

Erweiterte Zeigerfunktionen in C

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 1

Erweiterte Zeigerfunktionen in C

- In diesem Kapitel wollen wir zeigen:
 - ➔ was Zeigerfelder sind
 - ➔ wozu man Zeigerfelder verwendet
 - ➔ wie man Kommandozeilenparameter an ein Hauptprogramm übergibt
 - ➔ wie man Zeiger auf Funktionen vereinbart
 - ➔ wozu man Zeiger auf Funktionen verwendet.
 - ➔ wozu man Zeiger auf Zeiger verwendet.

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 2

Zeigerfelder

- In C sind Zeiger Variable. Sie können zu
➡ **Zeigerfeldern** zusammengefasst werden.

Beispiele für die Definition von Zeigerfeldern:

```
#define max_zeiger 50
char *Zfeld [max_zeiger]; /* Zeigerfeld auf char mit
                           max_zeiger = 50 Zeigern vereinbaren */
float *f_zeiger [10];      /* Zeigerfeld auf float mit
                           10 Zeigern vereinbaren */
```

- Zeigerfelder vereinfachen und beschleunigen z.B. das
Sortieren von Matrizen erheblich:

Man sortiert z.B. Zeiger auf Matrixzeilen.

Das geht wesentlich schneller als alle Elemente einer
Matrixzeile zu sortieren.

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 3

Zeigerfelder: Parameterübergabe an *main (...)* (1)

- An Programme können beim Aufruf
➡ **Kommandozeilenparameter**
(*command line arguments*) übergeben werden.

Beispiel: Aufruf des Programms *test.exe* unter *MS DOS* mit
Kommandozeilenparameter:

I:\MS VISUAL C++\...\test.exe E2B, Programmieren in C, "sehr gut"

Programm
Kommandozeilenparameter

- Auch unter *MS DOS / Windows* können an ein C-
Hauptprogramm *main (...)* **Kommandozeilenparameter**
übergeben werden.
- Zur Parameterübergabe an *main (...)* werden
➡ **Zeigerfelder verwendet.**

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 4

Zeigerfelder: Parameterübergabe an *main (...)* (2)

- Unter *MS DOS / Windows* können an ein **Hauptprogramm Kommandozeilenparameter** übergeben werden.
- Dazu benutzt *main (...)* die 2 Kommandozeilenparameter:
 - ➔ `int argc`
 - ➔ `char* argv [ ]`
- `argc` ist ein **Integer-Parameter** und enthält die **Anzahl der Kommandozeilenargumente**.
- `argv [ ]` ist ein **Zeigerfeld** auf **eventuelle weitere Parameter**, die jeweils als **strings** übergeben werden

02.06.2004



Beispiel für die Parameterübergabe an *main (...)* (1)

- Das folgende *BorlandC*-Programm *test.c* unter *MS DOS / Windows* gibt die **Kommandozeilenparameter** auf dem Bildschirm aus:

Beispiel: Kommandozeilenparameter *MSVC++ / MS DOS*:

```
/* Projekt: test, Datei: test.c HAW/VGL 11.4.04 */
#include <stdio.h> /* Bibliothek für printf (..) einbinden */
int main ( int argc, char *argv[ ] )
{
    int ianz_par ; /* Schleifenparameter ianz_par vereinbaren */
    printf ( " ianz_par = %u\n", argc ) ; /* Parameteranzahl ausgeben */
    for ( ianz_par = 0; ianz_par < argc ; ianz_par++ )
        printf ( " %s\n", argv [ ianz_par ] ) ; /* Parameter ausgeben */
    return 0;
}
```

- Beim **Übersetzen** wird das Programm *test.exe* erzeugt.

02.06.2004



Beispiel für die Parameterübergabe an *main (...)* (2)

- Beim **Übersetzen** erzeugt der **MSVC-Compiler** aus dem **C-Projekt test** das ausführbare **DOS-Programm test.exe**.
- Die **Kommandozeilenparameter** sind **optional**.

Das Programm **test.exe** kann man in einer **MS DOS-Kommandozeile** (→ **Start** → **Ausführen**) starten:

I:\MS VISUAL C++\...\test.exe E2B, Programmieren in C, "sehr gut"

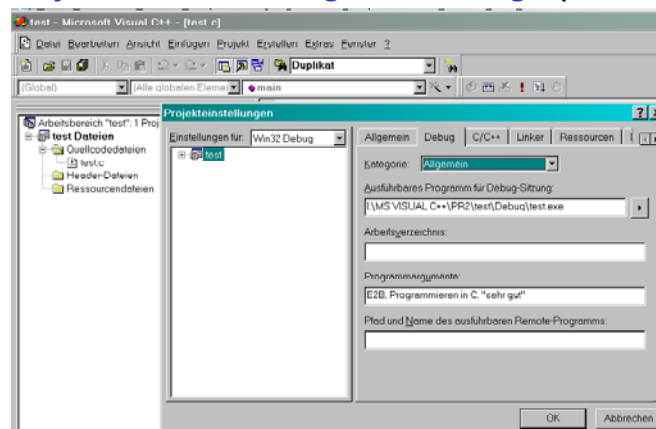
- Das Programm kann auch in der integrierten **Benutzeroberfläche von MSVC++ mit Argumenten** aufgerufen werden.

02.06.2004



Beispiel für die Parameterübergabe an *main (...)* (3)

- Ausführen des Programms **test.exe** in **MSVC++** mit **folgenden Kommandozeilenargumenten** im Menü **Projekt** → **Einstellungen** → **Debug** (**ALT+F7**)



02.06.2004



Beispiel für die Parameterübergabe an *main (...)* (4)

- Ausführen des Programms *test.exe* in *MSVC++* mit
Erstellen → *! Ausführen von ...* (*STRG+F5*)

```

C:\I:\MS VISUAL C++\PR2\test\Debug\test.exe" E2B, Programmieren in C, "sehr gut"
ianz_par = 6
I:\MS VISUAL C++\PR2\test\Debug\test.exe
E2B,
Programmieren
in
C,
sehr gut
Press any key to continue

```

02.06.2004



Beispiel für die Parameterübergabe an *main (...)* (5)

- Ausführen des Programms *test.exe* in *MSVC++* mit
Erstellen → *! Ausführen von test.exe* (*STRG+F5*)

