

So schön sieht jetzt das  
Informatiklabor aus, aber keiner  
sieht es ☹️

# Informatik 2

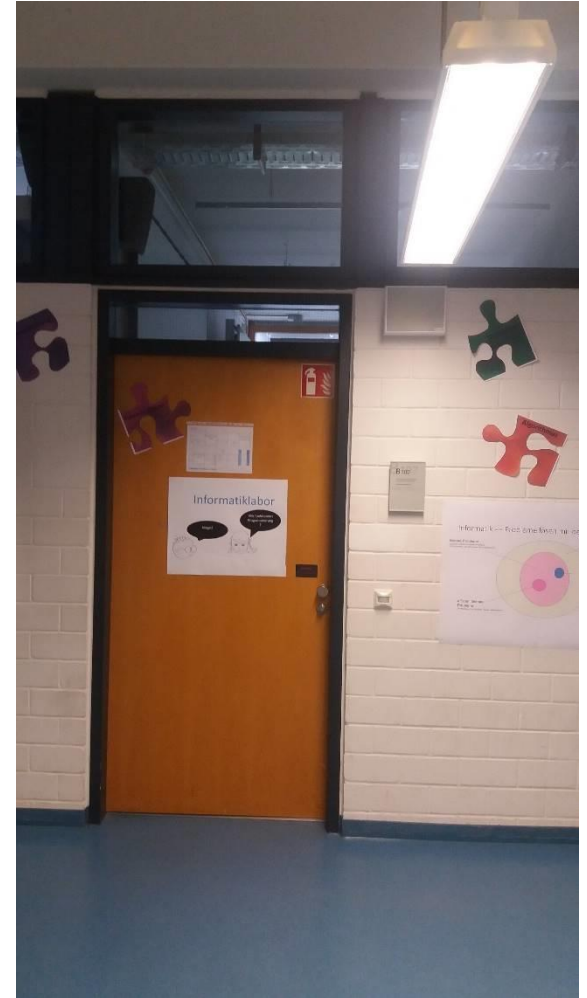
## Strukturen in C

**Irene Rothe**

Zi. B 241

[irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)

Instagram: irenerothedesign



# Das Marie Kondo Prinzip

→ If it doesn't spark joy, get rid of it.

Wenn etwas keinen Spaß macht, ändere es!

If a function or line doesn't spark joy, get rid of it.

Quelle: <https://www.karim-geiger.de/blog/das-marie-kondo-software-design-principle>

# Informatik: 2 Semester für Ingenieure

## Informatik = Lösen von Problemen mit dem Rechner

✓ Zum Lösen von Problemen mit dem Rechner braucht man **Programmierfähigkeiten** (nur mit Übung möglich): Was ist Programmieren?

✓ Was ist ein Flussdiagramm?

### → **Programmiersprache C:**

- ✓ Elementare Datentypen
- ✓ Deklaration/Initialisierung
- ✓ Kontrollstrukturen: if/else, while, for
- ✓ Funktionen
- ✓ Felder (Strings)
- ✓ Zeiger
- struct
- Speicheranforderung: malloc
- Listen
- Bitmanipulation

✓ Wie löst der Rechner unsere Probleme? → mit **Dualdarstellung** von Zeichen und Zahlen und mit Hilfe von **Algorithmen**





→ Ein Beispiel für ein Problem: **Kryptografie**

→ Sind Rechner auch Menschen? → **Künstliche Intelligenz**

→ Für alle Probleme gibt es viele Algorithmen. Welcher ist der Beste? → **Aufwand** von Algorithmen



