### Content of the experimenter kit:

- 1× blue protection box
- 1× development board
- 1× logic analyzer + USB cable + connection cable
- 1× plastic tube
- $N$× poker chips (e.g. white, black, red, blue, green)
- 1× screwdriver for potentiometer adjustment
- 10× connection cable

## Safety Instructions

- ☛ The board may only be connected to the USB port of a computer.
- ☛ Additional external hardware must be connected only after consultation of the supervisors.
- ☛ Self-reliant soldering on the board is forbidden.
- ☛ The board must be stored dry and protected from direct sunlight.
- ☛ Defective circuit boards must be reported immediately*.


**Danger! Small parts. Not suitable for children under 3 years.**

* Components can always break - In your own interest, you should report any damage as soon as possible, so that we can replace defective parts and provide you with a working board as quickly as possbile.

Practical Exercise – Microcontroller Techniques

Prof. Dr. L. Reindl

Laboratory for Electrical Instrumentation

# Cheat sheet
## for microcontroller C programming

**Version 1.4 - October 2016**

# C cheat sheet - Course "Microcontroller Techniques"

## VARIABLES

`int nameOfVariable;` // Initializes an integer variable. Possible values: -2^15 to 2^15 - 1

`char nameOfVariable;` // Variable for a character. Possible values -128 to +127

`unsigned char nameOfVariable;` // An (unsigned) byte (0 to 255)

**Values** for **variables** are specified in one of these two formats:

Decimal: 123
Hexadecimal: 0x7B

Also possible is the initialization with characters, but this usually makes only sense for variables of type `char`.

In addition, variables can also be initialized as an array:

```
int[6] numbers; // Creates an array with 6 entries - so it can
                // store 6 int-Numbers
numbers[0] = 1; // initialize first flement (0, not 1!) with 1
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
numbers[5] = 6; // initialize last element with 6 (size 6 means that 5 is the biggest element!)
```

## CONSTRUCTS

```
for (Start statement; Condition; Action after each loop) {
    // Instructions per loop pass
    continue; // Cancel current loop pass
              // and proceed with the next
}
```

```
while (Condition) {
  // Actions per loop pass
  break; // cancel loop
}
```

`break` and `continue` can be used in each loop types.

```
if (1st condition) {
  // Actions if 1st condition is true.
}
else if (2nd condition) {
  // Actions if 1st condition is not true and 2nd condition is true.
}
else {
  // Actions if 1st and 2nd conditions are not true.
}
```

```
switch (var) {
  case 0: <instructions>;           // if var == 0, do something
  break;  // The rest of the cases are skipped (important for default!)
  case 3: <instructions>;           // do something else if var == 3
  break;
  case 5: <instructions> // ...(Note the missing break statement!)
    ...
  default: <A default statement>;
           // if var != 0 && var != 3, but also in var == 5
           // ⇒(because of the lack of break!)
  break;   // Optional
}
```

## OPERATORS

**Assignment operator: =**

`number = 4;` // Initializes the variable number and assigns the value to 4.

**Comparative operators: ==, >=, <=, !=**

| | | | |
|---|---|---|---|
| `1 == 1` equals 1. | `4 >= 2` equals 1. | `2 <= 8` equals 1. | `4 != 6` equals 1. |
| `7 == 3` equals 0. | `3 >= 7` equals 0. | `6 <= 2` equals 0. | `3 != 3` equals 0. |

**Mathematical operators: +=, -=, /=, *=, ++, –, +, -, /, ∗, %**

```
int number = 0;
number += 9;                    // number is increased by 9 → 9
number -= 3;                    // number is reduced by 3 → 6
number /= 2;                    // number is divided by 2 → 3
number *= 4;                    // number is multiplied by 4 → 12
number++;                       // number is increased by 1 → 13
number--;                       // number is decreased by 1 → 12
number = (5 + 1 * 2 - 3) / 4;   // number is set to 1 (PEMDAS!)
number = 15 % 4;                // number is set to 3 (modulo)
```

**Binary operators: |, &, ^, ~, <<, >> ( also |=, &=, ^=)**

The binary **OR**

```
4 | 2  results 6 because:  4 = 0100 ; 2 = 0010 ; 4 | 2 = 0110
4 | 4  results 4 because:  4 = 0100 ; 4 = 0100 ; 4 | 4 = 0100
3 | 2  results 3 because:  3 = 0011 ; 2 = 0010 ; 3 | 2 = 0011
```

The binary **AND**

```
4 & 2  results 0 because:  4 = 0100 ; 2 = 0010 ; 4 & 2 = 0000
4 & 4  results 4 because:  4 = 0100 ; 4 = 0100 ; 4 & 4 = 0100
3 & 2  results 2 becuase:  3 = 0011 ; 2 = 0010 ; 3 & 2 = 0010
```

The binary **exclusive OR / XOR**

```
4 ^ 2  results 6 because:  4 = 0100 ; 2 = 0010 ; 4^2 = 0110
4 ^ 4  results 0 because:  4 = 0100 ; 4 = 0100 ; 4^4 = 0000
3 ^ 2  results 1 because:  3 = 0011 ; 2 = 0010 ; 3^2 = 0001
```

The binary **NOT**

```
~0010  results  1101
~1111  results  0000
```

**Shifting**

```
4 << 1  equals 8   because:  4 = 0100 4 << 1 = 1000   (shift to left)
3 << 2  equals 12  because:  3 = 0011 3 << 1 = 1100   (shift to left)
3 >> 2  equals 0   because:  3 = 0011 3 >> 2 = 0000   (shift to right)
3 >> 1  equals 1   because:  3 = 0011 3 >> 1 = 0001   (shift to right)
```

## COMMENTS

**Single-line comments:**

```
// this text is ignored
```

**Multiline comments:**

```
/*
   Everything
   - even line breaks -
   are ignored
*/
```

## FUNCTIONS

```
int testFunction (char a, int b) // Defines a function with the
                                 // name testFunction, with two
                                 // arguments a  (type: char)
                                 // and b (type: int). Type of the
                                 // return value of the function is int.
{ // Beginning of the actual function
  // Here is the code to be executed within the function.
  int i = 14;
  i = i * 2;
  return i; // Return the value and complete of the function.
}           // Syntactic completion of the function.

testFunction(    , 2);          // Exemplaric call of the function
int y = testFunction(14, 0x14); // Alternative call in which the
                                // return value is stored in y
```