C-Programm:

```

/* Projekt: test, Datei: test */
#include <stdio.h>

int main ( int argc, char *argv[ ] )
{ int ianz_par ;
  printf ( „ianz_par: %u\n“, argc ) ;
  for ( ianz_par = 0; ianz_par < argc ;
        ianz_par++ )
    printf ( \"%s\n“, argv [ ianz_par ] ) ;
  return 0; }

```

Bildschirmausgabe:

```

ianz_par = 6
I:\MS Visual C++\ ... \test.exe
E2B,
Programmieren
in
C,
sehr gut

```

02.06.2004



Beispiel für die Parameterübergabe an *main (...)* (6)

- Eigenschaften der Parameterübergabe an *test.exe*:

- ➔ Das Betriebssystem übergibt den 1. Parameter *argc* (Anzahl der Argumente) als *int*.
- ➔ Der folgenden Parameter werden als **Zeigerfeld auf Strings** übergeben: *char* argv []*.
- ➔ Das **1. Zeigerfeld** enthält den **Namen des aufgerufenen Programms mit vollständigem Pfad**.
- ➔ Jedes Leerzeichen (*blank*) wird als **Trennzeichen zwischen den Strings** interpretiert.
Das Komma hat keine Steuerfunktion.
- ➔ Soll ein Parameter Leerzeichen enthalten, so muss er in Anführungszeichen **"..."** gesetzt werden.

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 11

Aufgabe: Sortieren von Matrixzeilen

Gegeben ist die folgende Matrix:

short m [5] [3] = { {1, 2}, {11, 22, 33}, {-11,-22,-33}, {-1,-2,-3}, {111} };

Schreiben Sie die Funktion *sort_ZE*, die in einem Zeigerfeld anzeigt, wie die Zeilen der Matrix angeordnet sein müssen, wenn jeweils die Elemente der 1. Spalte ➔ *m [i] [0]* nach der Größe in aufsteigender Reihenfolge sortiert sind!

m [i] [0] ↓

In diesem Falle:

<i>m [2] [0] = -11</i>	<i>m [2] [1] = -22</i>	<i>m [2] [2] = -33</i>
<i>m [3] [0] = -1</i>	<i>m [3] [1] = -2</i>	<i>m [3] [2] = -3</i>
<i>m [0] [0] = 1</i>	<i>m [0] [1] = 2</i>	<i>m [0] [2] = 0</i>
<i>m [1] [0] = 11</i>	<i>m [1] [1] = 22</i>	<i>m [1] [2] = 33</i>
<i>m [4] [0] = 111</i>	<i>m [4] [1] = 0</i>	<i>m [4] [2] = 0</i>

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 12



Aufgabe: Sortieren von Matrixzeilen (1)

Lösung:

Zunächst werden alle 1. Spaltenindizes des Feldes im Zeigerfeld `z_m` in der ursprünglichen Reihenfolge abgelegt.

Es wird eine *in-place* Sortierung durchgeführt:

Es wird zunächst das aktuelle Minimum gesucht und dessen Zeiger mit dem von `z_m[0][0]` getauscht.

Danach wird der Algorithmus mit `z_m[i][0]`, $i = 1, 2, \dots, z$ fortgesetzt.

Vorteile:

- ➡ schnelle Sortierung
- ➡ wenig zusätzlicher Speicherbedarf

`short sort_ZE (short m [ ] [max], short z, short s, short *z_m [max])`

/ 2-dimensionales Feld m [max] [max]*

Zeilenanzahl (m) = z,

Spaltenanzahl (m) = s,

*z_m [max] Zeigerfeld auf sortierte 1. Zeilenelemente vereinbaren */*

/ sort_ZE = 0, wenn keine Feldgrenzen überschritten wurden*

*und =1, wenn $s > max$ oder $z > max$. Im Fehlerfall wird nicht sortiert */*

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 13



Aufgabe: Sortieren von Matrixzeilen (2)

Programm (1): Definitionen und Prototypen:

`#define max 20` */* maximale Zeilen- & Spaltenanzahl der Matrix */*

`#include <stdio.h>` */* Bibliothek für scanf (), printf () */*

`short sort_ZE (short m[ ] [max], short, short, short *f [max]);`

/ Prototypenvereinbarung von sort_ZE */*

/ 2-dimensionales Feld m [max] [max],*

Zeilenanzahl von m = z, Spaltenanzahl von m = s

*Zeigerfeld auf sortierte 1. Zeilenelemente */*

/ sort_ZE = 0, wenn keine Feldgrenzen ueberschritten wurden*

und =1, wenn $s > max$ oder $z > max$.

*Im Fehlerfall wird nicht sortiert */*

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 14



Aufgabe: Sortieren von Matrixzeilen (3)

Programm (2): Hauptprogramm (1):

```
int main (void)                /* Hauptprogramm */
{
    short i, j, error=1, m [max] [max] =
    { {1, 2}, {11, 22, 33}, {-11,-22,-33}, {-1,-2,-3}, {111} };
    /* Laufindizes i, j, Fehleranzeiger error, Feld m */
    short s=3, z=5, *temp; /* Spalten-& Zeilenzahl (m), Zeiger temp */
    short *pm [max];      /* Zeigerfeld auf sortierte 1 Zeilenelemente */
    printf ("Unsortiertes Feld:\n\n");
    for ( i=0; i<5; i++ )    /* unsortiertes Feld ausgeben */
    {
        for ( j=0; j<3; j++ ) printf (" m[%1d][%1d] = %3d ", i, j, m[ i ][ j ] );
        printf("\n");
    }
    error = sort_ZE (m, z, s, pm); /* Funktion sort_ZE aufrufen */
    if (error) printf (" Fehler: Feldgrenzen ueberschritten!");
}
```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 15



Aufgabe: Sortieren von Matrixzeilen (4)

Programm (3): Hauptprogramm (2):

```
/* if (error) printf (" Fehler: Feldgrenzen ueberschritten!"); */
else
{ printf ("\n\nSortiertes Feld:\n\n");
  for ( i=0; i<5; i++ ) /* sortiertes Feld ausgeben */
  {
      temp = pm [ i ]; /* Pointer auf 1. Zeilenelemente setzen */
      for ( j=0; j<3; j++ )
      { printf (" m [%1d] [%1d] = %3d ", i, j, *(temp++));
        /* m sortiert ausgeben */ }
      printf("\n");
  }
}
printf("\n\n*** Programmende: Return-Taste druecken ! ***");
getchar ( ); /* Ausgabefenster bis ↵-Tastendruck anzeigen */
return 0; /* Ruecksprung ins Betriebssystem */
}
```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 16

