

Programmieren in C: Zeiger (*pointer*)

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 1

Zeiger in C

- In diesem Kapitel wollen wir zeigen:
 - ➡ was Adressen sind
 - ➡ was Zeiger (*pointer*) sind
 - ➡ wie man Zeiger vereinbart
 - ➡ welche Zeigeroperatoren es gibt
 - ➡ wozu man Zeiger verwendet
 - ➡ welcher Zusammenhang zwischen Feldern und Zeigern besteht.

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 2

Datenspeicher und Adressen (1)

- C verwaltet den **Datenspeicher selbständig** und für den Anwender ➡ **transparent** (unsichtbar).
- Der Datenspeicher besteht aus ➡ **Speicherzellen**, die bei vielen Computern eine Information von 1 Byte enthalten.
- Jede Speicherzelle besitzt eine **eindeutige Nummer**, mit welcher der Computer sie anspricht.
Diese Nummer heißt:
➡ **Adresse der Speicherzelle.**

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 3

Datenspeicher und Adressen (2)

- Jeder Computer spricht seine Speicherzellen mit Hilfe von eindeutigen Nummern an,
den ➡ **Adressen der Speicherzellen.**

Beispiel: `char string [ ] = "Ich bin ein string!" ;`

- C verwaltet den Datenspeicher selbst:

Wir nehmen an, dass das **byte-breite char-Feld** `string` auf der ➡ **Adresse 0x100** beginnt.

Speicherbelegung:

↓ ↓ Speicherzelle: Byte mit der Adresse 0x100

HEX-Adresse	100	101	102	103	104	105	106	107	108	109	10a	10b	10c	10d	10e	10f	110	111	112	113
Inhalt	'I'	'c'	'h'	' '	'b'	'i'	'n'	' '	'e'	'i'	'n'	' '	's'	't'	'r'	'i'	'n'	'g'	'!'	'\0'

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 4

Adressen in der Assemblersprache

- Jeder Prozessor benutzt eine **eigene Maschinensprache**, die **➡ Assemblersprache**.
 - In der Assemblersprache programmiert man **elementare Befehle**, die **die Adressen** des Speichers **direkt ansprechen**.
 - C ist eine **hardwarenahe Hochsprache**, die es dem C-Compiler erleichtert, den C-Code in die Assemblersprache umzusetzen.

Deshalb ist die Verwendung von Zeigern in C sehr effizient:

Der Compiler kann damit einen sehr kurzen und schnell ablaufenden Assemblercode erzeugen.

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 5

27.05.2004

Beispiel: Motorola 68000 Assemblerbefehl

- Jeder Prozessor besitzt eine eigene Maschinensprache, die **➡ Assemblersprache**.

Beispiel Motorola 68000-Befehl: **move.b** **\$2000,\$4000**

Kopiere das Byte aus der Speicherzelle mit der Hex-Adresse 2000 in die Speicherzelle mit der Hex-Adresse 4000

- Das **➡ Assemblerprogramm** übersetzt diesen Befehl in den **➡ Objectcode** (Binärcode) des 68000-Prozessors: **0x11f8** **2000** **4000**

move.b **\$2000,\$4000**

- Nur in der Binärdarstellung kann ein Prozessor einen Befehl 'verstehen'.

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 6

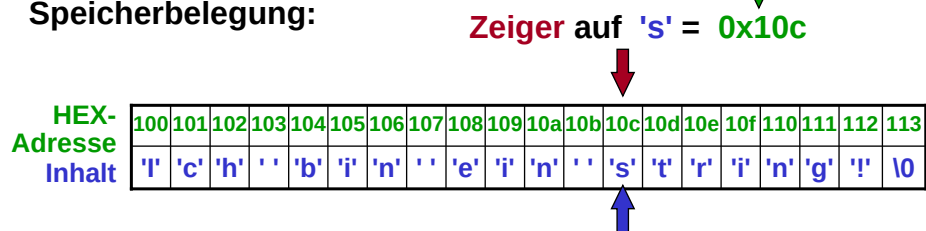
27.05.2004

Was sind Zeiger (*pointer*) ?

Beispiel: `char string [ ] = "Ich bin ein string!";`
 /* Annahme: gespeichert ab Adresse 0x100 */

- In **→ Zeigern** werden **→ Adressen** gespeichert.
- Ein **Zeiger** zeigt auf den **→ Datenwert**, der auf der im Zeiger abgelegten Adresse gespeichert ist.

Speicherbelegung:



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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 7

Wie vereinbart man Zeiger (*pointer*)?

- Einen **Zeiger** auf einen Datenwert vereinbart man durch Voranstellen des **Operators** **→ *** auf den **Namen des Zeigers**.

☛ Der ***** gehört nicht zum Namen des Zeigers.



Beispiel:

`char *ptr;`

- **ptr** ist ein **→ Zeiger** auf einen Datenwert des Typs **→ char**.

☛ Nach der Definition zeigt der Zeiger **ptr** **zunächst** auf **'nichts'**: er enthält eine **zufällige Adresse**!

