

# PROGRAMMIEREN I

WS 2022

Prof. Dr.-Ing. Kolja Eger  
Hochschule für Angewandte Wissenschaften Hamburg



## Präsenzklausur im Labor (180min)

Dynamische Speicherallokation

Praktikum 7

Strukturen

Praktikum 6

Dateien

Praktikum 5

Zeiger

Praktikum 4

Vektoren (Arrays)

Praktikum 3

Funktionen

Praktikum 2

Kontrollstrukturen

Datentypen & Operatoren

Erste Programme

Praktikum 1

# Unser Weg durch das Semester



# Was machen wir heute?

- Zeiger
  - Einführung
  - Erste Beispiele
- Dateien
  - Einführung
  - Beispiel
  - Textdateien

# ZEIGER (POINTER)

# Zeiger

- Ein Zeiger
  - ist eine Variable, die auf eine Stelle im Speicher zeigt
  - engl. Pointer
- Zum Vergleich ein Vektor
  - ist eine Verkettung von Variablen
  - engl. Array
- Beide in C eng verwandt

# Eine Variable hat 4 Eigenschaften

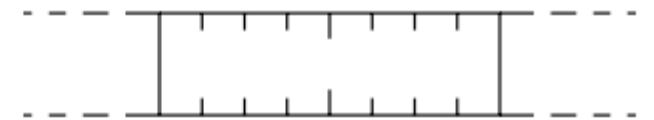
- Typ
  - definierter Datentyp, z.B. int, char, ..
- Name
  - um Variablen von anderen zu unterscheiden
- Adresse/Speicher
  - eine Variable wird an einer Stelle im Speicher abgelegt, die durch eine eindeutige Adresse definiert ist.
- Wert
  - eine Variable hat einen Wert

```
double zahl = 1.2345;
```

Typ: double

Name: zahl

Speicher:



Wert: 1,234

# Zeiger (II)

- Jede Speicherzelle im Arbeitsspeicher eines Computers hat eine eindeutige Adresse
- Eine Information, z.B. der Zahlenwert einer Zahl vom Typ `int`, wird unter einer Adresse abgelegt
- Er kann über die Adresse wieder abgerufen oder verändert werden
- Vieles rund um Adressen läuft für den Programmierer im Hintergrund ab
- Adressen können in C gezielt genutzt werden – wie und warum behandeln wir im Detail
- Wird in C mit Adressen gearbeitet, so ist sie nicht selbst die Information, sondern die Adresse welche auf die Stelle im Speicher zeigt, an der die Information liegt  
→ Zeiger

# Einfache Zeiger

Allgemeine Syntax zur Definition eines Zeigers

```
<Datentyp> *<Variablenname>;
```

## Beispiel

```
int *pInt;
```

- Datentyp `int` deutet an, dass der Speicher, auf den der Zeiger zeigt
- Stern gibt an, dass es sich um einen einfachen Zeiger handelt
- Zusammen ➔ "Zeiger auf Datentyp `int`".
- Name des Zeigers, hier `pInt`, ist frei wählbar



# Einfache Zeiger (2)

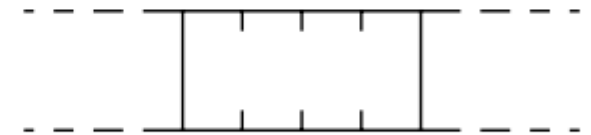
- Ein Zeiger belegt bei einem 32 Bit System vier Byte
- und auf einem 64 Bit System 8 Byte

```
int *pInt;
```

Typ:  $\text{int}^*$

Name:  $\text{pInt}$

Speicher:



Wert:

?

