

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?

Dateien

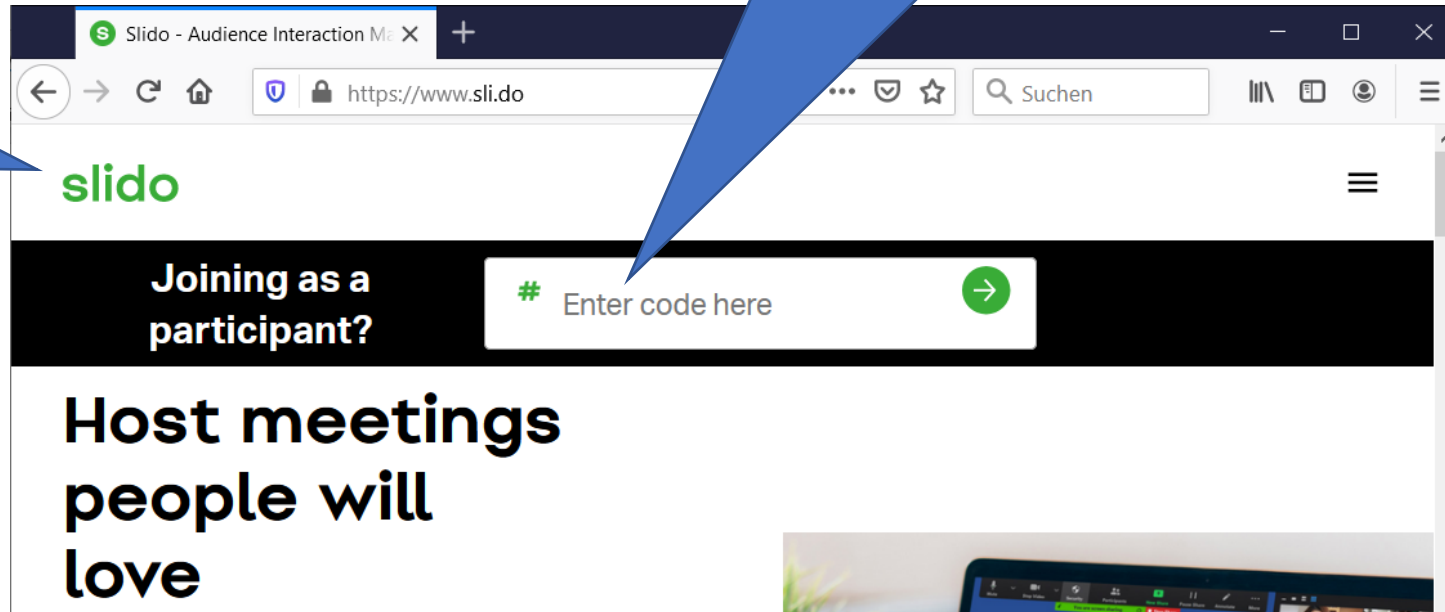
- Binärdateien

Zeiger

- Zeiger & Vektoren
- Zeigervektoren
- Zeiger auf Zeiger

Slido.com

REE-PR1



BINÄRDATEIEN

Dateien - Wiederholung

In C sind Dateien vom Typ `FILE`

Funktionen zur Verarbeitung von Dateien:

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

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

```
int fputc ( int character, FILE * stream );
```

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

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

Binärdateien

- Öffnen wieder mit `fopen`
 - Mode für Binärdateien ist „b“, z.B. „wb“ oder „rb“
- Verarbeitung mit `fwrite()` und `fread()`

In diesem Stream wird geschrieben

Größe eines Elementes, welches geschrieben werden soll (Hilfsfunktion `sizeof()` nützlich)

