„C-Programmierung - Strukturen“

Marcel Lebek

Einleitung

- Strukturen
  - Was sind Strukturen?
  - Syntax
  - Verwaltung im Speicher
  - Bitfelder
  - Beispiel: Liste
- Unions
  - Aufbau
  - Unterschiede
  - Wann Sinnvoll?
- Quellen

Was sind Strukturen?

- Zusammenschluss mehrerer:
  - Elementare Variablen (int, char, double, float,…)
  - Pointer
  - Arrays
- Einleitung durch den Befehl *struct*
- Verbessert Übersicht unheimlich
- Ermöglicht Schritt in Richtung OOPM
- Optimal für Listen und Bäume

Syntax

```
char Vorname[20] = "Hans";
char Nachname[20] = "Meier";
int PLZ = 11111;
char Wohnort[20] = "Hamburg";
```

```
struct Adresse {
        char Vorname[20];
        char Nachname[20];
        int PLZ = 11111;
        char Wohnort[20];
} Adresse1={ "Hans", "Meier", 11111, "Hamburg"};
```

Syntax

```
char Vorname[20] = "Hans";
char Nachname[20] = "Meier";
int PLZ = 11111;
char Wohnort[20] = "Hamburg";

char Vorname[20] = "Peter";
char Nachname[20] = "Pan";
int PLZ = 22222;
char Wohnort[20] = "Berlin";

char Vorname[20] = "Uschi";
char Nachname[20] = "Blubb";
int PLZ = 33333;
char Wohnort[20] = "Muenchen";

char Vorname[20] = "Egon";
char Nachname[20] = "Mueller";
int PLZ = 44444;
char Wohnort[20] = "Luebeck";
```

```
struct Adresse {
        char Vorname[20];
        char Nachname[20];
        int PLZ = 11111;
        char Wohnort[20];
} Adresse1={"Hans", "Meier", 11111, "Hamburg"},
  Adresse2={"Peter", "Pan", 22222, "Berlin"},
  Adresse3={"Uschi", "Blubb", 33333, "Muenchen"},
  Adresse4={"Egon", "Mueller", 44444, "Luebeck"};
```

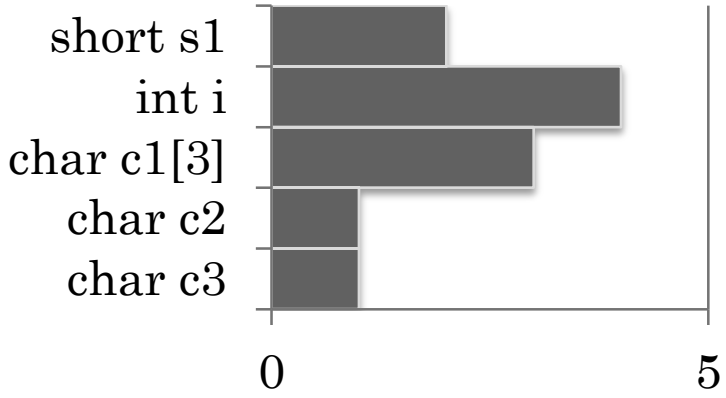
Syntax

- Strukturtypdeklaration: struct Adresse {…};
- Gesamtlänge der Struktur: *sizeof(Struktur)*
- Zugriff auf einzelne Komponenten durch Punktnotation: *Adresse1.Vorname = "Peter";*
- Weitere Strukturen hinzufügen:
  struct Adresse Adresse5;
- Weiteres hinzufügen von Komponenten während der Laufzeit **nicht** möglich.

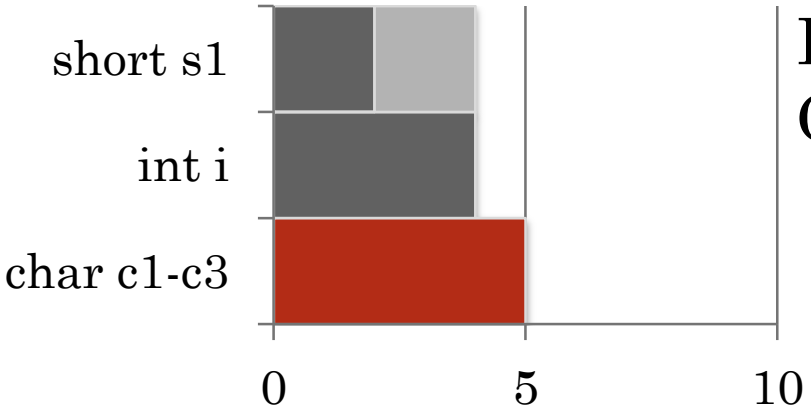
Verwaltung im Speicher

- Variablen liegen hinter einander im Speicher
- Reihenfolge wie in Deklaration
- Addierte Menge bildet Gesamtgröße der Struktur
- C99 §6.7.2.1: „*There may be unnamed padding within a structure object, but not at its beginning.*“[1]