# Initialisierung eines einfachen Zeigers

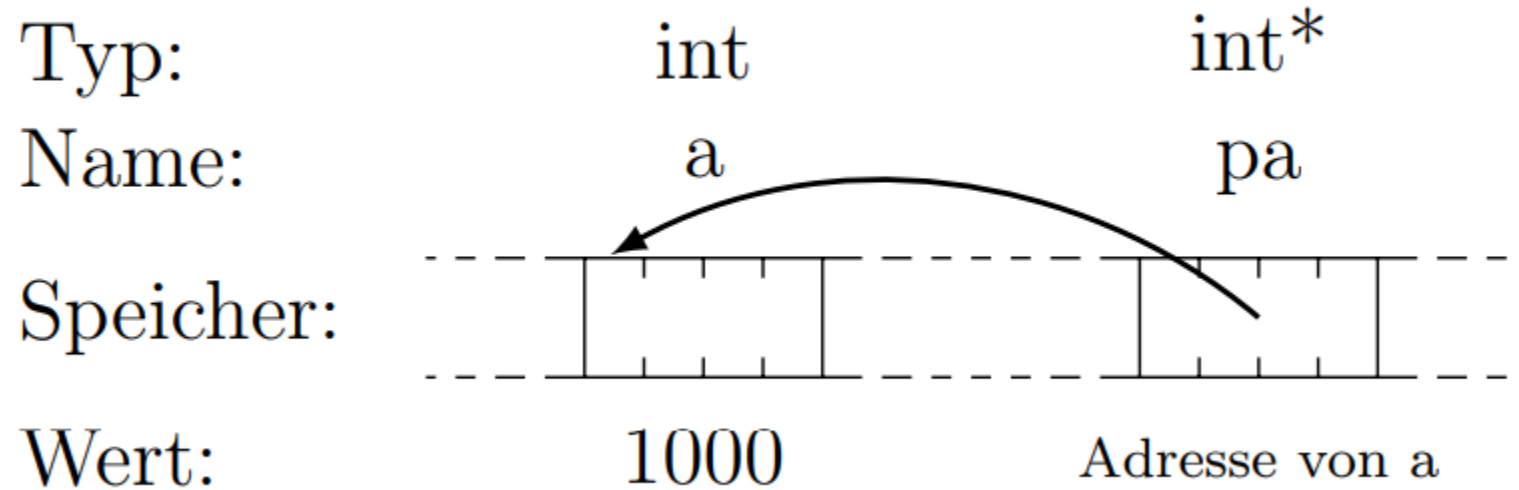
- Nachdem ein Zeiger definiert wurde, zeigt er an eine unbekannte Stelle im Speicher
- Um ihn auf eine sinnvolle Stelle zeigen zu lassen, benötigen wir den Adressoperator &
- Symbol & wird hier als unärer Operatoren genutzt d.h. mit nur einem Operanden
- Im Gegensatz dazu benötigen binäre Operatoren zwei Operanden (z.B. für bitweise AND, z.B.  $x \& y$ )

```
int a=1000;  
int *pa ;  
pa = &a ;
```

Oder kürzer mit Definition und Initialisierung in einer Zeile

```
int a=1000;  
int *pa=&a;
```

# Initialisierung eines einfachen Zeigers (2)



## → Beispiel in Visual Studio

```
// Erstes Beispiel mit Zeigern
#include<stdio.h>

int main() {
    int a = 1000;
    int* pa; // Definition eines Zeigers mit *
    pa = &a; // Zuweisung der Adresse mit &
    printf("int= %d; pointer= %p\n", a, pa); // Ausgabe von Wert & Adresse
    printf("int= %d; pointer= %p\n", *pa, pa); // Ausgabe des Werts über Zeiger
    return 0;
}
```

# Arbeiten mit einfachen Zeigern - Beispiel

```
1  int a=4, b;      /* zwei int-Variablen */
2  int z[6];        /* ein Vektor mit sechs int-Variablen */
3  int *pi;         /* ein Zeiger auf int */
4
5  pi = &a;         /* jetzt zeigt pi auf a */
6  b = *pi;         /* jetzt hat b den Wert 4 */
7  *pi = 7;         /* jetzt hat a den Wert 7 */
8  pi = &z[2];      /* nun zeigt pi auf z[2] */
9  *pi = 9;         /* jetzt hat z[2] den Wert 9 */
10 pi++;           /* ... und jetzt zeigt pi auf z[3] */
```





# Beispiel: Zwei Zahlen austauschen, SwapInt





Es sollen die Werte für die Variable a und b ausgetauscht werden:

```
tmp = a ;  
a = b ;  
b = tmp ;
```