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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 8

Wie weist man einem Zeiger eine Adresse zu ?

- Der vorangestellte **Adressoperator**
 **&** erzeugt die **Adresse einer Variablen**.

Beispiel:

```
char *p_c ; /* Vereinbarung: Zeiger auf eine char-Variable*/
char c = 'C' ; /* Vereinbarung: Variable c = 'C' */
p_c = &c ; /* Adresszuweisung: Zeiger p_c zeigt auf c */
```



- Die Anzahl der Bits eines Zeigers hängt nicht mit der Länge der Variablen zusammen, auf die der Zeiger zeigt:

Je nach Compilereinstellung sind Zeiger (auch auf **char**) beispielsweise 16 oder 32 bit breit!

27.05.2004

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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 9

Wie erhält man den Datenwert, auf den ein Zeiger zeigt?

- Der vorangestellte **Verweisoperator** (Inhaltsoperator)
 ***** gibt den **Datenwert zurück**, auf den der Zeiger zeigt.
- Er liefert das  **Objekt** des Zeigers

Beispiel:

```
char *p_c ; /* Zeiger auf eine char-Variable*/
char c = 'C', c1 ; /* char-Variable c, c1: c = 'C' */
p_c = &c ; /* Zeiger p_c zeigt auf c */
c1 = *p_c ; /* c1 = 'C': c1 ist das Objekt von *p_c */
```



27.05.2004

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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 10

Die Funktion des Operators *

- Die mehrfache Bedeutung des Zeichens $\Rightarrow$ * in C ist für Anfänger verwirrend:

1. Multiplikationsoperator:

c * = **a** ; /* weise **c** das Produkt aus **c** und **a** zu */


2. Deklarationsoperator für Zeiger:

int ***pt** ; /* **pt** ist ein Zeiger auf eine **int**-Variable */


3. Verweisoperator:

long ***i_p** ; /* **hier** ist **i_p** ein **Zeiger** auf eine **long**-Variable */
x = ***i_p** ; /* **hier**: weise **x** das **Objekt** des Zeigers **i_p** zu */



4. Teil eines Kommentaroperator: **/* Kommentar */**

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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 11

27.05.2004

Ausgabe von Zeigeradressen

- Man kann die Adressen von Zeigern mit Hilfe der $\Rightarrow$ %p -Formatanweisung ausgeben.

Beispiel:

```
#include <stdio.h>
int main (void)                                /* Hauptprogramm */
{
  double *pointer ;
  double d1 = 13579.02468, d2 ;                /* Variablen vereinbaren */
  pointer = &d1 ;
  d2 = *pointer ;
  printf("d1 = %f, d2 = %f \n Pointeradresse = %p", d1, d2, pointer) ;
  return 0 ;                                   /* Rücksprung ins Betriebssystem */
}
```

- Ergebnis:

d1 = 13579.02468, d2 = 13579.02468

Pointeradresse = 1BB2 (maschinen- & programmabhängig!)

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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 12

27.05.2004

Zeigeroperatoren

Es macht keinen Sinn, zu Lasten der Verständlichkeit eines Programms 'mit Gewalt' Zeiger zu verwenden!

Ein 'Pointerkomiker' hat das folgende Programm geschrieben:

```
#include <stdio.h>
int main (void)
{
    char *zeiger ;
    char Kuh = 'Q', q1;
    zeiger = &Kuh ;
    q1 = *zeiger ;
    printf ("Kuh = %c, q1 = %c", Kuh, q1) ;
    return 0 ;
}
```

1. Kommentieren Sie jede Zeile des Programms!
2. Welche Werte werden ausgegeben?
3. Vereinfachen Sie das Programm!

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 13

27.05.2004

✓ Zeigeroperatoren (1)

1. Kommentieren

```
#include <stdio.h> /* Bibliothek für printf (..) einbinden */
int main (void)    /* Hauptprogramm */
{
    char *zeiger1;  /* Zeiger zeiger1 auf char vereinbaren */
    char Kuh = 'Q', q1; /* char-Variable Kuh = 'Q' und q1 vereinbaren */
    zeiger1 = &Kuh ; /* zeiger1 zeigt auf Kuh */
    q1 = *zeiger1 ;  /* In q1 wird das Objekt von zeiger1 gespeichert */
    printf ("Kuh = %c, q1 = %c", Kuh, q1) ;
                                /* Ausgabe:    Kuh = Q, q1 = Q */
    return 0 ;                /* Rücksprung ins Betriebssystem */
}
```

2. Ausgabe

Kuh = Q, q1 = Q

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 14

27.05.2004

✓ Zeigeroperatoren (2)

3. Programmvereinfachung

```
#include <stdio.h> /* Bibliothek für printf (..) einbinden */
int main (void)    /* Hauptprogramm */
{
    char Kuh = 'Q', q1; /* char-Variable Kuh = 'Q' und q1 vereinbaren */
    q1 = Kuh ;          /* q1 wird der Inhalt von Kuh zugewiesen */
    printf ("Kuh = %c, q1 = %c", Kuh, q1) ;
                        /* Ausgabe:    Kuh = Q, q1 = Q */
    return 0 ;         /* Rücksprung ins Betriebssystem */
}
```

Die Zeigerarithmetik in der Aufgabenstellung ist überflüssig!
Sie erschwert die Lesbarkeit des Programms!