# Design der Folien

-  hinterlegt sind alle Übungsaufgaben. Sie sind teilweise sehr schwer, bitte absolut nicht entmutigen lassen! Wir können diese in Präsenz besprechen oder über Fragen im Forum.
-  hinterlegte Informationen und grüne Smileys sind wichtig und klausurrelevant.
- Alles hinter „**Achtung**“ unbedingt beachten!
-  verwende ich, wenn überraschende Probleme auftreten können. Wenn Sie schon programmiererfahrend sind, können das eventuell besonders große Überraschungen für Sie sein, wenn Sie eine andere Sprache als C kennen.
- „Tipp“ benutze ich, um Ihnen einen Weg zu zeigen, wie ich damit umgehen würde.
- „Bemerkung“ in Folien beziehen sich meist auf Sonderfälle, die nicht unbedingt klausurrelevant sind, aber für Sie beim Programmieren eine Bedeutung haben könnten
-  hinter diesem Symbol ist ein Link fürs Anhören bzw. Gucken weiterer Infos

### Aufbau der Folien:

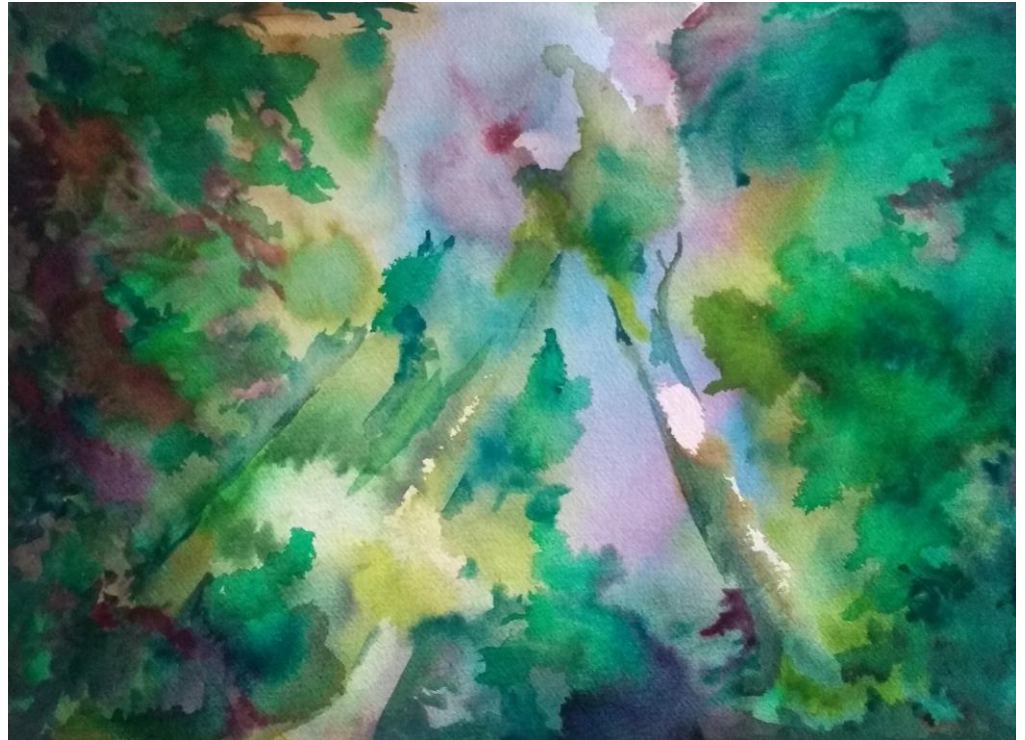
- Am Anfang motiviere ich gerne mit einem Beispiel, das eventuell schwer verständlich ist. Wem das nicht zusagt, dem empfehle ich, diese Folien zu überspringen.
- Weiter arbeite ich mit vielen Beispielen, die oftmals immer wieder das Gleiche erklären nur auf unterschiedliche Arten. Hat man einen Sachverhalt einmal verstanden, braucht man eventuell diese Beispiele nicht.
- Folien, die mit **Einschub** beginnen, beinhalten Zusatzinformationen, die nicht nötig für das Verständnis des Themas sind.
- Grün hinterlegte Informationen sind das, was Sie aus der Vorlesung rausnehmen sollen, alles andere sind vertiefende Informationen und Motivation.



