	10100
			1101	10101
			1110	10110
			1111	.....

# BINARY BASE COUNTING

0 = 0	10 = 2	100 = 4	1000 = 8	10000 = 16
1 = 1	11 = 3	101 = 5	1001 = 9	10001 = 17
		110 = 6	1010 = 10	10010 = 18
		111 = 7	1011 = 11	10011 = 19
			1100 = 12	10100 = 20
			1101 = 13	10101 = 21
			1110 = 14	10110 = 22
			1111 = 15	.....



# COMPUTER MEMORY

```
11101101101001010101101011001100111110011001
11111011011000101100111011011010010101011010
11001100111110011001111110110110001011001110
11011010010101011010110011001111100110011111
10110110001011001110110110100101010110101100
11001111100110011111101101100010110011101101
10100101010110101100110011111001100111111011
01100010110011101101101001010101101011001100
11111001100111111011011000101100111011011010
01010101101011001100111110011001111110110110
00101100111011011010010101011010110011001111
10011001111110110110001011001110110110100101
01011010110011001111100110011111101101100010
```

# LIST OF NUMBERS

6153464468964240

Could be:

- 6, 1, 5, 3, 4, 6, 4, 4, 6, 8, 9, 6, 4, 2, 4, 0
- 61, 53, 46, 44, 68, 96, 42, 40
- 6, 15, 346, 4468, 96424, 0
- 6153, 46, 446, 6896, 4240
- Anything!

# LIST OF FIXED WIDTH NUMBERS

6153464468964240,  $w = 4$

= 6153, 4644, 6896, 4240

# LIST OF FIXED WIDTH NUMBERS

6153464468964240,  $w = 4$

= 6153, 4644, 6896, 4240

How to store 1, 2, 3 like this?

000100020003

## LIST OF FIXED WIDTH NUMBERS

6153464468964240,  $w = 4$

= 6153, 4644, 6896, 4240

How to store 1, 2, 3 like this?

000100020003

How to store 5 digit numbers like this?

e.g. 65535 Can't be done :- (

# BINARY GROUPS

- Most convenient group size?
  - 8 bits = 1 byte
  - example:
    - 00100110
    - 11110111
    - 00001000

# BYTE TRIVIA

- Why 8 bits in a byte?
  - # needed to encode a single character of text
  - Early computers used smaller bytes
  - ASCII is 7 bits, 128 characters
    - 52 Alphabet (lower and upper case)
    - 10 Numbers
    - Punctuation (.,;'{ etcetera)
    - Control characters (newline, end of file, ...)
  - But 8 is a power of 2
  - 8 became de facto standard

# COMPUTER MEMORY

00110011	10110011	11101100	11100100
10110011	11101100	10110011	01001000
01000101	01001100	01010000	00100001
00100000	01001001	00100111	01101101
00100000	01110100	01110010	01100001
01110000	01110000	01100101	01100100
00100000	01101001	01101110	00100000
01100001	00100000	01100010	01111001
01110100	01100101	00100000	01100110
01100001	01100011	01110100	01101111
01110010	01111001	00100001	00000000
00110011	10110011	11101100	11100100
10110011	11101100	10110011	10110011



# WRITING BINARY IN ARDUINO

How to tell if `10` is decimal or binary?

- Default numbers are decimal
- Binary starts with `B`: `B10`, `B1011`, `B001`


Even more bases exist:

- Hexadecimals start with `0x`: `0x1A`, `0xFF`
- Octal starts with `0`: `010`, `077`, `0123`

*That's a zero!*



*Also a zero!*



Watch out for accidental octal!

# DATATYPES IN C

- char
  - 1 byte (= -128 to 127)
- int
  - 2 bytes (= -32768 to 32767)
- long
  - 4 bytes (= -2147483648 to 2147483647)
- float
  - 4 bytes (= -3.4028235E+38 to 3.4028235E+38)\*

\*but you can't count from one to the other! You lose precision.

# DATATYPES IN C

- char
  - 1 byte (= -128 to 127)
- int
  - 2 bytes (= -32768 to 32767)
- long
  - 4 bytes (= -2147483648 to 2147483647)
- float
  - 4 bytes (= -3.4028235E+38 to -3.4028235E+38)\*

\*but you can't count from one to the other! You lose precision.

# SIGNED OR UNSIGNED

- Use one bit for sign
  - $00000001 = 1$
  - $11111111 = -1^*$
  - $11111110 = -2$
  - $11111100 = -3$
- All data types are signed per default
- use `unsigned` keyword
  - Get 1 bit higher range!