Wie sieht eine entsprechende Funktion aus?

```
void SwapInt (int value1, int value2)  
/* falsch oder richtig? */  
{  
    int tmp;  
    tmp = value1;  
    value1 = value2;  
    value2 = tmp;  
}
```

# Falscher Ansatz: Variable wird kopiert!

Typ:	int	int	int	int
Name:	a	b	value1	value2
Speicher:				
Wert:	5	9	9	5

# Beispiel: Zwei Zahlen austauschen, SwapInt

Richtige Lösung mit Zeigern

```
void SwapInt (int *pValue1, int
*pValue2)
{
    int tmp;
    tmp = *pValue1;
    *pValue1 = *pValue2;
    *pValue2 = tmp;
}
```

Funktionsaufruf mit Zeigern

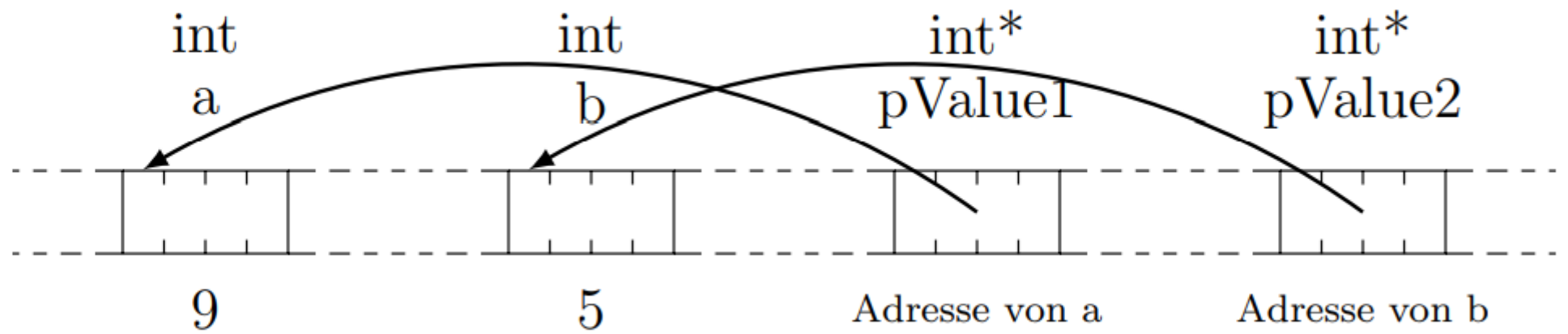
```
int a=5;
int b=9;
SwapInt(&a, &b);
```

Typ:

Name:

Speicher:

Wert:



# UMGANG MIT DATEIEN



# Unterschiedliche Dateien

## Textdatei

- ASCII-Dateien (im engeren Sinne)
  - Enthält Zeichen nach *American Standard Code for Information Interchange* Zeichensatzes
- nur druckbare Zeichen mit 7 Bit (0...127) kodiert (aber als 1 Byte abgelegt)
- Allgemein: Dateien, die nur lesbare Zeichen enthalten
- Leicht zu erstellen und mit Editoren lesbar
- Dateien eher größer

## Binärdateien

- Beliebige Zahlenwerte, nicht zwingend druckbares Zeichen
- Z.B. *EXE*-Dateien mit ausführbarem Code (CPU-Befehle)
- nicht mit Texteditor auslesbar
- Kompakte Datenhaltung
- Alle Dateien können als Binärdateien verarbeitet werden
- In C gibt es unterschiedliche Funktionen um mit Text- und Binärdateien zu arbeiten

# Textdateien – Beispiel: Datei kopieren

1. Dateien öffnen

2. Datei kopieren

3. Dateien schließen

- In C sind Dateien vom Typ `FILE`
- Dateioperationen sind in der Standard-Bibliothek `stdio.h`

# FILE

- Alle Interaktionen mit Dateien erfolgen über die Struktur FILE
- Beim Öffnen der Datei wird die Struktur erstellt und ein Zeiger auf die Struktur zurückgegeben
- Alle anderen Funktionen erwarten den Zeiger als Input-Parameter
- Die Struktur enthält einen Puffer (engl. *Buffer*) damit nicht jedes Zeichen einzeln vom Datenträger gelesen werden muss
- Beim ersten Lesen werden z.B. 4096 Bytes gepuffert und erst beim 4097.Byte muss aus der Datei erneut gelesen werden
- Somit wird die Programmlaufzeit tw. deutlich reduziert!

