

Einführung in die Programmierung

Grundbegriffe

Prof. Dr. Peter Jüttner

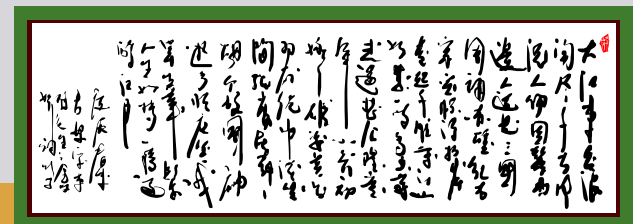
Inhalt

- **Grundbegriffe**
 - Grundstruktur eines C Programms
 - Variable
 - Konstante
 - Typen
 - Ein-/Ausgabe
 - Operatoren
 - Kontrollstrukturen
 - Vektoren (Arrays, Felder)
 - Typen

Typen

Motivation

- Informationen haben in der Regel einen Typ
 - Name als Zeichenkette
 - Alter als Zahl
 - Geburtsdatum z.B. als 2-Folgen von Zahlen (23.12.99)
 - Zeitungsartikel als Text
- Informationstyp legt fest
 - Größe der Information (sofern limitiert)
 - Operationen, die auf einer Information erlaubt sind (z.B. arithmetische Operationen auf Zahlen, Zusammenfügen von Texten)



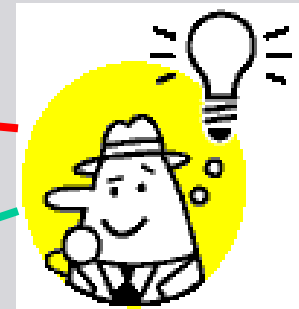
Typen

Implementierung in einer Programmiersprache

- Standardtypen
 - char, float, short, int, long, double, ...
 - charakterisiert („nach außen“) durch
 - Typname
 - Wertebereich
 - Literalkonstante
 - Operationen
 - charakterisiert („nach innen“) durch
 - interne Darstellung
 - Realisierung der Operationen

muss man kennen

sollte man kennen



Typen

Implementierung in einer Programmiersprache

- Ganzzahltypen (Wiederholung)
 - char (1Byte), short (2 Byte), int (2 oder 4 Byte), long (4 Byte)
 - char in „Doppelrolle“
 - vorzeichenlos (unsigned)
 - vorzeichenbehaftet (signed)
 - Konstante dezimal (mit oder ohne Vorzeichen)
 - 5, 67, +7, -22
 - Konstante oktal oder hexadezimal (mit oder ohne Vorzeichen)
 - 0x10, -0x22
 - Buchstabenkonstante (inkl. Ersatzdarstellung, s. ASCII Code)
 - 'x', , '\n'
 - Operationen: Arithmetik, Shift, Vergleich, Bitoperatoren, ...
 - intern als Bitfolge im 2-er Komplement (signed)



Typen

Implementierung in einer Programmiersprache

- Zeichenketten (eigentlich kein Standardtyp in C)
 - Vektor aus Buchstaben (`char zk[15]`)
 - maximale Länge maschinenabhängig
 - Konstante
 - “Konstante Zeichenkette”
 - mit implizitem Abschlußzeichen (`'\0'`)
 - Operationen: zeichenweiser Zugriff über Index, Bibliotheksfunktionen
 - intern als Folge von ASCII-Zeichen



Typen

Implementierung in einer Programmiersprache

- Gleitkommtypen (Wiederholung)
 - float (4Byte), double (8 Byte), long double (10 Byte)
 - Konstante mit Punkt (mit oder ohne Vorzeichen)
 - 5.0, 67.6, +7.1, -22.9
 - Konstante in Exponentenschreibweise (mit oder ohne Vorzeichen)
 - -10e5, 22e-77
 - Operationen: Arithmetik, Vergleich, ...
 - intern als Bitfolge aus Vorzeichen, Mantisse und Exponent



Typen

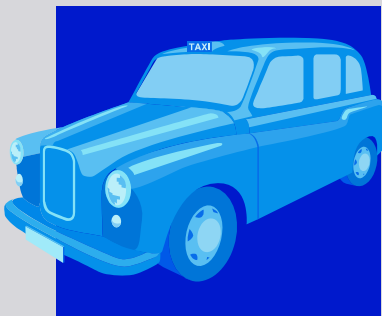
Implementierung in einer Programmiersprache