✓ Aufgabe: Sortieren von Matrixzeilen (5)

Programm (4): Funktion sort_ZE (1):

```
short sort_ZE (short m [ ][max], short z, short s, short *z_m [ ])
/* 2-dimensionales Feld m,
   Zeilenanzahl von m = z, Spaltenanzahl von m = s
   Zeigerfeld auf sortierte 1. Zeilenelemente vereinbaren */
/* sort_ZE = 0, wenn Feldgrenzen nicht ueberschritten wurden
   und 1, wenn s > max oder z > max. Im Fehlerfall wird nicht sortiert
   und *z_m ist undefiniert. */
{
  short i, j, ja, *t1;      /* Zeigerfeld, Minimum (Zwischenspeicher),
                           Laufindizes, temporaeren Pointer vereinbaren*/
  if ( (s > max-1) || (z > max-1) ) return 1; /* Fehler: Feldgrenzen
   ueberschritten */
  else
    /* { ... */
```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 17

✓ Aufgabe: Sortieren von Matrixzeilen (6)

Programm (5): Funktion sort_ZE (2):

```
/* else */
{ for (i=0; i<z; i++) z_m[ i ] = &m[ i ][ 0 ]; /* Pointer→1. Zeilenelement */
  for (i=0; i<z+1; i++) /* alle Zeilen bis auf die letzte durchlaufen */
  {
    t1 = z_m[ i ]; /* Min-Adresse → t1: Annahme: m[0][0] = Minimum*/
    ja = i;      /* Zeilenindex des angenommenen Minimums → ja */
    for ( j=i+1; j<z; j++ ) /* alle restlichen Zeilen durchlaufen */
    {
      if ( *(z_m[ j ]) < *t1 ) /* neues Minimum ?*/
      {
        /* ja: */
        ja = j;      /* Index des aktuellen Minimums in ja merken */
        t1 = z_m[ j ]; /* Adresse des aktuellen Minimums → t1 */
      }
    }
    z_m[ ja ] = z_m[ i ]; /* Minimalwert mit z_m[i] tauschen */
    z_m[ i ] = t1;      /* i erhoehen */
  }
  return 0; /* kein Fehler */ }
```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 18



Aufgabe: Sortieren von Matrixzeilen (7)

Ausdruck:

Unsortiertes Feld:

<code>m [0] [0] = 1</code>	<code>m [0] [1] = 2</code>	<code>m [0] [2] = 0</code>
<code>m [1] [0] = 11</code>	<code>m [1] [1] = 22</code>	<code>m [1] [2] = 33</code>
<code>m [2] [0] = -11</code>	<code>m [2] [1] = -22</code>	<code>m [2] [2] = -33</code>
<code>m [3] [0] = -1</code>	<code>m [3] [1] = -2</code>	<code>m [3] [2] = -3</code>
<code>m [4] [0] = 111</code>	<code>m [4] [1] = 0</code>	<code>m [4] [2] = 0</code>

Sortiertes Feld:



<code>m [0] [0] = -11</code>	<code>m [0] [1] = -22</code>	<code>m [0] [2] = -33</code>
<code>m [1] [0] = -1</code>	<code>m [1] [1] = -2</code>	<code>m [1] [2] = -3</code>
<code>m [2] [0] = 1</code>	<code>m [2] [1] = 2</code>	<code>m [2] [2] = 0</code>
<code>m [3] [0] = 11</code>	<code>m [3] [1] = 22</code>	<code>m [3] [2] = 33</code>
<code>m [4] [0] = 111</code>	<code>m [4] [1] = 0</code>	<code>m [4] [2] = 0</code>

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 19

02.06.2004

Zeiger auf Funktionen (1)

- In C kann man ➡ **Zeiger auf Funktionen** definieren.
- Zeigern auf Funktionen haben den Vorteil, dass man die aufgerufene Funktion **dynamisch** - also erst zu Laufzeit des Programms - **festlegen kann**.

Beispiel für die Definition eines Zeigers auf eine Funktion:

```
int (*z1) (char a, char b) ; /* Zeiger z1 auf eine Funktion mit den
                             2 Parametern a und b vom Typ char.
```

Die Funktion gibt eine *int*-Variable zurück*/

- Der Zeiger auf eine Funktion zeigt auf die
➡ **Einsprungsadresse** der Funktion:

Dies ist die Adresse, an der die **Ausführung des Programms** der Funktion beginnt.

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 20

02.06.2004

Zeiger auf Funktionen (2)

Beispiel für die Definition eines Zeigers auf eine Funktion:

```
int (*z1) (char a, long b); /* Zeiger auf eine Funktion mit den
                             2 Parametern a vom Typ char und b (long).
                             Die Funktion gibt eine int-Variable zurück.*/
```

↑ ↑ ↑

- Der Typ des Zeigers auf eine Funktion ist der Typ des Rückgabewertes der Funktion.

☛ Man beachte: Wegen der Vorrangregeln in C sind die Klammern um den Zeiger **z1** auf die Funktion unbedingt erforderlich!

☛ Man vergleiche mit:

```
int *z2 (short x);
```

Die Funktion **z2** gibt einen Zeiger auf **int** zurück !

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 21

02.06.2004

Zeiger auf Funktionen (3)

Definition eines Zeigers auf eine Funktion:

```
int (*z1) (char a, char b) ;
```



- Funktion der Klammern um den Zeiger **z1**:

Die Klammern (...) gehören der **Prioritätsgruppe 1** an.

Die **Bearbeitungsrichtung** ist → (von links nach rechts).

Der Ausdruck wird folgendermaßen interpretiert:

1. '**z1**' ist ein Zeiger auf '.
2. '**(char a, char b)**' ist eine Funktion mit den Eingangsparametern **a** und **b** des Typs **char** '.

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 22

02.06.2004

Zeiger auf Funktionen (4)

Definition eines **Zeigers auf eine Funktion**:

```
int ( *z1 ) ( char a, char b ) ;
```

- Der obige Ausdruck wird folgendermaßen interpretiert:
 1. 'z1' ist ein Zeiger auf ' '.
 2. ' (char a, char b) ' eine Funktion mit den Eingangsparametern a und b des Typs char'.