Erlaubte Zeigeroperationen in C

- Ein Zeiger enthält eine **ganze Zahl**, die
➡ **Adresse der Speicherzelle**, ab der die
Variable im Speicher abgelegt ist.
- **Zeiger** und **ganze Zahlen** sind in C **nicht äquivalent**.
- Für Zeiger sind **nur** die folgenden Operationen **erlaubt**:
 - ➡ Zuweisungen von Zeigern des gleichen Typs
 - ➡ Addition und Subtraktion von ganzen Zahlen zu Zeigern
 - ➡ Subtraktion oder Vergleich von 2 Zeigern auf Elemente desselben Feldes.
 - ➡ Zuweisung oder Vergleich eines Zeigers mit Null.



Einige verbotene Zeigeroperationen in C

- **Zeiger** und **ganze Zahlen** sind in C nicht äquivalent.
- Für Zeiger sind beispielsweise folgende Operationen verboten:
 - ☛ Subtraktion von Zeigern auf **unterschiedliche Variable**
 - ☛ Multiplikation oder Division von Zeigern
 - ☛ Anwendung logischer Operationen auf Zeiger
 - ☛ Anwendung von Schiebeoperationen auf Zeiger.
 - ☛ Zuweisen eines Zeigers auf einen Datentyp an einen Zeiger auf einen anderen Datentyp **ohne explizite Typumwandlung** (Ausnahme: Zeiger auf **void**.)

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 17

Beispiele verbotener Zeigeroperationen in C

- Für Zeiger sind beispielsweise folgende Operationen verboten:

Beispiel:

```
char *p0, *pa, *p ;           /* Zeiger auf 3 char-Variable*/
char c = 'C', c1[ ] = "so?" ; /* char-Variable: c = 'C', c1[0 .. 3] = 'so?\\0' */
p0 = &c ;                     /* Zeiger p0 zeigt auf c */
pa = &c1[1] ;                  /* Zeiger pa zeigt auf c1 [1] */
```

- ☛ Addition oder Subtraktion von Zeigern auf **unterschiedliche Variable**:
 ~~$p = p0 + pa;$~~
- ☛ Multiplikation oder Division von Zeigern: ~~$p = p0 * pa;$~~
- ☛ Schiebeoperationen mit Zeigern: ~~$pa = p0 >> 2;$~~
- ☛ logische Operationen mit Zeigern: ~~$p0 \&\& = 0xa;$~~

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 18

Zeiger und Typkonversion (1)

- Bei vielen 'normalen' Datentypen nimmt der C-Compiler eine ➡ **automatische Typkonversion** vor, falls er die Prototypenvereinbarung einer Funktion vorfindet.

Beispiel:

```
#include <stdio.h>
#include <math.h> /* Prototypenvereinbarung in math.h:
...
double _Cdecl _FARFUNC sqrt (double __x);
...
*/
int main (void) /* Hauptprogramm */
{ float Ergebnis ;
  int x = 2 ; /* Variablen vereinbaren */
  Ergebnis = sqrt (x); /* in math.h sind alle Parameter vom Typ double */
  printf ("integer x = %d, Wurzel (x) = %f", x, Ergebnis) ;
  return 0 ; /* Rücksprung ins Betriebssystem */ }
```

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 19

Zeiger und Typkonversion (2)

Beispiel:

```
#include <stdio.h>
#include <math.h> /* Prototypenvereinbarung in math.h:
...
double _Cdecl _FARFUNC sqrt (double __x);
...
*/
int main (void) /* Hauptprogramm */
{ float Ergebnis ;
  int x = 2 ; /* Variablen vereinbaren */
  Ergebnis = sqrt (x); /* in math.h sind alle Parameter vom Typ double */
  printf ("integer x = %d, Wurzel (x) = %f", x, Ergebnis) ;
  return 0; /* Rücksprung ins Betriebssystem */ }
```

- Die Funktion **sqrt** erwartet einen **double-Parameter** und gibt einen **double-Parameter** zurück.
- Der C-Compiler erkennt an der **Prototypenvereinbarung** in **math.h** die Diskrepanz der Typen und wandelt **x** und **Ergebnis** entsprechend.

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 20

Zeiger und Typkonversion (3)

☛ Verwendet man Zeiger, so funktioniert die automatische Typkonversion nicht **in jedem Fall**:

Die **direkte Zuweisung eines Zeigers auf einen Datentyp an eine Variable anderen Datentyps** ist verboten.