- Vektoren (strukturierter Typ)
 - über einem **Basistyp**, z.B.
 - **char** zk[10]
 - **int** zahlfeld[100];
 - Konstante basistypabhängig
 - Initialisierung mittels { ... }
 - Operation: Indizierung
 - interne Darstellung als Folge von Elementen des Basistyps

Typen

Selbstdefinierte Typen - Motivation

- mit Standardtypen nicht oder nur „mühsam“ darstellbar
 - Aufzählungen, z.B. Ampelfarben (rot, gelb, grün)
 - komplexe, strukturierte Informationen
 - Person (Name, Adresse, Geb.-Datum, Tel.-Nr.)
 - Pkw (Halter, Typ, Kennzeichen)



Typen

Selbstdefinierte Typen

- Schlüsselwort **typedef**
- Mittels **typedef** wird ein neuer Bezeichner für einen Datentyp eingeführt. Variablen können danach in folgenden Deklarationen unter Benutzung des neuen Typ-Bezeichners deklariert werden.
- Anwendung
 - **typedef** Neuer_Typ Typdefinition;

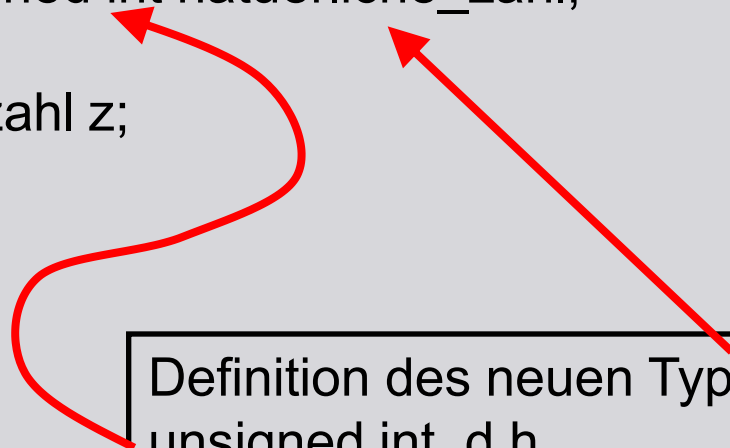
Name des neu definierten Typs

Definition des neuen Typs

Typen

Selbstdefinierte Typen

- „Einfachster“ Fall: Umdefinition eines Standardtyps
 - `typedef unsigned int natuerliche_zahl;`
...
`natuerliche_zahl z;`



Definition des neuen Typs `natuerliche_zahl` als `unsigned int`, d.h.

- *unsigned int* hat jetzt auch den Bezeichner *natuerliche zahl*
- überall wo *natuerliche_zahl* steht, kann auch *unsigned int* stehen
- *natuerliche_zahl* und *unsigned int* sind 100% typkompatibel

Typen

Selbstdefinierte Typen

- „Einfachster“ Fall: Umdefinition eines Standardtyps
 - Beispiel

```
#include <stdio.h>

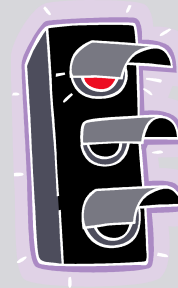
/* Umdefinitionn des Typs unsigned int zu natuerliche_zahl */
typedef unsigned int natuerliche_zahl;

int main()
{
    natuerliche_zahl n1 = 5;
    unsigned int i1 = 7;
    n1 = n1 + 3;
    n1 = n1 * i1;
    printf("n1:%d\n",n1);
    getchar();
}
```

Typen

Selbstdefinierte Typen

- Aufzählungstypen - Motivation
 - Darstellung einer beschränkten Wertemenge, die eine Variable annehmen kann
 - Ampelfarben rot, gelb, grün
 - Geschlecht weiblich, männlich
 - Karosserie open, locked, double locked
 - Nationalfarben schwarz, rot, gold



Typen

Selbstdefinierte Typen

- Aufzählungstypen
 - `typedef enum ampelfarbe { rot, gelb, gruen};`

alternativ mit konkreten Werten:

- `typedef enum color { red = 10, yellow = 20, green = 30};`

alternativ mit Lücken

- `typedef enum couleur { rouge, jaune = 10, bleu, vert};`



Typen

Selbstdefinierte Typen

- Aufzählungstypen
 - ... sind so weit kompatibel mit int, dass
 - farbe f = rot;
int i = f;
möglich ist (**umgekehrt nicht!**)
 - Elemente haben einen Zahlenwert:
 - ... { rot, gelb, gruen } → rot = 0, gelb = 1, gruen = 2
(Durchnummeriert aufsteigend beginnend mit 0)
 - ... {rouge, jaune = 10, bleu, vert} → rouge = 0, jaune = 10, bleu = 11, vert = 12
(Belegt teilweise mit konkreten Werten)



Typen

Selbstdefinierte Typen

- Aufzählungstypen - Beispiele

```
#include <stdio.h>

typedef enum ampelfarbe { rot, gelb, gruen };

typedef enum couleur { rouge, jaune= 10, bleu, vert };

typedef enum color { red = 10, yellow = 20, green = 30 };

int main()
{
    ampelfarbe farbe;

    farbe = rot; printf("rot =%d\n", farbe);
    farbe = gelb; printf("gelb =%d\n", farbe);
    farbe = gruen; printf("gruen =%d\n", farbe);

    ...
}
```


Typen

Selbstdefinierte Typen

- Aufzählungstypen - Beispiele

```
#include <stdio.h>

typedef enum ampelfarbe { rot, gelb, gruen };

typedef enum color { red = 10, yellow = 20, green = 30 };

typedef enum couleur { rouge, jaune= 10, bleu, vert };

int main()
{
    ...
    printf("\nAufzaehlungstyp mit konkreten Werten:\n");

    color c;
    c=red; printf("red =%d\n", c);
    c=yellow; printf("red =%d\n", c);
    c=green; printf("red =%d\n", c);
    ...
}
```

Typen

Selbstdefinierte Typen

- Aufzählungstypen - Beispiele

```
#include <stdio.h>

typedef enum ampelfarbe { rot, gelb, gruen };

typedef enum color { red = 10, yellow = 20, green = 30 };

typedef enum couleur { rouge, jaune= 10, bleu, vert };

int main()
{
    ...
    printf("\nAufzaehlungstyp mit einzelnen konkreten Werten:\n");

    couleur c1;
    c1 = rouge; printf("rouge =%d\n", c1);
    c1 = jaune; printf("jaune =%d\n", c1);
    c1 = bleu; printf("bleu =%d\n", c1);
    c1 = vert; printf("vert =%d\n", c1); ...
}
```

Typen

Selbstdefinierte Typen

- Aufzählungstypen
 - Elemente von Aufzählungstypen müssen in einem Programm disjunkt sein:

...

```
typedef enum farbe = { rot, gelb, blau, schwarz };  
typedef haarfarbe = { schwarz, blond, grau, weiss}
```

...

geht nicht!

Typen

Selbstdefinierte Typen

- Strukturierte Typen
 - „Zusammenbau“ eines neuen Typs aus bereits bestehenden Typen (oder Vektoren davon)
 - Beispiel: Typ Person besteht aus
 - Vorname vom Typ char[20]
 - Nachname vom Typ char[20]
 - Körpergröße (in Zentimeterern) vom Typ unsigned char
 - Gewicht (in Kilo) vom Typ unsigned char
 - Haarfarbe vom Typ Farbe (Aufzählungstyp)

Typen

Selbstdefinierte Typen

- Strukturierte Typen
 - Beispiel: Typ Person in C-Notation:

```
typedef struct person
{
    char Vorname[20];
    char Nachname[20];
    unsigned char Koerpergroesse; /* in Zentimetern */
    unsigned char Gewicht; /* in Kilo */
    haarfarbe Haar;
};
```

Typen

Selbstdefinierte Typen

- Strukturierte Typen

- Beispiel: Typ Person in C-Notation:

```
typedef struct person
```

```
{
```

```
    char Vorname[20];
```

```
    char Nachname[20];
```

```
    unsigned char Koerpergroesse; /* in Zentimetern */
```

```
    unsigned char Gewicht; /* in Kilo */
```

```
    haarfarbe Haar;
```

```
};
```

Definition des neuen Typs
namens person

als Struktur

mit den Komponenten
Vorname
Nachname
...