Zum Vergleich das folgende Beispiel:

```
int * z1 ( char a, char b ) ;
```

Der Operator * hat eine niedrigere Priorität als die Klammern (...).

Deshalb wird zunächst z1 (char a, char b) ausgewertet und als **Funktion interpretiert**.

Diese gibt einen Zeiger auf int zurück.

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 23

02.06.2004

Beispiel für Zeiger auf Funktionen (1)

- Wir verwenden einen  Zeiger auf eine Funktion, um die 2 Zahlen zu subtrahieren:

```
#include <stdio.h>
int Differenz ( int, int ); /* Prototypenvereinbarung: 2 int-Parameter */
int main (void)           /* Hauptprogramm */
{
    int (*z) (int a, int b); /* Zeiger auf Funktion mit 2 int-Parametern: a,b */
    int Ergebnis;
    z = Differenz;           /* Hole die Einsprungsadresse von Differenz */
    Ergebnis = (*z) (100,200); /* Subtrahiere 200 von 100 */
    printf ("a-b = %d\n\n", Ergebnis); /* Ergebnis ausgeben */
    return 0;               /* Rücksprung ins Betriebssystem */
}

int Differenz ( int x, int y ) /* Funktion Differenz */
{
    return x - y;
}
```

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 24

02.06.2004

Beispiel für Zeiger auf Funktionen (2)

- Das Beispiel hat gezeigt, wie man mit einem **Zeiger** auf eine **Funktion** zugreifen kann.
- Der Nutzen des vorherigen Programms ist **nicht direkt einsichtig**: Man kann dieselbe Operation auch einfacher durchführen.
- Der Vorteil von Zeigern auf Funktionen besteht darin, dass sich **Zeiger manipulieren** lassen. Somit kann man erst zur Laufzeit des Programms festlegen, welche Funktion aufgerufen wird.
- Wir zeigen dies in einer Übungsaufgabe, in der wir **unterschiedliche Funktionen** für die **Addition**, die **Subtraktion** und die **Multiplikation** von 2 Variablen verwenden.

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 25

Aufgabe: Felder von Zeigern auf Funktionen

Schreiben Sie ein Programm, das mit Hilfe eines Zeigerfeldes auf Funktionen die Addition, die Subtraktion und die Multiplikation von 2 einzulesenden int-Variablen vornimmt.

Die Art der Operation wird ebenfalls eingelesen, z.B. 0 = Addition, 1 = Subtraktion, 2 = Multiplikation.

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 26

✓ Aufgabe: Felder von Zeigern auf Funktionen (1)

Definitionen und Prototypen:

```
#include <stdio.h>
```

```
int Summe ( int, int ) ; /* Prototypenvereinbarung: 2 int-Summanden */
int Differenz ( int, int ) ; /* Prototypenvereinbarung: 2 int-Summanden */
int Produkt ( int, int ) ; /* Prototypenvereinbarung: 2 int-Summanden */
int (*z[3]) ( int x, int y ) = { Summe, Differenz, Produkt } ;
/* Initialisiere das Zeigerfeld auf die Funktionen*/
```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 27

✓ Aufgabe: Felder von Zeigern auf Funktionen (2)

Hauptprogramm:

```
int main (void) /* Hauptprogramm */
{
    int Ergebnis, i, j, ok; /* Variablen vereinbaren */
    unsigned int op; char c[3] = {'+', '-', '*'};
    while (1)
    {
        printf (" Gib die Integer-Zahlen ein i, j ? : ");
        ok = scanf ("%d, %d", &i, &j); /* Operanden einlesen */
        if (ok != 2) break; /* falsche Eingabe beendet Programm! */
        printf (" Gib die Operation ein: 0 = + , 1 = - , 2 = * \n");
        scanf ("%u", &op); /* Operation einlesen */
        if ( op > 2) break; /* falsche Eingabe beendet Programm! */
        Ergebnis = (*z[op]) ( i, j ); /* Verarbeite i, j */
        printf ("%d %c %d = %d\n", i, c[op], j, Ergebnis); /* Ergebnis = */
    }
    return 0; /* Rücksprung ins Betriebssystem */
}
```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 28

✓ Aufgabe: Felder von Zeigern auf Funktionen (3)

Funktionen:

```
int Summe ( int x, int y )      /* Funktion Summe */
{ return x + y; }

int Differenz ( int x, int y )  /* Funktion Differenz */
{ return x - y; }

int Produkt ( int x, int y )    /* Funktion Produkt */
{ return x * y; }
```

02.06.2004



Aufgabe: Zeigeroperationen (2)

2. Gegeben ist folgender Programmausschnitt:

```
...
char *p = "Ich bin ein string.";
printf (p);
...
```

- 2.1 Dieser Programmausschnitt ist korrekt.
Geben Sie einen Zeigerausdruck an, mit dem Sie den '.' in der Zeichenkette durch ein '!' ersetzen!
- 2.2 Geben Sie an Stelle des oben angegebenen Programmausschnitts einen mit der gleichen Funktion an, indem Sie statt des Zeigers ein Feld verwenden! Welchen Vorteil hat die Definition eines Stringzeigers gegenüber der Definition mit eines Feldes?
- 2.3 Geben Sie einen Feldindexausdruck an, mit dem Sie den '.' in der Zeichenkette durch ein '!' ersetzen!

02.06.2004



✓ Aufgabe: Zeigeroperationen (2)

Lösungen:

Programmausschnitt:

```
...
char *p = "Ich bin ein string.";
printf (p);
...
```

2.1 Folgender Zeigerausdruck ersetzt den '.' durch ein '!':

```
*(p + 18) = '!';
```

2.2 Der Vorteil gegenüber der Definition von z.B.

```
q [ ] = "Ich bin ein string.";
```

liegt darin, dass der Zeiger *p* verändert werden kann (z.B. mit Hilfe von *p++*), der Feldname *q* aber nicht.

2.3 Feldindexausdruck für das Ersetzen des '.' durch ein '!':

```
q [18] = '!';
```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 31

✎ Aufgabe: Zeigeroperationen (3)

3. Gegeben ist folgender Programmausschnitt mit einem Zeigerfeld auf Strings mit Fehlermeldungen:

```
...
char *Fehler [ ] =
{ "Die Datei existiert nicht!",
  "Zu wenig Speicherplatz vorhanden!",
  "Die angegebene Datei existiert bereits!",
  "Schreibfehler: Ist die Datei schreibgeschützt?" };
...
```

Eine Funktion gibt als Fehlermeldung die ganze Zahl *error* des Typs *short* im Bereich von 0 ... 3 zurück, die auf den aufgetretenen Fehler verweist.

