

Felder in C

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 1

Felder in C

- In diesem Kapitel wollen wir zeigen,
 - ➔ was Felder (*arrays*) sind
 - ➔ wie man in C Felder vereinbart
 - ➔ welche Eigenschaften Felder haben
 - ➔ dass Felder als Eingangsparameter in Funktionen auf andere Weise behandelt werden als normale Variable.
 - ➔ wozu man Felder verwendet.

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 2

Felder (*arrays*) in C (1)

- Was sind Felder?
- Felder sind Namensvereinbarungen, die **mehr als eine Variable** oder **Konstante desselben Datentyps** enthalten.
 - Ein Feld enthält **Feldelemente** (*components*), auf die mit einem **gemeinsamen Namen** und **unterschiedlichen Indizes** zugegriffen wird.

Für die Indizes in C **gilt immer**:

➡ $0 \leq \text{Index} \leq \text{Anzahl der Komponenten}-1$

Beispiel:

```
int vektor [4] ; /* Das Feld vektor enthält die 4 Komponenten:  
                vektor [0], vektor [1], vektor [2], vektor [3] */
```

02.12.2004



Feldvereinbarungen in C

- Wie vereinbart man Felder?
- Der C-Compiler erkennt Felder an den auf einen Namen folgenden **eckigen Klammern []**.
- Im Unterschied zu anderen Programmiersprachen ist in C kein **eigenes Schlüsselwort für Felder** vorgesehen.

Beispiel:

```
int vektor [4] ; /* Das int-Feld vektor enthält die 4 Komponenten:  
                vektor [0], vektor [1], vektor [2], vektor [3] */
```

02.12.2004



Initialisierung von Feldern in C

- Bei der **Initialisierung von Feldern** werden die Feldelemente nacheinander den Werten in den geschweiften Klammer zugewiesen, beginnend mit der Komponente ... **[0]**.
- Werden zu wenige Initialisierungswerte angegeben, so ist der  **Rest der Komponenten gleich 0**.
 Leere Werte sind unzulässig.
- Ist die  **Feldlänge nicht angegeben**, so dimensioniert der Compiler das Feld selbst.

Beispiele:

`int f [4] = {20, 30, 45} ; /* f [0] = 20, f [1] = 30, f [2] = 45, f [3] = 0 */`

`int f [4] = {20, ,30, 45} ; /* Fehlermeldung des C-Compilers */`

`int a [ ] = {4, 500, -1} ; /* Feld a: a [0] = 4, a [1] = 500, a [2] = -1 */`

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 5

02.12.2004

Zugriff auf Felder in C

- Man kann mit **beliebigen (auch gefährlichen!) Integerausdrücken** auf die Feldkomponenten zugreifen.

Beispiele:

`int k =10, vektor [ 5 ] = { 5, 4, 3, 2, 1 }, i = 1, j = 2 ;`

`float feld [10] = { 0. }, f [4] = { 0. }; /* { 0.} initialisiert
alle Feldelemente mit 0.`

`Eine leere Klammer { } ist hier verboten! */`

...

`feld [9] = 1000. ;`

`while ( k>5 ) feld [--k]++ ;`

...

`f [ j+1 ] = 3*feld [ k ] + feld [ 9 ] ;`

...

`for ( i = 0 ; i < 5 ; ) vektor [ i ] *= ( 2 + i++ ) ; /* * * */`

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 6

02.12.2004

Funktion der eckigen Klammer [] in C

💣 **Man beachte:** Die Zahl in den eckigen Klammern [] hat **unterschiedliche Bedeutung:**

➡ Bei der **Felddeklaration und –initialisierung** gibt sie die **Anzahl der Feldelemente** an.

Beispiel:

`float feld [ 10 ] = { 0. } ;` /* [10] ist die Anzahl der Elemente im Float-Feld. Alle werden auf 0 gesetzt. */

➡ Beim **Feldzugriff** bezeichnet sie den **Index des Feldelementes**.

Beispiel:

`feld [ 3 ] = 7 * feld [ 2 ] ;` /* Das Feldelement mit dem Index 2 wird mit 7 multipliziert und im Feldelement mit dem Index 3 gespeichert */



💣 Fehlerträchtiger Zugriff auf Felder in C

- Beim Zugriff auf Feldkomponenten mit den Inkrement- und Dekrement-Operatoren ist Vorsicht geboten, weil die **Reihenfolge bei der Abarbeitung der Operatoren in C nicht festgelegt** ist.
- Das **Ergebnis** eines Programms, welches diese Regel missachtet, ist unter Umständen **compilerabhängig**.



schlechtes Beispiel:

`int i = 1 ;` /* .. */
`for ( i = 0 ; i < 5 ; ) vektor [ i ] *= ( 2 + i++ ) ;`

- In der Regel berechnet der **MSVC-Compiler** die **Präoperationen bevor** und die **Postoperationen, nachdem alle anderen Operationen der aktuellen Anweisung ausgeführt sind**.





Felder und Reihenfolge der Operatoren

Untersuchen Sie den folgende schlechten Programmausschnitt!

1. Geben Sie 2 mögliche alternative Programmschleifen an, bei denen der Ablauf der Operationen eindeutig, das Ergebnis jedoch nicht dasselbe ist.
2. Ermitteln Sie das Ergebnis für den *MSVC*-Compiler! Aus Ihrer Lösung soll erkennbar sein, wie Sie zu Ihren Werten kommen! Beachten Sie die Abarbeitung der Postinkrementvariablen bei *MSVC* !
3. Überzeugen Sie sich vom Ergebnis mit Hilfe eines Testprogramms!
4. Schreiben Sie statt des schlechten Schleifenbeispiels einen eindeutigen Programmausschnitt gleicher Funktion!

```
/* ... */
int i, v[3] = { 1, 2, 3 };
for (i = 0; i < 3; ) v[i] *= (2 + i++);
/* ... */
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 9



Felder und Reihenfolge der Operatoren (1)

1. Programmalternativen:

- Programm 1:

```
for (i = 0; i < 3 ;)
{
    v[i] = v[i] * (2 + i);
    i++;
}
```
- Programm 2:

```
for (i = 0; i < 3 ; )
{
    Zwischenspeicher = 2+i ;
    i++ ;
    v[i] = v[i] * Zwischenspeicher ;
}
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 10

✓ Felder und Reihenfolge der Operatoren (2)

2. MSVC 6.0:

• Programm

```
int i,
v[3] =
{1, 2, 3};

/* ... */

for (i = 0; i < 3;)
v[i] *=
(2 + i++);
```

