

Unionen (*unions*) und Bitfelder (*bit fields*) in C

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 1

Unionen und Bitfelder in C

- In diesem Kapitel wollen wir zeigen:
 - ➔ was Unionen sind
 - ➔ wie man Unionen vereinbart
 - ➔ wozu man Unionen verwendet
 - ➔ was Bitfelder sind
 - ➔ wie man Bitfelder vereinbart
 - ➔ wozu man Bitfelder verwendet

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 2

Was sind Unionen (*unions*) ?

Unionen sind ➡ **Strukturen mit Alternativen.**

- Die Variablen von Unionen können **unterschiedlichen Typs** sein.
- **Zu jedem Zeitpunkt** kann aber nur auf **eine der Variablentypen** zugegriffen werden.
- Man vereinbart Unionen, mit der ➡ **union-Anweisung.**

Beispiel:

```

union utyp1 {
    int i ;
    char c [2] ;
    float f ;
} test ;
  
```

Zugriff **entweder**
nur auf **i** oder
nur auf **c [2]** oder
nur auf **f** !

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
 Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
 K16_ Unionen & Bitfelder.ppt, Seite 3

Unionen (1): Verwendung

- Der **Speicherplatz von Unionen** wird bei der Kompilierung **statisch festgelegt**
- Er kann zur Laufzeit des Programms nicht mehr verändert werden.
- Man verwendet Unionen, um
 - **Speicherplatz zu sparen**
 - Variablen in **unterschiedlicher Form** zu interpretieren
 - einfacher auf **einzelne Bits (Bitfelder)** zugreifen zu können.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
 Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
 K16_ Unionen & Bitfelder.ppt, Seite 4

Unionen (2): Vereinbarung

Beispiel:

```
union utyp1 {
    int i ;
    char c [2] ;
    float f ;
} test ;
```

- Die Union **utyp1** enthält die 3 Variablen **i**, **c** und **f** im **gleichen** Speicherbereich.
- **utyp1** ist der **Bezeichner** (Etikett = *tag*) **der Struktur**.
utyp1 ist kein Variablenname und belegt keinen Speicher.
- Der einzige **Variablenname** ist **test**.

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 5

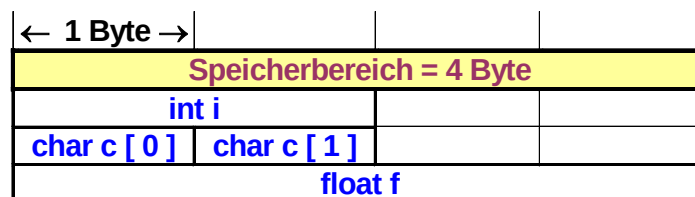
06.06.2004

Unionen (3): Speicheranordnung

Beispiel:

```
union utyp1 {
    int i ;
    char c [2] ;
    float f ;
} test ;
```

- **utyp1** enthält die 3 Variablen **i**, **c** und **f** im **gleichen Speicherbereich**.
- Der Speicher für **test** wird so bemessen, dass die **größte Variable Platz hat**:



Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 6

06.06.2004

Unionen (4): Zugriff

Beispiel:

```
union utyp1 {
    int i ;
    char c [2] ;
    float f ;
} test ;
```

- **utyp1** enthält die 3 Variablen **i**, **c [2]** und **f** im **gleichen Speicherbereich**.
- Der Variablen **test** kann **jeweils nur eine** der in der **union** enthaltenen Variablen - hier **i**, **c [2]** und **f** - zugewiesen werden.
- Die Zuweisung muss jedoch **konsistent** bleiben:
Der Programmierer muss sicher stellen, **dass der Datentyp, welcher entnommen wird, derselbe ist, der zuletzt gespeichert wurde.**

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 7

Unionen (5): alternative Vereinbarung, Felder und Zeiger

- Unionen können mit  **typedef** vereinbart werden.
- Es dürfen auch  **union-Felder** vereinbart werden.
- Auch  **Zeiger auf union-Felder** sind zulässig.



