

Strukturen (*structures*) und Parameterübergabe in C

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 1

Strukturen in C

- In diesem Kapitel wollen wir zeigen:
 - ➔ wie man die *typedef* - Anweisung verwendet
 - ➔ was Strukturen (*structures*) sind
 - ➔ wie man Strukturen vereinbart
 - ➔ welche Operatoren für Strukturen existieren
 - ➔ wie man Strukturen verwendet
 - ➔ welche Arten der Parameterübergabe an Funktionen möglich sind.
 - ➔ welche Vor- und Nachteile die verschiedenen Arten der Parameterübergabe haben.

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 2

Die *typedef*-Anweisung (1)

- Man verwendet die **→ typedef** – Anweisung, um **neue Namen für Datentypen** zu vereinbaren.

Beispiel: `typedef` `unsigned long` `ULONG;`

- Dabei wird dem **bekannten** Datentyp ➡ **unsigned long** der **neu definierte** Name ➡ **ULONG** zugeordnet.
- Die *typedef* – Anweisung kann **keine neuen Datentypen** konstruieren.
 - Die *typedef* - Anweisung wird gern verwendet, um **Datenformate** einfach **an unterschiedliche Prozessoren** anpassen zu können.

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 3

2705 2004

Die *typedef*-Anweisung (2)

- Im folgenden Beispiel wird der Typ **FACHHOCHSCHULE** vereinbart.
- Man beachte die Unterschiede bei der **Groß- und Kleinschreibung** von **FACHHOCHSCHULE** bzw. **Fachhochschule**:

Beispiel:

```
typedef char STUDIENRICHTUNG [5] ;
typedef STUDIENRICHTUNG STUDIENGANG [10] ;
typedef STUDIENGANG FACHBEREICH [30] ;
typedef FACHBEREICH FACHHOCHSCHULE [50] ;
FACHHOCHSCHULE Fachhochschule ;
char Etechnik = Fachhochschule [20] [12] [2] [4] ;
```

07052004

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



Programmieren in ANSI C:
K11 Strukturen.ppt, Seite 4

Die typedef-Anweisung (3)

Beispiel:

```
typedef char          STUDIENRICHTUNG [5] ;
typedef STUDIENRICHTUNG STUDIENGANG [10] ;
typedef STUDIENGANG  FACHBEREICH [30] ;
typedef FACHBEREICH  FACHHOCHSCHULE [50] ;

FACHHOCHSCHULE      Fachhochschule ;
char Etechnik = Fachhochschule [20] [12] [2] [4] ;
```

- Im Beispiel wird der **char**-Variablen **Etechnik** die **Studienrichtung** 4 des **Studiengangs** 2 des **Fachbereichs** 12 der **Fachhochschule** 20 zugeordnet:
- Alternative Definition von **FACHHOCHSCHULE**:

```
typedef char FACHHOCHSCHULE [50] [30] [10] [5] ;
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 5

27/05.2004

Strukturen (1)

- Im Unterschied zu Feldern (*arrays*) enthält eine **Struktur** Komponenten (*members, fields, elements*) aus in der Regel **unterschiedlichen Datentypen**.
- Die unterschiedlichen Datentypen werden unter dem **gemeinsamen Strukturnamen** zusammengefasst.

Beispiel:

```
struct BIBLIOTHEK {          /* Etikett der Struktur bibliothek */
    char Autor_Nachname [40] ; /* Nachname des Autors */
    char Autor_Vorname [20] ;  /* Vorname des Autors */
    char Buchtitel [100] ;     /* Buchtitel */
    unsigned short Datum [3] ; /* Veröffentlichungsdatum */
    unsigned char Auflage ;    /* Auflage */
    float Preis ;              /* Preis des Buches */
} Buch ;                      /* Definition der Strukturvariablen Buch */
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 6

27/05.2004

Strukturen (2)

Beispiel:



```
struct BIBLIOTHEK {           /* Etikett der Struktur bibliothek */
char Autor_Nachname [40] ; /* Nachname des Autors */
char Autor_Vorname [20] ; /* Vorname des Autors */
char Buchtitel [100] ;      /* Buchtitel */
unsigned short Datum [3] ; /* Veröffentlichungsdatum */
unsigned char Auflage ;     /* Auflage */
float Preis ;               /* Preis des Buches */
} Buch ;
```

- Die Struktur **BIBLIOTHEK** enthält Daten von Veröffentlichungen.

- **BIBLIOTHEK** ist der

➡ **Bezeichner** (Etikett = *tag*) **der Struktur**.

- **BIBLIOTHEK** ist kein Variablenname!

- Der einzige **Variablenname** ist ➡ **Buch**.

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 7

Zugriff auf Felder und Strukturen

- Im Folgenden verwenden wir für einen Bestandteil eines **Feldes** den Begriff **Element** und für den einer **Struktur** den Begriff **Komponente**.
- Man greift bei **Feldern** (*arrays*) auf die Elemente (*elements*) mit Hilfe von ➡ **Indizes** zu:

Beispiel: **Feldelementzugriff mit Index**

```
... short j, i [ ] = {0, 10, 20} ; /* Vereinbarung der Variablen j, i [ ] */
j = i [1] ;                       /* j = i [1] = 10 zuweisen */
...
```



- Dagegen greift man bei **Strukturen** (*structures*) auf die Komponenten (*members*) mit den **Namen der Komponenten** zu.

