# Lesson 10

Christian Schwarz, Jakob Krebs

15.12.2019

## Contents

## Sources and Solutions

- we publish all code written in this course at `https://github.com/jkrbs/c_lessons`
- we will publish example solutions of the tasks on same site
- send us questions or your solutions to c-lessons@deutschland.gmbh

## Bit fields

## basics

repeatition of the bit operations

| Symbol | Operation | Example |
|---|---|---|
| \| | logical or | 0110 \| 0101 == 0111 |
| & | logical and | 0110 & 0101 == 0100 |
| ^ | logical xor | 0110 ^ 0101 == 0011 |
| ~ | logical negation | ~0110 == 1001 |
| << | shift to the left | 0110 << 2 == 011000 |
| >> | shift to the right | 0110 >> 2 == 0001 |

## bit fields

We want to use our memory as efficient as possible, so we can have smaller datatypes, when we need them.

```
struct Box
{
  unsigned int  opaque        : 1;
  unsigned int  fill_color    : 3;
  unsigned int                : 4; // fill to 8 bits
  unsigned int  show_border   : 1;
};


sizeof(Box) = 2
```

## bit fields

Now we can adress the variables like every other member of a struct.

The behavior of c struct is implementation defined, therefore it is undefined behavior, what happpens, when you assign bigger values.

If you want to avoid the risk of making mistakes (which you might not notice with your compiler), you can just use a normal `int` and use it to emulate a bit field (by shifting and asigning single bits with `|=` and `&=`)

## example

```

/* Sets the gameControllerStatus using OR */
void KeyPressed( int key ) { gameControllerStatus |= key; }

/* Turns the key in gameControllerStatus off
using AND and ~ (binary NOT)*/
void KeyReleased( int key ) { gameControllerStatus &= ~key; }

/* Tests whether a bit is set using AND */
int IsPressed( int key ) { return gameControllerStatus & key; }
```

## project