Beispiel:

```
typedef union {
    int i ;
    char c [2] ;
    float f ;
} UTYP1;

UTYP1 uv [2] ;
UTYP1 *p_uv ;
```



06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 8

Unionen (6): Initialisierung

- Bei Unionen kann **nur** die
 ➔ **1. Variable initialisiert werden.**
- Bei union-Feldern gleicht die Initialisierung derjenigen von Feldern:

Der **konstante Initiator** steht in
 ➔ **geschweiften Klammern.**

Beispiel:

```
typedef union {
    int i ;
    char c [2] ;
    float f ;
} UTP1;

UTP1 uv [2] = { 4321, 7856 }, *p_uv ;
```

- **Hier dürfen nur int-Konstanten, die sich auf i beziehen, zur Initialisierung verwendet werden.**

Hochschule für angewandte Wissenschaften Hamburg
 Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
 K16_ Unionen & Bitfelder.ppt, Seite 9

06.06.2004

Unionen (7): Operationen (1)

Beispiel:

```
union utyp1 {
    int i ;
    char c [2] ;
    float f ;
};

union utyp1 v ;
int x ;
short d ;
```

- Bei Unionen verwendet C **dieselben Operatoren wie bei Strukturen** für

➔ die Zuweisung einer union-Komponente: **v.i = 321 ;**

➔ die Auswahl von union-Komponenten: **x = v.i ;**
d = v.c[1] ;

Hochschule für angewandte Wissenschaften Hamburg
 Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
 K16_ Unionen & Bitfelder.ppt, Seite 10

06.06.2004

Unionen (8): Operationen (2)

Beispiel:

```
typedef union { int i ;
                char c [2] ;
                float f ;
            } UTP1;
UTP1 xyz [3] , *p_xyz ;
unsigned long w ;
int x ;
```

- Für Unionen verwendet C **dieselben Operatoren wie für Strukturen** für

➔ den Speicherbedarf:

```
w = sizeof (UTP1) ;
```

➔ Zeiger und Adressen:

```
p_xyz = & xyz [2] ;
```



```
x = p_xyz -> f ;
```

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_Unionen & Bitfelder.ppt, Seite 11

06.06.2004

Was sind Bitfelder (*bit-fields*) ?

- Ein ➔ **Bitfeld** ist eine **Komponente** einer **Struktur, die bitweise organisiert** ist.
- Man verwendet Bitfelder beispielsweise in der hardwarenahen Programmierung, um
 - **Bits zusammenzufassen**
 - auf Peripheriebausteine **ohne die Operatoren** (definiert für **int** oder **char**)
& (bitweise UND),
| (bitweise ODER),
^ (bitweise ANTIVALENZ)
auf einzelne Bits zugreifen zu können.

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_Unionen & Bitfelder.ppt, Seite 12

06.06.2004

Bitfelder: Vereinbarung

- Man vereinbart Bitfelder ähnlich wie Strukturen, z.B. mit einer ➡ **typedef**-Anweisung

Beispiel:

```
typedef struct {
    unsigned int    ein_bit : 1 ;
    unsigned int    zwei_bit : 2 ;
    unsigned int    drei_bit : 3 ;
    unsigned int    noch_2_bit : 2 ;
} BITFELD_1 ;
```

- Wie werden die Bits in **BITFELD_1** angeordnet?

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_Unionen & Bitfelder.ppt, Seite 13

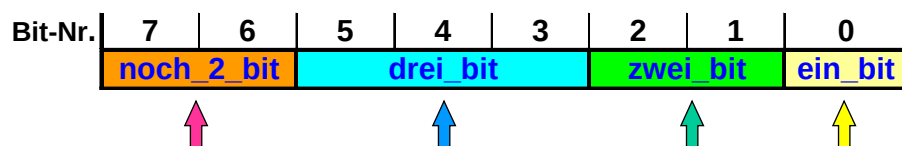
Bitfelder: Speicheranordnung

- Bitfelder sind nur **bedingt portierbar**:
Die Speicherbelegung hängt von der **Implementierung des Compilerherstellers** ab.
- Meistens (auch bei *MSVC*) werden die Bitfelder **nahtlos, beginnend mit dem LSB** aneinandergefügt.