3.1 Welchen Wert muss die Variable *error* enthalten, um auf den Fehler "Zu wenig Speicherplatz vorhanden!" zu verweisen?

3.2 Verwenden Sie eine Feldindexoperation, um den Fehler "Zu wenig Speicherplatz vorhanden!" mit `printf ( )` auszugeben

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 32

Aufgaben: Zeigeroperationen (4)

4. Die Verwendung von Zeigern in C ist komplex.

Zur Übung erläutern Sie bitte jede der folgenden Anweisungen!

Beachten Sie dabei die Vorrangbedingungen (*priority list*) der Operatoren in C!

- 4.1 `float (*p) ( short *x );`
- 4.2 `float (*p ( short *x )) [10];`
- 4.3 `float p ( short (*x) [] );`
- 4.4 `float p ( short *x [] );`
- 4.5 `float *p ( short x [] );`
- 4.6 `float *p ( short (*x) [] );`
- 4.7 `float *p ( short *x [] );`
- 4.8 `float (*p) ( short (*x) [] );`
- 4.9 `float *(*p) ( short *x [] );`
- 4.10 `float (*p [10]) ( short x );`
- 4.11 `float *(*p [10]) ( short *x );`

!

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 33

✓ Aufgaben: Zeigeroperationen (4) (1)

Lösungen (1):

- 4.1 `float (*p) ( short *x );`
p ist ein Zeiger auf eine Funktion, die eine **float**-Variable zurückgibt und einen Zeiger auf eine **short**-Eingangsvariable erwartet.
- 4.2 `float (*p ( short *x )) [10];`
p ist eine Funktion, die einen Zeiger auf ein **float**-Feld mit 10 Elementen zurückgibt und einen Zeiger auf eine **short**-Eingangsvariable erwartet.
- 4.3 `float p ( short (*x) [] );`
p ist eine Funktion, die eine **float**-Variable zurückgibt und einen Zeiger auf ein **short**-Feld als Eingangsvariable erwartet.
- 4.4 `float p ( short *x [] );`
p ist eine Funktion, die eine **float**-Variable zurückgibt und ein Zeigerfeld auf **short**-Variablen als Eingangsvariable erwartet.

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 34

✓ Aufgaben: Zeigeroperationen (4) (2)

Lösungen (2):

4. 5 `float *p ( short x [ ] ) ;`

`p` ist eine Funktion, die einen Zeiger auf eine `float`-Variable zurückgibt und ein `short`-Feld als Eingangsvariable erwartet.

4. 6 `float *p ( short (*x) [ ] ) ;`

`p` ist eine Funktion, die einen Zeiger auf eine `float`-Variable zurückgibt und einen Zeiger auf ein `short`-Feld als Eingangsvariable erwartet.

4. 7 `float *p ( short *x [ ] ) ;`

`p` ist eine Funktion, die einen Zeiger auf eine `float`-Variable zurückgibt und ein Zeigerfeld auf `short` als Eingangsvariable erwartet.

02.06.2004



✓ Aufgaben: Zeigeroperationen (4) (3)

Lösungen (3):

4. 8 `float (*p) ( short (*x) [ ] ) ;`

`p` ist Zeiger auf eine Funktion, die eine `float`-Variable zurückgibt und einen Zeiger auf ein `short`-Feld als Eingangsvariable erwartet.

4. 9 `float *(*p) ( short *x [ ] ) ;`

`p` ist Zeiger auf eine Funktion, die einen Zeiger auf eine `float`-Variable zurückgibt und ein Zeigerfeld auf `short` als Eingangsvariable erwartet.

4.10 `float (*p [10]) ( short x ) ;`

`p` ist Zeigerfeld mit 10 Elementen auf Funktionen: Jede Funktion gibt eine `float`-Variable zurück und erwartet eine `short`-Eingangsvariable.

02.06.2004



✓ Aufgaben: Zeigeroperationen (4) (4)

Lösungen (4):

4.11 `float *(*p [10]) ( short *x ) ;`

`p` ist Zeigerfeld mit 10 Elementen auf Funktionen : Jede Funktion gibt einen Zeiger auf eine `float`-Variable zurück und erwartet einen Zeiger auf eine `short`-Eingangsvariable.

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 37

Zeiger auf Zeiger

- Man sollte **komplizierte Konstrukte** in C nur dann einsetzen, wenn sie **unbedingt gebraucht werden**.
- Zu den komplizierten Konstrukten zählen auch
 ➡ **Zeiger auf Zeiger**
 - Zeiger auf Zeiger werde bei **hardwarenahen Anwendungen**, wie Interrupts, gebraucht.
 - Außerdem werden sie bei **2-dimensionalen Feldern** implizit verwendet.
- Andererseits erlauben Zeiger auf Zeiger **flexibleres und eleganteres Programmieren**, z.B. bei
 - ➡ der Fehlerbehandlung komplexer Programme
 - ➡ beim Sortieren von Daten mit `qsort(..)`

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 38

Zeiger auf Zeiger (2)

- In C sind **Zeiger auf Zeiger** relativ leicht zu implementieren, weil der *****-Operator beliebig häufig rekursiv verwendet werden kann:
- Der Ausdruck
 $\Rightarrow$ `char * zeiger`
 definiert einen **Zeiger**, der auf ein **char-Objekt** **verweist**.
- Der Ausdruck
 $\Rightarrow$ `char ** zeiger`
 definiert somit einen **Zeiger auf einen Zeiger**, der auf ein **char-Objekt** **verweist**.