Verwaltung im Speicher



Theorie:
Gesamtlänge = 11



Praxis:
Gesamtlänge = 16

Verwaltung im Speicher

- Jeder Compiler handhabt es anders
- Gesamtlänge der Struktur ist immer ein Vielfaches der Größe der größten Variable
- Jede Variable beginnt an einer geraden Adresse
- Einige Compiler bieten Funktionen, um Paddingbytes zu umgehen

Bitfelder

- Nutzbar wenn für Werte eine Obergrenze existiert (z.B. Tage im Monat)
- Nur mit *unsigned* Integern nutzbar
- Platzsparend….
- ….auf Kosten von Zugriffszeit

Bitfelder

```
struct Zeit {
        int Stunden;
        int Minuten;
};
```

Strukturgröße: 64 bit

```
struct Zeit {
        unsigned int Stunden : 5;
        unsigned int Minuten : 6;
};
```

Strukturgröße: 11 bit

Bitfelder

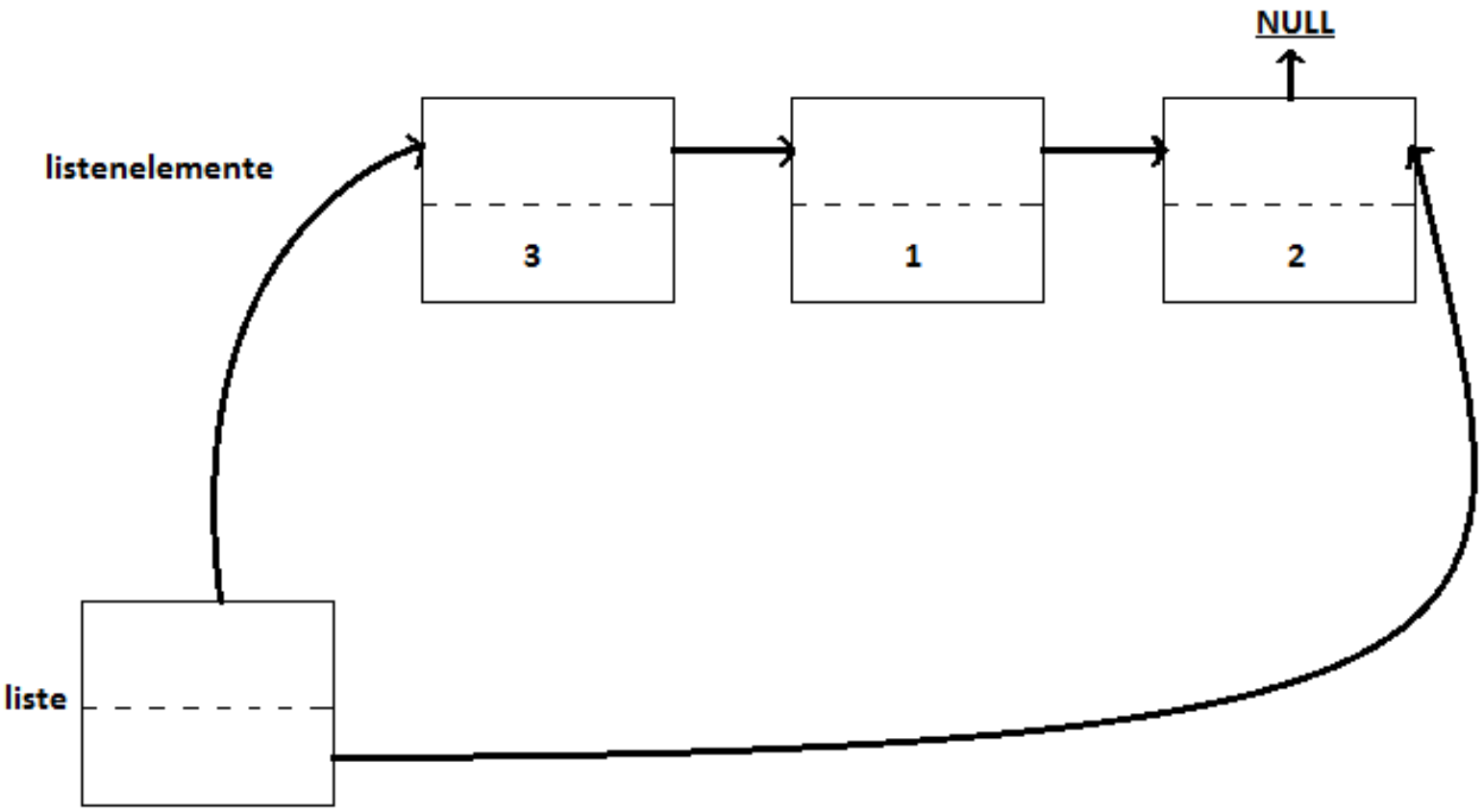
- Bei 1 Million Zeiten:
- Struktur: 7,63 MB
- Bitfeld: 1,31 MB