• Schleifendurchläufe

1.	2.	3.
$i = 0 < 3$ $v[0] = 2$ $i++ = 1$	$i = 1 < 3$ $v[1] = 6$ $i++ = 2$	$i = 2 < 3$ $v[2] = 12$ $i++ = 3$
4. $i = 3 \geq 3 \Rightarrow$ Schleifenabbruch		



✓ Felder und Reihenfolge der Operatoren (3)

3. C Testprogramm:

/* Datei: k7_a0.c U. Vogelsang, HAW Hamburg, FB E&I 1.9.2000 */

/* Programm testet Reihenfolge der Operationen in MSVC */

#include <stdio.h>

int main (void)

```
{   int i, v[3] = { 1, 2, 3 };
    printf ("Programm testet Reihenfolge der Operationen in"
           "MSVC 6.0 \n\n ");
    printf (" Vektor v initialisiert mit: v [0] =1, v [1] =2, v [2] =3\n\n") ;
    for ( i = 0; i < 3; )
    { printf ( "\n\n Vor der Operation: v [i] *= ( 2 + i++ );\n"
              " i=%d, v [i]= %d\n\n", i, v [i] ) ;
      v [i] *= ( 2 + i++ ) ;
      printf ( "Nach der Operation:\n  i-1=%d, v [i-1]=%d,"
              "i=%d\n", i-1, v [i-1], i ) ;
    }
    return 0 ; }
```



✓ Felder und Reihenfolge der Operatoren (4)

4. Besseres Schleifenprogramm:

```
/* ... */  
int i, v [3] = { 1, 2, 3 } ;  
for ( i = 0; i < 3; i++)    v [ i ] = v [ i ] * ( 2 + i ) ;  
/* ... */
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 13

Feldgrenzen in C (1)

- Der **Programmierer** ist für die Einhaltung der Feldgrenzen **verantwortlich**:

☛ Greift man auf Komponenten **jenseits der** definierten **Feldgrenzen** zu, so hat das **katastrophale Folgen**:

➡ Der **Lesezugriff** liefert einen **undefinierten Wert**.

➡ Beim **Schreibzugriff überschreibt** man **fremden** Daten- oder Programmspeicher.

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 14

Feldgrenzen in C (2)

- ☛ Greift man auf Komponenten **jenseits der** definierten **Feldgrenzen** zu, so hat das **katastrophale Folgen**:

Beispiel:

```
/* .... */
```

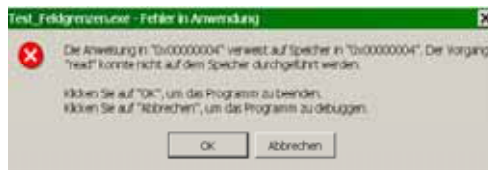
```
int vektor [5], i=1 ;
```

```
for ( i = 0 ; i <= 6 ; i++ ) vektor [i] = i - 2 ;
```

```
/* i = 5, 6:
```

nicht-zugeordneter Speicher wird überschrieben! */

➔ **MSVC-Fehlermeldung:**



- Eine Fehlermeldung **muss** nicht auftreten!

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 15

Zeichenketten (strings) in C

- Zeichenketten (ASCII-Zeichenfolgen) werden in Feldern vom Typ **char** gespeichert. Dabei belegt jedes **ASCII-Zeichen** ein Feldelement.
- Die **char**-Zeichenkette wird **immer** mit **ASCII NUL = 0x00** abgeschlossen.
Deshalb ist der Speicherbedarf des **ASCII-Zeichenfeldes** immer um **1 Byte größer** als die Anzahl der Buchstaben.
- Im Unterschied zu einzelnen **ASCII-Zeichen (string-literals)** wird eine Zeichenkette (**string**) in **Anführungszeichen " ... "** eingeschlossen.

Beispiel:

Der String **s[20]** enthält maximal 19 Buchstaben = 20 **ASCII-Feldelemente**

```
"Ich bin ein string!";
```

```
/* s [0] = 'I', s [1] = 'c', s [2] = 'h',  
s [3] = '...', s [18] = '!', s [19] = 0x00 */
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 16

Initialisierung von Zeichenketten (*strings*) in C

- Zeichenketten (*ASCII*-Zeichenfolgen, *strings*) werden in *char*-Feldern gespeichert
- Dabei belegt jedes Zeichen ein Feldelement.
Die *char*-Zeichenkette wird immer mit *ASCII NUL* = *0x00* abgeschlossen.

➔ Strings können sehr **einfach initialisiert** werden:
Die Feldgrenzen müssen nicht angegeben werden.

Beispiel: String von 19 Buchstaben = 20 *ASCII*-Feldelementen

char s [] = "Ich bin ein string!"; /* s [0] = 'I', s [1] = 'c', s [2] = 'h',
s [3] = '...' , s [18] = '!', s [19] = *0x00* */



02.12.2004



Einlesen von Zeichenketten (*strings*) (1)

- Die *NUL*-(*0x00*)-**Terminierung** erlaubt die **einfache Eingabe** von *char*-Zeichenketten mit Hilfe des ➔ *%s*-Formats.
- Der String wird normalerweise **nur bis zum 1. whitespace eingelesen** und danach wird die Eingabe abgebrochen.

1. Beispiel: Programm für die „normale“ Tastatureingabe von Strings

```
/* ... */
char string [ 80 ] ;
fflush ( stdin ) ;
scanf ( "%s", string );
/* ... */
```

02.12.2004



Einlesen von Zeichenketten (*strings*) (2)

- Der String wird normalerweise **nur bis zum 1. whitespace eingelesen** und danach wird die Eingabe abgebrochen.

1. Beispiel: Programm für die „normale“ Tastatureingabe von Strings

```
/* ... */
char string [ 80 ] ;
fflush ( stdin ) ;
scanf ( "%s", string ) ;    /* ... */
```

Tastatureingabe:

Ich bin ein string!↵

Eingelesener string: "Ich"

Der Einlesevorgang wird mit dem 1. **Whitespace: " "** (blank) abgebrochen!

