

## 3 Pointer-Variablen

### 3.1 Einführung

Im vorherigen Kapitel haben wir gesehen wie mit dem Adress-Operator «&» die Adresse einer Variablen bestimmt werden kann. In der Programmiersprache C kann man nicht nur Adressen bestimmen, sondern man kann diese auch in speziellen Variablen speichern.

Speziell ist, dass in C nicht nur die Adresse bekannt sein muss sondern auch welcher Datentyp unter dieser Adresse gespeichert ist.

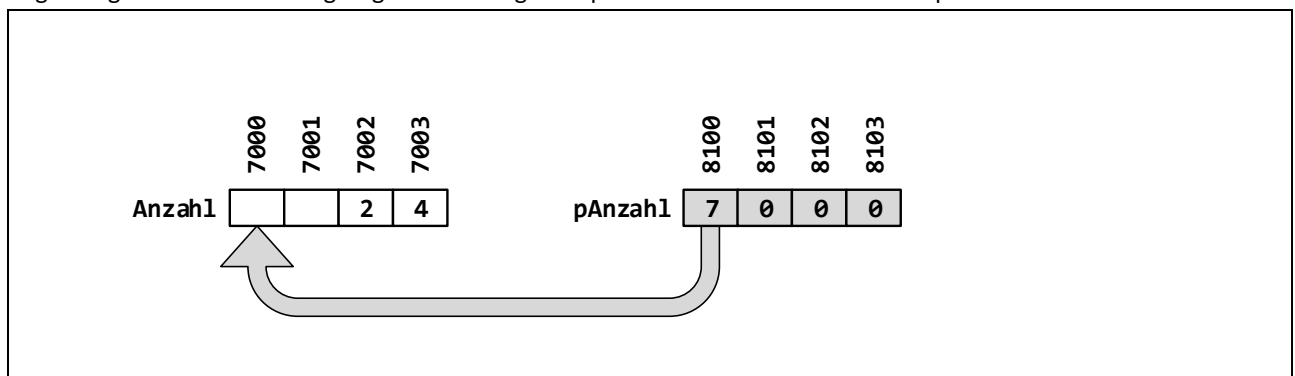
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int    Anzahl = 24;
    int    *pAnzahl = &Anzahl;

    printf("Inhalt von Anzahl = %i\n", Anzahl);
    printf("Adresse von Anzahl = %p\n", pAnzahl);
    system("pause");
    return 0;
}
```

Im obigen Beispiel wird neu eine Variable «pAnzahl» definiert, die eine Adresse einer Integer-Variable speichert. Eine Variable die eine Adresse speichert bezeichnet man als **Pointer** (auf Deutsch Zeiger).

Die Adresse ist eine ganzzahlige Zahl, welche der Nummer einer Speicherstelle im Arbeitsspeicher entspricht. Die folgende grafische Darstellung zeigt für das obige Beispiel den Ausschnitt des Arbeitsspeichers:



### 3.2 Zeiger deklarieren (Der Indirektionsoperator \* bei der Deklaration)

Wie bereits erwähnt, ist ein Zeiger eine numerische Variable. Wie alle Variablen muss man auch Zeigervariablen deklarieren, bevor man sie verwenden kann. Die Benennung von Zeigervariablen folgt den gleichen Regeln wie für andere Variablen. Der Name muss eindeutig sein.

**Vereinbarung:** Wir halten uns an die Konvention, dass wir für den Variablennamen eines Zeigers immer die Bezeichnung «pName» und nicht nur «Name» verwenden. Also vor dem eigentlichen Variablennamen immer ein kleines «p», für Pointer, schreiben.

Dies ist allerdings nicht zwingend. Sie können Ihren Zeigern beliebige Namen geben, solange sie den C-Regeln entsprechen.

Eine Zeigerdeklaration weist die folgende Form auf:

Abstrakt:	<b>typename</b>	<b>*Zeigername;</b>
Konkret:	<b>int</b>	<b>*pName = NULL;</b>

«Typname» steht für einen beliebigen Datentyp von C und spezifiziert den Typ der Variablen, auf die der Zeiger verweist. Der Stern (\*) macht bei der Deklaration deutlich, dass «Zeigername» ein Zeiger ist.

### 3.3 Zeiger initialisieren (mit Hilfe des Adressoperators &)

Nachdem Sie einen Zeiger deklariert haben, müssen Sie sich darum kümmern, dass er auf ein sinnvolles Ziel – sprich eine Variable – verweist. Analog zu normalen Variablen gilt: Wenn Sie nicht-initialisierte Zeiger verwenden, sind die Ergebnisse unvorhersehbar und unter Umständen katastrophal.

**Achtung:** Initialisieren Sie stets einen Zeiger indem sie ihn auf eine konkrete Variable referenzieren lassen oder setzen sie den Zeiger auf NULL (NULL = Leer)

**Aufgabe 01: Pointer-Variablen deklarieren**

An die Deklaration eines Pointers muss man sich etwas gewöhnen, weil vor dem Variablennamen ein «\*» (Stern) steht: «int \*pZahl;». Dabei ist «pZahl» also eine Pointer-Variable, welche auf eine ganze Zahl zeigt. Demnach ist «int \*» die Definition für einen Pointer.

Tippen Sie das folgende Programm ab und bringen Sie es zum Laufen.

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

int main()
{
    int    Anzahl    = 24;
    int *pAnzahl1    = &Anzahl;
    int*   pAnzahl2   = &Anzahl;

    printf("Inhalt von Anzahl    = %i\n", Anzahl);
    printf("Inhalt von pAnzahl1 = %p\n", pAnzahl1);
    printf("Inhalt von pAnzahl2 = %p\n", pAnzahl2);
    system("pause");
    return 0;
}
```