\*negative numbers are  
actually encoded differently

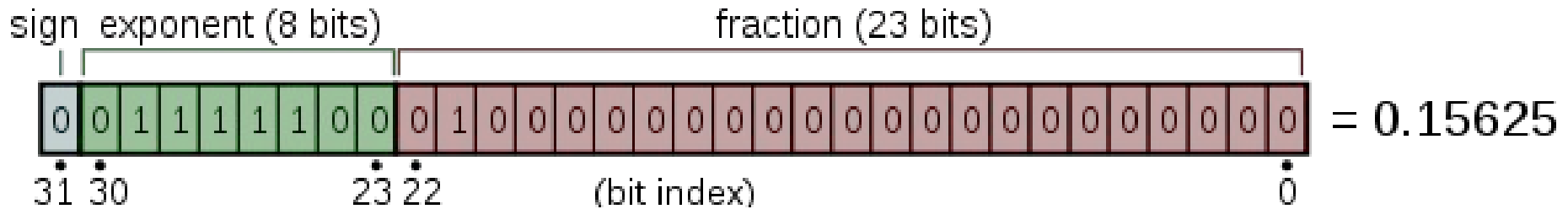
# UNSIGNED DATATYPES IN C

- (unsigned char) OR byte
  - 1 byte (= 0 to 255)
- unsigned int
  - 2 bytes (= 0 to 65535)
- unsigned long
  - 4 bytes (= 0 to 4294967295)
- float
  - can not be unsigned

# DATATYPES IN C

- char
  - ascii text and small numbers
- int
  - normal range numbers, counters
- long
  - large numbers (time, flash memory)
- float
  - fractions, slow, limited precision

# FLOATS



- Avoid because not supported in Arduino hardware
- Use only when you absolutely need fractions

# BOOLEANS

- Yes/no, True/false
- Is actually a `char` in disguise
- No real "true" and "false"
  - 0 is false
  - Non-0 (anything but 0) is true

```
int test = (4 > 3); // test = 1
```

```
If (test) { ... }
```

```
If (42) { ... }
```



# COMPUTER MEMORY

char

00110011	10110011	11101100	11100100
10110011	11101100	10110011	01001000
01000101	01001100	01010000	00100001
00100000	01001001	00100111	01101101
00100000	01110100	01110010	01100001
01110000	01110000	01100101	01100100
00100000	01101001	01101110	00100000
01100001	00100000	01100010	01111001
01110100	01100101	00100000	01100110
01100001	01100011	01110100	01101111
01110010	01111001	00100001	00000000
00110011	10110011	11101100	11100100
10110011	11101100	10110011	10110011

# COMPUTER MEMORY

char

int

00110011

10110011

11101100

11100100

10110011

11101100

10110011

01001000

01000101

01001100

01010000

00100001

00100000

01001001

00100111

01101101

00100000

01110100

01110010

01100001

01110000

01110000

01100101

01100100

00100000

01101001

01101110

00100000

01100001

00100000

01100010

01111001

01110100

01100101

00100000

01100110

01100001

01100011

01110100

01101111

01110010

01111001

00100001

00000000

00110011

10110011

11101100

11100100

10110011

11101100

10110011

10110011

# COMPUTER MEMORY

char	int		long
00110011	10110011	11101100	11100100
10110011	11101100	10110011	01001000
01000101	01001100	01010000	00100001
00100000	01001001	00100111	01101101
00100000	01110100	01110010	01100001
01110000	01110000	01100101	01100100
00100000	01101001	01101110	00100000
01100001	00100000	01100010	01111001
01110100	01100101	00100000	01100110
01100001	01100011	01110100	01101111
01110010	01111001	00100001	00000000
00110011	10110011	11101100	11100100
10110011	11101100	10110011	10110011

# VARIABLE DECLARATIONS

Must have:

- Type (char, int, long....)
- Name
  - Anything you like, but be descriptive
  - Must not be a keyword (char, for, while...)
  - Must not start with a number
  - Case sensitive! “Thing” and “thing” are different

May have:

- unsigned modifier
- initial value

# VARIABLE DECLARATIONS

```
int x = 1; // correct, but mystery  
float maxTemperature = 39.5;  
unsigned char age;  
int counter; // no initial value  
unsigned int sheep = 12;
```

# SCOPE { }

- Globals
  - Declared outside of functions
  - Can be accessed everywhere in your program
- Locals
  - Declared inside functions
  - Cease to exist at }
  - Even when you repeat (as in `loop()`)
- static
  - Declared in a function
  - Value is kept between function calls
  - Initialized only once
  - Only available in the function!

# CONSTANTS: PREPROCESSOR

- Variables that do not change value
- `#define LEDPIN 13`
  - Means: Before compiling, replace the string "LEDPIN" with "13"
- so:
  - `digitalWrite(LEDPIN, HIGH);`
  - is rewritten as
  - `digitalWrite(13, HIGH);`

# CONSTANTS: PREPROCESSOR

Try in setup():

```
#define CONSTANT 4;
```

```
Serial.begin(9600);
```

```
Serial.println(CONSTANT);
```



# CONSTANTS: PREPROCESSOR

**BEWARE**

**the following:**

```
#define LEDPIN 13;  
digitalWrite(LEDPIN, HIGH);
```

**becomes:**

```
digitalWrite(13, HIGH);
```

# SERIE VARIABLEN

- `int ledPinOne = 11;`
- `int ledPinTwo = 13;`
- `int ledPinThree = 5;`
- `ledPinX ?`