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 19

Einlesen von Zeichenketten (*strings*) (3)

- Man möchte **ganze Sätze** einlesen, wobei **scanf (..)** die **Whitespaces ignoriert**:
- In diesem Fall kann man mit **[^..]** **ASCII-Zeichen** angeben, mit dem die Eingabe beendet wird, z.B.:
 ➡ **enter = ↵ = \n**

2. Beispiel: Programm für Tastatureingabe von Strings

```
/* ... */
char string [ 80 ] ;
fflush ( stdin ) ;
scanf ( "%[^\\n]", string ) ; /* [^\\n] akzeptiert nur ↵ als
                               Eingabeendezeichen */
/* ... */
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 20

Einlesen von Zeichenketten (*strings*) (4)

- Wenn man über die Tastatur **ganze Sätze** eingeben möchte, kann man mit `scanf ( "%[^\n]", string ) ;` **ASCII-Zeichen** angeben, mit dem die Eingabe beendet wird, z.B.: $\Rightarrow$ **enter = $\backslash$ n**

Tastatureingabe:

Ich bin ein string! $\backslash$ n

Eingelesener string:

"Ich bin ein string!"

Der Eingabe wurde mit

enter = $\backslash$ n

beendet!

02.12.2004



Einlesen von Zeichenketten (*strings*) (5)

- Man möchte **ganze Sätze** einlesen und die Eingabe **nur mit enter = $\backslash$ n** beenden.

Außerdem sollen **nicht mehr als k ASCII-Zeichen** eingelesen werden: `scanf ( "%k[^\n]", string ) ;`

3. Beispiel: Programm für Tastatureingabe von ganzen Sätzen mit einer Beschränkung der Anzahl der ASCII-Zeichen auf 10

```
/* ... */
char string [ 80 ] ;
fflush ( stdin ) ;
scanf ( "%10[^\n]", string ) ;
/* ... */
```

02.12.2004



Einlesen von Zeichenketten (*strings*) (6)

- Wenn man über die Tastatureingabe **mit**
enter = ↵ = \n **beenden** und außerdem **die Anzahl**
der eingelesenen Zeichen beispielsweise auf 10
begrenzen will, so verwendet man:

```
scanf ( "%10[^\n]", string ) ;
```

Tastatureingabe:

Ich bin ein string!↵

Eingelesener String: "Ich bin ei"

Die Eingabe wurde nach 10 Zeichen abgebrochen!

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 23

Vereinfachte Ausgabe von Zeichenketten

- In C ermöglicht die **NUL-(0x00)-Terminierung** eine
vereinfachte Ausgabe von **char**-Zeichenketten.

1.Beispiel: Bildschirmausgabe einer Zeichenkette

```
/* ... */
char string [ ] = "Ich bin ein string!" ;
printf ( "%s", string ) ;
/* ... */
```

2.Beispiel: einfachere Version (u.U. **gefährlich** ☹️): **printf (..)** verlangt minimal nur 1 String als Parameter

```
/* ... */
char string [ ] = "Ich bin ein string!" ;
printf (string) ;
/* ... */
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 24

Verarbeitung von Zeichenketten (*strings*)

- In C ermöglicht die **NUL**-(0x00)-Terminierung eine einfache Verarbeitung von **char**-Zeichenketten.

Beispiel: Kopieren eines Strings

```
/* ... */
int i = 0, k = 0;
char string [100], s [ ] = "Ich bin ein string! " ;
    while ( string [i++] = s [k++] ) ;
    printf ( "s = %s \nstring = %s", s, string ) ;
/* ... */
```

02.12.2004



Felder als Funktionsparameter (1)

- Bei ➡ **Feldern als Parameter von Funktionen** wird die **Funktionsdeklaration** anders gehandhabt als der **Funktionsaufruf**.
 - Bei der **Deklaration der Funktion** muss der Compiler wissen, dass ein Feld übergeben wird. Dies erkennt er **an den eckigen Klammern**.
 - Bei **1-dimensionalen Feldern** muss die **Feldlänge nicht angegeben** werden.

Beispiel für eine Funktionsdeklaration: ↓ ↓

```
unsigned short copy_char0 ( char str1 [ ], char str [ ],
                           char such_zeichen )
```

```
{ /* ... */
}
```

02.12.2004



Felder als Funktionsparameter (2)

- Bei **Feldern als Parameter von Funktionen** wird die **Deklaration** anders gehandhabt als der **Funktionsaufruf**.
- Beim **Funktionsaufruf** mit einem **1-dimensionalen Feld** wird nur der **Feldname ohne eckige Klammern** verwendet.

Beispiel für einen Funktionsaufruf:

```
int main ( void )
{
    char string [ 100 ] ;
    char string1 [ ] =
        "Ich studiere an der HAW Hamburg Elektrotechnik." ;
    char such_zeichen = /* .. */ ;
    /* ... */
    ➔ such_zeichen_nr = copy_char0 ( string1, string, such_zeichen ) ;
    /* ... */
}
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 27

Felder als Funktionsparameter (3)

- Wird einer Funktion ein **Feld als Funktionsparameter** übergeben, so ist der Funktion **die Anzahl der Feldelemente unbekannt**.
- Im allgemeinen muss man **die Feldlänge als zusätzlichen Parameter übergeben**, damit die **Feldgrenzen in der Funktion bekannt** sind.

Beispiel einer Funktionsdeklaration:

```
void func ( float feld1 [ ], short anzahl_elemente )
{
    short element_zaeher ;
    /* ... */
    ➔ for ( element_zaeher = 0; element_zaeher < anzahl_elemente ;
        element_zaeher ++ )
        /* ... */
}
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 28

Felder als Funktionsparameter (4)

- Wenn man **Felder als Parameter von Funktionen** verwendet, so werden sie **anders behandelt als normale Variable**:
 - Ändert man „normale“ **Eingangsvariable innerhalb eines Unterprogramms**, so merkt das aufrufende Programm davon nichts: Die Werte **im Hauptprogramm** bleiben **unverändert**.
 - ✱ Ändert man dagegen **Feldvariable, die Eingangsparameter einer Funktion sind, innerhalb eines Unterprogramms**, so werden die Variablen **auch im Hauptprogramm geändert!**

02.12.2004



Stringfelder als Funktionsparameter

- Ein **Sonderrolle haben Stringfelder**:
- Wird einer Funktion ein **Stringfeld als Funktionsparameter** übergeben, so ist es **häufig nicht nötig**, die Anzahl der Feldelemente anzugeben. ↓
- Die **Terminierung** eines Stringfeldes mit **0x00** kann in Schleifen **zur Feldgrenzenüberwachung** verwendet werden.

Beispiel einer Funktionsdeklaration mit einem Stringfeld:

```
void func ( char stringfeld [ ] )
{
    short element_zaeher ;
    /* ... */
    while ( stringfeld [ element_zaeher ] != 0 )
    { /* ... */ element_zaeher++ ; } /* ... */ }
```

→

02.12.2004



Mehrdimensionale Felder in C

- Häufig werden auch **mehrdimensionale Felder** benötigt:

➔ Das 2-dimensionale Feld **matrix [z] [s]**
enthält **$z * s$** Elemente:
z Zeilen und **s** Spalten

➔ Das 3-dimensionale Feld **tensor [l] [m] [n]**
enthält **$l * m * n$** Elemente.