02.06.2004



Zeiger auf Zeiger bei 2-dimensionalen Feldern

- Wir betrachten folgendes **Beispiel eines 2D-Feldes**:
 $\Rightarrow$ `int feld_2d [3] [2] ;`
- Jedes Element von `feld_2d` ist wiederum ein Feld:
`feld_2d [0] = &feld_2d [0] [0] = * feld_2d`
`feld_2d [1] = &feld_2d [1] [0] = *( feld_2d + 1 )`
`feld_2d [2] = &feld_2d [2] [0] = *( feld_2d + 2 )`
- Nun wenden wir den *****-Operator an und erhalten:
`*feld_2d [0] = feld_2d [0] [0] = ** feld_2d`
`*feld_2d [1] = feld_2d [1] [0] = *( *( feld_2d + 1 ) )`
`*feld_2d [2] = feld_2d [2] [0] = *( *( feld_2d + 2 ) )`
- In allgemeiner Schreibweise:
`feld_2d [m] [n] = *( *( feld_2d + m ) + n )`

02.06.2004



Fehlerausgabe mit Zeigern auf Zeiger (1)

Gegeben ist folgender Programmausschnitt mit einem Zeigerfeld auf Strings mit Fehlermeldungen:

```
...
char *Fehler [ ] =
{ "Die Datei existiert nicht!",
  "Zu wenig Speicherplatz vorhanden!",
  "Die angegebene Datei existiert bereits!",
  "Schreibfehler: Ist die Datei schreibgeschützt?" } ;
...
```

Eine Funktion gibt als Fehlermeldung die ganze Zahl **error** des Typs **short** im Bereich von 0 ... 3 zurück, die auf den aufgetretenen Fehler verweist.

Schreiben Sie ein Testprogramm, das in **main (...)** das Zeigerfeld **char *Fehler []** definiert und mit **scanf (...)** Fehlernummern einliest. Das Zeigerfeld und **error** wird an das Unterprogramm **Fehlerausgabe (...)** übergeben und die entsprechende Meldung dort mit **printf (...)** ausgegeben.

02.06.2004



✓ Fehlerausgabe mit Zeigern auf Zeiger (1)

Lösung:

Programmausschnitt mit Fehlermeldungen:

```
char *Fehler [ ] =
{ "Die Datei existiert nicht!",
  "Zu wenig Speicherplatz vorhanden!",
  "Die angegebene Datei existiert bereits!",
  "Schreibfehler: Ist die Datei schreibgeschützt?" } ;
```

3.1 **error** muss den Wert 1 enthalten.

3.2 Feldindexoperation für den Fehler
"Zu wenig Speicherplatz vorhanden!" :

```
#include <stdio.h>
```

```
...
printf (Fehler [error]) ;
```

02.06.2004



✓ Fehlerausgabe mit Zeigern auf Zeiger (2)

```

/* Programm zeigt die Handhabung von Zeigerfeldern und **-Operator */
#include <stdio.h>
void Fehlerausgabe ( unsigned int error, char ** Meldung ) ; /* Prototyp */
int main (void) /* Hauptprogramm */
{
    char *Meldung[ ] = { "",
        "\n Fehler: Ueberschreiten der Feldgrenzen!",
        "\n Fehler: Zu wenig Speicherplatz vorhanden!",
        "\n Fehler beim Oeffnen: Datei ist nicht vorhanden!",
        "\n Unzulaessige Fehlernummer in Funktion 'Fehlerausgabe' aufgetreten!\n\n" };
    int ok = 0;
    short error;
/* Programmtest: */
    do
    {
        printf ("\n\n Fehlernummer (error) (short)? : ");
        ok = scanf ("%hd", &error);
        Fehlerausgabe(error, Meldung) ;
    } while (ok ==1) ;
    return 0 ; } /* Ende Programmtest */

```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 43

✓ Fehlerausgabe mit Zeigern auf Zeiger (3)

/* Programm zeigt die Handhabung von Zeigerfeldern und **-Operator */
/* Funktion Fehlerausgabe */

```

void Fehlerausgabe ( unsigned int Fehlernr, char ** Meldung )
{
    if ( Fehlernr >=4 ) printf ( Meldung [4] ) ;
    else printf ( Meldung [Fehlernr] ) ;
}

```

02.06.2004

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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 44

Zeiger auf Zeiger: *qsort* (..) (1)

- In den 60er Jahren des letzten Jahrhunderts wurde ein schnelles Sortierverfahren entwickelt:
 ➔ **Quicksort** (*qsort* (..))
- *qsort* (..) sortiert ein Feld nach einer vom Anwender vorzugebenden Vergleichsfunktion.
- Es ist in der Bibliothek *stdlib.h* vorhanden.
- Weil die Vergleichsfunktion vom Anwender geschrieben werden muss, ist *qsort* (..) sehr flexibel.
- Bei *qsort* (..) werden u.U. ebenfalls **Zeiger auf Zeiger** benötigt,

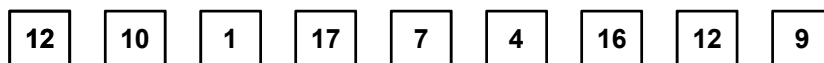
02.06.2004



Funktionsweise von *Quicksort* (1)

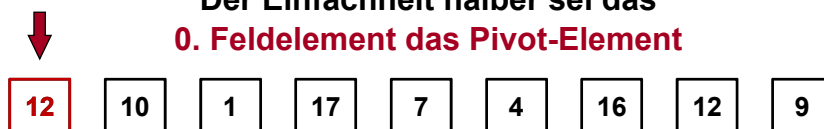
- Der Quicksort-Algorithmus funktioniert nach dem **Prinzip „Teile und Herrsche“**

Beispiel



1. Zunächst wählt man ein **beliebiges Vergleichselement** des zu sortierenden Feldes aus, das
 ➔ **Pivot-Element**

Der Einfachheit halber sei das
0. Feldelement das Pivot-Element

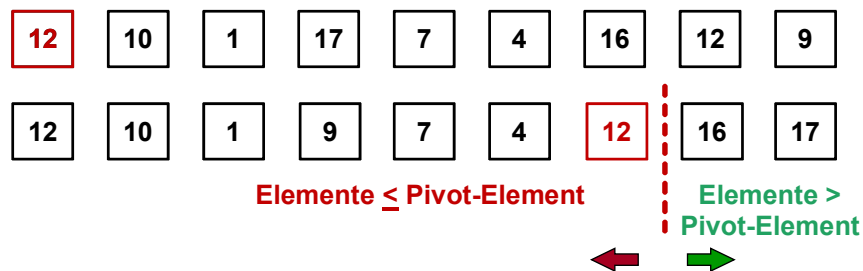


02.06.2004



Funktionsweise von Quicksort (2)

- Man sortiert nun das Feld so um, dass
 - alle Elemente links des Pivot-Elements kleiner gleich dem Pivot Element sind und
 - alle Elemente **rechts des Pivot-Elements größer als** das Pivot-Element sind.