→ Strukturen

Bitfelder

Union



# Strukturen: Motivation

Felder sind ja nicht schlecht, um gleiche Dinge abzuspeichern. Aber ich will gerne auch unterschiedliche Dinge abspeichern, um z.B. all meine Klamotten zu organisieren nach folgenden Merkmalen:

- wann gekauft
- wie teuer
- Name des Kleidungsstück
- Farbe
- schon mal angezogen
- Wohlfühlfaktor

# Strukturen: Motivation

In einem Feld (Array) können mehrere Dinge **desselben** Typs zusammengefasst werden.

Manchmal möchte man aber Dinge **verschiedenen** Typs zusammenfassen, wenn sie logisch zusammen gehören.

```
struct Klamotte{  
    int wanngekauft;  
    double wieteuer;  
    char name[30];  
    char farbe;  
    char schonmalangezogen;  
    int wohlfuehlfaktor;  
}; //Semikolon
```

Strukturname


Komponenten



# Strukturen: Beispiel

```
struct Klamotte{
    int wanngekauft;
    double wieteuer;
    char name[30];
    char farbe;
    char schonmalangezogen;
    int wohlfuehlfaktor;
};

int main(){
    struct Klamotte kleid1, kleid2;
    kleid1.wieteuer=99.99;
    kleid1.schonmalangezogen='j';
    kleid1.wohlfuehlfaktor=55;
    kleid2.wieteuer=199.66;
    printf("Preis:%lf, Wohlfuehlfaktor=%i", kleid1.wieteuer,
           kleid1.wohlfuehlfaktor);
    return 0;
}
```



Dies ist nur eine Definition!

# Strukturen: Beispiel: Mehrfamilienhaus

Haus mit Wohnungen gleichen Grundrisses:

- Wohnzimmer
- Schlafzimmer
- Bad
- Küche
- Flur

Definition der Struktur Wohnung:

```
struct Wohnung{  
    int wohnzimmer;  
    int schlafzimmer;  
    int bad;  
    int kueche;  
    int flur;  
};
```

Deklaration der Wohnung im Haus:

```
struct Wohnung erdgeschossWohnung, ersteEtageWhg, dachWhg;
```



# Strukturen: Definition

```
struct Konto{
```

```
    int kontonummer;  
    double betrag;
```

}

Zusammenfassung von Dingen  
verschiedenen  
Typs

```
}; //Semikolon!!
```

Struktur-Definition:

```
struct Strukturname{
```

```
    <Datentyp> Komponentennamen1;
```

```
    <Datentyp> Komponentennamen2;
```

```
    ...
```

```
};
```



# Strukturen: Zugriff - Punktoperator

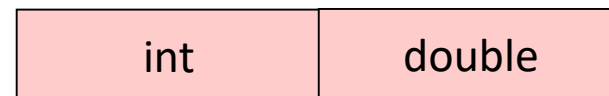
```
#include <stdio.h>
```

```
struct Konto{  
    int kontonummer;  
    double betrag;  
};
```

← Struktur-Definition

```
int main(){  
    struct Konto k1; ← Variablen-Deklaration  
    struct Konto k2={42,1.99}; ← Variablen-Deklaration und  
                                Variablen-Initialisierung  
                                (dies sind nur Beispielwerte)  
    k1.kontonummer = 2323;  
    k1.betrag = 999.99; } ← Zuweisung  
  
    return 0;  
}
```

Reservierung von Speicher hintereinander:



# Strukturen: Zugriff mit Pfeiloperator

```
#include <stdio.h>
```

```
struct Konto{  
    int kontonummer;  
    double betrag;  
};
```

Struktur-Definition

```
int main(){  
    struct Konto k,*pk;  
    pk=&k;
```

```
    pk->kontonummer=7;  
    pk->betrag = 5.01;
```

→ Sehr üblich!

```
    return 0;
```

```
}
```



# Strukturen: Vorteil – einfaches Kopieren

```
#include <stdio.h>
```

```
struct Konto{  
    int kontonummer;  
    double betrag;  
};
```

```
int main(){  
    struct Konto k1,k2;  
    k1.kontonummer=7;  
    k1.betrag=5.01;
```

```
    k2 = k1;//Praktisch: ganze Struktur wird kopiert
```

```
    return 0;
```

```
}
```