# → Visual Studio

```
1  #define _CRT_SECURE_NO_DEPRECATED
2  #include <stdio.h>
3
4  int main() {
5      FILE *inp = NULL;    // Zeiger auf Eingabedatei-Struktur
6      FILE* out = NULL;    // Zeiger auf Ausgabedatei-Struktur
7
8      char InpFileName[] = "Beispiel.txt";    // Name für Eingabedatei
9      char OutFileName[] = "Beispiel.bak";    // Name für Ausgabedatei
10
11     char ch;    // Zeichen zum Kopieren
12
13     // Schritt 1: Dateien öffnen
14     inp = fopen(InpFileName, "rt");
15     if (inp == NULL)
16         printf("Fehler: InpFile konnte nicht geöffnet werden\n");
17
18     out = fopen(OutFileName, "w");
19     if (out == NULL)
20         printf("Fehler: OutFile konnte nicht geöffnet werden\n");
21
22     // 2.Schritt Datei kopieren
23     if (inp && out) {
24         ch = fgetc(inp);
25         while (!feof(inp)) {
26             fputc(ch, out);
27             ch = fgetc(inp);
28         }
29     }
30
31     // 3.Schritt: Dateien schließen
32     if (inp) fclose(inp);
33     if (out) fclose(out);
34
35     // Schlussmeldung
36     if (inp && out)
37         printf("Datei erfolgreich kopiert\n");
38
39     return 0;
40 }
```

# Datei öffnen mit `fopen`

```
FILE * fopen ( const char * filename, const char * mode );
```

filename                      Dateiname

mode                          Zugriffsmodus auf die Datei

Following containing a the access modes it can be:

"r"	<b>read:</b> Open file for input operations. The file must exist.
"w"	<b>write:</b> Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
"a"	<b>append:</b> Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations ( <code>fseek</code> , <code>fsetpos</code> , <code>rewind</code> ) are ignored. The file is created if it does not exist.
"r+"	<b>read/update:</b> Open a file for update (both for input and output). The file must exist.
"w+"	<b>write/update:</b> Create an empty file and open it for update (both for input and output). If a file with the

- Zusätzlich kann mit ,t' oder ,b' im Mode die Datei als Text- oder Binärdatei geöffnet werden
- Text-Datei ist „default“-Einstellung, d.h. ,t' kann auch weggelassen werden
- Funktion gibt einen Zeiger auf die Struktur FILE zurück; bei Fehler wird NULL zurückgegeben



# Datei verarbeiten

- Im Beispiel wird ein Zeichen gelesen und geschrieben
- Lesen mit `int fgetc ( FILE * stream );`
- Schreiben mit `int fputc ( int character, FILE * stream );`
- Prüfen ob Dateiende erreicht mit `int feof ( FILE * stream );`
- Im Hintergrund werden Daten blockweise (4096 Bytes) von der Festplatte in den Arbeitsspeicher geladen, um Zugriffszeit zu reduzieren
- Auch beim Schreiben werden Daten zuerst im Arbeitsspeicher gepuffert und dann als Block weggeschrieben

# Dateien schließen mit `fclose`

```
int fclose ( FILE * stream );
```

- Dateien werden auch automatisch geschlossen wenn Programm beendet wird
- Trotzdem: Dateien schließen, wenn sie nicht mehr genutzt werden, um Overhead zu vermeiden

# Fehler-Handling

- Typische Fehler
  - Zu öffnende Datei existiert nicht
  - Datei in die geschrieben werden soll ist schreibgeschützt und/oder bereits geöffnet
- `NULL` gibt für den Zeiger auf eine Datei an, dass sie noch nicht geöffnet wurde oder ein Fehler aufgetreten ist
- Zeiger sollte am Anfang auf `NULL` gesetzt werden
- Vor dem Zugriff ist zu prüfen, ob der Zeiger ungleich `NULL` ist

**VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!**