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

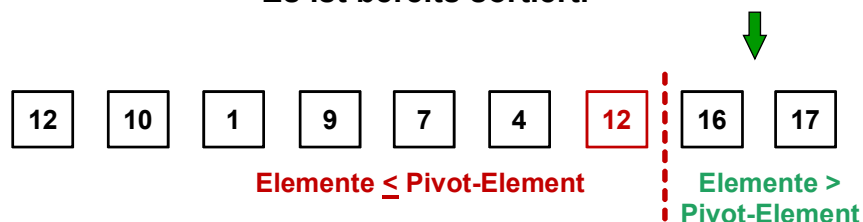


Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 47

02.06.2004

Funktionsweise von Quicksort (3)

- Man hat nun das Feld in 2 Teilfelder zerlegt und kann diese nun durch rekursives Aufrufen von **quicksort(..)** erneut sortieren.
 - In diesem speziellen Fall muss das
 - **rechte Teilfeld** nicht weiter sortiert werden: Es ist bereits sortiert.



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

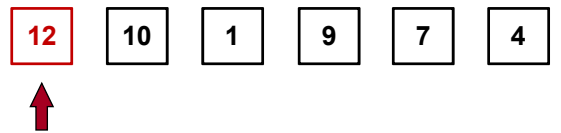


Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 48

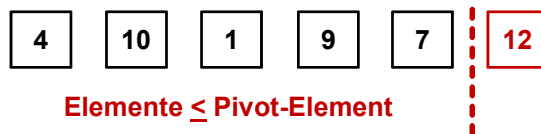
02.06.2004

Funktionsweise von Quicksort (4)

2. Man sortiert nun durch rekursives Aufrufen von *quicksort (..)* das **linke Teilfeld**.



Pivot-Element des linken Teilfeldes



Elemente $\leq$ Pivot-Element

3. Das rechte Teilfeld muss in diesem Spezialfall nicht sortiert werden.

02.06.2004

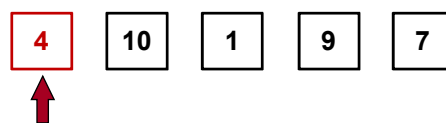
Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



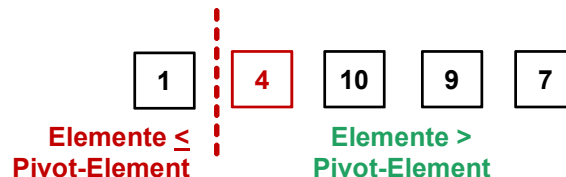
Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 49

Funktionsweise von Quicksort (5)

4. Man sortiert nun durch rekursives Aufrufen von *quicksort (..)* wiederum das **linke Teilfeld**.



Pivot-Element des linken Teilfeldes



**Elemente $\leq$
Pivot-Element**

**Elemente $>$
Pivot-Element**

5. Das 2. linke Teilfeld muss in diesem Spezialfall nicht sortiert werden.

02.06.2004

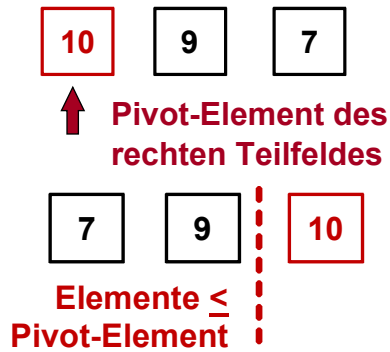
Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 50

Funktionsweise von Quicksort (6)

6. Man sortiert nun durch rekursives Aufrufen von *quicksort(..)* in diesem Schritt das **rechte Teilfeld**.



7. Nach diesem Schritt ist das Feld sortiert.

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

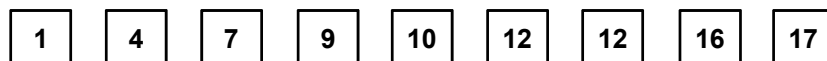


Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 51

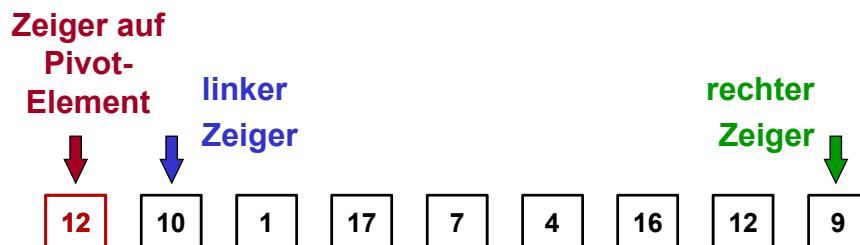
02.06.2004

Funktionsweise von Quicksort (7)

7. Nach diesem Schritt ist das Feld sortiert:



- Viele Quicksort-Algorithmen **arbeiten mit 3 Zeigern**:



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

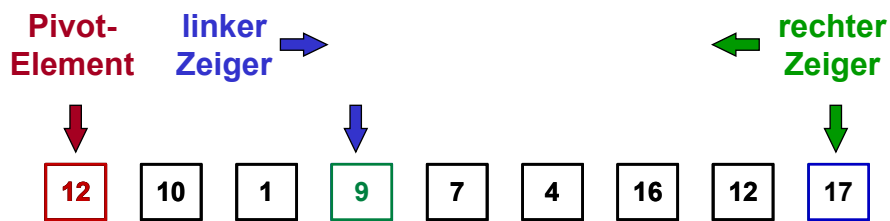


Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 52

02.06.2004

Beispiel einer programmtechnischen Realisierung von *Quicksort* (1)

- Der **linke Zeiger** läuft nach rechts bis er ein Element findet, das **größer ist als das Pivot-Element**.
- Der **rechte Zeiger** läuft nach links bis er ein Element findet, das **kleiner gleich dem Pivot-Element** ist.
- Die beiden Elemente werden **vertauscht**:



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

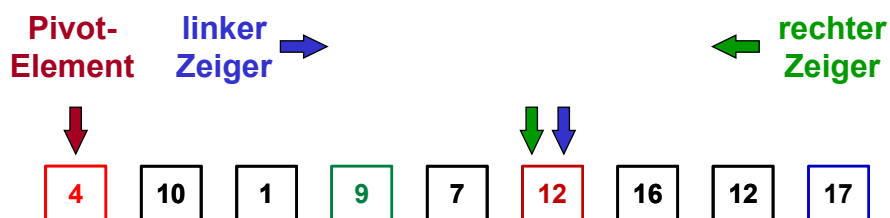


Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 53

02.06.2004

Beispiel einer programmtechnischen Realisierung von *Quicksort* (2)

- Beide Zeiger werden nach innen weiterbewegt bis sie sich kreuzen.
- Danach wird das **Kreuzungselement mit dem Pivot-Element vertauscht**.