# Strukturen: Struktur in einer Struktur - Beispiel

```
#include <stdio.h>
#include <string.h>

struct Name{
    char vorname[20];
    char nachname[20];
};

struct Adresse{
    struct Name adressant;
    char strasse[20];
    int hausnummer;
};

int main(){
    struct Adresse student={{ "Anna", "Mueller" }, "Musterstr.", 1};
    strcpy(student.adressant.nachname, "Schmidt");
    printf("Nachname: %s", student.adressant.nachname);
    return 0;
}
```



# Strukturen: noch ein Beispiel mit Feld

```
#include <stdio.h>
#include <string.h>

struct Buch{
    int buchnummer;
    char autor[20];
    char titel[50];
    int jahr;
};

int main(){
    struct Buch katalog[100];
    katalog[2].buchnummer=1234;
    strcpy(katalog[2].autor, "Irene Rothe");
    strcpy(katalog[2].titel, "Exponentielle Algorithmen");
    printf("Autor: %s, Titel: %s", (katalog+2)->autor, katalog[2].titel);
    return 0;
}
```



# Strukturen: Beispiel - Mehrfamilienhaus

```
struct Wohnung{
    int wohnzimmer;
    int schlafzimmer;
    int bad;
    int kueche;
    int flur;
};

struct Haus{
    struct Wohnung erdgeschoss;
    struct Wohnung ersteEtage;
    struct Wohnung dach;
};

Deklaration:
struct Haus meinHaus;
meinHaus.erdgeschoss.bad=10;//in quadratmeter
```

# Strukturen: Bemerkungen

- **structs** werden als call-by-value übergeben (im Gegensatz zu Feldern!).
- Bei größeren **structs** kostet das möglicherweise viel Speicher.
- Deshalb ist es günstiger, die Eingabeparameter als Zeiger auf **struct** zu erklären.

```
void function(struct Adresse *p) {...}
int main(){
    struct Adresse student,*pstudent;
    pstudent=&student;
    function(pstudent);
    return 0;
}
```

Achtung: Der Rückgabewert einer Funktion darf vom **struct**-Typ sein, siehe Praktikumsaufgabe.



# Strukturen: Vorteile



- übersichtlich
- einfach zu kopieren
- können von Funktionen zurückgegeben werden

# Strukturen: Wiederholung

3 Zuweisungsmöglichkeiten:

```
Punktoperator: k.kontonummer = 2323;
```

```
Pfeiloperator: pk->kontonummer = 2323;
```

```
(Selten: Gemischte Form: (*pk).kontonummer = 2323;)
```

Coco versteht Strukturen: <https://www.youtube.com/watch?v=OrEFakXTD1E>



Coco startet mit der C Programmierung Teil 6 Strukturen

**Achtung:** Leider  
fehlen im Filmchen  
die runden  
Klammern bei  
main.



# Strukturen: Übung

Wo könnte man beim Spiel Schiffe versenken, sinnvoll Strukturen verwenden?

Achtung: Auf der nächsten Folie ist die Lösung zu sehen.