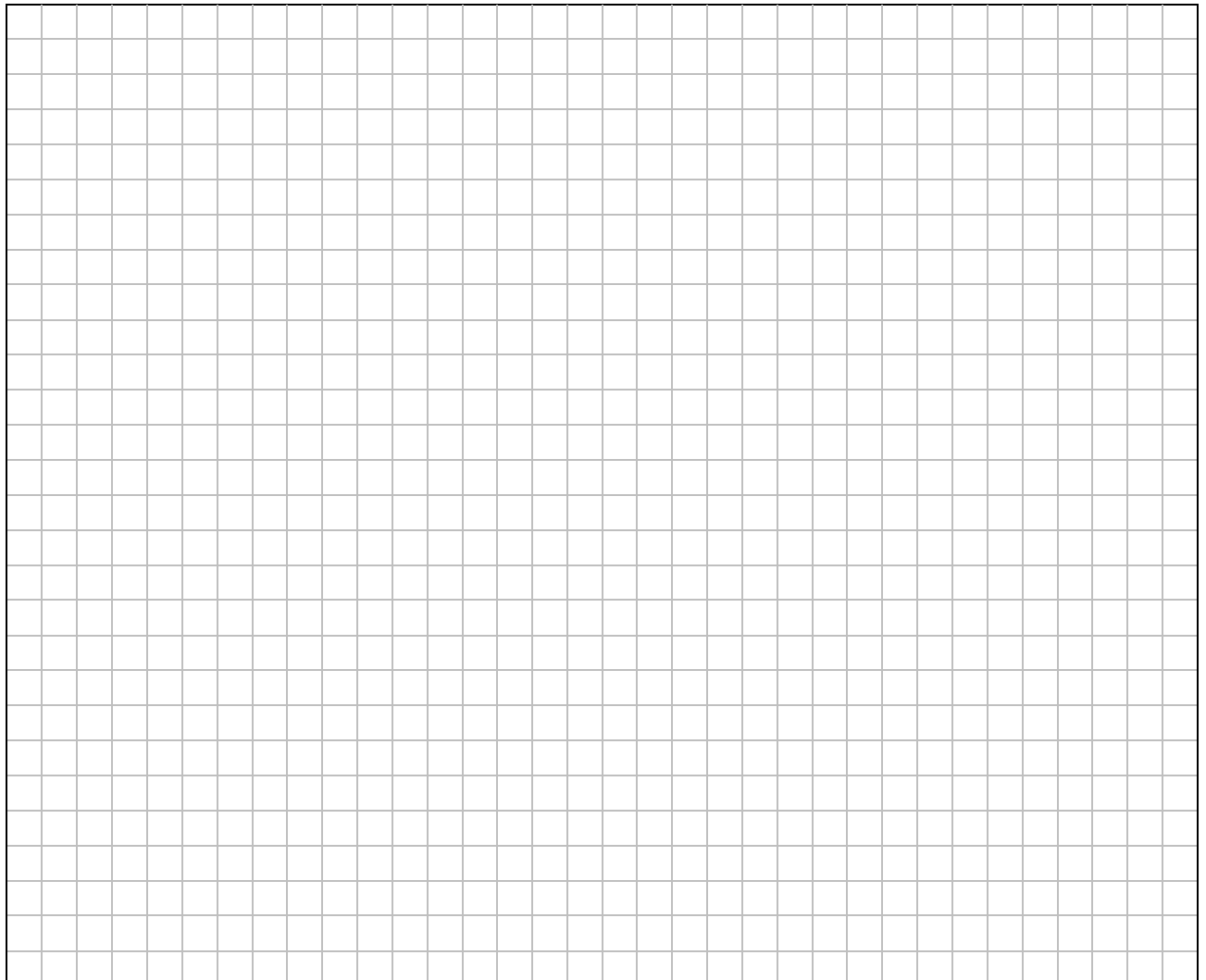
Spielt es einer Rolle ob die Variable «pAnzahl1» bzw. «pAnzahl2» als

