

PR: Programmiersprache C (Linux)
Einheit 5: Datentypen, Pointer (I)

Roland A. Eggetsberger
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

Datentypen

- Strukturen
- Unions
- Aufzählungen
- Bitfelder

Strukturen (I)

- Modell einer Struktur (Bsp.)

Strukturen (II)

- Es werden verschiedene Datentypen zusammengefasst.
- Vorteile
 - Parameterübergabe (elementar)
 - Datensätze als logische Einheit
 - Platzsparend

Strukturen (III)

- Definition (Bsp.)

```
struct adresse {
    char strasse[30];
    int hausnummer;
    int plz;
    char ort[20];
} eineadresse;
```

adresse ist dabei ein Tag, es wird kein Datentyp definiert.

Strukturen (IV)

- Definition (Bsp.)

```
struct adresse {
    char strasse[30];
    int hausnummer;
    int plz;
    char ort[20];
};
struct adresse eineadresse;
```

Strukturen (V)

- Initialisierung (Bsp.)

```
struct adresse eineadresse = {
    "Altenbergerstr.",
    69,
    4040,
    "Linz"
};
```

Strukturen (VI)

- Zuweisung (Bsp.)

```
:
int einehausnummer;
int eineplz;
:
eineadresse.hausnummer = einehausnummer;
eineadresse.plz = eineplz;
```

Strukturen (VII)

- Datentypdefinition (Bsp.)

```
typedef struct adresse {
    char strasse[30];
    int hausnummer;
    int plz;
    char ort[20];
} adressentyp;
adressentyp eineadresse;
```

Strukturen (VIII)

- Felder von Strukturen (Bsp.)

```
int eineplz;
adressentyp adressen[10];
:
eineplz = adressen[0].plz;
```

Strukturen (IX)

- Operationen

Übergabe an Funktion (by value)
Rückgabe aus Funktion
Zuweisung an eine Struktur
Anwendung von Adress- und sizeof-Operator

Strukturen (X)

- Alignment (I)

Die Anordnung im Speicher ist nicht vorhersehbar.

```
struct {
    char c;
    long l;
} test;
printf("%ld\n", sizeof(test));
```

... liefert daher 5, 6 oder 8.

Strukturen (XI)

- Alignment (II)
Probleme daher bei:
Direktem Speicherzugriff
Zugriff auf Dateien

Unions (I)

- Modell einer Union (Bsp.)



Unions (II)

- Auch hier werden verschiedene Typen zusammengefasst.
- Der Zusammenhang ist jedoch enger.

Unions (III)

- Definition (Bsp.)

```
union zahl {
    short shortzahl;
    long longzahl;
} einezahl;
```

Auch hier ist `zahl` wieder ein Tag.

Unions (IV)

- Zuweisung (Bsp.)

```
:
short zweitezahl;
:
zweitezahl = einezahl.shortzahl;
```

Unions (V)

- Bemerkung
Oft ist eine Kombination von Strukturen und Unions günstig.

Unions (VI)

- Probleme
Lesen nach dem Schreiben
Frage - welcher Typ?

Aufzählungen (I)

- Erklärung
Oft sind bei `int` nur gewisse Werte interessant.
Diese lassen sich mit Namen versehen.

Aufzählungen (II)

- Definition (Bsp.)

```
enum tage {
    mon,
    die,
    ...
} woche;
```

Aufzählungen (III)

- Definition (Bsp.)
Dabei ist es möglich die Bereiche zu verschieben
(ein Startwert wird festgelegt).

```
enum tage {
    mon = 1,
    die,
    ...
} woche;
```

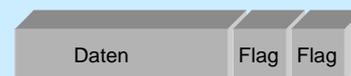
Aufzählungen (IV)

- Es können auch Typen definiert werden.

```
typedef enum {
    FALSE,
    TRUE,
} bool;
```

Bitfelder (I)

- Modell eines Bitfeldes (Bsp.)