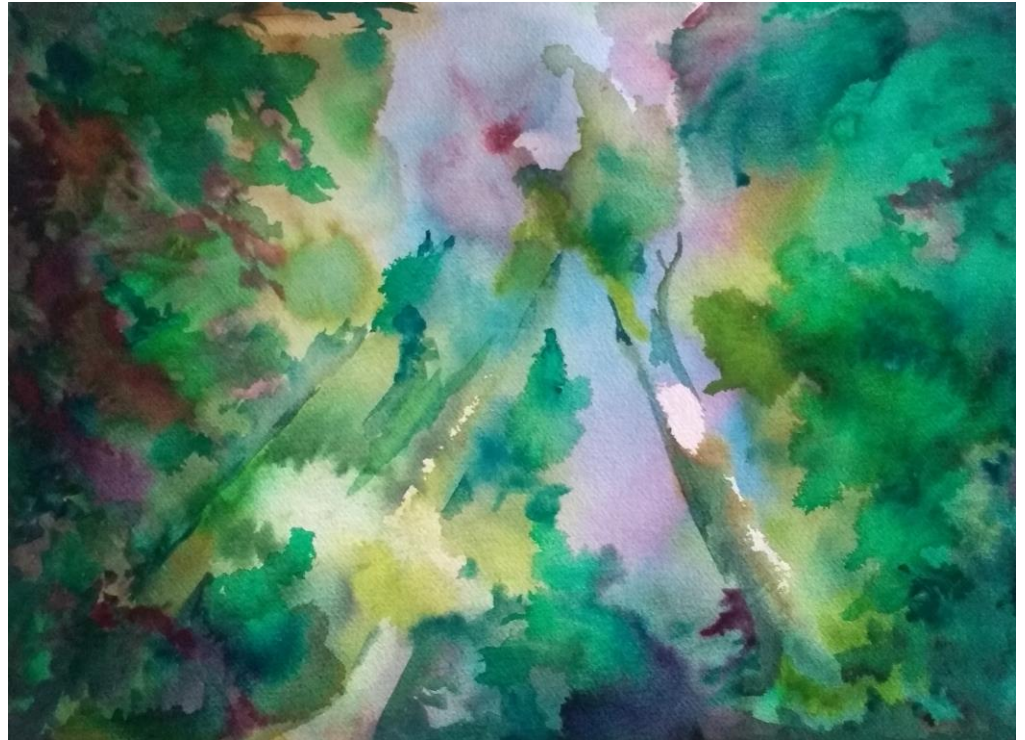
# Strukturen: Lösung - Schiffe versenken

Eine Möglichkeit:

```
struct Spieler{
    char spielfeld[10][10];
    char loesungsfeld[10][10];
    int anzahltreffer;
    char spielername[20];
};
...
int main(){
    struct Spieler spielerA, spielerB;
    ...
    spielerA.spielfeld[5][6]='S';
    ...
    return 0;
}
```



✓ Strukturen  
→ Bitfelder  
Union



# Nicht abgefragt in der Klausur: Definition von Bitfeldern

Motivation: manchmal hat man nur wenig Platz

Deshalb: Unterbringung von Infos auf eng begrenztem Speicherplatz → in C können einzelnen Bitfeldern Namen gegeben werden:

Beispiel:

```
struct Printerstatus{
    unsigned:4; //nicht verwendet, könnten Werte von
                0-15 enthalten
    unsigned error:1; //0 bedeutet Druckerfehler
    unsigned select:1; //1 bedeutet Drucker online
    unsigned paper:1; //1 bedeutet kein Papier
    unsigned busy:1; //bedeutet Drucker bereit
};

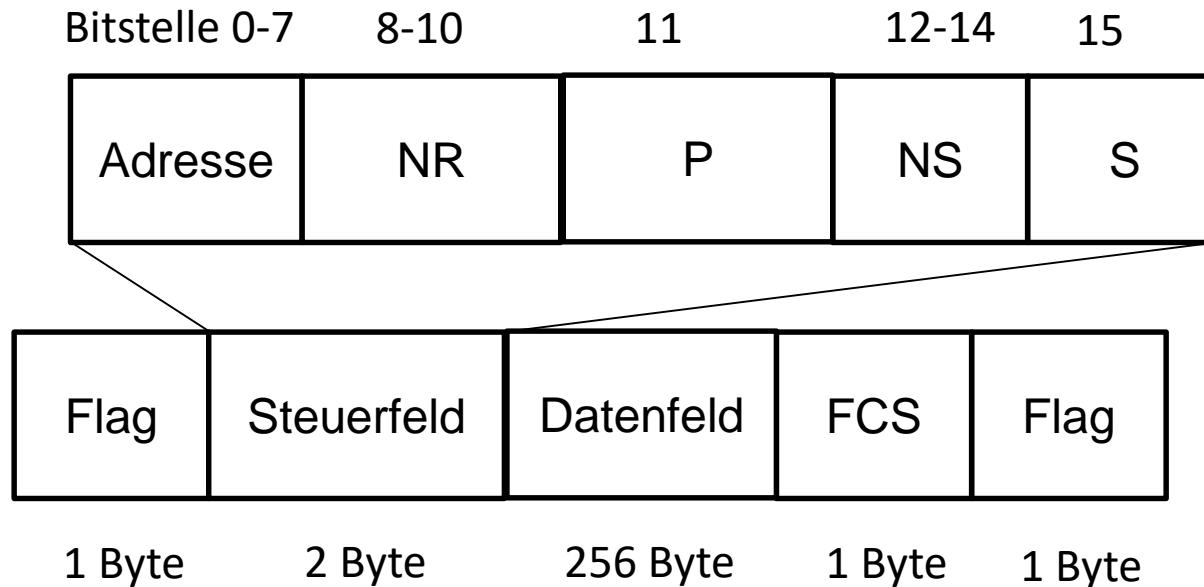
int main(){
    struct Printerstatus printerstatus;
    if (printerstatus.busy==1 && printerstatus.select==1){
        printf("Drucker bereit");
    }
    else{...}
    return 0;
}
```