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 8

Zugriff auf Strukturen (1)

Beispiel:

```
struct LITERATUR {
    char Autor_Nachname [40]; /* Feld: Nachname des Autors */
    char Autor_Vorname [20]; /* Feld: Vorname des Autors */
    char Titel [100];         /* Feld: Titel */
    unsigned short Datum [3]; /* Feld: Veröffentlichungsdatum */
    unsigned char Auflage;    /* char: Auflage (0 ...255) */
};
```

- Man greift auf die **Komponenten** von Strukturen mit Hilfe **des Operators** ➡ **.** mit den ➡ **Komponentennamen** zu:
➡ **Variablenname. Komponentenname**

Beispiele für Strukturkomponentenzugriffe:

```
struct LITERATUR roman, sachbuch;
...
➡ sachbuch.Titel = "Programmieren in C";
sachbuch.Datum = { 30, 6, 1997 };
roman.Datum [2] = sachbuch.Datum [2];
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 9

27.05.2004

Strukturen als Komponenten

- Strukturen können als **Komponenten** ebenfalls ➡ **Strukturen** enthalten.

Beispiel:

```
struct DATUM
{
    unsigned int tag;
    char *monat;
    unsigned int jahr;
};
/* ... */

struct LIT {
    char *Autor_Nachname; /* alternative Feldvereinbarung */
    char *Autor_Vorname; /* alternative Feldvereinbarung */
    char Titel [100];     /* Titel als Feldvereinbarung */
    ➡ struct DATUM Pub_Datum; /* Struktur DATUM
                               in Struktur LITERATUR */
    unsigned char Auflage; /* Auflage */
};
/* ... */
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 10

27.05.2004

Strukturfelder (1)

➔ **Strukturfelder** sind ebenfalls zulässig.

Beispiel:

```
struct DATUM
{
    unsigned int tag ;
    char *monat ;
    unsigned int jahr ;
};
/* ... */
struct LIT
{
    char *Autor_Nachname ;      /* Feld fuer Nachnamen-string */
    char *Autor_Vorname ;      /* Feld fuer Vornamen-string */
    char *Titel ;              /* Feld fuer Titel-string */
    struct DATUM Pub_Dat [10] ; /* Strukturfeld in Struktur */
    unsigned char Auflage[10] ; /* Auflage */
};
/* ... */
int main (void)
{
    struct LIT books [100] ;    /* Feld von Strukturen */
    /* ... */
}
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 11

Strukturfelder (2)

Beispiel:

```
struct DATUM
{
    unsigned int tag ;
    char *monat ;
    unsigned int jahr ;
};
struct LIT
{
    char *Nachname ;
    char *Vorname ;
    char *Titel ;
    struct DATUM Pub_Dat [10] ;
    unsigned char Auflage [10] ;
};
/* ... */
int main ( void )
{
    struct LIT books [100] ;
    /* Fortsetzung des Beispiels: */
    books [0].Nachname = "Gans";
    books [0].Vorname = "Gustav";
    books [0].Titel = "Glück mit C";
    books [0].Pub_Dat [0].jahr = 1997;
    books [0].Pub_Dat [0].monat= "Mai";
    books [0].Pub_Dat[0].tag = 30;
    books [0].Auflage[0] = 1;
    books [0].Pub_Dat [1].jahr = 2002;
    books [0].Pub_Dat [1].monat="Juli";
    books [0].Pub_Dat [1].tag = 12;
    books [0].Auflage[1] = 2;
    /* ... */
    books [1].Nachname = "Duck";
    books [1].Vorname = "Donald";
    books [1].Titel = "Pech mit C";
    books [1].Pub_Dat [0].jahr = 1994;
    /* ... */
    return 0 ; }
/* ... */
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 12

Unterschiedliche Strukturvereinbarungen (1)

- Strukturen können als
 - ➔ **namenlose Strukturen** oder
 - ➔ **namensbehaftet** mit Hilfe von **typedef** vereinbart werden.

Beispiel für **namenlose** Struktur:

```
struct      /* Strukturen Startdatum, Enddatum vereinbaren */
{ unsigned int Tag, Jahr;
  char *Monat ;
} Anfangsdatum, Enddatum;
```

Beispiel für **namensbehaftete** Struktur:

```
typedef struct /* Struktur des Typs DATUM vereinbaren */
{ unsigned int Tag, Jahr;
  char *Monat ;
} DATUM;
```

DATUM Startdatum, Enddatum; /* struct ist hier verboten! */

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 13

27.05.2004

Unterschiedliche Strukturvereinbarungen (2)

- In **namensbehafteten Strukturen** können
im Unterschied zu **namenlosen** Strukturen
 - ➔ **weitere Variable** vereinbart werden.

Beispiel:

```
typedef struct      /* Struktur des Typs DATUM vereinbaren */
{ unsigned int Tag, Jahr;
  char *Monat ;
} DATUM ;
```

DATUM Startdatum, Enddatum ; /* Das Schlüsselwort struct ist hier verboten! */

...