Typen

Selbstdefinierte Typen

- Strukturierte Typen
- Beispiel: Alternative (!) Typ Person in C-Notation:

```
typedef struct
{
    char Vorname[20];
    char Nachname[20];
    unsigned char Koerpergroesse; /* in Zentimetern */
    unsigned char Gewicht; /* in Kilo */
    haarfarbe Haar;
} person;
```

Typen

Selbstdefinierte Typen

- Strukturierte Typen
 - Vereinbarung einer Variablen vom Typ Person („wie gewohnt“):
Typname Variablenname;

```
person p1;
```

- Zugriff auf eine Komponente einer Variablen vom Typ Person:
Variablenname.Komponente

```
strcpy(p1.Vorname, "Hans");  
strcpy(p1.Nachname, "Meier");  
p1.Koerpergroesse = 180;  
p1.Gewicht = 78;  
p1.Haar = schwarz;
```


Typen

Selbstdefinierte Typen

- Strukturierte Typen
- Zugriff auf Teile einer Komponente einer Variablen vom Typ Person:

```
...  
p1.Nachname[1] = 'a';  
...
```

Arrayzugriff „wie gewohnt“

Komponente Nachname
vom Typ char[20]

Typen

Selbstdefinierte Typen

- Strukturierte Typen – Typen in Strukturen
 - ```
typedef struct person;
{
 char Vorname[20];
 char Nachname[20];
 char Strasse[30]
 char Ort[20];
};
```
  - ```
typedef struct angestellter  
{ person pers_daten;  
  char personalnummer[8];  
  g_klasse gehalt;  
};
```
- ```
typedef enum g_klasse { tarif, uebertarif, leitend }
```

# Typen

## Selbstdefinierte Typen

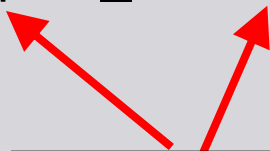
- Strukturierte Typen – Typen in Strukturen - Zugriff

```
typedef struct angestellter
{ person pers_daten;
 char personalnummer[8];
 g_klasse gehalt;
};
```

```
angestellter a1;
```

```
...
```

```
strcpy(a1.pers_daten.Nachname , "Müller");
```



Mehrfacher Komponentenzugriff

# Typen

## Selbstdefinierte Typen

- Strukturierte Typen – Abbildung im Speicher

```
typedef struct person
{
 char Vorname[20];
 char Nachname[20];
 unsigned char Koerpergroesse;
 /* in Zentimetern */
 unsigned char Gewicht;
 /* in Kilo */
 haarfarbe Haar;
};

...
person p;
```

Adr. xyz / p

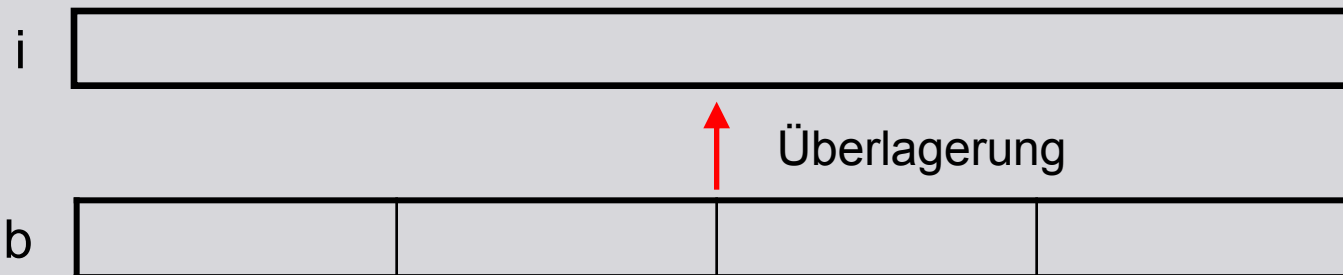
Komponenten  
werden nacheinander  
im Speicher abgelegt

| Arbeitspeicher          |  |
|-------------------------|--|
|                         |  |
| 20 Byte für<br>Vorname  |  |
| 20 Byte für<br>Nachname |  |
| 1 Byte für k.-Groesse   |  |
| 1 Byte für Gewicht      |  |
| 4 Byte für Haar         |  |
|                         |  |