Anzahl der Bits



# Bitfelder: Übung

Definieren Sie eine Struktur für eine Datenkommunikation der folgenden Art:

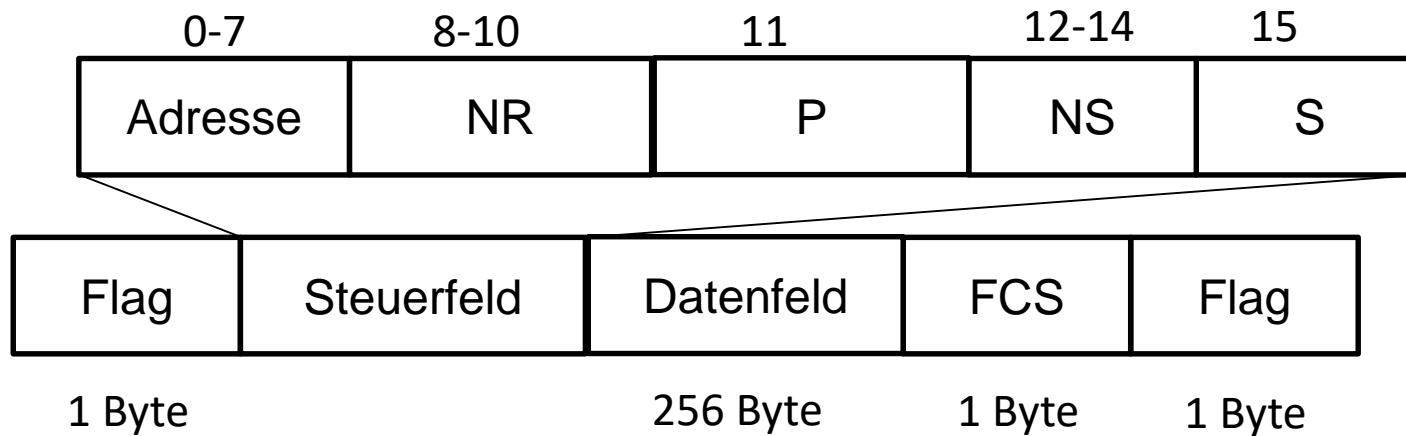


- Flag ist das Blockbegrenzungszeichen
- Steuerfeld enthält die Steuerinformationen für den Datenaustausch
- Datenfeld enthält die Informationen
- FCS enthält das Blockprüfzeichen

Achtung: auf der nächsten Folien ist die Lösung!



