Hochschule **Rosenheim**
University of Applied Sciences

Pozedurale Programmierung

Abgeleitete Datentypen

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

Prof. Dr. F.J. Schmitt

Überblick – abgeleitete Datentypen

- Strukturen (`struct`) → wurden bereits komplett behandelt

- Aufzählungen (`enum`)
- Typdefinition mit `typedef`
- Variante Strukturen (`union`)
- Bitfelder

ENUM

Aufzählungen (1)

- Werden in C verwendet, um mehrere Konstanten zu definieren und zu einem Typ zu kombinieren
- Eigenschaften der Konstanten:
  - Alle müssen ganzzahlig sein
  - Zahlen werden der Reihe nach aufsteigend mit 0 beginnend vergeben
- Beispiel:

```
enum Enumname_e
{
   NAME1,   // ist 0
   NAME2,   // ist 1
   NAME3    // ist 2
};
```

```
enum Boolean_e
{
   FALSE,
   TRUE
};
```

Aufzählungen (2)

- Den einzelnen Konstanten können auch explizit ganzzahlige Werte zugeordnet werden

```
enum Farbe_e
{
   FARBE_ROT   = 1,
   FARBE_GRUEN = 2,
   FARBE_BLAU  = 4
};
```

Aufzählungen (3)

- Werden Zahlenwerte weggelassen, so werden die Zahlen wieder aufsteigend vergeben

```
enum Monate_e
{
  MONAT_JANUAR = 1,
  MONAT_FEBRUAR,
  MONAT_MAERZ,
  MONAT_APRIL,
  MONAT_MAI,
  MONAT_JUNI,
  MONAT_JULI,
  MONAT_AUGUST,
  MONAT_SEPTEMBER,
  MONAT_OKTOBER,
  MONAT_NOVEMBER,
  MONAT_DEZEMBER
};
```

Aufzählungen (4)

- Definition einer Variablen

```
enum Enumname_e Variablenliste;
```

- Beispiel:

```
enum Farbe_e ferrari1 = FARBE_ROT;
enum Farbe_e ferrari2 = FARBE_GELB;
```

TYPEDEF

Definition neuer Typnamen (1)

- `typedef`-Vereinbarung definiert einen weiteren Namen für einen bestehenden Datentyp
- Beispiel
  - Bisher:

```
enum Boolean_e
{
   FALSE,
   TRUE
};
```

```
enum Boolean_e var;
```

  - Definition eines neuen Typnamens `Boolean_t`

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

```
Boolean_t var;
```

Definition neuer Typnamen (2)

- Beispiel: Kombination `typedef`-Vereinbarung mit Strukturdefinition

```
typedef struct Adresse_s
{
    char name[30];
    long plz;
    char ort[20];
    char strasse[50];
    long nummer;
} Adresse_t;
```

darf weggelassen werden

- Definition eines Feldes von Datentyp Adresse_t

```
Adresse_t Adressbuch[100];
```

Definition neuer Typnamen (3)

- Umdefinition von elementaren Datentypen
  - wegen anderer Verwendung,
    z.B. char zum Speichern von Zahlen

```
typedef unsigned char byte;
```

  - wegen Portierbarkeit
    - C-Standard garantiert in vielen Fällen (z.B. long) nur eine Mindestlänge
    - d.h. auf verschiedenen Systemen können Datentypen unterschiedliche Wertebereiche haben
    - Lösung:
      - Umdefinition aller Standarddatentypen in einer Header-Datei
      - keine Verwendung von Standardtypen im Code
      - Bei Portierung auf anderes System: nur diese Header-Datei ändern

```
typedef int i32_t;
```

UNION

Variante Strukturen – union (1)

- Werden eingesetzt, wenn
  - verschiedene Attribute zu einer Struktur zusammengefasst werden sollen, die jedoch nicht gleichzeitig auftreten können
  - unterschiedliche Sichtweisen auf die gleichen Daten benötigt werden
- Definition

```
union Unionname_u
{
  Typ Attributname;
  //...
};
```

```
union Zahl_u
{
  double punktZahl;
  long   ganzeZahl;
};
```