Beispiel:

```
typedef struct { unsigned int    ein_bit : 1 ;
                unsigned int    zwei_bit : 2 ;
                unsigned int    drei_bit : 3 ;
                unsigned int    noch_2_bit : 2 ;
                } BITFELD_1 ;
```

LSB



06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_Unionen & Bitfelder.ppt, Seite 14

Bitfelder: Portierbarkeit (1)

- Bitfelder sind nur **bedingt** auf andere Prozessoren **portierbar**.
- Der *ANSI C Standard* lässt für Bitfelder nur die Datentypen **int**, **unsigned int** und **signed int** zu.
 - *MSVC* erlaubt zusätzlich die Datentypen **char**, **long** mit dem Zusatz **signed** und **unsigned**.
 - Beim Datentyp **signed** wird das **MSB** (prozessorabhängig) als **Vorzeichenbit einer 2-Komplement-Darstellung** interpretiert.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 15

Bitfelder: Portierbarkeit (2)

- Die Anordnung der Bits in einem **Bitfeld** ist **compiler- und prozessorabhängig**.
- Beispielsweise kann die Reihenfolge der Bits auch mit dem **MSB** beginnen:

MSB

Bit-Nr.	7	6	5	4	3	2	1	0
	ein_bit	zwei_bit		drei_bit			noch_2_bit	

06.06.2004

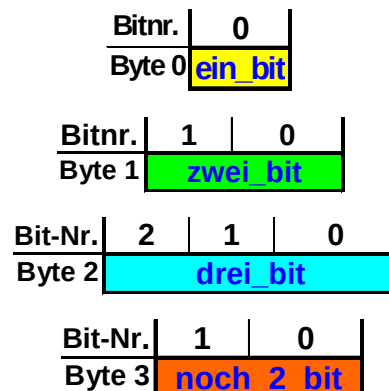
Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 16

Bitfelder: Portierbarkeit (3)

- Die Anordnung der Bits in einem **Bitfeld** ist **compiler- und prozessorabhängig**.
- Die Bitfelder können u.U. auch auf verschiedene Bytes verteilt werden:



06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 17

Bitfelder: Operatoren (1)

- Auf Bitfelder können **fast alle Operatoren** angewendet werden, **die auch für Strukturen zulässig sind**.
- Mathematische Operationen, die über den **Zahlbereich einer Bitfeldkomponente von k Bit Länge hinausgehen**, werden als

➔ **modulo(2^k)-Rechenoperationen** ausgeführt.

Beispiel:

```
...
struct { unsigned int  bit1 : 1;
          unsigned int  bit2 : 3; } bf;
...
bf.bit2 = 9; /* Diese Operation mit der Dezimalzahl 9 wird
              modulo 2^3 ausgeführt, weil die Komponente
              bit2 3 Bit lang ist. */
```

Ergebnis: **bf.bit2 = 9 modulo (2^3) = 9 modulo 8 = 1**

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 18

Bitfelder: Operatoren (2)

- Komponenten von Bitfeldern

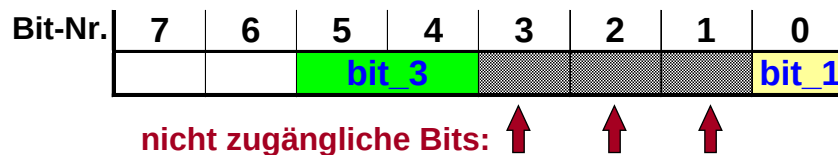
➔ **müssen nicht zwingend benannt werden.**

Man kann dann darauf mit dem Bitfeld **nicht zugreifen.**

- Auf diese Weise kann man bei der hardwarenahen Programmierung **nicht zugängliche Bits in Registern berücksichtigen.**

Beispiel:

```
typedef struct { unsigned int bit_1: 1;
                ➔ unsigned int      : 3;
                unsigned int bit_3: 2; } BITF_1;
```



Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_Unionen & Bitfelder.ppt, Seite 19

06.06.2004

Bitfelder: Operatoren (3)

- Auf Bitfelder können **fast alle Operatoren** angewendet werden, **die auch für Strukturen zulässig sind.**

Eine Ausnahme bildet der

➔ **Adressoperator &Variable.Bitfeldkomponente**

- Der Adressoperator & darf in *Borland C* **nicht auf eine Bitfeldkomponente angewendet werden**, da nicht sichergestellt ist, dass die Komponente auf einer zulässigen Adresse liegt.
- Andere Compilerhersteller erlauben den Gebrauch des Adressoperators für Bitfeldkomponenten, in der Regel falls jede Komponente in einem **eigenen Byte oder Wort** abgelegt wird.