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



Programmieren in ANSI C:
K14_erweiterte Zeigerfunktionen.ppt, Seite 54

02.06.2004

Sortieren mit *qsort* (..) (1)

- In der Bibliothek *stdlib.h* befindet sich die Sortierfunktion
 ➡ **Quicksort** (*qsort* (..))
- Prototypendeklaration von *qsort* (..) :

```
void qsort ( void *base, size_t num, size_t width,
int (__cdecl *compare )
(const void *elem1, const void *elem2 ) );
```
- size_t* : vergleiche Kapitel *Speicherverwaltung*
- *compare (const void *elem1, const void *elem2)*
 ist ein **Zeiger auf eine vom Anwender zu schreibende Funktion**, die 2 Elemente des zu sortierenden Feldes vergleicht und einen Integerwert zurückgibt.

02.06.2004



Sortieren mit *qsort* (..) (2)

- int (__cdecl *compare) (const void *elem1, const void *elem2)*
 ist ein Zeiger auf eine vom Anwender geschriebene Funktion, die 2 Elemente *elem1* und *elem2* vergleicht und einen Integerwert zurückgibt:

Rückgabewert	Bedeutung
<0	<i>elem1</i> < <i>elem2</i>
=0	<i>elem1</i> = <i>elem2</i>
>0	<i>elem1</i> > <i>elem2</i>

__cdecl ist ein Microsoft-spezifischer Parameter, den wir nicht beachten müssen.

Mit der *__cdecl* –Option wird **bei jeder Compilereinstellung** nach jedem Funktionsaufruf der Stack aufgeräumt.

02.06.2004



Sortieren mit *qsort* (..) (3)

- Prototypendeklaration von *qsort* (..):

```
void qsort ( void *base, size_t num, size_t width,
int (__cdecl *compare )
(const void *elem1, const void *elem2 ) );
```
- *void *base* **Zeiger auf die Startadresse** der zu sortierenden Elemente, die in einem Feld abgelegt sind.
- *size_t num* **Anzahl der Elemente** im zu sortierenden Feld (Datentyp *size_t*)
- *size_t width* **Wortlänge der Elemente** im zu sortierenden Feld in Bytes (Datentyp *size_t*)

02.06.2004



Sortieren mit *qsort* (..) (4)

- Prototypendeklaration von *qsort* (..):

```
void qsort ( void *base, size_t num, size_t width,
int (__cdecl *compare )
(const void *elem1, const void *elem2 ) );
```
- *const void *elem1* **Zeiger auf das Vergleichselement**
- *const void *elem2* **Zeiger auf das zu vergleichende Element im Feld**
- Das Schlüsselwort *const* in *const void *elem1* bzw. *const void *elem2* bezieht sich auf die Zeiger, die von der Funktion *compare* (..) nicht verbogen werden dürfen!

02.06.2004



Sortieren mit *qsort* (..) (1)

Das folgende Programm liest die Kommandozeilenparameter und benutzt die Bibliotheksfunktion *qsort* (..), um die Parameter zu sortieren. Anschließend werden die sortierten Parameter ausgegeben.

Analysieren und testen Sie das Programm!
Machen Sie sich z.B. mit dem *Hilfe*-Menü von *MSVC++* mit der Bibliotheksfunktion *_stricmp* (..) vertraut!

Erläutern Sie die Anweisung
*return _stricmp(* (char**) arg1, * (char**) arg2) ;*
in der Funktion *compare* (..) !

02.06.2004



Sortieren mit *qsort* (..) (2)

```
#include <stdlib.h> /* Fuer qsort (..) */
#include <string.h> /* Fuer _stricmp (..) */
#include <stdio.h> /* Fuer printf (..) */

int compare ( const void *arg1, const void *arg2 ) ;
/* Prototypendeklaration der vom Anwender zu schreibenden
   Funktion compare (..) */

/* ... */
```

02.06.2004





Sortieren mit *qsort* (..) (3)

```
void main ( int argc, char **argv )
{
    int i ;

    /* Loesche argv[0] vor dem Sortieren */
    argv++ ;
    argc-- ;

    /* Sortiere die uebrigen Parameter mit dem Quicksort-
       Algorithmus qsort (..) : */
    qsort ( (void *) argv, (size_t) argc, sizeof ( char * ), compare ) ;

    /* Ausgabe der sortierten Liste: */
    for( i = 0; i < argc; ++i )
        printf ( "%s\n ", argv [i] ) ;
    printf ( "\n" ) ;
}
```

02.06.2004



Sortieren mit *qsort* (..) (4)

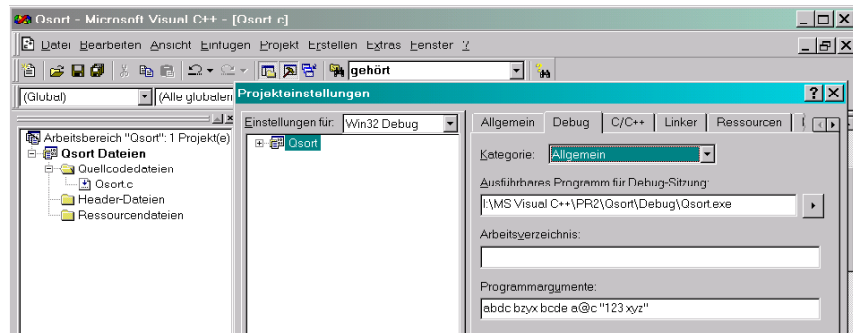
```
int compare ( const void *arg1, const void *arg2 )
{
    /* Vergleich der beiden Strings: */
    return _stricmp ( * ( char** ) arg1, * ( char** ) arg2 ) ;
}
```

02.06.2004



✎ Sortieren mit *qsort* (..) (5)

Das Programm wird mit folgenden
Kommandozeilenparametern aufgerufen:
abdc bzyx bcde a@c "123 xyz"



✎ Sortieren mit *qsort* (..) (6)

Das Programm wird mit folgenden
Kommandozeilenparametern aufgerufen:
abdc bzyx bcde a@c "123 xyz"