Beispiel:

```
#include <stdio.h>
int main (void)                /* Hauptprogramm */
{
    short *pointer ;
    double d1 = 13579.02468, d2 ; /* Variablen vereinbaren */
    pointer = &d1 ;              /* fehlerhafte Zuweisung ! */
    d2 = *pointer ;
    printf ("d1 = %f, d2 = %f", d1, d2) ;
    return 0 ;                  /* Rücksprung ins Betriebssystem */
}
```

- Warnung des MSVC-Compilers: " **warning C4133: '=' : Inkompatible Typen – von 'double *' zu 'short *'** "
- Ergebnis: **d1 = 13579.024680, d2 = -10129.000000**

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 21

Zeiger und Typkonversion (4)

Beispiel:

```
#include <stdio.h>
int main (void)                /* Hauptprogramm */
{
    short *pointer;
    double d1 = 13579.02468, d2 ; /* Variablen vereinbaren */
    pointer = &d1;              /* fehlerhafte Zuweisung ! */
    d2 = *pointer ;
    printf ("d1 = %f, d2 = %f", d1, d2) ;
    return 0 ;                  /* Rücksprung ins Betriebssystem */
}
```

- Man hätte hoffen können, dass der **ganzzahlige Teil** von **d1** richtig an **d2** übergeben wird.
- Dies ist jedoch nicht der Fall: Es werden **ohne Rücksicht auf die Bedeutung die untersten 2 Bytes** der 8 Bytes der double Variablen **d1** angesprochen und kopiert.
Das Ergebnis ist falsch!

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 22



Aufgabe: Zeigeroperationen

1. Mit den folgenden Zeigeroperationen wollte der Programmierer erreichen, dass der ganzzahlige Teil der **long double** Variablen **d** ausgegeben wird:

```
...
signed char c ;
long double *p_d, d = -12.02468 ; /* Variablen vereinbaren */
p_d = &d ;
c = *p_d ;
...
```

- 1.1 Ist dieser Programmausschnitt korrekt? – Begründung !
 1.2 Vermeiden Sie unnötige Zeigeroperationen und geben Sie einen möglichst einfachen Programmausschnitt zur Speicherung des ganzzahligen Anteils von **d** in der char-Variablen **c** an!

27.05.2004

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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 23



Aufgabe: Zeigeroperationen (1)

Lösungen (1):

```
...
signed char c ;
long double *p_d, d = -12.02468 ; /* Variablen vereinbaren */
p_d = &d ;
c = *p_d ;
printf ("long double d = %Lf, c = %d", d, c) ;
...
```

- 1.1 Dieser Programmausschnitt ist im Rahmen der Typumwandlungsregeln richtig:

Der Compiler identifiziert den Ausdruck ***p_d** als **long double Variable**. Diese wird einer (**signed**) **char-Variablen** zugewiesen. Folglich ist das Ergebnis korrekt, wenn der ganzzahlige Teil im darstellbaren Bereich von **signed char**-Variablen liegt: **-128 ... + 127**. Andernfalls wird das Ergebnis entsprechend den Umwandlungsregeln für die Variablen verfälscht.

Das Ergebnis ist **hier** korrekt:

long double d = -12.02468, c = -12

27.05.2004

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



Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 24

✓ Aufgabe: Zeigeroperationen (2)

Lösungen (2):

- 1.2 Vermeiden Sie unnötige Zeigeroperationen und geben Sie einen möglichst einfachen Programmausschnitt zur Speicherung des ganzzahligen Anteils von **d** in der char-Variablen **c** an:

```
...
signed char c ;
long double d = -12.02468 ;    /* Variablen vereinbaren */
c = (signed char) d ;
printf ("long double d = %Lf, c = %d", d, c) ;
...
```

Der **cast-Operator** (**signed char**) kann entfallen, wenn dem Compiler die **Typvereinbarungen** von **c** und **d** bekannt sind (weil beispielsweise Prototypenvereinbarungen getroffen wurden). Dies ist wichtig bei Funktionsaufrufen mit den Parametern bzw. Rückgabewerten von **c** bzw. **d**.

Man beachte, dass hierbei Zahlbereichsüberschreitungen auftreten können!



Aufgabe: Zeigeranwendung (1)

Ein Programmierer hat die Funktion **tausch** zum Vertauschen von 2 char-Variablen geschrieben:

```
#include <stdio.h>
void tausch (char, char) ;
int main (void)
{
    char a = 'a', Z = 'Z' ;
    tausch (a, Z) ;
    printf (" a = %c   Z = %c", a, Z) ;
    return 0 ;
}

void tausch (char c1, char c2)
{
    char speicher ;
    speicher = c1;  c1 = c2 ;   c2 = speicher ;
}
```





Aufgabe: Zeigeranwendung (2)

1. Analysieren Sie das Programm!
2. Welche Werte werden ausgegeben? – Begründung!
3. Korrigieren Sie gegebenenfalls die Funktion **tausch**!