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_Unionen & Bitfelder.ppt, Seite 20

06.06.2004

**Aufgabe: Bitfelder (1)**

Untersuchen Sie das folgende Programm mit Bitfeldern!

```
typedef struct { unsigned int    ein_bit : 1 ;
                unsigned int    zwei_bit : 2 ;
                unsigned int    drei_bit : 3 ;
                unsigned int    noch_2_bit : 2 ;
                unsigned int      : 7 ;
                unsigned int    mehr_bits : 3 ; } BITFELD_1 ;

int main (void)
{ BITFELD_1 bitfeld, bitfeld1, *p_bt = &bitfeld ;
  char *p_bt3 = &bitfeld.drei_bit ;
  bitfeld.ein_bit = 1 ;      bitfeld.zwei_bit = 2 ;      bitfeld.drei_bit = 5 ;
  bitfeld.noch_2_bit = 10 ;
  if ( bitfeld.drei_bit == 5 ) bitfeld.drei_bit += 3 ;
  bitfeld.mehr_bits = 3 ;                      bitfeld1 = *p_bt ;
  bitfeld1.zwei_bit ^= 5 ;                      bitfeld1 ^= 5 ;
  bitfeld1.mehr_bits = *p_bt3 ;
  return 0 ; }
```

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 21

06.06.2004

**Aufgabe: Bitfelder (2)**

1. Welche Fehler enthält das obige Programm?
2. Korrigieren Sie die Fehler!
3. Beobachten Sie die Bitfeldvariablen im *Borland C-Debugger* im Einzelschrittbetrieb mit der *watch*-Funktion!

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K16_ Unionen & Bitfelder.ppt, Seite 22

06.06.2004

✓ Aufgabe: Bitfelder (1)

1. Fehler:

```
typedef struct { unsigned int    ein_bit : 1 ;
                unsigned int    zwei_bit : 2 ;
                unsigned int    drei_bit : 3 ;
                unsigned int    noch_2_bit : 2 ;
                unsigned int      : 7 ;
                unsigned int    mehr_bits : 3 ; } BITFELD_1 ;

int main (void)
{ ...
  char *p_bt3 =
    &bitfeld.drei_bit ;
  ...
  bitfeld1.zwei_bit ^= 5 ;
  bitfeld1 ^= 5 ;
  ...
  bitfeld1.mehr_bits = *p_bt3 ;
  ...
  return 0; }
```

← '&' auf Bitfeld ignoriert
'initializing': inkompatible Datentypen -
von 'unsigned int *' zu 'char *'

← ^= Ungültig fuer struct



✓ Aufgabe: Bitfelder (2)

1. korrigiertes Programm:

```
typedef struct { unsigned int    ein_bit : 1 ;
                unsigned int    zwei_bit : 2 ;
                unsigned int    drei_bit : 3 ;
                unsigned int    noch_2_bit : 2 ;
                unsigned int      : 7 ;
                unsigned int    mehr_bits : 3 ; } BITFELD_1 ;

int main (void)
{ BITFELD_1 bitfeld, bitfeld1, *p_bt = &bitfeld ;
  /* char *p_bt3 =&bitfeld.drei_bit ; fehlerhafte Zeile auskommentiert */
  bitfeld.ein_bit = 1 ;          bitfeld.zwei_bit = 2 ;
  bitfeld.drei_bit = 5 ;
  bitfeld.noch_2_bit = 10 ;
  if ( bitfeld.drei_bit == 5 )   bitfeld.drei_bit += 3 ;
  bitfeld.mehr_bits = 3 ;        bitfeld1 = *p_bt ;
  bitfeld1.zwei_bit ^= 5 ; /* bitfeld1 ^= 5 ; falsche Zeile kommentiert */
  bitfeld1.mehr_bits = bitfeld.drei_bit ; /* korrigierte neue Zeile */
  return 0; }
```