Beispiele:

int matrix [4] [6] ; */* matrix enthält $l*m = 4*6 = 24$ Elemente */*
float tensor [3] [6] [4] ; */* tensor enthält $l*m*n = 3*6*4 = 72$ Elemente */*

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 31

02.12.2004

Initialisierung mehrdimensionaler Felder in C

- Man initialisiert **mehrdimensionale** Felder
ähnlich wie 1-dimensionale Felder.
- Dabei werden die Feldelemente **zeilenweise** zugewiesen.

Beispiel:

int m1 [2] [3] = { {1, 2}, {11, 22, 33} } ;
zeilenweise Initialisierung:

➔ **m1 [0] [0] = 1 m1 [0] [1] = 2 m1 [0] [2] = 0**
 ➔ **m1 [1] [0] = 11 m1 [1] [1] = 22 m1 [1] [2] = 33**

Das Feldelement **m1 [0] [2]** enthält **0**.

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 32

02.12.2004

Speicherzuordnung bei mehrdimensionalen Feldern in C

- Die Feldelemente werden im Speicher **zeilenweise** abgelegt:

➔ Der nächste Speicherplatz nach **Feld [i, j]** enthält das Feldelement **Feld [i, j+1]**

Beispiel:

Die Elemente des Feldes

```
int f [3] [4] = { {1, 2}, {11, 22, 33} } ;
```

werden folgendermaßen angeordnet:

➔ **f [0] [0] = 1 f [0] [1] = 2 f [0] [2] = 0 f [1] [0] = 11**
f [1] [1] = 22 f [1] [2] = 33 ...

Alle übrigen Feldelemente enthalten **0**.

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 33



Speicherzuordnung bei mehrdimensionalen Feldern in C

➔ Es ist guter Programmierstil, mehrdimensionale Felder bei der Initialisierung zeilenweise aufzuschreiben.



schlechtes Beispiel:

```
int f [3] [4] = { {1, 2}, {11, 22, 33}, {101, 202, 303, 404} } ;
```



gutes Beispiel:

```
int f [3] [4] = {
    {1,    2},
    {11,   22, 33},
    {101, 202, 303, 404 }
};
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 34

Felder

- Untersuchen Sie den folgende Programmausschnitt!
Geben Sie dazu in der richtigen Reihenfolge den Wert oder die Werte an, die die jeweiligen Variablen annehmen!
Aus Ihrer Lösung soll erkennbar sein, wie Sie zu Ihren Werten kommen!

```
/* ... */
int k = 6 ;
float feld a, [6] = { 0., 1., 2., 3., 4., 5. } ;
while ( k > 4 )      a = feld [ --k ] ++ ;
/* ... */
```

02.12.2004



✓ Felder (1)

- Man beachte, dass beispielsweise das **vektor [5]** die Feldelemente **vektor [0], vektor [1], ... , vektor [4]** enthält.
- Man beachte beim Ausdruck **feld [--k]++** den Unterschied zwischen den folgenden Operatoren:
 - k :** der Feld**index k** wird in *MSVC* im allgemeinen um 1 erniedrigt, **bevor k das erste Mal** verwendet wird.
 - feld [...]+:** Das Feldelement **feld [...]** wird in *MSVC* im allgemeinen um 1 erhöht, **nachdem** alle anderen Operationen der Anweisung ausgeführt sind.

02.12.2004



✓ **Felder (2)**

- Programm
- Schleifendurchläufe

1.	2.	3.
<pre>int k = 6 ; /* ... */ float a, feld [6] = {0.,1.,2.,3.,4.,5.} ; while (k > 4) a = feld [-k] ++ ;</pre>	<pre>k = 6 > 4 --k = 5 a = feld[-k] = 5 feld[-k] ++ = 6</pre>	<pre>k = 5 > 4 --k = 4 a = feld[-k] = 4 feld[-k] ++ = 5</pre>
		<pre>k = 4 <= 4 ⇒ Schleifen- abbruch Der Rest des Feldes bleibt unverändert: feld[k] k = 3 .. 0 : { 3, 2, 1, 0 }</pre>

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 37

✓ **Felder (3)**

• Testprogramm:

```
/* Projekt: Operatorenreihenfolge.dsw, U. Vogelsang, HAW HH,
23.11. 04: Programm testet Reihenfolge der Operationen in MSVC */
#include <stdio.h>
int main ( void )
{ int k = 6 ;
  float a, feld [6] = { 0., 1., 2., 3., 4., 5. } ;
  printf ("Programm testet Reihenfolge der Operationen in"
          " MSVC 6.0\n\n" );
  printf (" feld initialisiert mit: feld[0]...feld[5] = 1.0 ...5.0\n\n" );
  while ( k > 4 )
  { printf ( "Vor der Operation: a = feld [ -k ] ++ :\n k=%d", k ) ;
    if (k!=6) printf ( " , feld [k]= %f\n", feld [k] ) ;
    feld [ -k ] ++ ;
    printf ( "\nNach der Operation:\n k=%d, feld [k]= %f\n\n", k,
            feld [k] ) ;
    for ( k = 3; k >= 0; k-- ) printf ( "k=%d, a=%f, feld [k]= %f\n\n",
                                       k, a, feld [k] ) ;
  }
  return 0 ;
}
```

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 38

Parameterübergabe: mehrdimensionale Felder

- Weil C die **Einhaltung der Feldgrenzen nicht überwacht**, braucht der C-Compiler bei
 ➔ **Feldern als Funktionsparametern nur die 2. Felddimension** zur Berechnung der Feldelemente.

Beispiel:

```
void f_m ( int f [2] [3] ) /* Definition der Funktion statt eines Prototyps */
{
    ➔ f [1] [2] = 2 ;      /* relativer Speicherplatz: 1 * 3 + 2 */
}

int main ( void )        /* Hauptprogramm */
{
    int feld [2] [3] = { {10, 20, 30}, {100, 200, 300} } ;
                                /* 2-dimensionale Feldinitialisierung */
    f_m ( feld ) ;          /* Funktionsaufruf */
    /* ... */              /* feld [1] [2] enthält nun 2 */
    return 0 ;
}
```

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



Programmieren in ANSI C:
 K8_C-Felder.ppt, Seite 39

02.12.2004

Parameterübergabe: mehrdimensionale Felder (2)

- Beim Ablauf eines Programms wird bei 2-dimensionalen Feldern bei jedem Zugriff auf ein Feldelement dessen **aktueller Speicherplatz** berechnet:

Aktueller Speicherplatz = Speicherplatz des ersten Feldelements + (Zeilenindex * Anzahl der Spalten) + Spaltenindex

- Die **Zeilenanzahl** wird bei der Speicherplatzberechnung **nicht benötigt**. Sie ist in der Kopfzeile einer Funktion überflüssig.

➔ Falls der **formale Parameter** einer Funktion ein **Feld** ist, so darf der **1. Klammerparameter leer bleiben**. ➔

Beispiele:

```
void func_1 ( int f [ ] )
void f_m ( int f [ ] [4] )
```

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



Programmieren in ANSI C:
 K8_C-Felder.ppt, Seite 40

02.12.2004

Beispiel: 2D-Felder als Funktionsparameter

- Das folgende einfache Beispiel zeigt,
- ➔ wie man Felder als Funktionsparameter verwendet
- ➔ dass im **Unterschied zu anderen Parametern** bei **Feldparametern** in Funktionen die Feldkomponenten **auch im Hauptprogramm** verändert werden
- ➔ wie man Felddimensionen mit dem **Präprozessor-kommando #define** festlegen kann
- ➔ wie man Felder mit Hilfe von Programmschleifen ausgeben kann.

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 41

Beispiel: 2D-Felder als Funktionsparameter (1)

- Das Programmbeispiel zeigt den Umgang mit **2-dimensionalen Feldern** als **Parametern in Funktionen**:
 - In der Funktion **f_m** wird einem Element des initialisierten Feldes **feld** der neue Wert **wert** zugewiesen.
 - Der Zeilen- und Spaltenindex werden **[z] [s]** eingegeben.
 - Das Feld **feld [DIM1] [DIM2]** wird im Unterprogramm **f_m** an der Stelle **[z] (= Zeilennummer) [s] (= Spaltennummer)** mit **wert** überschrieben.

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 42

Beispiel: 2D-Felder als Funktionsparameter (2)

/* Autor: Vogelsang / HAW Hamburg

Datum: 7.12.2001 aktualisiert 24.11.2004

*/

#include <stdio.h>

#define DIM1 3

#define DIM2 4

void f_m (short feld [] [DIM2], short, short, short) ; /* Prototyp */

int main (void) /* Hauptprogramm */

{

short i, j, z, s, ok, wert = 0 ;

short feld [DIM1] [DIM2] = /* 2-dimensionale Feldinitialisierung */

```

    {
        {10, 20, 30, 40},
        {100, 200, 300, 400},
        {1000, 2000, 3000, 4000}
    }

```

};

02.12.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.SauvagerdProgrammieren in ANSI C:
K8_C-Felder.ppt, Seite 43**Beispiel: 2D-Felder als Funktionsparameter (3)**

do

{

printf ("\n\n feld [%2hd] [%2hd]:", DIM1, DIM2) ;

for (j=0; j<DIM1; j++) /* Ausgabe aller DIM1 Zeilen */

{

printf ("\n") ;

for (i=0; i<DIM2; i++) printf(" %4hd", feld [j] [i]) ;

/* Ausgabe der DIM2 Elemente einer Zeile */

}

printf ("\n\n An welcher Stelle [z] [s] soll feld mit 'wert' "

"ueberschrieben werden?\n z s wert: ?\n ") ;

fflush (stdin) ; /* Tastaturpuffer loeschen */

ok = scanf ("%hd %hd %hd", &z, &s, &wert) ;

f_m (feld, z, s, wert) ; /* Funktionsaufruf */

/* feld [z] [s] enthaelt nun wert */

02.12.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.SauvagerdProgrammieren in ANSI C:
K8_C-Felder.ppt, Seite 44

Beispiel: 2D-Felder als Funktionsparameter (4)

```

f_m ( feld, z, s, wert );           /* Funktionsaufruf */
/* feld [z][s] enthaelt nun wert */
printf ( " \n\n geaendert: feld [z=%2hd] [s=%2hd] = %hd",
        z, s, wert ) ;
for ( j=0; j<DIM1; j++ ) /* Ausgabe aller DIM1 Zeilen */
{
    printf ("\n") ;
    for ( i=0; i<DIM2; i++ ) printf(" %4hd", feld [j] [i]);
    /* Ausgabe der DIM2 Elemente einer Zeile */
}
} while (ok == 3) ;
return 0 ;
}

void f_m ( short donald_duck [ ] [DIM2], short goofy, short daisy,
          short mickey )
{ donald_duck [goofy][daisy] = mickey ;
  /* Feldplatz donald_duck [goofy] [daisy] wird
    mit mickey ueberschrieben */
}

```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 45

Beispiel: 2D-Felder als Funktionsparameter (5)

- Bildschirmdarstellung des Programms:

feld [3] [4]:

10	20	30	40
100	200	300	400
1000	2000	3000	4000

An welcher Stelle [z] [s] soll feld mit 'wert'
ueberschrieben werden?

z s wert ?

2 3 0

geaendert: feld [z= 2] [s= 3] = 0

10	20	30	40
100	200	300	400
1000	2000	3000	0

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 46

 Stringfelder (1)

Schreiben Sie ein möglichst allgemein verwendbares C-Programm, das ohne Gebrauch von String-Bibliotheksfunktionen den String (*string2*)

" im Semester E1A"

orthografisch richtig am Ende der Zeichenkette (*string1*)

"Ich studiere an der Fachhochschule Hamburg
Elektrotechnik."

einfügt und nach *string* kopiert!

string1 und *string2* bleiben unverändert!

Erstellen sie zuvor eine sinnvolle Programmstruktur mit
geeigneten Funktionen und einen
Programmablaufplan!

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 47

 Stringfelder (1)

- Im folgenden Programm sucht die Funktion *copy_char0* den abschließenden Punkt (.) in *string1* und kopiert den String ausschließlich des Punktes nach *string*. Ferner gibt es den zugehörigen Feldindex (*index*) des Punktes an das Hauptprogramm zurück.
- Die Funktion *copy_char1* fügt den einzuschiebenden Text aus *string2* an der richtigen Stelle in *string* ein und kopiert anschließend den Rest des *string1* nach *string*. Ferner gibt es den Index (*end_index*) des letzten kopierten Zeichens in *string* an das Hauptprogramm zurück.

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 48

✓ Stringfelder (2)

- Definitionen und Funktionsprototypen:

```
#include      <stdio.h>
# define      DOT      '.'      /* Suchzeichen definieren */
# define      MAX_L     100     /* max. Stringlänge definieren */

unsigned short copy_char0 ( char [ ], char [ ], char ) ;