«int \*pAnzahl1;»

oder

«int\* pAnzahl1;»

definiert wird?





### Aufgabe 03: Ich liebe Pointer

Ergänzen Sie das folgende Programm zur Berechnung einer Rechteckfläche, indem Sie für den fehlenden Code nur noch die Pointer-Variablen «pLaenge», «pBreite» und «pFlaeche» verwenden.

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

int main()
{
    int Laenge    = 0;
    int Breite    = 0;
    int Flaeche    = 0;
    int *pLaenge  = &Laenge;
    int *pBreite  = &Breite;
    int *pFlaeche = &Flaeche;

    // Eingabe von Länge und Breite
    ...
    ...
    // Verarbeitung der Eingabe, Berechnung der Fläche
    ...
    ...
    // Ausgabe von Länge, Breite und Fläche
    ...
    ...
    system("pause");
    return 0;
}
```

## Aufgabe 04: Compiler-Error

Wenn Sie das folgende kleine Programm abtippen und kompilieren, werden Sie einen Compilerfehler erhalten:

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

int main()
{
    double *pQuadrat = 81.0;

    printf("Quadrat = %lf\n", *pQuadrat);
    printf("Zahl = %lf\n", sqrt(*pQuadrat));
    system("pause");
    return 0;
}
```

Schreiben Sie den Grund für diesen Fehler und eine eventuelle Korrektur auf:

[illegible]

### Aufgabe 05: Programmcode nach Anweisungen umsetzen

Schreiben Sie ein Programm gemäss den folgenden Anweisungen und testen Sie es! Die Zahlen der folgenden Teilaufgaben korrespondieren mit den Kommentaren der untenstehenden Codevorlage.

01. Es werden zwei int-Variablen «Zahl» und «Wert» deklariert, von denen «Zahl» mit dem Wert «345» initialisiert wird.
02. Es werden zwei Zeiger («pZeig1» und «pZeig2») deklariert, welche die Adresse einer int- Variablen speichern können.
03. Der Zeigervariablen «pZeig1» wird die Adresse von «Zahl» zugewiesen
04. Der Zeigervariablen «pZeig2» wird die Adresse von «Wert» zugewiesen
05. Über den Indirektionsoperator wird über den Zeiger «pZeig2» der Variablen «Wert» der Wert «45» zugewiesen.
06. Dem Zeiger «pZeig2» wird die Adresse, die der Zeiger «pZeig1» enthält, zugewiesen. Dadurch zeigt «pZeig2» jetzt ebenfalls auf die Adresse der Variablen «Zahl».
07. Über den Zeiger «pZeig2» wird der Variablen «Zahl» nun der Wert «100» zugewiesen.
08. Die Funktion printf gibt den Wert und die Adresse der Variablen «Zahl» aus
09. Die Funktion printf gibt die Adresse, die in der Pointervariablen «pZeig1» gespeichert ist und den Wert, auf den diese Adresse zeigt, aus (Tipp: Ausgabe der Adresse mit %u oder %p).
10. Die Funktion printf gibt den Wert und die Adresse der Variablen «Wert» aus
11. Die Funktion printf gibt die Adresse, die in der Pointervariablen «pZeig2» gespeichert ist und den Wert, auf den diese Adresse zeigt, aus (Tipp: Ausgabe der Adresse mit %u oder %p).