Beispiel: Liste

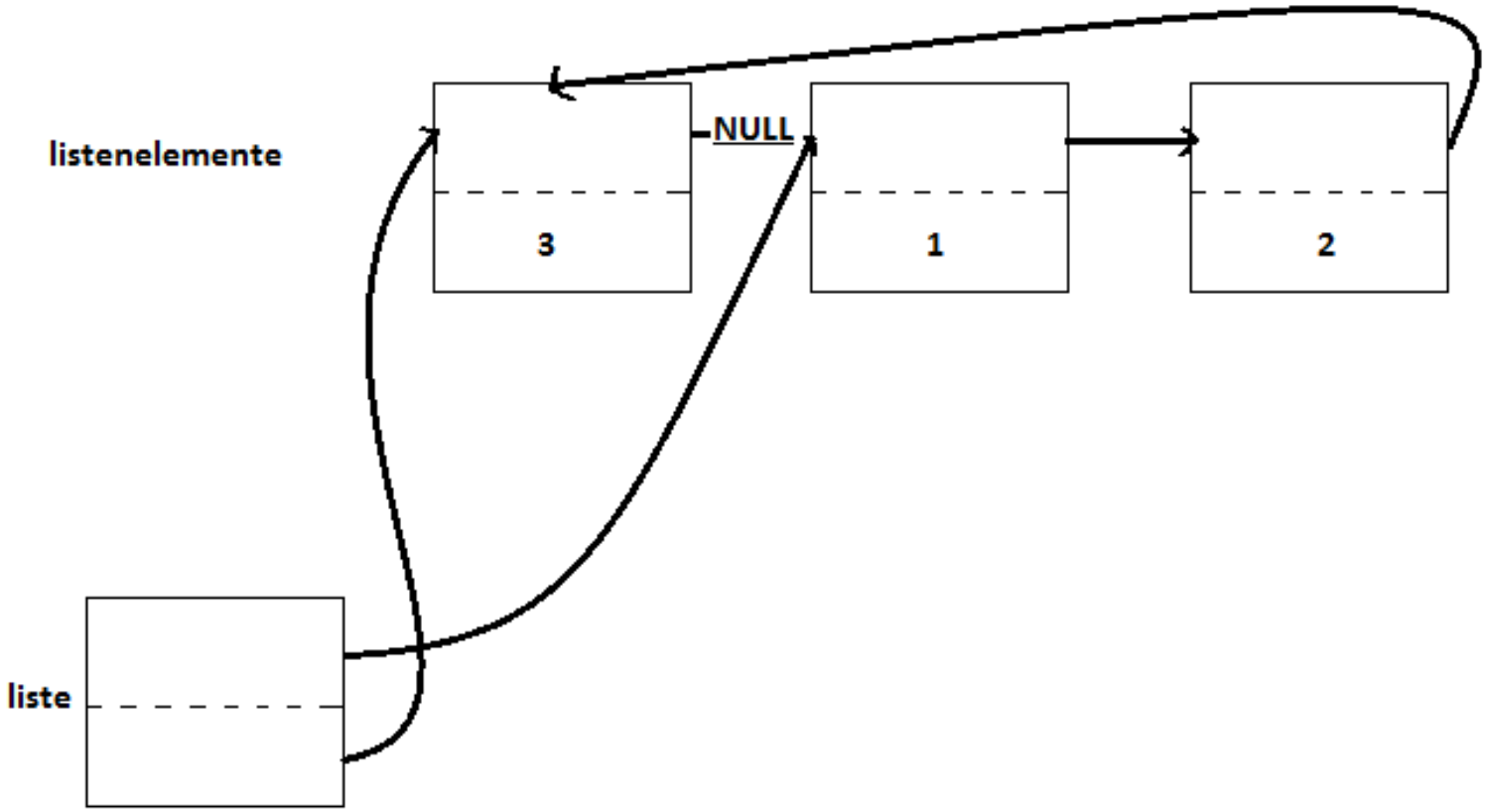
```
Struct listenelement {
        struct listenelement * nachfolger;
        int daten;
};


Struct liste {
        struct listenelement * start;
        struct listenelement * ende;
};
```