/* copy_char kopiert str1[ ] bis zum Suchzeichen such_zeich nach
str[ ] und gibt index von such_zeich in str1[ ] zurueck */

unsigned short copy_char1 ( char [ ], char [ ], char [ ], unsigned
short) ;

/* copy_char1 erwartet den Index (index), ab dem str2[ ] in str[ ]
eingefuegt werden soll. Danach wird der Rest von str1[ ] nach dem
Suchzeichen hinter str2[ ] kopiert. Der Index des letzten in str[ ]
kopierten Zeichens wird zurueckgegeben. */
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 49

✓ Stringfelder (3)

- Hauptprogramm:

```
int main(void)
{ char string [MAX_L] ;
  char string1 [ ] =
    "Ich studiere an der HAW Hamburg Elektrotechnik." ;
  char string2 [ ] = " im Semester E1A" ;
  unsigned short max_st = MAX_L ;
  char such_zeichen = DOT ;
  unsigned short end_index, j = 0, such_zeichen_nr = 0 ;
  while (j < max_st) string [j++] = 0x0 ;
      /* Sicherheit: string mit 0x0 füllen */
  such_zeichen_nr = copy_char0 ( string1, string, such_zeichen ) ;
  printf( " such_zeichen_nr=%u\n string1=%s\n"
    " string=%s\n\n", such_zeichen_nr, string1, string ) ;
  end_index = copy_char1 (string1, string2, string, such_zeichen_nr) ;
  printf( " end_index=%u\n string=%s", end_index, string ) ;
  return 0 ; }
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 50

✓ Stringfelder (4)

- Funktionen (1):

```
unsigned short copy_char0 ( char str1 [ ], char str [ ],
                          char such_zeichen )
{
    /* copy_char kopiert str1[ ] bis zum Suchzeichen such_zeich nach
    str[ ] und gibt index von such_zeich in str1[ ] zurueck */
    unsigned short k=0 ;      /* k vereinbaren und initialisieren */
    while ( str1 [k] && str1 [k] != such_zeichen ) str [k] = str1 [k++] ;
    return k ; /* Position des Suchzeichens im String zurueckgeben */
}
```

02.12.2004



✓ Stringfelder (5)

- Funktionen (2):

```
unsigned short copy_char1 ( char str1[ ], char str2[ ], char str[ ],
                          unsigned short index )
{
    /* copy_char1 erwartet den Index (index), ab dem str2[ ] in str[ ]
    eingefuegt werden soll. Danach wird der Rest von str1[ ] nach
    dem Suchzeichen hinter str2[ ] kopiert und der Index des letzten
    in str[ ] kopierten Zeichens wird zurueckgegeben. */
    unsigned short i=0, j = index ;
    while ( str2 [ i ] ) str [index++] = str2 [i++] ; /* str 2 einfuegen */
    while ( str1 [ j ] ) str [index++] = str1 [j++] ;
    /* Rest von str1 anfüegen */
    return index-1 ; /* Index des letzten kopierten Zeichens in str
    zurueckgeben */
}
```

02.12.2004



✓ Stringfelder (6): Struktogramm (1)

`string[100], index_suchzeichen` deklarieren

`string1[ ]`="Ich studiere ... Elektrotechnik." deklarieren

`string2[ ]`=" im Semester E1b" deklarieren

Suchen von '.' in `string1` und kopieren nach `string` bis (ohne) Suchzeichen '.'

`Index_Suchzeichen = copy_char0(String1, String, '.')`

`printf()` `string1, string, index_suchzeichen`

Anhängen von `string2` an `string` ab auchzeichen '.', danach Anhängen des Restes von `string1` an `string`.

Gesamtfeldlänge = `copy_char1(string1, string2, string, index_suchzeichen)`

`printf()` `string, stringlaenge Gesamtstring`

← Zum Betriebssystem

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 53

✓ Stringfelder (7): Struktogramm (2)

`index_Suchzeichen = copy_char0(str1[ ], str [ ], char z)`

`k=0`

`while (!End_of_string1 && ( str1[k]!= z ) )`

`str[k] = str1[k++] ; /* bis Suchzeichen z von */
/*str1[] nach Str[] kopieren */`

← Ruecksprung aus `copy_char0 ( ), return k`

`k` ist der Index für das Suchzeichen '.' in `Str1`

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 54

Aufgabe: Transponierte einer Matrix

Man transponiert eine Matrix, indem man die Zeilen und die Spalten vertauscht:

Beispiel:

```
short m [2] [3] = { {1, 2}, {11, 22, 33} }, m_t [3] [2];
```

```
m [0] [0] = 1    m [0] [1] = 2    m [0] [2] = 0
```

```
m [1] [0] = 11   m [1] [1] = 22   m [1] [2] = 33
```

Das Feld von **2 Zeilen** und **3 Spalten** wird  **transponiert:**

```
m_t [0] [0] = 1    m_t [0] [1] = 11
```

```
m_t [1] [0] = 2    m_t [1] [1] = 22
```

```
m_t [2] [0] = 0    m_t [2] [1] = 33
```

Schreiben Sie ein Hauptprogramm mit einer Funktion **transp** zum Transponieren einer **int**-Matrix!

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 55

02.12.2004

Aufgabe: Transponierte einer Matrix (1)

Programm (1):

```
#include <stdio.h>          /* Bibliothek fuer scanf(..), printf(..) */
#define MAX 20              /* Vereinbarung fuer max */
#define DIM1 2
#define DIM2 3              /* Felddimensionen definieren */

void transp (int m [ ] [MAX], int m_t [ ] [MAX], int z, int s);
/* Prototyp: transp mit 2 Feldern m: Originalmatrix (Quelle)
   m_t: transponierte Matrix (Ziel)
   z: Anzahl der Zeilen von m
   s: Anzahl der Spalten von m */

int main (void)             /* Hauptprogramm */
{
    int m [MAX] [MAX] = { {1, 2}, {11, 22, 33} }, m_t [MAX] [MAX];
    int s=DIM1, z=DIM2, i, j; /* m, m_t & s, z, i, j vereinbaren */
}
```

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 56

02.12.2004

✓ Aufgabe: Transponierte einer Matrix (2)

Programm (2):

```

printf ("Matrix:")          /* Matrix m ausgeben */
for (j=0; j<DIM1; j++)     /* Schleife über alle Zeilen */
{
    printf ("\n") ;
    for (i=0; i<DIM2; i++)  printf(" %4hd", m [j] [i]);
                           /* Ausgabe der Elemente 1 Zeile */
}
transp (m, m_t, z, s) ;    /* Funktion transp ( ) aufrufen */
printf (" \n\n transponierte Matrix:") ;
for (j=0; j<DIM2; j++)     /* Schleife über alle Zeilen */
{
    printf ("\n") ;
    for (i=0; i<DIM1; i++)  printf(" %4hd", m_t [j] [i]) ;
                           /* Ausgabe der Elemente 1 Zeile */
}
return 0 ;                 /* Ruecksprung ins Betriebssystem */ }

```

02.12.2004



✓ Aufgabe: Transponierte einer Matrix (3)

Programm (3):