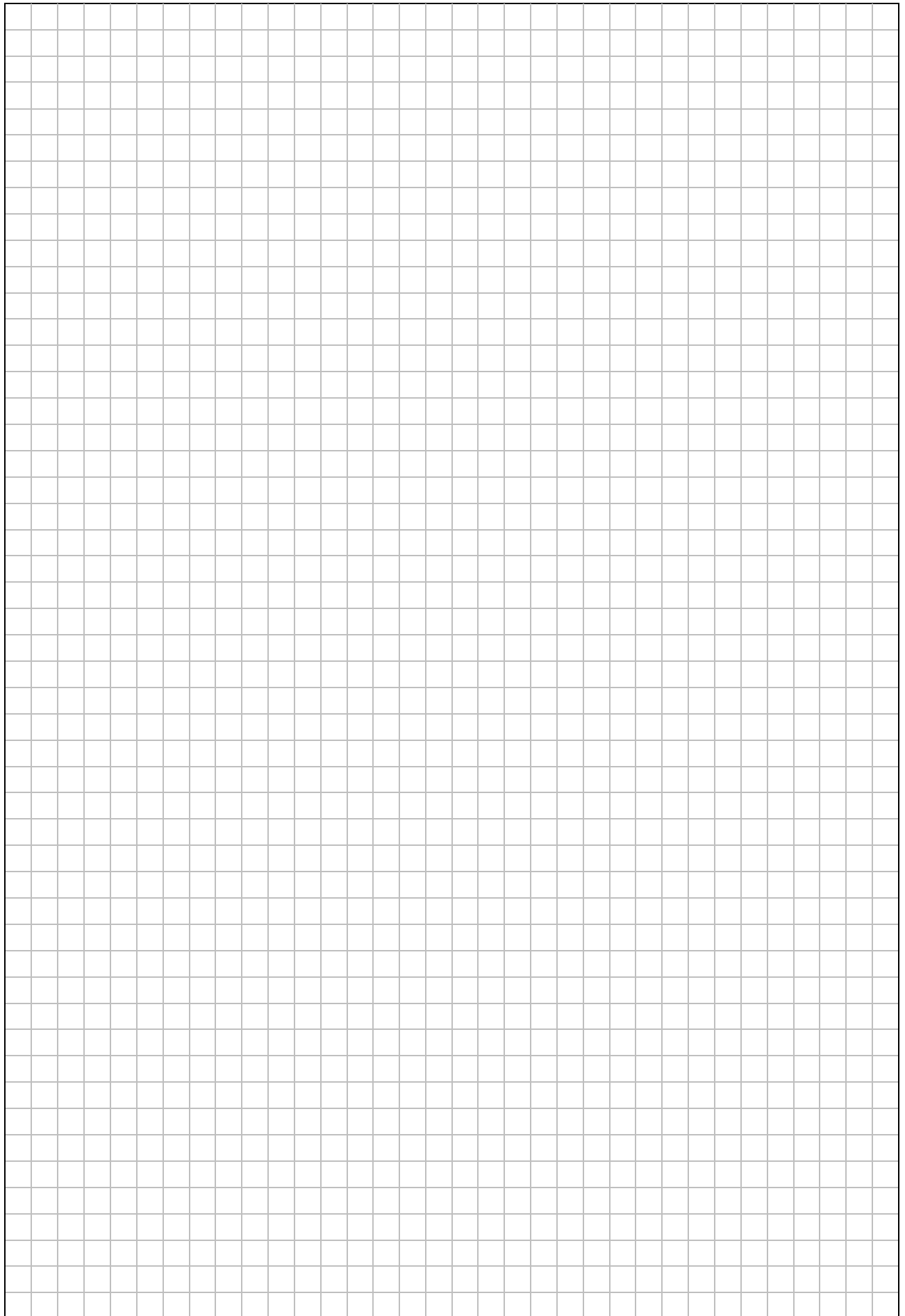
Verwenden Sie für das Programm, die folgende Vorgabe!