# Typen

## Selbstdefinierte Typen

- Strukturierte Typen mit Überlagerung
  - Motivation: Ein bestimmter Speicherbereich soll unterschiedlich interpretiert werden, z.B.  
die einzelnen Bytes einer Integerzahl  $i$  sollen als Buchstabenvektor  $b$  interpretiert werden:



# Typen

## Selbstdefinierte Typen

- Strukturierte Typen mit Überlagerung
  - Implementierung in C

```
typedef union u
{ int i;
 unsigned char c[4];
};
u u_v;
```

- Die Komponenten i und c von u\_v belegen den selben Speicherplatz.  
Je nach Auswahl der Komponente (u\_v.i oder u\_v.c) wird der Speicherplatz unterschiedlich interpretiert.

# Typen

## Selbstdefinierte Typen

- Aufzählungstypen - Beispiele

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

```
int main()
```

```
{
```

```
 int j;
```

```
 ...
```

```
 typedef union u
```

```
 { int i;
```

```
 unsigned char c[4];
```

```
 };
```

```
 u v;
```

```
 v.i = 0xFFFFFFFF;
```

```
 for (j=0;j<4;j++)
```

```
 printf("%d\n",v.c[j]);
```

```
 ...
```

```
}
```



```
C:\Dokumente und Einstellungen\Admin\Eigene Dateien\FH Deggendorf\Vorlesung Einföhru...
255
255
255
255
```

# Typen

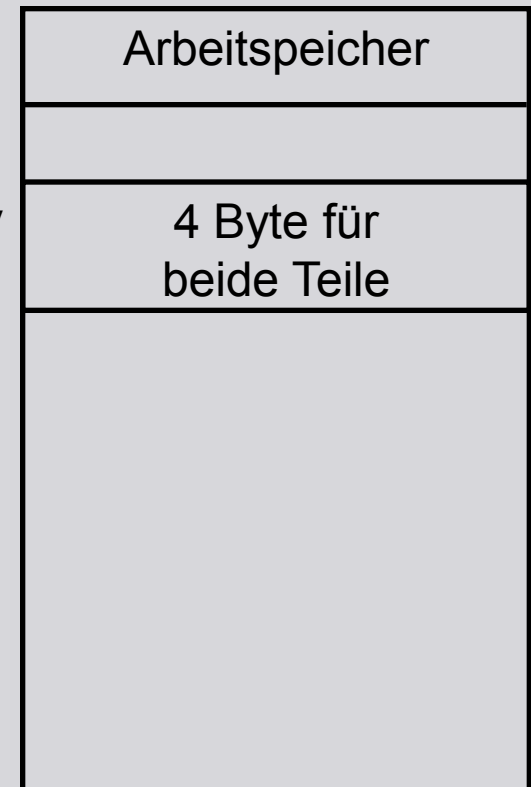
## Selbstdefinierte Typen

- Strukturierte Typen mit Überlagerung
  - Abbildung im Speicher

```
typedef union u
{ int i;
 unsigned char c[4];
};
u u_v;
```

Adr. xyz / u\_v

Komponenten  
werden überlagert im  
Speicher abgelegt





# Typen

## Selbstdefinierte Typen

- können aufgebaut werden aus bereits vorhandenen (und ggf. auch selbst definierten Typen)
  - in Strukturen: `typedef struct { ... } neuer_Typ`
  - in Vektoren: `neuer_Typ Feld_aus_neuem_Typ[10];`
- wird nur eine einzige Variable eines selbst definierten Typs benötigt, so kann diese direkt (ohne Typdefinition) definiert werden
  - `enum { rot, gelb, gruen } ampelfarben;`
  - `struct { int i; int j } zahlenstruktur;`

# Typen

**Zum Schluss dieses Abschnitts ...**

**Noch Fragen ??**