```

void transp (int m [ ] [MAX], int m_t [ ] [MAX], int z, int s)
/* transp mit 2 Feldern      m: Originalmatrix (Quelle)
                           m_t: transponierte Matrix (Ziel)
                           z: Anzahl der Zeilen von m
                           s: Anzahl der Spalten von m
                           vereinbaren */

{
    int i, j;
    for ( i=0; i<z; i++ )
    {
        for ( j=0; j<s;j++ ) m_t [ i ] [ j ] = m [ j ] [ i ] ; /* m_t berechnen */
    }
}

```

02.12.2004





komplexeres Programm (1)

Elektronische Würfel für das Spiel *Monopoly* (1)

Sie sind die Suche nach heruntergefallenen Würfeln und das Kopfrechnen beim Spiel ‚*Monopoly*‘ leid.

Außerdem sind Sie es leid, sich mit Mitspielern zu streiten, wer als nächster an der Reihe ist.

Deshalb beschließen Sie, ein C-Programm zu schreiben, welches das Würfeln und die Würfelregeln des Spiels nachbildet.

Würfelregeln für *Monopoly* (1):

1. Als Erstes ist einzugeben, mit **wie vielen Spielern** (2-6) gespielt wird.
Ferner sind die Namen der Spieler in der Reihenfolge einzugeben, in der Sie am Zug sind (maximal 10 *ASCII*-Zeichen).
2. Das Programm ermittelt mit Hilfe einer geeigneten Zufallszahl, welcher Spieler **anfangen** darf.
Danach wird die vorgegebene Reihenfolge der Spieler fortgesetzt.

02.12.2004

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



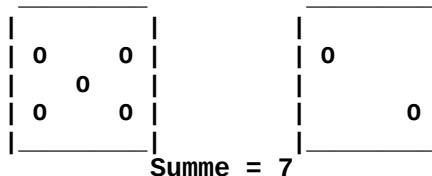
Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 59



komplexeres Programm (2)

Würfelregeln für *Monopoly* (2):

3. Es wird mit 2 normalen 6-seitigen Würfeln gewürfelt.
Die 2 Würfel sollen mit geeigneten *ASCII*-Zeichen grafikähnlich dargestellt werden:



Die beiden Würfel müssen **nicht nebeneinander** abgebildet werden. Sie dürfen auch untereinander stehen!

3. Die Gesamtsumme der Punkte beider Würfel eines Wurfes soll ausgegeben werden.
4. Jeder Wurf wird mit einiger beliebigen Taste quittiert (Ausnahme: 3 mal *Pasch* gewürfelt!).
5. Es wird reihum gewürfelt. Es soll angezeigt werden, welcher Spieler als nächster am Zug ist.

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 60

komplexeres Programm (3)

Würfelregeln für Monopoly (3):

6. Würfelt ein Spieler ein **Pasch** (= beide Würfel zeigen dieselbe Zahl), so darf er noch einmal würfeln.
Die Anzahl der aufeinander folgenden **Pasch**-Würfe eines Spielers ist anzuzeigen!
7. Hat ein Spieler 3 mal hintereinander **Pasch** gewürfelt, so muss er für 3 Runden in das **Gefängnis** (= 3 Runden aussetzen!).
Dies ist anzuzeigen und bei der Anzeige der Reihenfolge der Spieler zu berücksichtigen.
Der Spieler kann sich aber aus dem Gefängnis **freikaufen**.
Deshalb fragt das Programm, ob sich der Delinquent freigekauft hat und berücksichtigt dies bei der Reihenfolge der Spieler.
8. Mit dem ASCII-Zeichen **#** wird das Spiel **beendet**.

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 61

02.12.2004

✓ komplexeres Programm (1)

Header & main (1)

```
/* HAW VGL 27.11.2003 V 2.12.2004 */
/* Projektname: Monopoly.dsw */
#include <time.h>      /* Bibliotheken einbinden */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>    /* fuer strcmp (..) */

short wurf ( void ) ; /* Prototypenvereinbarung: wurf */

int main (void)      /* Hauptprogramm MSVC-Version */
{
    unsigned char Anzahl_Spieler ;
    char Spieler [6] [12] = { 0 } ; /* Sicherheitsmassnahme */
    /* Spieler [...] [10] = 0x0 (Stringende fuer %s), Spieler [...] [11] =
    Anzahl der Runden im Gefaengnis */
    char i, j, dran, pasch = 0, in = '\0';
    short ok, strok = 1, w1, w2;
    long ticks ;      /* Variable fuer Zufallsgenerator */
```

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 62

02.12.2004

✓ komplexeres Programm (2)

main (2)

```

srand (time(&ticks) % 37) ;
fflush ( stdin ) ;
system ("cls") ; /* Bildschirm loeschen */
printf (" Das Spiel wuerfelt nach Monopoly-Regeln.\n\n"
" Es gibt jeweils den naechsten Spieler an und wuerfelt.\n\n"
" Jeder Wurf wird mit einer beliebigen Taste quittiert.\n\n"
" Die Tasten #, f bzw. F haben Sonderfunktionen:\n\n"
" Nach einem Pasch(bei beide Wuerfel haben dieselbe Punktzahl)\n"
" darf derselbe Spieler noch einmal wuerfeln.\n\n"
" Wuerfelt derselbe Spieler 3 mal hintereinander Pasch,\n"
" so muss er im Gefaengnis 3 Runden aussetzen.\n\n"
" Das Programm ueberwacht die Dauer des Aussetzens.\n\n"
" Kauft sich der Spieler aus dem Gefaengnis frei, so wird das\n"
" bei einer Quittierung mit der Taste f bzw. F beruecksichtigt.\n\n"
" # beendet das Spiel.\n\n\n") ;

```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 63

✓ komplexeres Programm (3)

main (3)