➔ **DATUM Geburtsdatum, Diplomdatum ;**

- **Definieren Sie Strukturen ausschließlich mit typedef!**

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 14

27.05.2004

Initialisierung von Strukturen

- Strukturen und Strukturfelder können ähnlich initialisiert werden wie Felder.

Beispiele:

```
typedef struct          /* Struktur des Typs DATUM vereinbaren */
{ unsigned int Tag ;
  char *Monat ;
  unsigned int Jahr ;
} DATUM ;

DATUM Geburtsdatum = { 1, "Juli", 1949 } ;

DATUM Familiengeburtstage [4] = { { 1, "Juli", 1949 },
                                   { 1, "August", 1949 },
                                   { 26, "April", 1981 },
                                   { 24, "Mai", 1984 } } ;
```

27/05/2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 15

Strukturen als Funktionsparameter

- Übergibt man eine **Struktur** an eine Funktion, so wird die **gesamte Struktur kopiert**.
- Bei **Feldern** wird dagegen **nur die Anfangsadresse** übergeben und **kopiert**.

Beispiel für **direkte Übergabe** einer Struktur:

```
typedef struct { float r, i ; } KOMPLEX ;
/* Strukturtyp für komplexe Zahl KOMPLEX definieren */

KOMPLEX quadrat ( KOMPLEX x )      /* komplexe Funktion quadrat
{                                   definieren */
  KOMPLEX square ;
  square.r = x.r * x.r - x.i * x.i ; /* Realteil von square */
  square.i = 2 * x.r * x.i ;         /* Imaginärteil von square */
  return square ;                   /* Rückgabe der Quadratzahl */
}
```

27/05/2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 16

Zeiger auf Strukturen (1)

- Übergibt man eine Struktur an eine Funktion, so wird die **gesamte Struktur kopiert**.
- Die Kopie einer Struktur mit vielen Parametern benötigt eine lange Rechenzeit und viel Speicherplatz.
- Deshalb übergibt man häufig stattdessen
➡ **Zeiger auf Strukturen**

Beispiel für **Zeigerübergabe** einer Struktur:

```
typedef struct { float r, i ; } KOMPLEX ; /* KOMPLEX definieren */
                                     ↓           ↓
void p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y )
{
    /* komplexe Funktion p_quadrat definieren */
    (*p_y).r = (*p_x).r * (*p_x).r - (*p_x).i * (*p_x).i ; /* Realteil berechnen */
    (*p_y).i = 2 * (*p_x).r * (*p_x).i ; /* Imaginärteil berechnen */
}
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 17

Zeiger auf Strukturen (2)

Beispiel:

```
typedef struct { float r, i ; } KOMPLEX ; /* KOMPLEX definieren */
                                     ↓           ↓
void p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y )
{
    /* komplexe Funktion p_quadrat definieren */
    (*p_y).r = (*p_x).r * (*p_x).r - (*p_x).i * (*p_x).i ; /* Realteil berechnen */
    (*p_y).i = 2 * (*p_x).r * (*p_x).i ; /* Imaginärteil berechnen */
}
```

- Im Beispiel werden die **Anfangsadressen** ➡ **p_x** und **p_y** der Strukturen übergeben.

Vorteile:

- ➡ Es werden nur die Anfangsadressen und **nicht alle** Parameter kopiert.
- ➡ Die Funktion hat **Zugriff auf die Originale** des aufrufenden Programms.

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 18

Zeiger auf Strukturen (3)

Beispiel:

```
typedef struct { float r, i ; } KOMPLEX ; /* KOMPLEX definieren */

void p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y )
{
    /* komplexe Funktion p_quadrat definieren */
    (*p_y).r = (*p_x).r * (*p_x).r - (*p_x).i * (*p_x).i ; /* Realteil berechnen */
    (*p_y).i = 2 * (*p_x).r * (*p_x).i ; /* Imaginärteil berechnen */
}
```

- Die Schreibweise (z.B. $\Rightarrow$ **`(*p_y).r`**) ist wegen der **Vorrangregeln der Operatoren** notwendig.
- Deshalb erfordert das Programm **relativ viel Schreibaufwand** und ist **unübersichtlich**.

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 19

Der Operator -> (1)

- Wegen der schnelleren Befehlsausführung werden in Funktionen häufig statt Strukturen **Zeiger auf Strukturen** übergeben.
- Man kann auf die Komponenten einer Struktur mit Hilfe eines Zeigers mit **weniger Schreibaufwand** und **übersichtlicher** mit dem **Operator $\Rightarrow$ ->** zugreifen.
- Der Ausdruck **`a->b`** ist eine **Kurzschreibweise** für den Ausdruck **`(*a).b`**.

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 20

Der Operator -> (2)

- Man kann auf die Komponenten einer Struktur mit Hilfe eines Zeigers mit **weniger Schreibaufwand** und **übersichtlicher** mit dem **Operator** ➡ ➡ **->** zugreifen.
- Der Ausdruck **a->b** ist die **Kurzschreibweise** des Ausdrucks **(*a).b**.

Beispiel:

```
typedef struct { float r, i ; } KOMPLEX ; /* KOMPLEX definieren */
void p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y )
{
    p_y->r = p_x->r * p_x->r - p_x->i * p_x->i ; /* Realteil berechnen */
    p_y->i = 2 * p_x->r * p_x->i ; /* Imaginärteil berechnen */
}
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 21

27.05.2004

Parameterübergabe bei Funktionen

- Bei der Parameterübergabe an Funktionen unterscheidet man
 - ➡ **Namensaufruf** (*call by name*)
 - ➡ **Wertaufruf** (*call by value*)
 - Beim **Namensaufruf** kann das **Original des Parameters verändert werden** beim **Wertaufruf nicht**.
- Die Begriffe Namensaufruf und Wertaufruf sind in der Literatur für viele Programmiersprachen weit verbreitet.
 - Für C treffendere Begriffe sind
 - ➡ **Adressübergabe** statt Namensaufruf
 - ➡ **Wertübergabe** statt Wertaufruf.

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 22

27.05.2004

Wert- und Adressübergabe: einfache Variable

➔ **Wertübergabe:**
Original kann nicht
verändert werden

Beispiel:

```
void func (int i)
{ ...
  ... = i ;
  i = ... ;
}
```

Beispiele zum Aufruf:

```
int j ;
func (7) ;
func (j) ;
```

➔ **Adressübergabe:**
Original kann
verändert werden

Beispiel:

```
void func (int *p_i)
{ ...
  ... = *p_i ;
  *p_i = ... ;
}
```

Beispiele zum Aufruf:

```
int j, *p = &j ;
func (&j) ;
func (p) ;
```



Wert- und Adressübergabe: Feld

- Eine Wertübergabe ist bei Feldern **nicht** möglich.

➔ **Adressübergabe:** Original kann verändert werden

Beispiel 1:

```
void func ( int f [ ] )
{ ...
  ... = f [0] ;
  f [3] = ... ;
}
```

Beispiel 2:

```
void func ( int *p_f )
{ ...
  ... = *p_f ;
  *(p_f + 3) = ... ;
}
```

Beispiele zum Aufruf:

```
int feld [5], *p_feld = feld ;
...
func ( &feld [0] ) ;
func ( p_feld ) ;
```



Wert- und Adressübergabe: Struktur	
➔ Wertübergabe: Original kann <u>nicht</u> verändert werden Beispiel: <pre>typedef struct { int tag, monat, jahr; } DAT ; void func (DAT d) { = d.tag ; d.monat = ... ; }</pre> Beispiel zum Aufruf: <pre>DAT datum ; func (datum) ;</pre>	➔ Adressübergabe: Original kann verändert werden Beispiel: <pre>typedef struct { int tag, monat, jahr; } DAT ; void func (DAT *p) { = (*p).tag ; (*p).monat = ... ; }</pre> Beispiele zum Aufruf: <pre>DAT datum, *p_d = &datum ; func (&datum) ; func (p_d) ;</pre>
Hochschule für angewandte Wissenschaften Hamburg Prof. Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagard Programmieren in ANSI C: K11_Strukturen.ppt, Seite 25	

Aufgabe: Zeiger auf Strukturen (1)

Untersuchen Sie das folgende Beispiel, welches als Funktionsparameter Zeiger auf Strukturen verwendet:

```
typedef struct { float r, i ; } KOMPLEX ; /* KOMPLEX definieren */
void p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y )
{
    /* komplexe Funktion p_quadrat definieren */
    (*p_y).r = (*p_x).r * (*p_x).r - (*p_x).i * (*p_x).i ; /* Realteil berechnen */
    (*p_y).i = 2 * (*p_x).r * (*p_x).i ; /* Imaginärteil berechnen */
}
```

1. Erläutern Sie in den Anweisungen

```
void KOMPLEX p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y ) ;
```

und

```
(*p_y).i = 2 * (*p_x).r * (*p_x).i ; /* Imaginärteil berechnen */
```

für jeden * - Operator dessen Funktion!

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

Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 26



Aufgabe: Zeiger auf Strukturen (2)

Untersuchen Sie das folgende Beispiel, welches als Funktionsparameter Zeiger auf Strukturen verwendet:

```
typedef struct { float r, i ; } KOMPLEX ; /* KOMPLEX definieren */
void p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y )
{
    /* komplexe Funktion p_quadrat definieren */
    (*p_y).r = (*p_x).r * (*p_x).r - (*p_x).i * (*p_x).i ; /* Realteil berechnen */
    (*p_y).i = 2 * (*p_x).r * (*p_x).i ; } /* Imaginärteil berechnen */
```

- Überzeugen Sie sich davon, dass die angegebene Funktion mathematisch richtig ist, indem Sie die Parameter x und y in gewohnter komplexer Schreibweise schreiben und das Ergebnis für RE(y) und IM(y) ausrechnen:

$$x = \text{RE}(x) + j \text{IM}(x) \quad y = \text{RE}(y) + j \text{IM}(y)$$

$$(j = \text{imaginäre Einheit} = (-1)^{1/2})$$
- Begründen Sie, warum bei den Ausdrücken des Typs **(*a).b** die Klammern notwendig sind!
- Auf welche Parametertypen würde versucht zuzugreifen, wenn man die Klammern weggelassen hätte?