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 27



Aufgabe: Zeigeranwendung (1)

1. Programmanalyse

```
#include <stdio.h>
void tausch (char c1, char c2) ; /* Prototypenvereinbarung: tausch */

int main (void)                /* Hauptprogramm */
{
    char a = 'a', Z = 'Z' ;    /* char a, Z vereinbaren: a = 'a', Z = 'Z' */
    tausch (a , Z) ;           /* Funktion tausch (a,Z) aufrufen */
    printf (" a = %c   Z = %c", a , Z) ; /* a und Z ausgeben */
    return 0 ;                 /* Rücksprung ins Betriebssystem */
}

void tausch (char c1, char c2) /* Funktion tausch */
{
    char speicher ;
    speicher = c1;   c1 = c2 ;   c2 = speicher ;
    /* Die Inhalte von c1 und c2 sollen vertauscht werden */
}
```

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 28

✓ **Aufgabe: Zeigeranwendung (2)**

2. Ausgabe: a = a Z = Z **tausch ist falsch!**

Das Vertauschen hat nicht funktioniert. Fehlerursache?

In C werden beim Funktionsaufruf **immer Kopien von allen Parametern** erstellt:

c1 = 'a' c2 = 'Z'

Die Funktion **tausch** nimmt folgende Operationen vor:

speicher = c1; c1 = c2; c2 = speicher;

main

a

'a'

Z

'Z'

am Ende von tausch

c1

'Z'

c2

'a'

Die Kopien sind vertauscht, die Originalvariablen nicht!

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

Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 29

✓ **Aufgabe: Zeigeranwendung (3)**

Speicherbelegung im falschen Programm **tausch**:

Die Adressen der Speicherzellen sind hier willkürlich vorgegeben. C verwaltet den Speicher selbst!

Variable	Byte-Adresse	Inhalt am Anfang von tausch	Inhalt am Ende von tausch
a	0x100	'a'	'a'
Z	0x101	'Z'	'Z'
	...	...	...
c1	0x2500	'a'	'Z'
c2	0x2501	'Z'	'a'
speicher	0x2502	'a'	'a'

speicher = c1;
c1 = c2;
c2 = speicher;

In **tausch** sind nur die **Kopien** von 'a' (Variable **c1**) und 'Z' (Variable **c2**) vertauscht!

Die **Originale** a und Z bleiben erhalten!

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

Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 30



Aufgabe: Zeigeranwendung (4)

3. Korrektes Programm

```
#include <stdio.h>
#include <conio.h>
void swap (char *, char *) ; /* Prototypenvereinbarung von swap */
int main (void)              /* Hauptprogramm */
{
    char a = 'a', Z = 'Z' ;   /* char a, Z vereinbaren: a = 'a', Z = 'Z' */
    swap (&a, &Z) ;           /* Funktion swap (a,Z) aufrufen */
    fflush (stdin) ;          /* Fuer system ("cls") Tastaturpuffer loeschen */
    system ("cls") ;          /* Bildschirm loeschen */
    printf (" a = %c   Z = %c", a, Z) ; /* a und Z ausgeben */
    getch ( ) ;               /* Output-Fenster offen bis Tastendruck */
    return 0 ;                /* Rucksprung ins Betriebssystem */
}

void swap (char *p1, char *p2) /* Funktion swap */
{
    char speicher ;
    speicher = *p1 ; *p1 = *p2 ; *p2 = speicher ;
    /* Die Objekte von p1 und p2 werden vertauscht */
}
```

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 31

27.05.2004



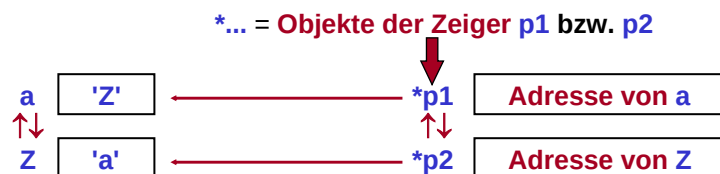
Aufgabe: Zeigeranwendung (5)

3. Ausgabe: a = Z Z = a

Das Vertauschen der Variablen hat funktioniert. Warum ?

main

am Ende von **tausch**



Die Originalvariablen werden vertauscht!

Auch hier werden die **Inhalte der Zeiger (Adressen)** kopiert.

Die Kopien enthalten aber **dieselben Adressen** wie die **Originalen**!

Mit Hilfe des **Verweis-Operators *** wird korrekt auf die **Inhalte (Objekte)** der kopierten Zeiger zugegriffen!

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 32

27.05.2004

✓ **Aufgabe: Zeigeranwendung (6)**

Speicherbelegung im richtigen Programm swap:

Variable	Byte-Adresse	Inhalt am Anfang von swap	Inhalt am Ende von swap
p1 → a	0x100	'a'	'Z'
p2 → z	0x101	'Z'	'a'
	...	...	...
&a	0x250	0x100	0x100
&z	0x254	0x101	0x101
	...	...	...
p1	0x2000	0x100	0x100
p2	0x2004	0x101	0x101
speicher	0x2008	'a'	'a'

speicher = *p1;
***p1 = *p2;**
***p2 = speicher;**

In swap wird mit den **Kopien** von **&a** (Pointer **p1** auf 'a') und **&z** (Pointer **p2** auf 'Z') mit dem **Verweisoperator *** auf die Originale zugegriffen: Die **Originale a und z** sind vertauscht!