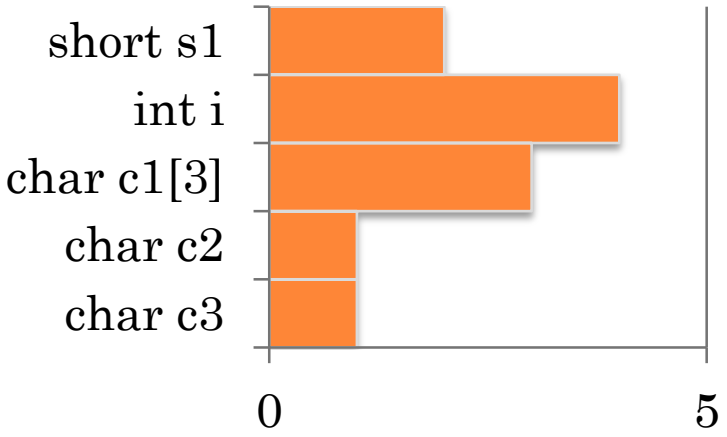
Beispiel: Liste



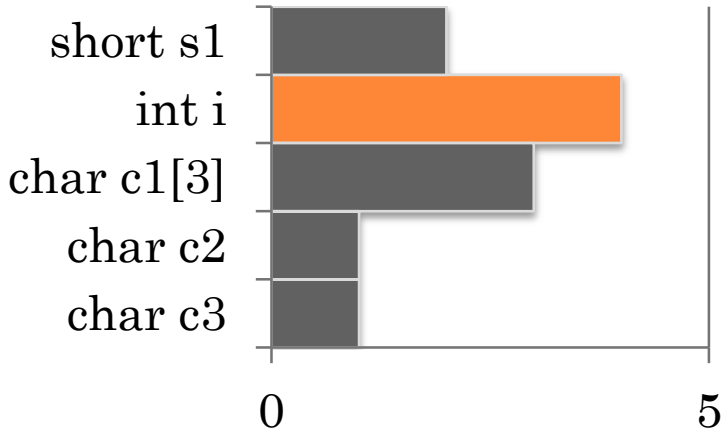
Beispiel: Liste



Unions - Aufbau



Theoretische Länge: 12



Theoretische Länge: 4
(längste Variable)

Unions - Aufbau

- Syntax ähnlich wie bei Strukturen

```
union test {
        int zahl;
        float gk;
} test1;
```

test1.zahl = 1;	*zahl = 1* *gk = 0.0*
test1.gk = 1.2345	*zahl = 1067320345* *gk = 1.2345*