Bitfelder (II)

- Verwendung
Strukturen lassen sich auch dichter packen
- Vorteile
Es wird automatisch maskiert
Kein Überschreiben möglich

Bitfelder (III)

- Definition (Bsp.)

```
struct bits {
    unsigned int daten:14;
    unsigned int flag1:1;
    unsigned int flag2:1;
};
```

Bitfelder (IV)

- Nicht möglich
Felder in Bitfeldern

Pointer

Verwendung Beschreibung Definition Zuweisung Untypisierte Pointer Null Pointer Speicherreservierung Speicherfreigabe	I/O Datenstrukturen
---	------------------------

Verwendung (I)

- Oft nicht anders möglich
- Effizienter Code
- Mächtiges Werkzeug

Verwendung (II)

- Arrays
- Strukturen
- Funktionen

Verwendung (III)

- Beispiel
Typisch ist das Erreichen eines call by reference.

```
void zaehle(int *zahl_ptr) {
    (*zahl_ptr)++;
}
```

Beschreibung (I)

- Ein Pointer ist eine Variable.
- Inhalt ist die Speicheradresse einer anderen Variable.



Beschreibung (II)

- Wie bekommt man die Adresse?
Adressoperator (&)
- Wie erhält man den Inhalt?
Umleitungsoperator (*)

Definition

- Form
`int *zahl_ptr;`
Dieser Pointer enthält noch keine Adresse.
Der Typ ist aber festgelegt.

Zuweisung (I)

- Adresszuweisung
`zahl_ptr = wert_ptr;`
- Wertzuweisung
`*zahl_ptr = *wert_ptr;`

Zuweisung (II)

- Beispiel
`int zahl, wert = 1;`
`int *ptr;`

`ptr = &zahl;`
`*ptr = 2;`
`wert = *ptr;`
`wert = ptr;`

Zuweisung (III)

- Nicht erlaubt ist

```
int zahl;
*zahl;
```

- Sehr wohl erlaubt ist

```
int *ptr;
&ptr;
```

Untypisierte Pointer

- Form

```
void *ptr;
```

- Kann beliebig typisierten Pointern zugewiesen werden und umgekehrt.

Null Pointer

- Bemerkung

In der Standardlibrary ist eine Konstante definiert, die für Pointer, die keinen Speicher referenzieren, gedacht ist.

```
NULL
```

Speicherreservierung (I)

- Form

```
#include <stdlib.h>
void *malloc(typ groesse);
```

Speicherreservierung (II)

- Beispiel

```
ptr = malloc(sizeof(int));
if (ptr != NULL)
    *ptr = 1;
else
    printf("Zu wenig Speicher!\n");
```

Speicherreservierung (III)

- Bemerkung

Oft ist folgende Angabe hilfreich

```
...sizeof(*ptr)...
```

Im Allgemeinen falsch ist jedoch

```
...sizeof(ptr)...
```

Speicherfreigabe (I)

- Form

```
#include <stdlib.h>

void free(void *ptr);
```

Speicherfreigabe (II)

- Bemerkung

Trotz Speicherfreigabe zeigt der Pointer noch an die ursprüngliche Stelle.

I/O

- Form

Auch für Pointer gibt es eine Formatanweisung.

```
%p
```

Datenstrukturen (I)

- Listen (I)

```
struct liste {
    datentyp var;
    struct liste *naechstes;
};
struct liste *ptr;
```

Datenstrukturen (II)

- Listen (II)

Der Zugriff auf Komponenten erfolgt via:

```
(*ptr).var
```

Dafür gibt es eine Abkürzung:

```
ptr->var
```

Datenstrukturen (III)

- Listen (III)

Operationen auf Listen
Liste initialisieren
Einfügen von Elementen
Löschen von Elementen
usw.

Datenstrukturen (IV)

- Listen (IV)

Es besteht auch die Möglichkeit eine Liste mehrfach zu verketten.

```
struct liste {
    datentyp var;
    struct liste *naechstes;
    struct liste *voriges;
};
```

Datenstrukturen (V)

- Bäume

```
struct baum {
    datentyp var;
    struct baum *linkes_blatt;
    struct baum *rechtes_blatt;
};
```

Datenstrukturen (VI)

- Stacks

Aufgebaut wie eine Liste.
Jedoch eingeschränkte Funktionen.

Datenstrukturen (VII)

- Queues

Aufgebaut wie eine Liste.
Jedoch eingeschränkte Funktionen.