27.05.2004

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

Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 33

Zeiger- und Feldoperationen (1)

- In C sind Zeiger und Felder eng miteinander verwandt. Häufig kann man statt Feldindizes auch Zeiger verwenden.
- Ein **Feldname** enthält die **nicht veränderbare Adresse des ersten Feldelements**.

Beispiel:

```
char string [ ] = "Ich bin ein string!"; *p_s1, *p_s2 ;
/* Ein string namens string der Länge 20 Bytes und die 2 Zeiger
   auf char p_s1 und p_s2 werden vereinbart */
p_s1 = &string[0] ;
/* p_s1 enthält nun die Anfangsadresse von string [0] */
p_s2 = string ;
/* p_s2 enthält nun ebenfalls die Anfangsadresse von string [0] */
/* Es gilt: p_s1 = p_s2 ! */
```

27.05.2004

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

Programmieren in ANSI C:
 K10_Zeiger.ppt, Seite 34

Zeiger- und Feldoperationen (2)

- Ein **Feldname** enthält die Adresse des ersten Feldelements:
Sie darf aber nicht verändert werden:

☛ ein **Feldname** ist **kein veränderbarer Zeiger**!

Beispiele:

```
char string [ ] = "Ich bin ein string!", *zs ;
string = &string [2] ;
```

/* Die letzte Anweisung ist falsch: Ein Feldname ist mit dem Feld fest verbunden. Ihm kann keine neue Adresse zugewiesen werden.

⇒ Der **MSVC-Compiler** gibt die **Fehlermeldung**:
'error C2106: '=' : Linker Operand muss ein L-Wert sein Fehler beim Ausführen von cl.exe.' aus! */

```
zs = &string ;
```

/* Diese Anweisung ist unsinnig,
weil *string* ja bereits die Anfangsadresse des Feldes enthält.

⇒ Der **Borland C-Compiler** ist gnädig: Er gibt die **Warnung**:
'Suspicious pointer conversion' aus
und weist *zs* die Anfangsadresse des Feldes zu! */

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 35

Zeiger- und Feldoperationen (3)

- Häufig verwendet man in C statt Feldindizes auch Zeigeroperationen wie z.B. **pointer++** bzw. **--pointer**.

Beispiel: `short *pa, s [3] = { 10, 256, 513 } ;` **/* short belegt 2 Byte */**
`pa = s ;` **/* Annahme: s ist ab Adresse 0x200 gespeichert */**
`... pa++ ;`

	<code>pa = s</code>		<code>pa++</code>			
	↓ <code>s[0]</code>		↓ <code>s[1]</code>			<code>s[2]</code>
Adresse	0x200	0x201	0x202	0x203	0x204	0x205
Inhalt	0xa	0x0	0x0	0x1	0x1	0x2

bei INTEL-PCs: LS-Byte MS-Byte LS-Byte MS-Byte LS-Byte MS-Byte

Bei einer Zeigervereinbarung muss ein Datentyp angegeben werden, damit der **C-Compiler** mit den Operatoren **++** bzw. **--** auf die Feldelemente **richtig zugreifen kann**.

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 36

Zeiger- und Feldoperationen (4)

- Man verwendet man in C **statt Feldindizes** häufig **Zeigeroperationen**.
- Additions- und Subtraktionsoperatoren mit Zeigern verweisen **stets** auf die Feldelemente, wenn man die **Feldgrenzen nicht überschreitet**.

Beispiel:

```
long feld [5] = {40, 30, 20, 10, 0}, *p1, *p2, i, j, k;
/* long feld feld der Länge 5, die Zeiger p1 und p2
auf long und die 3 long-Variablen i, j, k vereinbaren */

p1 = feld; /* p1 enthält die Anfangsadresse von feld [0] */
p2 = &feld [3]; /* p2 zeigt auf 10 */
i = feld [4]; /* i = 0 */
j = *(++p1); /* j = 30 (p1 zeigt auf 30) */
k = *(p2--); /* k = 10 (p2 zeigt auf 20 nach der Operation) */
```

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 37

27.05.2004

Zeiger- und Feldoperationen (5)

- Man kann Felder mit **Feldoperationen** oder mit **Zeigeroperationen** ansprechen.

Beispiel: Zugriff mit ➡ **Feldoperation**

```
...
char s1, s [ ] = "Schon wieder ein string!";
short i = 6 ;
s1 = s [i++]; /* s1 = 'w'; i = 7 nach der Operation */
```



Beispiel: Zugriff mit ➡ **Zeigeroperation**

```
...
char s1, s [ ] = "Schon wieder ein string!";
short i = 6 ;
s1 = *(s + i++); /* s1 = 'w'; i = 7 nach der Operation */
```



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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 38

27.05.2004

Zeiger- und Feldoperationen (6)

- Man kann **konstante Felder** auch mit
➔ **Zeigeroperationen** definieren.

Dabei kann man den **Zeiger direkt verändern**.

Beispiel: Definition eines Strings mit einer **Zeigeroperation**

```
...
char *s = "Noch'n string!"; *p_s = s;
++p_s;          /* p_s zeigt nun auf 'o' */
...
```

- Eine **einfache Erweiterung** des Feldes ist dann allerdings nicht mehr möglich.