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

int main() {
    // 01.
    // 02.
    // 03.
    // 04.
    // 05.
    // 06.
    // 07.
    // 08.
    // 09.
    // 10.
    // 11.
    system("pause");
    return 0;
}
```

Schreiben Sie hier auf, welche Ausgaben Sie erhalten haben:

[illegible]



### 3.4 Pointer-Arithmetik

(siehe auch "C Programmieren von Anfang an" von Helmut Erlenkötter, Kapitel 10.4, Pointer-Arithmetik)

Einer der grossen Vorteile bei der Verwendung von Pointern in C ist die Pointer-Arithmetik. Pointer können addiert bzw. subtrahiert werden, wie das folgende Beispiel zeigt:

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

int main()
{
    int Zahl = 20;
    int *pZahl = &Zahl;

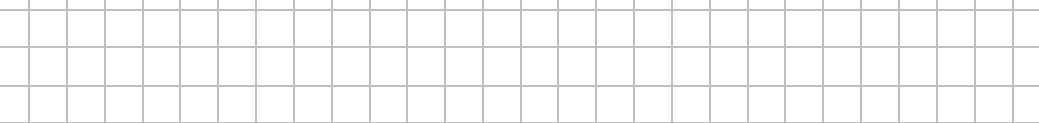
    printf("\n");
    printf("Adresse auf die pZahl zeigt          = %p\n", pZahl);
    printf("Inhalt der Adresse auf die pZahl zeigt = %i\n", *pZahl);
    pZahl++;
    printf("\n");
    printf("Adresse auf die pZahl zeigt          = %p\n", pZahl);
    printf("Inhalt der Adresse auf die pZahl zeigt = %i\n", *pZahl);
    system("pause");
    return 0;
}
```

Das obige Programm erzeugt die folgende Ausgabe:

```
Adresse auf die pZahl zeigt      = 001EFEE0
Inhalt der Adresse auf die pZahl zeigt = 20

Adresse auf die pZahl zeigt      = 001EFEE4
Inhalt der Adresse auf die pZahl zeigt = -858993460
```

Versuchen Sie diese Ausgabe zu erklären!



### Erkenntnis:

A blank sheet of graph paper with a grid pattern. The grid consists of small squares formed by thin gray lines. There are 20 columns and 15 rows of squares. The entire grid is enclosed within a black rectangular border.



## Aufgabe 06: Programmcode mit Zeiger verstehen

Welche der folgenden Zeilen ergibt einen Compilerfehler [CF], eine Compilerwarnung [CW] oder einen Laufzeitfehler [LF]?

```

01: #include <stdio.h>
02: #include <stdlib.h>
03:
04: int main()
05: {
06:     // Variablen deklarieren und initialisieren
07:     int Zahl = 10.0;
08:     char Zeichen = 'A';
09:     double Wert = 0.00;
10:
11:     // Pointervariablen deklarieren und initialisieren
12:     int *pZahl = null;
13:     char *pZeichen = &Zeichen;
14:     double *pWert = &Wert;
15:     pZahl = &Zahl;
16:
17:     // Ausgaben
18:     printf ("%i, %i\n", Zahl, pZahl);
19:     printf ("%c, %c\n", &Zeichen, *pZeichen);
20:     printf ("%lf, %lf\n", *pWert + 1, *Wert);
21:
22:     // Veränderungen
23:     Zahl = *pZahl++;
24:     Zeichen = "A";
25:     pWert = --Wert + 8;
26:
27:     // Ausgaben 2
28:     printf ("%i, %i\n", Zahl, *pZahl);
29:     printf ("%c, %c\n", Zeichen, *pZeichen);
30:     printf ("%lf, %lf\n", *(pWert + 5), Wert);
31:
32:     system("pause");
33:     return 0;
34: }
```

Beschreiben Sie und korrigieren Sie den jeweiligen Fehler in der folgenden Tabelle analog dem gemachten Beispiel!

Zeile	Fehlerbeschreibung und Korrektur	Fehlerart [KF, KW, LF]
07	<b>Fehler:</b> Ein double Wert wird einer Integer-Variablen zugewiesen → Möglicher Genauigkeitsverlust  <b>Korrektur:</b> int iWert = 10;	KW

<b>Zeile</b>	<b>Fehlerbeschreibung und Korrektur</b>	<b>Fehlerart [KF, KW, LF]</b>