```

do
{
printf ( " Wie viele Spieler nehmen am Spiel teil? "
" 1 < Anzahl_Spieler < 6: " ) ;
fflush ( stdin ) ;
ok = scanf ( "%u",&Anzahl_Spieler ) ;
} while ( !ok || Anzahl_Spieler < 2 || Anzahl_Spieler > 6 ) ;
printf ( "\n Geben Sie die Spieler in der richtigen Reihenfolge"
" der Zuege an!" ) ;
for ( i = 0; i <= Anzahl_Spieler-1; i++ )
{
printf( "\n Wie heisst der %1u. Spieler? (maximal 10 Zeichen) \n",
i+1 ) ;
fflush ( stdin ) ;
scanf ( "%10s", &Spieler [ i ] ) ;
}

```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 64

✓ komplexeres Programm (4)

main (4)

```

j = i ;
while ( j > 0 ) /* strcmp (..) in string.h: gleiche Spielernamen ? */
{
    strok = strcmp ( Spieler [ --j ], Spieler [ i ] ) ;
    if ( strok == 0 )
    {
        printf ( "\n Gleiche Spielernamen sind nicht erlaubt!\n" ) ;
        i-- ;
        break ;
    }
}
if ( strok == 0 )      continue ;
printf ( "\n Der %1u. Spieler heisst: %s\n", i+1, Spieler [ i ] ) ;
}

```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 65

✓ komplexeres Programm (5)

main (5)

```

dran = rand ( ) % Anzahl_Spieler ;
printf ( "\n %s beginnt!\n\n"
        " o.k.? (beliebige Taste / !=Programmende):", Spieler [dran] ) ;
fflush ( stdin ) ;
scanf ( "%c", &in ) ;
while ( in != '#' )
{
    if ( !Spieler [dran] [11] ) /* enthaelt Anzahl der Absitzrunden */
    {
        for ( pasch =0; pasch < 3 && (in != '#'); pasch++ )
        {
            /* Pasch-Auswertung */
            w1 = wurf ( ) ;
            w2 = wurf ( ) ;
            /* w2 = w1; /* +++ Nur zu Testzwecken +++ */
            printf ( "\nDie Summe der Punkte betraegt: %u\n", w1+w2+2 ) ;

```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 66

✓ komplexeres Programm (6)

main (6)

```

if ( w1 != w2 ) break ;
if ( pasch == 2 )
{
    printf ( "3. Pasch!: %s muss fuer 3 Runden ins Gefaengnis\n "
            " Freikaufen ? (f,F,j,J / andere Taste):", Spieler [dran] ) ;
    fflush ( stdin ) ;
    scanf ( "%c", &in ) ;
    if ( in != 'f' && in != 'F' && in != 'j' && in != 'J' )
        Spieler [dran] [11] = pasch+1 ; /* Gefaengnis merken */
    break ;
}

printf ( "%1u. Pasch!: %s ist noch einmal am Zug!\n\n"
        "o.k.? (beliebige Taste / #=Programmende):", pasch+1,
        Spieler [dran] ) ;
fflush ( stdin ) ;
scanf ( "%c", &in ) ;
if ( in == '#' ) break;
}

```

02.12.2004



✓ komplexeres Programm (7)

main (7)

```

if ( in == '#, ) break ;
printf( "\n naechster Zug? (beliebige Taste / #=Programmende):" ) ;
fflush ( stdin ) ;
scanf ( "%c",&in ) ;
if (in == '#') break ;
fflush ( stdin ) ;
system ( "cls" ) ; /* Bildschirm loeschen */
dran = ++dran % Anzahl_Spieler; /* naechster Spieler */
if (Spieler [dran] [11] ) /* 1 Runde weniger absitzen */
{
    Spieler [dran] [11]--;
    if ( Spieler [dran] [11] )
        printf ( "%s sitzt noch fuer %1u Runde(n) im Gefaengnis\n",
                Spieler [dran], Spieler [dran][11] ) ;
    else
        printf ( "%s kommt aus dem im Gefaengnis frei.\n",
                Spieler [dran] ) ;
}

```

02.12.2004



✓ komplexeres Programm (8)

main (8)

```

    else
        printf ( "\n%s ist am Zug!\n",Spieler [ dran ] ) ;
    }
    printf ( "\n\n\n          PROGRAMMENDE!\n\n" ) ;
    return 0;
}

```

PAGE 6

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 69

✓ komplexeres Programm (9)

wurf (1)

short wurf (void)

```

{
char play, wuerfel [6] [6] [12] =    /* Darstellung des Wuerfels */
{ /* 1 Punkt*/
{"      \n"}, /* Statt der Hexadezimaldarstellung */
{"|      |\n"}, /* kann auch die aequivalente ASCII- */
{"|      |\n"}, /* Darstellung verwendet werden, Beispiel: */
{"|      O |\n"}, /* '|', '|', '|', '|', '|', 'O', '|', .....*/
{"|      |\n"},
{"|      |\0"}},
/* 2 Punkte*/
{"      \n"},
{"|      |\n"},
{"|      O |\n"},
{"|      |\n"},
{"|      O |\n"},
{"|      |\0"}},

```

1212300

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



Programmieren in ANSI C:
K8 C-Felder.ppt, Seite 70

✓ komplexeres Programm (10)

wurf (2)

/* 3 Punkte*/

```
{{"  \n"},
 {"| \n"},
 {"|  O  \n"},
 {"|    O \n"},
 {"|      O \n"},
 {"|____\0"}}
```

/* 4 Punkte*/

```
{{"  \n"},
 {"| \n"},
 {"|  O  O \n"},
 {"|    O \n"},
 {"|      O \n"},
 {"|____\0"}}
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 71

✓ komplexeres Programm (11)

wurf (3)

/* 5 Punkte*/

```
{{"  \n"},
 {"| \n"},
 {"|  O  O \n"},
 {"|    O \n"},
 {"|      O \n"},
 {"|____\0"}}
```

/* String-Darstellung des Würfels */

/* 6 Punkte*/

/* Hexadezimal-Darstellung des Würfels */

/* schlechtes Beispiel : */

```
{0x20,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x20,'\n'},
{0x7c,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x7c,'\n'},
{0x7c,0x20,0x4f,0x20,0x20,0x4f,0x20,0x20,0x4f,0x20,0x7c,'\n'},
{0x7c,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x7c,'\n'},
{0x7c,0x20,0x4f,0x20,0x20,0x4f,0x20,0x20,0x4f,0x20,0x7c,'\n'},
{0x7c,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x5f,0x7c,'\0'}} ;
```

02.12.2004

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



Programmieren in ANSI C:
K8_C-Felder.ppt, Seite 72

✓ komplexeres Programm (12)

wurf (4)

```
play = rand ( ) %6 ;  
    /* Zahlbereich von 0 ... 5: 3D-Stringfeld fuer Wuerfel [0] ... [6] */  
printf ( "%s\n", wuerfel [play] ) ;  
return play ;  
}
```

02.12.2004