Beispiel: **Fehlerhafte** Stringerweiterung

```
...
char *s = "Noch'n string!";
*(s + 14) = "!\0";    /* falsch */
```

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 39

27.05.2004

Zeigeroperationen

Geben Sie für das folgende C-Programm nach jeder Zeile die Inhalte aller Variablen an! *Annahme: Das Feld beginnt bei Adresse 0x1000.*

```
int main(void)
{
    int i, *Z_Feld ;
    int Feld[ ] = {10, 20, 30} ;
    Z_Feld = Feld ;
    Z_Feld = &Feld[1] ;
    i = *Z_Feld ;
    i = *Z_Feld + + ;
    i = *Z_Feld - - ;
    i = *+ + Z_Feld ;
    i = *- - Z_Feld ;
    i = (*Z_Feld) + + ;
    i = (*Z_Feld) - - ;
    i = + + (*Z_Feld) ;
    i = - - (*Z_Feld) ;
    return 0 ;
}
```

/* Feldvereinbarung */
/* Annahme: jedes int-Feld-Element belegt 2 Bytes! */

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 40

27.05.2004

✓ Zeigeroperationen					
	Z_Feld	i	Feld[0]	Feld[1]	Feld[2]
int i, *Z_Feld;					
int Feld[] = {10, 20, 30};	?	?	10	20	30
Z_Feld = Feld;	0x1000	?	10	20	30
Z_Feld = &Feld[1];	0x1002	?	10	20	30
i = *Z_Feld;	0x1002	20	10	20	30
i = *Z_Feld++;	0x1004	20	10	20	30
i = *Z_Feld--;	0x1002	30	10	20	30
i = *++Z_Feld;	0x1004	30	10	20	30
i = *--Z_Feld;	0x1002	20	10	20	30
i = (*Z_Feld)++;	0x1002	20	10	21	30
i = (*Z_Feld)--;	0x1002	21	10	20	30
i = ++(*Z_Feld);	0x1002	21	10	21	30
i = --(*Z_Feld);	0x1002	20	10	20	30

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 41

27.05.2004

Funktionen: Parameterübergabe bei Feldern

- Eine **Funktion** kann mit dem **Feldnamen** = Zeiger auf Adresse des 1. Feldelements auf den **Inhalt des Originals** zugreifen.
- Sie kann somit das **Original des Feldes verändern**.
- Um Speicherplatz zu sparen, **vermeidet man in C Kopien von Feldern** und übergibt statt dessen den **Feldnamen**.

Beispiel:

```
void func1 ( int f [3] )    /* Eine Funktionsdefinition zu Anfang
                           /* vermeidet eine Prototypenvereinbarung */
    { f [1] = f [2];        /* Alternative: *(f + 1) = *(f + 2); */
int main(void)             /* Hauptprogramm */
    {
        int feld [3] = {1, 2, 3};
        ➔ func1 ( feld );    /* Alternative: func1 ( &feld [0] ); */
        ...;                /* feld [1] enthält nun 3 */
        return 0;
    }
```

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 42

27.05.2004

Zeiger und mehrdimensionale Felder (1)

- In C werden mehrdimensionale Felder
➡ **zeilenweise** abgelegt.

Beispiel:

```
short *p_m1, m1 [2] [3] = { {1, 2}, {11, 22, 33} };
```

Das Feld von **2 Zeilen** und **3 Spalten** wird **zeilenweise** initialisiert:

$m1[0][0] = 1$ $m1[0][1] = 2$ $m1[0][2] = 0$
 $m1[1][0] = 11$ $m1[1][1] = 22$ $m1[1][2] = 33$

Beispiel:

$p_m1 = \&m1[0][1]$ /* Zeiger p_m1 zeigt auf $m1[0][1] = 2$ */
 $++p_m1;$ /* Zeiger p_m1 zeigt auf $m1[0][2] = 0$ */
 $++p_m1;$ /* Zeiger p_m1 zeigt auf $m1[1][0] = 11$ */

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 43

27.05.2004



Aufgabe: Stringkopieren mit Zeigern

In Kapitel 7 haben wir den folgenden Programmausschnitt zum Kopieren eines Strings geschrieben:

```
...
int i = 0, k = 0;
char string [100], s [ ] = "Ich string so vor mich hin." ;
while ( string [i++] = s [k++] )
...
```

- Schreiben Sie die möglichst kurze Funktion **cpy_st (*org, *copy)** zum Kopieren eines Strings mit den Zeigern **org** auf das Originalfeld und **copy** das kopierte Feld als Parametern! Verwenden Sie eine **for**-Schleife und für den Zugriff auf die Feldelemente einen Operator der Form ***(Feld+Index)**!
- Schreiben Sie die möglichst kurze Funktion **cpy_str (*org, *copy)** zum Kopieren eines Strings. Beim Aufruf werden nun ein Zeiger auf die Anfangsadresse des Originalfeldes und ein Zeiger auf **string[10]** als Startadresse für das kopierte Feldes übergeben!