- Länge einer Varianten Struktur entspricht der Länge des größten Attributs

Variante Strukturen (2)

- Definition von Variablen

```
union Unionname_u Variablenliste;
```

```
union Zahl_u punktOderGanzeZahl;
```

- Syntax entspricht der von `struct`

Vergleich struct – union

```
struct bsp_s
{
  char a;
  int b;
  double c;
} bsp;
```

```
union bsp_u
{
  char a;
  int b;
  double c;
} bsp;
```

Speicherplatzbedarf:

- 1 + 4 + 8 = 13 Byte
- max(1, 4, 8) = 8 Byte

Zugriff: `bsp.a = 'A';`

- hat keinen Einfluss auf b und c
- verändert b und c

Variante Strukturen – Beispiel

```
struct Kreis_s
{
  double x,y;
  double r;
};

struct Rechteck_s
{
  double x,y;
  double h,b;
};

union GeoObjekt_u
{
  struct Kreis_s kreis;
  struct Rechteck_s rechteck;
};
```

Variante Strukturen
Beispiel (Erweiterung)

Hochschule **Rosenheim**
University of Applied Sciences

```
enum GeoTyp_e
{
  eKREIS, eRECHTECK
};

struct GeoObjekt_s
{
  enum GeoTyp_e typ;

  union GeoObjekt_u
  {
    struct Kreis_s kreis;
    struct Rechteck_s rechteck;
  } objekt;
};
```

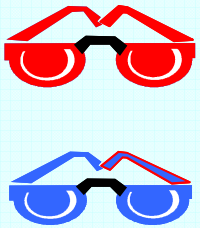
Variante Strukturen – Sichtweisen

- Speicher wird mit unterschiedlichen Brillen betrachtet
- es werden verschiedene Zugriffsvarianten angeboten
- genaue Kenntnis der Datentypgröße erforderlich (systemabhängig)

```
typedef unsigned char byte;        //  8-Bit
typedef unsigned short word;       // 16-Bit

typedef union
{
  word w;
  byte b[4]
} doppel_t;

doppel_t d;
d.w = 0x4142;
// bedeutet: d.b[0] == 0x42 ('B') und d.b[1] == 0x41 ('A')
```



word w	0x42	0x41	ungenutzt	
byte b[4]	b[0]	b[1]	b[2]	b[3]

BITFELDER

Bitfelder

- Bitfelder ermöglichen den einfachen Zugriff auf einzelne Bits
- wird oft benötigt bei system- bzw. hardwarenaher Programmierung
- wie Bitfelder im Speicher liegen ist compilerabhängig, erschwert die Portierbarkeit

Bitfelder – Beispiel

```
typedef union
{
  word w;
  struct
  {
    unsigned bit0_3       : 4;     // Länge 4 Bit
    unsigned bit4         : 1;     // Länge 1 Bit
    unsigned bit5         : 1;
    unsigned bit6         : 1;
    unsigned bit7         : 1;
    unsigned bit8_15      : 8;
  } bitweise;
} t_doppel;

t_doppel d;
d.w=0xFFFF;
d.bitweise.bit4=0;          // d.w = 0xFFEF;
d.bitweise.bit5=0;
d.bitweise.bit6=0;
d.bitweise.bit6=5;          // es werden nur die unteren n Bit berücksichtigt,
                            // n = Länge des Elements; => bit6 = 1
d.bitweise.bit6=6;          // => bit6 = 0,
d.bitweise.bit0_3 = 0xF;    // alle 4 Bit == 1
```

Zusammenfassung

- Aufzählungen
  - Schlüsselwort „enum“
  - ermöglicht Verwendung von symbolischen Werten statt Zahlen
- Variante Strukturen
  - Schlüsselwort „union“
  - verschiedene Sicht auf die gleichen Daten
- Typdefinitionen
  - Schlüsselwort „typedef“
  - Einführung benutzerdefinierter Datentypenbezeichnungen