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 27

27.05.2004



Aufgabe: Zeiger auf Strukturen (1)

1. Operator *:

```
void p_quadrat ( KOMPLEX *p_x, KOMPLEX *p_y )
(*p_y).i = 2 * (*p_x).r * (*p_x).i ; } /* Imaginärteil berechnen */
```



- Definition eines Zeigers auf die Struktur **KOMPLEX**
- Objekt-Operator des Zeigers auf eine Struktur
- Multiplikation
- Teil des Kommentar-Operators **/*** bzw. ***/**

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 28

27.05.2004

✓ Aufgabe: Zeiger auf Strukturen (2)

2. Mathematische Richtigkeit:

$$\begin{aligned}
 x &= \text{RE}(x) + j \text{IM}(x) & y &= \text{RE}(y) + j \text{IM}(y) \\
 y &= x^2 \\
 \text{RE}(y) + j \text{IM}(y) &= (\text{RE}(x) + j \text{IM}(x))^2 \\
 &= (\text{RE}(x) + j \text{IM}(x)) * (\text{RE}(x) + j \text{IM}(x)) \\
 &= \text{RE}(x) * \text{RE}(x) + j (\text{RE}(x) * \text{IM}(x)) + j (\text{IM}(x) * \text{RE}(x)) + \\
 &\quad + j^2 (\text{IM}(x) * \text{IM}(x)) \\
 &= \text{RE}(x) * \text{RE}(x) + 2j (\text{RE}(x) * \text{IM}(x)) - \text{IM}(x) * \text{IM}(x)
 \end{aligned}$$

Koeffizientenvergleich:

$$\text{RE}(y) = \text{RE}(x) * \text{RE}(x) - \text{IM}(x) * \text{IM}(x)$$

$$\text{IM}(y) = 2 (\text{RE}(x) * \text{IM}(x))$$

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 29

✓ Aufgabe: Zeiger auf Strukturen (3)

3. Klammern bei Ausdrücken des Typs (*a).b:

Die Klammern werden benötigt, weil der Punkt-Operator '.' eine höhere Priorität als der Verweisoperator '*' hat.

Hier muss zunächst auf den Inhalt des Zeigers auf die Struktur zugegriffen werden und erst danach auf die Komponente.

4. Keine Klammern bei Ausdrücke des Typs *a.b:

Der Ausdruck ohne Klammern hätte dieselbe Bedeutung wie *(a.b).
b ist aber kein Zeiger, sondern eine Komponente der Struktur **a**.

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 30



Aufgabe: Strukturen (1)

1. Ist der folgende Programmausschnitt richtig? – Wurden Fehler gemacht? Wenn ja, welche? Korrigieren Sie die Fehler!

```
struct { short a, long b, char *c ; } STRUKTUR str ;

{ ...
  b = a;
  ... }
```

2. Ist der folgende Programmausschnitt richtig? – Wurden Fehler gemacht? Wenn ja, welche? Begründung!

```
typedef struct { short a ; float b ; } STRUKTUR ;
struct STRUKTUR s, *z_s;

int main (void)
{ ...
  *z_s = &s;
  z_s.b = z_s.a;
  ... }
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 31



Aufgabe: Strukturen (1) (1)

Zu 1:

```
struct { short a, long b, char *c ; } STRUKTUR str ;

{ ...
  b = a;
  ... }
```

➡ Die **Deklaration der Parameter** bei der Strukturdefinition ist falsch. Richtig wäre:

```
struct STRUKTUR { short a ; long b ; char *c ; } str ;
```

➡ Die **Reihenfolge der Parameter** bei der Strukturdefinition ist falsch. Richtig wäre:

```
struct STRUKTUR { short a ; long b ; char *c ; } str ;
```

➡ Der **Zugriff auf die Strukturkomponenten** ist falsch. Richtig wäre:

```
str.b = str.a;
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 32

✓ Aufgabe: Strukturen (1) (2)

Zu 2:

```
typedef struct { short a ; float b ; } STRUKTUR ;
➡ struct STRUKTUR s, *z_s ;

int main (void)
{ ...
  ➡ *z_s = &s ;
  ➡ z_s.b = z_s.a ;
  ... }
```

➡ struct ist hier falsch. Richtig wäre: **STRUKTUR s, *z_s ;**

➡ Der **Verweisoperator** ist falsch. Richtig wäre:
z_s = &s ;

➡ Der **Zugriff auf die Strukturkomponenten** ist falsch.
 Richtig wäre:
z_s->b = z_s->a ;
 oder:
(*z_s).b = (*z_s).a ;

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



Programmieren in ANSI C:
 K11_Strukturen.ppt, Seite 33

27.05.2004



Aufgabe: Strukturen (2)

Schreiben Sie eine Variante der Funktion

void p_quadrat (KOMPLEX *p_x, KOMPLEX *p_y)

mit dem Namen **quadrat_p (...)**:

Die Funktion **quadrat_p (...)** soll ebenfalls eine komplexe Zahl quadrieren und der Eingangsparameter soll ebenfalls ein Zeiger auf eine Struktur des Typs **KOMPLEX** sein, welche die zu quadrierende Eingangszahl enthält.