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 44

27.05.2004



Aufgabe: Stringkopieren mit Zeigern (1)

1. Programm mit dem Operator *(Feld+Index):

```
#include <stdio.h>
void cpy_st (char *, char *); /* Prototypenvereinbarung: cpy_st */
int main (void)             /* Hauptprogramm */
{
    char string [100], s [ ] = "Ich string so vor mich hin.";
    cpy_st (s, string);      /* Funktion cpy_st ( ) aufrufen */
    printf (" string = %s \n s = %s", string , s ); /* string, s ausgeben */
    return 0;                /* Ruecksprung ins Betriebssystem */
}

void cpy_st (char *z0, char *z1) /* Funktion cpy_st */
{ /* cpy_st kopiert den '\0'-terminierten String mit der Anfangsadresse
   z0 in den '\0'-terminierten String mit der Anfangsadresse z1 */
    int i;
    for (i=0; *(z0+i) != 0; i++) *(z1+i) = *(z0+i);
    /* String, auf den z0 zeigt, in den String, auf den z1 zeigt kopieren*/
    *(z1+i) = *(z0+i); /* for-Befehl!/: String-Terminator \0 kopieren! */ }
```

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 45



Aufgabe: Stringkopieren mit Zeigern (2)

2. möglichst kurzes Programm

```
#include <stdio.h>
#include <conio.h>
void cpy_str (char *c1, char *c2); /* Prototypenvereinbarung: cpy_str */
int main (void)                 /* Hauptprogramm */
{ char string [100], s [ ] = "Ich string so vor mich hin.";
  /* char string[100], s[ ] vereinbaren */
  fflush ( stdin );             /* Fuer system ("cls") Tastaturpuffer loeschen */
  system ( "cls" );            /* Bildschirm loeschen */
  cpy_str (&s[0], &string[10]); /* Funktion cpy_str (...) aufrufen */
  printf (" s = %s \n string[10] -> = %s", s ,&string[10] );
  /* string, s ausgeben */
  return 0;                    /* Rücksprung ins Betriebssystem */ }

void cpy_str (char *z1, char *z2) /* Funktion cpy_str */
{ /* cpy_str kopiert den '\0'-terminierten String mit der Anfangsadresse
   z1 in den '\0'-terminierten String mit der Anfangsadresse z2 */
  while (*z2++ = *z1++);
  /* String, auf den z1 zeigt, in den String, auf den z2 zeigt kopieren*/ }
```

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 46

Parameterübergabe: mehrdimensionale Felder (1)

- Bei statisch definierten Feldern braucht der C-Compiler **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; /* Alternative: *(f + 1 * 3 + 2) = 2; */
}

int main ( void ) /* Hauptprogramm */
{
    int feld [2] [3] = { {10, 20, 30}, {100, 200, 300} };
    /* 2-dimensionale Feldinitialisierung */
    f_m ( feld ); /* Alternative: f_m = ( &feld [0] ) ; */
    /* 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:
K10_Zeiger.ppt, Seite 47

27.05.2004

Parameterübergabe: mehrdimensionale Felder (2)

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

Feldelementadresse = Adresse des ersten Feldelements + (Zeilenindex * Anzahl der Spalten) + Spaltenindex

- Die **Zeilenanzahl** ist bei der Adressberechnung **nicht nötig**. 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:
K10_Zeiger.ppt, Seite 48

27.05.2004



Aufgabe: Transponierte einer Matrix (Zeigerversion)

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:
K10_Zeiger.ppt, Seite 49

27.05.2004



Aufgabe: Transponierte einer Matrix Zeigerversion (1)

Programm (1):

```
#define MAX 100          /* Vereinbarung für MAX */
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], s=2, z=3;
                                /* Matrizen m, m_t & s, z vereinbaren */
  transp (m, m_t, z, s);    /* Funktion transp ( ) aufrufen */
  return 0;                /* Ruecksprung ins Betriebssystem */ }
```

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 50

27.05.2004



Aufgabe: Transponierte einer Matrix Zeigerversion (2)

Programm (2):

```
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 *p_t, i, j;
  p_t = m_t;
  for (i=0; i<z; i++)
  {
    for (j=0; j<s; ) *p_t++ = m [ j++ ] [ i ]; /* 1 Zeile von m_t berechnen */
  }
}
```

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 51

Statische Feldvereinbarungen in C

- Statische Feldvereinbarungen haben einige Nachteile:

- ➡ es wird während der gesamten Ablaufzeit des Programms nicht benötigter Speicherplatz belegt
- ➡ wird ein größeres Feld als in der **#define**-Präprozessoranweisung angegeben, so muss das Programm neu übersetzt werden
- ➡ beim Debuggen sind die Felder unnötig groß: Man findet viele undefinierte Speicherplätze, die 'Müll' enthalten.
- ➡ Wir werden später eine häufiger gebrauchte und effizientere Methode kennen lernen, die
 - ➡ **dynamische Speicheranforderung**

27.05.2004

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



Programmieren in ANSI C:
K10_Zeiger.ppt, Seite 52