Unions - Unterschiede

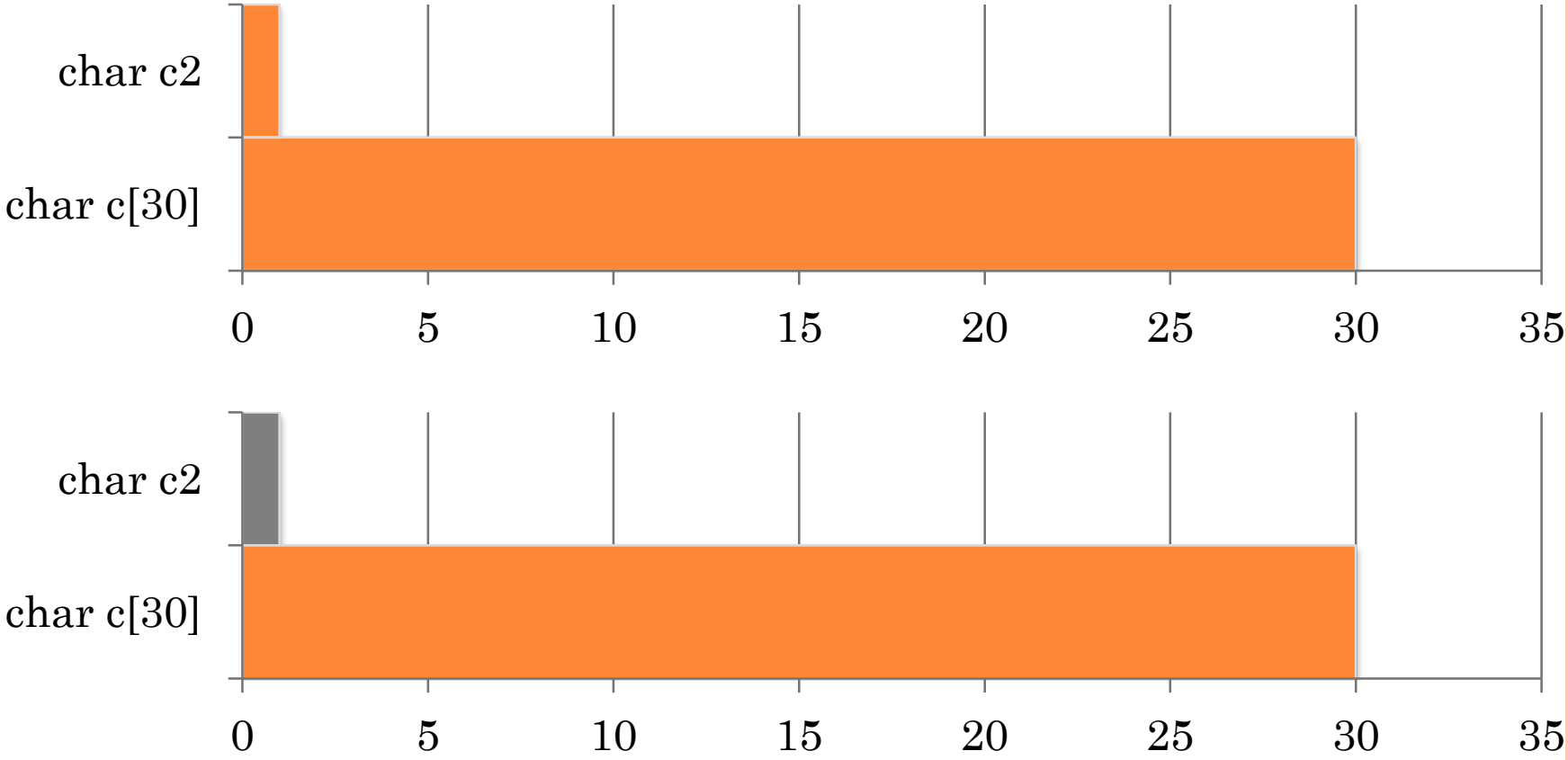
- Geringerer Speicherplatz
- Immer nur eine Komponente aktiv nutzbar
- Immer nützlich, wenn man verschiedene Variablen, aber nie zwei oder mehr gleichzeitig brauch.

Unions – Wann sinnvoll?

```
struct Adresse {
        char Vorname[20];
        char Nachname[20];
        int PLZ = 11111;
        char Wohnort[20];
};
```

```
union {
        struct Adresse1;
        struct Adresse2;
        struct Adresse3;
        struct Adresse4;
} Zieladresse;
```

Unions – Wann sinnvoll?



Zusammenfassung

- Strukturen ist eine Kombination aus mehreren Variablen vereint unter einem neuen Typ
- Die Länge der Struktur ist (oft) länger als die Summe der Variablen
- Strukturen erleichtern das erzeugen von Listen und Bäumen
- Unions sind wie Strukturen mit immer nur einer aktiven Komponente, nutzt dafür weniger Speicher.
- Bei ausschließlich (in der Größe) begrenzten Integern Bitfelder benutzen

Quellen

[1][ISO/IEC 9899:1999] ISO/IEC. *Programming Languages---C, 2nd ed* (ISO/IEC 9899:1999). Geneva, Switzerland: International Organization for Standardization, 1999.

- http://de.wikibooks.org/wiki/C-Programmierung
- http://stackoverflow.com/questions/2748995/c-struct-memory-layout
- http://www.c-howto.de
- Jürgen Wolf, „C von A bis Z“
- Helmut O.B. Schellong, „Moderne C-Programmierung“