Die Funktion **quadrat_p (...)** soll aber im Unterschied zu **p_quadrat** einen Zeiger auf eine Struktur des Typs **KOMPLEX** zurückgeben, die das Quadrat der komplexen Eingangszahl enthält!

Beachten Sie bitte unbedingt, dass Zeiger auf lokale Strukturen des Unterprogramms nur im Unterprogramm und nicht im Hauptprogramm gültig sind!

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



Programmieren in ANSI C:
 K11_Strukturen.ppt, Seite 34

27.05.2004

✓ Aufgabe: Strukturen (2) (1)

Definitionen von Strukturen und Funktionen:

```
#include <stdio.h>    /* Einbinden der Bibliothek stdio.h */
#include <stdlib.h>    /* Einbinden der Bibliothek conio.h */

typedef struct { float r, i ;} KOMPLEX ;
                        /* Strukturtyp KOMPLEX definieren */
KOMPLEX square, *p_square; /* komplexe Zahl global definieren */
KOMPLEX *quadrat_p ( KOMPLEX *p_x ) /* Funktion quadrat_p
    definieren, die einen Zeiger auf die Struktur KOMPLEX zurückgibt */
{
    square.r = p_x->r * p_x->r - p_x->i * p_x->i ;    /* Realteil von square */
    square.i = 2 * p_x->r * p_x->i ;                /* Imaginärteil von square */
    p_square = &square ;
    return p_square ;    /* Rückgabe des Zeigers auf die Quadratzahl */
}
/* Man beachte: p_square muss global definiert werden. Würde man
p_square lokal in der Funktion quadrat_p definieren, so würde der
Rückgabewert im Hauptprogramm auf eine Struktur zeigen, die nach
dem Rücksprung aus quadrat_p nicht mehr existiert! */
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 35

✓ Aufgabe: Strukturen (2) (2)

Hauptprogramm :

```
int main( void )    /* Hauptprogramm definieren */
{
    KOMPLEX a, *p_a = &a, Ergebnis ;
                        /* Strukturvariablen des Typs KOMPLEX definieren */
    fflush (stdin) ;
    system ("cls") ;    /* Bildschirm löschen */
    printf (" Realteil, Imaginärteil der zu quadrierenden Zahl: a.r, a.i ? ");
    scanf ("%f,%f", &a.r, &a.i) ;    /* Eingangsvariable einlesen */
    Ergebnis = *quadrat_p (p_a) ;
                        /* Ergebnis (Quadratzahl) zuweisen */
    printf (" Realteil(a) = %f, Imaginärteil (a) = %f\n"
        " Realteil (a^2) = %f, Imaginärteil (a^2) = %f",
        a.r,a.i, Ergebnis.r, Ergebnis.i) ;    /* Ergebnisse ausdrucken */
    return 0 ;    /* Rücksprung ins Betriebssystem */
}
```

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 36



Aufgabe: Strukturen (3) (1)

1. Definieren Sie die Strukturen *RECT* und *POLAR* mit folgenden Eigenschaften:
RECT enthält den Realteil *real* und den Imaginärteil *imag* einer komplexen *float*-Zahl (kartesische Koordinaten).
POLAR enthält den Betrag *abs* und die Phase *phase* (in Bogenmaß: $-\pi < \textit{phase} < +\pi$) einer komplexen *float*-Zahl (Polarkoordinaten).
2. Schreiben Sie die Funktion *ptor*, welche die Struktur *POLAR* in die Struktur *RECT* umwandelt und *RECT* zurückgibt! Verwenden Sie dazu gegebenenfalls geeignete Funktionen der Bibliothek *math.h*! Verwenden Sie keine Zeiger!
3. Schreiben Sie die Funktion *rtop*, welche die Struktur *RECT* in die Struktur *POLAR* umwandelt und *POLAR* zurückgibt! Verwenden Sie dazu gegebenenfalls geeignete Funktionen der Bibliothek *math.h*! Verwenden Sie keine Zeiger!

27.05.2004

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



Programmieren in ANSI C:
 K11_Strukturen.ppt, Seite 37



Aufgabe: Strukturen (3) (2)

4. Schreiben Sie die Funktion *rtop1*, welche die Struktur *RECT* in die Struktur *POLAR* umwandelt! Verwenden Sie dazu gegebenenfalls geeignete Funktionen der Bibliothek *math.h*! Verwenden Sie Zeiger!
5. Schreiben Sie die Funktion *ptor1*, welche die Struktur *POLAR* in die Struktur *RECT* umwandelt! Verwenden Sie dazu gegebenenfalls geeignete Funktionen der Bibliothek *math.h*! Verwenden Sie Zeiger!

27.05.2004

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



Programmieren in ANSI C:
 K11_Strukturen.ppt, Seite 38

✓ Aufgabe: Strukturen (3) (1)

1. Strukturdefinitionen:

```
typedef struct { float real, imag ; } RECT ;
/* Strukturtyp RECT definieren:
komplexe Zahl in kartesischen Koordinaten:
Komponenten vom Typ float: Realteil real,
Imaginärteil imag */

typedef struct { float abs, phase ; char error; } POLAR ;
/* Strukturtyp POLAR definieren:
komplexe Zahl in Polarkoordinaten:
Komponenten vom Typ float:
Betrag abs, Phase phase,
Fehlervariable error != 0: Fehler,
error = 0: kein Fehler */
```

27.05.2004



✓ Aufgabe: Strukturen (3) (2)

2. Funktion ptor:

```
#include <math.h>      /* Bibliothek math.h einbinden */

....
RECT ptor (POLAR x)    /* Funktion definieren */
{
    RECT y ;           /* Variable y des Typs RECT definieren */
    p_y->real = p_x->abs * cos ( p_x->phase ) ; /* Realteil berechnen */
    p_y->imag = p_x->abs * sin ( p_x->phase ) ; /* Imaginärteil
                                                berechnen */
    return y ;         /* Struktur y des Typs RECT zurückgeben */
}
```

27.05.2004



✓ Aufgabe: Strukturen (3) (3)

3. Funktion *rtop*:

```
#include <math.h>          /* Bibliothek math.h einbinden */
POLAR rtop (RECT x)        /* Funktion definieren */
{
    POLAR y;               /* Variable y des Typs POLAR definieren */
    p_y->abs = sqrt ( (double) p_x->real * p_x->real +
                      (double) p_x->imag * p_x->imag );
                          /* abs berechnen: cast verhindert
                          Zahlbereichsüberlauf */
    if (p_x->real != 0.)    /* kein Fehler: real != 0., phase berechnen */
    {
        p_y->error = 0;     /* kein Fehler: error = 0 */
        p_y->phase = atan2 ( (double) p_x->imag / p_x->real);
    } /* phase berechnen: cast verhindert Zahlbereichsüberlauf */
    else p_y->error = 1;    /* Fehler: real = 0., phase nicht berechnen */
    return y;              /* Struktur y des Typs POLAR zurückgeben */
}
```

27.05.2004



✓ Aufgabe: Strukturen (3) (4)

4. Funktion *ptor1*:

```
#include <math.h>          /* Bibliothek math.h einbinden */
....
void ptor1 (POLAR *p_x, RECT *p_y) /* Funktion definieren */
{
    /* Zeiger auf Eingangsvariable: p_y
    Zeiger auf Ausgangsvariable: p_x */
    p_y->real = p_x->abs * cos ( p_x->phase );
                          /* Kurzschreibweise mit -> */
    (*p_y).imag = (*p_x).abs * sin ( (*p_x).phase );
                          /* Langschreibweise mit (*...). */
}
```

27.05.2004





Aufgabe: Strukturen (3) (5)

5. Funktion *rtop1*:

```
#include <math.h>                /* Bibliothek math.h einbinden */
....
void rtop1 (RECT *p_x, POLAR *p_y) /* Funktion definieren */
{
    /* Zeiger auf Eingangsvariable: p_y
       Zeiger auf Ausgangsvariable: p_x */
    p_y->abs = sqrt ( (double) (p_x->real)) * p_x->real +
                (double) (p_x->imag) * p_x->imag );
    /* cast verhindert Zahlbereichsüberlauf */
    if (p_x->real != 0.) /* kein Fehler: real != 0., Phase berechnen*/
    {
        p_y->error = 0; /* kein Fehler: error = 0 */
        p_y->phase = atan2 ( (double) (p_x->imag) / p_x->real);
        /* cast verhindert Zahlbereichsüberlauf */
    }
    else p_y->error = 1; /* Fehler: real = 0., phase nicht berechnen */
}
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 43



Aufgabe: Strukturen (4)

- Schreiben Sie 2 Funktionen mit jeweils 2 Zeigern auf Strukturen des Typs **KOMPLEX** als Eingangsvariablen, die als Ergebnisse jeweils 1 Zeiger auf eine Struktur **D_KOMPLEX** zurückgeben. Die Struktur des Typs **KOMPLEX** besitzt als Komponenten **float**-Zahlen, die Struktur **D_KOMPLEX** verwendet als Komponenten **double**-Zahlen.

Die Funktion *k_add (...)* soll 2 komplexe Eingangsvariable addieren, die Funktion *k_mul (...)* soll 2 komplexe Eingangsvariable multiplizieren.

Lesen Sie im Hauptprogramm die Eingangsvariablen ein und verwenden Sie ein Zeigerfeld auf die 2 Funktionen, um in Abhängigkeit von einem Tastaturparameter entweder die Addition oder die Multiplikation zu aktivieren (vgl. auch *Übung zum Kapitel Zeiger, S. 68ff*)!

Geben Sie die Eingangsvariablen und die Ergebnisse auf dem Bildschirm aus!

Müssen Sie Vorkehrungen für eine Zahlbereichsüberschreitung treffen? – Welche?