# Bitfelder: Lösung



```
struct Control{
    unsigned adr: 8;
    unsigned nr: 3;
    unsigned p: 1;
    unsigned ns: 3;
    unsigned s: 1;
};

struct Datenkommunikation{
    char flag1;//Blockbegrenzung
    struct Control control;
    char data[256];
    char fcs;
    char flag2;
};
```





# Bitfelder: Platzsparend

```
struct ZeitHolzhammer{  
    int stunde;  
    int minute;  
};  
struct ZeitPlatzsparend{  
    unsigned stunde:5;  
    unsigned minute:6;  
};
```

8 Byte

11 Bit

Bemerkung: Maximum für Stunden ist 24, also reichen 5 Bits aus (Zahlen bis 31 darstellbar), bei Minuten braucht man 60 als Maximum, also 6 Bits (Zahlen bis 63 darstellbar)



# Bitfelder: Übung

Schreibe ein Programm, das den ASCII-Code eines Zeichens ausgibt!

Achtung: auf der nächsten Folien ist die Lösung!



# Bitfelder: Lösung 1

Definition der nötigen Strukturen:

```
struct Byte{  
    unsigned bit1: 1;  
    unsigned bit2: 1;  
    unsigned bit3: 1;  
    unsigned bit4: 1;  
    unsigned bit5: 1;  
    unsigned bit6: 1;  
    unsigned bit7: 1;  
    unsigned bit8: 1;  
};
```



# Bitfelder: Lösung - Hauptprogramm 1

Hauptprogramm:

```
int main() {
    struct Byte *byte;
    char zeichen;
    printf("Bitte gib ein Zeichen ein:\n");
    scanf("%c",&zeichen);
    byte=(struct Byte*)&zeichen;//casting auf Byte
    printf("Ausgabe des ASCII-Codes:");
    //umgekehrt ausgeben, damit es auf dem
    //Bildschirm hübsch aussieht
    printf("%i ",byte->bit8);
    printf("%i ",byte->bit7);
    printf("%i ",byte->bit6);
    printf("%i ",byte->bit5);
    printf("%i ",byte->bit4);
    printf("%i ",byte->bit3);
    printf("%i ",byte->bit2);
    printf("%i ",byte->bit1);
}
```