# SERIE VARIABLEN

- `int ledPin[1] = 11;`
- `int ledPin[2] = 13;`
- `int ledPin[3] = 5;`
- `ledPin[x] = ...`

# POINTERS



- Memory locations

- `char a = 5;`
- `char b = 9;`
- `char* pa = &a;`

1024	???
1025	???
1026	5
1027	9
1028	???
1029	???
1030	1026

## operators:

- `&a` = address of variable `a`
- `pa = 1026`
- `*pa` = Value of contents of memory address 1026 (= 5)

# POINTERS

Try in setup()

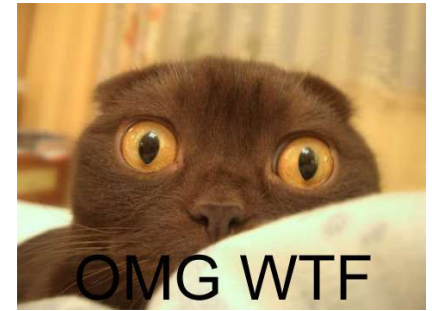
```
int value1 = 1000, value2 = 2000;
```

```
Serial.begin(9600);
```

```
Serial.println((int) &value1);
```

```
Serial.println((int) &value2);
```

# POINTERS



What is....

a	5
a + 1	6
pa	1026
pa + 1	1027
*pa	5
*pa + 1	6
*(pa + 1)	9

1024	???	
1025	???	
1026	5	a
1027	9	b
1028	???	
1029	???	
1030	1026	pa



# ARRAYS

- `int value[3];`
- `*value = 11;`
- `*(value+1) = 13;`
- `*(value+2) = 5;`

5996	5999	value
5997	???	
5998	???	
5999	11	
6000	13	
6001	5	
6002	???	
6003	???	
6004	???	

# ARRAYS

- `int value[3];`
- `* (value+0) = 11;`
- `* (value+1) = 13;`
- `* (value+2) = 5;`

5996	5999	value
5997	???	
5998	???	
5999	11	
6000	13	
6001	5	
6002	???	
6003	???	
6004	???	



# ARRAYS

- `int value[3];`
- `value[0] = 11;`
- `value[1] = 13;`
- `value[2] = 5;`

5996	5999	value
5997	???	
5998	???	
5999	11	
6000	13	
6001	5	
6002	???	
6003	???	
6004	???	

# INITIALISING ARRAYS

- `int value[3];`
- `value[0] = 11;`
- `value[1] = 13;`
- `value[2] = 5;`

5996	5999	value
5997	???	
5998	???	
5999	11	
6000	13	
6001	5	
6002	???	
6003	???	
6004	???	

- `int value[] = {11, 13, 5};`

# TEXT

char	int		long	char[]
00110011	10110011	11101100	11100100	
10110011	11101100	10110011	01001000	
01000101	01001100	01010000	00100001	
00100000	01001001	00100111	01101101	
00100000	01110100	01110010	01100001	
01110000	01110000	01100101	01100100	
00100000	01101001	01101110	00100000	
01100001	00100000	01100010	01111001	
01110100	01100101	00100000	01100110	
01100001	01100011	01110100	01101111	
01110010	01111001	00100001	00000000	
00110011	10110011	11101100	11100100	
10110011	11101100	10110011	10110011	

## TEXT

```
char *text = "blahdiblah";
```

is similar to:

```
char text[] = {98, 108, 97, 104,  
100, 105, 98, 108, 97, 104, 0};
```

the compiler puts an extra 0x00 at the end

## TEXT

```
char *text = "blahdiblah";
```

is similar to:

```
char text[] = { 'b', 'l', 'a',  
                'h', 'd', 'i', 'b', 'l', 'a', 'h',  
                '\0' };
```

`'0'` is the *character* 0, `'\0'` is the *value* 0

# ARRAY LENGTH

- Keep track of it yourself
  - `int nChars;`
- Use a "sentinel" value.
  - a value at the end of the array that will never appear in the array itself
- for strings: use `strlen()`
  - makes use of the `0x00` "sentinel" value at the end of strings

# ARRAY LENGTH

- Keep track of it yourself
  - `int nChars;`
- Use a "sentinel" value.
  - a value at the end of the array that will never appear in the array itself
- for strings: use `strlen()`
  - makes use of the `0x00` "sentinel" value at the end of strings

**`sizeof()` ?**

# SIZEOF

try in setup():

```
Serial.begin(9600);
```

```
int myArray[8] = {0, 1, 2, 3, 4, 5, 6, 7};
```

```
Serial.println(sizeof(myArray));
```

```
char *myText = "blahdiblah";
```

```
Serial.println(sizeof(myText));
```



# **sizeof**

...the amount of bytes a certain variable takes up in memory.

# SUMMARY

We have seen:

- History and context of C
- Binary counting
- Memory
- Variables
- Arrays/Pointers

# SUMMARY

We did not cover:

- functions
- loops
- program structure

Learn those by example

- Use the reference! <http://arduino.cc/en/Reference/HomePage>
- Search examples and modify them
- Keep track of your data when you combine examples!

# HAPPY CODING!