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 44

✓ Aufgabe: Strukturen (4) (1)

Definitionen von Strukturen und Prototypen:

```
#include <stdio.h> /* Einbinden der Bibliothek stdio.h */
#include <stdlib.h> /* Einbinden der Bibliothek conio.h */
typedef struct { float real, imag; } KOMPLEX ;
/* KOMPLEX definieren: kartesische Koordinaten:
Komponenten vom Typ float: Realteil real,
Imaginärteil imag, */
typedef struct { double d_real, d_imag; } D_KOMPLEX ;
/* D_KOMPLEX definieren: karthesische
Koordinaten:
Komponenten vom Typ double:
Realteil d_real, Imaginärteil d_imag */
D_KOMPLEX *k_add ( KOMPLEX *p1, KOMPLEX *p2 );
/* Prototypenvereinbarung: 2 Zeiger auf KOMPLEX */
D_KOMPLEX *k_mul (KOMPLEX *p1, KOMPLEX *p2 );
/* Prototypenvereinbarung: 2 Zeiger auf KOMPLEX */
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 45

✓ Aufgabe: Strukturen (4) (2)

Definitionen von globalen Variablen und der Funktionen *k_add*:

```
D_KOMPLEX *(*z[2]) (KOMPLEX *p1, KOMPLEX *p2) = {k_add, k_mul};
/* Initialisiere Zeigerfeld *z[2] auf die Funktionen k_add, k_mul.
Jede Funktion gibt einen Zeiger auf eine Struktur des Typs
D_KOMPLEX zurück */
D_KOMPLEX result; /* komplexe Zahl global definieren */
D_KOMPLEX *k_add ( KOMPLEX *p_x, KOMPLEX *p_y )
/* Funktion k_add definieren, die einen Zeiger auf eine
Struktur des Typs D_KOMPLEX zurückgibt */
{
result.d_real = (double) p_x->real + (double) p_y->real ; /* Re(result) */
result.d_imag = (double) p_x->imag + (double) p_y->imag ;
/* Im(result). Die casts (double) verhindern Zahlbereichsüberläufe */
return &result ; /* Rückgabe des Zeigers auf das Ergebnis */
/* Man beachte: result muss global definiert werden. Begründung:
siehe Erläuterung bei der Funktion k_mul */
}
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 46

✓ Aufgabe: Strukturen (4) (3)

Definitionen der Funktion *k_mul*:

```
D_KOMPLEX *k_mul ( KOMPLEX *p_x, KOMPLEX *p_y )
    /* Funktion k_mul definieren, die einen Zeiger auf eine Struktur
       des Typs D_KOMPLEX zurückgibt */
{
    result.d_real = (double) p_x->real * p_y->real -
        (double) p_x->imag * p_y->imag ; /* Realteil von result */
    result.d_imag = (double) p_x->real * p_y->imag +
        (double) p_x->imag * p_y->real ; /* Imaginärteil von result */
    /* Die casts (double) verhindern Zahlbereichsüberläufe */
    return &result ; /* Rückgabe des Zeigers auf das Ergebnis */
    /* Man beachte: result muss global definiert werden. Würde man
       result lokal in der Funktion k_mul definieren, so würde der
       Rückgabewert im Hauptprogramm auf eine Struktur zeigen, die
       nach dem Rücksprung aus der Funktion k_mul
       nicht mehr existiert! */
}
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 47

✓ Aufgabe: Strukturen (4) (3)

Hauptprogramm (1):

```
int main (void) /* Hauptprogramm */
{ /* Variablen vereinbaren: */
    KOMPLEX z1, z2 ;
    D_KOMPLEX Erg;
    int ok, Op;
    char Op1;
    fflush (stdin) ;
    system ("cls") ; /* Bildschirm löschen */
    while (1)
    {
        printf (" Komplexe float Eingangszahlen: Re1, Im1, Re2, Im2 ? : ");
        fflush (stdin); /* Tastaturpuffer löschen */
        ok = scanf ("%f, %f, %f, %f", &z1.real, &z1.imag,
            &z2.real, &z2.imag); /* Operanden einlesen */
        if (ok !=4) break; /* falsche Eingabe beendet Programm! */
        printf (" Operation eingeben: + oder * , sonst Abbruch ? : ");
        fflush (stdin); /* Tastaturpuffer löschen */
        scanf ("%c", &Op1); /* Operation einlesen */
    }
}
```

27.05.2004

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



Programmieren in ANSI C:
K11_Strukturen.ppt, Seite 48

✓ Aufgabe: Strukturen (4) (4)

Hauptprogramm (2):

```

scanf ("%c", &Op1);    /* Operation einlesen */

***
if (Op1 == '+') Op = Op1 - '+';    /* wenn Op1= '+': Op=0 */
else
if (Op1 == '*') Op = Op1 - '*' + 1;    /* wenn Op1= '*': Op=1 */
else break;    /* falsche Eingabe: Programmende! */
Erg = *(*z[Op]) ( &z1, &z2);    /* Verarbeite z1, z2 */
printf ("%f + j %f) %c (%f + j %f) =\n (%lf + j %lf)\n",
z1.real, z1.imag, Op1, z2.real, z2.imag,
Erg.d_real, Erg.d_imag);    /* Ergebnisse ausgeben */
}
printf("\n    Programmende!");
return 0;    /* Rücksprung ins Betriebssystem */
}

```

27.05.2004