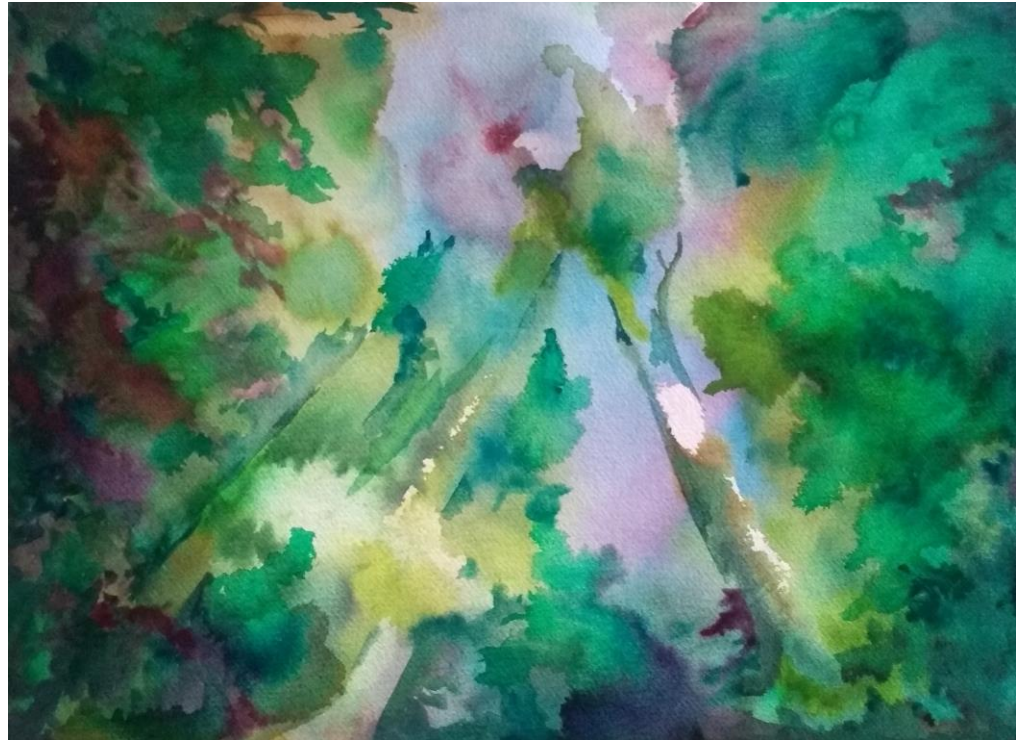
# Bitfelder: Lösung - Hauptprogramm 2

Hauptprogramm

```
int main(){
    int main(){
        struct Bitfolge a;
        scanf("%c",&a);
        printf("%i%i%i%i%i%i%i%i",a.bit7,a.bit6,a.bit5,
            a.bit4,a.bit3,a.bit2,a.bit1,a.bit0);
    }
}
```



- ✓ Strukturen
- ✓ Bitfelder
- Union





# Nicht abgefragt in der Klausur: Union (Variante)

- Daten verschiedenen Typs können im Laufe des Programms auf ein und demselben Speicherplatz gelegt werden (nicht gleichzeitig!!).
- Die Definition von Unions sieht syntaktisch wie ein **struct** aus.
- Aber die Komponenten einer Union haben im Speicher dieselbe Anfangsadresse, also muss der Speicherplatz so groß sein, wie die größte Komponente, die in der **union** abgespeichert werden könnte.

# Union

```
struct Zahl_s {  
    float punktZahl;  
    int    ganzeZahl;  
};
```

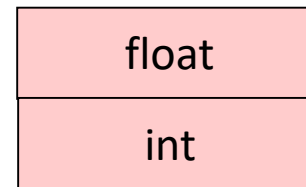
```
union Zahl_u {  
    float punktZahl;  
    int    ganzeZahl;  
};
```

```
int main(){  
    struct Zahl_s a;  
    union Zahl_u b;  
    return 0;  
}
```

Speicherung hintereinander



Speicherung übereinander



**punktZahl** und **ganzeZahl**  
in **Zahl\_u** sind nicht gleichzeitig  
ansprechbar!

# Union: Übung

Schreibe ein Programm, das den ASCII-Code eines Zeichens ausgibt!

Achtung: auf der nächsten Folie ist die Lösung!



# Union: Lösung

Definition der nötigen Strukturen:

```
struct Byte{
    unsigned bit1: 1;
    unsigned bit2: 1;
    unsigned bit3: 1;
    unsigned bit4: 1;
    unsigned bit5: 1;
    unsigned bit6: 1;
    unsigned bit7: 1;
    unsigned bit8: 1;
};
union ByteChar{
    char zeichen;
    struct Byte byte;
};
```



# Union: Lösung - Hauptprogramm

Hauptprogramm:

```
#include<stdio.h>
int main(){
    union ByteChar b;
    //char zeichen;
    printf("Bitte gib ein Zeichen ein:\n");
    scanf("%c",&b.zeichen);
    printf("Ausgabe des ASCII-Codes:");
    //umgekehrt ausgeben, damit es auf dem
    //Bildschirm hübsch aussieht
    printf("%i ",b.byte.bit8);
    printf("%i ",b.byte.bit7);
    printf("%i ",b.byte.bit6);
    printf("%i ",b.byte.bit5);
    printf("%i ",b.byte.bit4);
    printf("%i ",b.byte.bit3);
    printf("%i ",b.byte.bit2);
    printf("%i ",b.byte.bit1);
}
```

Der Hauptclou ist:  
ich gebe etwas als  
Zeichen ein, gebe  
es aber aus als  
Bitfeld



# Strukturen und Union: Padding

- Strukturen können verschiedene Längen haben, auch wenn ihre Komponenten gleich sind aber in verschiedener Reihenfolge stehen.
  - Für den schnellen Speicherzugriff füllt der Compiler (jeder macht das anders) leere Bytes rein, damit die Adressen ein Vielfaches ihrer Größe sind
  - Eine andere Anordnung der Komponenten (der Größe nach absteigend) kann das verhindern.
- Alles ist immer ein Kompromiss zwischen Speicherplatz und Laufzeit.



# Literatur:

- Peter Prinz und Ulla Kirch-Prinz: „C für PCs“