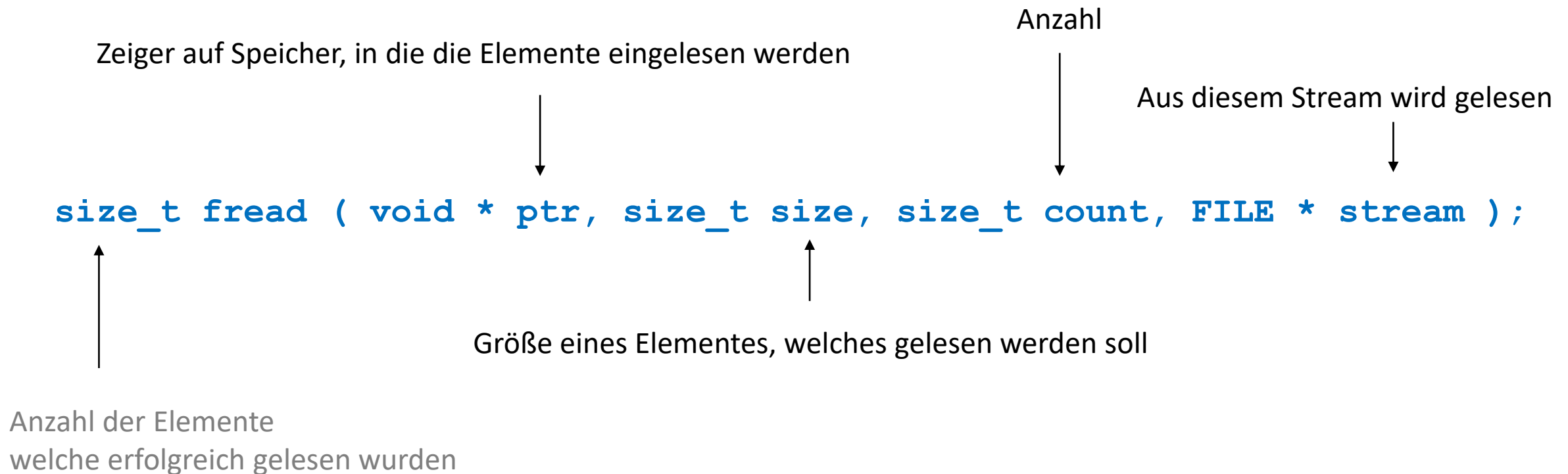
```
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream );
```

↑
Anzahl der Elemente
welche erfolgreich geschrieben wurden

↑
Zeiger auf Daten, die geschrieben werden sollen

↑
Anzahl der Elemente

Binärdateien – fread()



Übung: Binärdatei schreiben & lesen

- In EMIL finden Sie den Quellcode zum Schreiben einer Binärdatei (*schreibeBinaerdatei.c*)
- Testen Sie den Code wie folgt:
 - Führen Sie den Code aus. Wurde die Datei *Beispiel.dat* erstellt? In welchem Ordner wurde die Datei erstellt?
 - Setzen Sie die Datei im Date Explorer auf schreibgeschützt. Welche Ausgabe erfolgt bei erneuter Ausführung des Codes?
- In EMIL finden Sie auch die Datei *leseBinaerdatei.c*
 - Ergänzen Sie an den mit TODO markierten Stellen den Code in *leseBinaerdatei.c*, so dass die Datei *Beispiel.dat* eingelesen und ausgegeben wird

ZEIGER

- Zeigervektoren
- Zeiger auf Zeiger
- Beispiele

Zeiger - Wiederholung

- Was ist ein Zeiger?
- Wofür braucht man Zeiger?
- Welche zwei Operatoren werden bei Zeigern verwendet? Wofür?

Aufgabe 1.7: Welchen Wert haben die Variablen x und y nach dem folgenden Programmstück?

```
double x=1.2, y=1.5, *px=&x, *py=&y;  
*py = *px+0.8;  
*px = *px**py;
```

Zeiger – Beispiel

- Was macht diese Funktion?

```
int fct (char *text)
{
    char *pc=text;
    while(*pc != '\0') pc++;
    return pc-text;
}
```

- Die Funktion `strlen` ist Teil einer Standardbibliothek in C und kann mit der Headerdatei `<string.h>` eingebunden werden

VEKTOREN & ZEIGER (ARRAYS & POINTER)

Vektoren & Zeiger sind eng miteinander verwandt

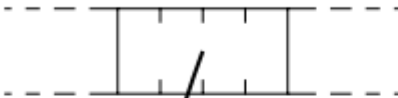
- Syntax kann man „vertauschen“
 - Man kann mit der Syntax für Zeiger auf Elemente des Vektors zugreifen
 - Und auch mit der Syntax für Vektoren den Zeiger verwenden
- Beispiel


```
int b[8];  
int *c = &b[0];
```

```
*(b+2) = 4;
```

```
c[2] = 4
```

Und nochmal Schritt für Schritt

Typ: `int*`
Name: `c`
Speicher: 
Wert: Adresse von `b[0]`

Typ: `int` `int` `int` `int` `int` `int` `int` `int`
Name: `b[0]` `b[1]` `b[2]` `b[3]` `b[4]` `b[5]` `b[6]` `b[7]`
Speicher: 
Wert: `?` `?` `?` `?` `?` `?` `?` `?`

```
int b[8];  
int *c = &b[0];
```

Und nochmal Schritt für Schritt (2)

```
int b[8];  
int *c = &b[0];
```

```
b[0] = 2;  
*c=2;
```



Beide Zeilen haben die gleiche Wirkung!

```
b[2] = 4;  
*(c+2) = 4;
```



Beide Zeilen haben die gleiche Wirkung!

Und nochmal Schritt für Schritt (3)

```
int b[8];  
int *c = &b[0];
```

```
b[0] = 2;  
*b = 2;  
  
c[0] = 2;  
*c = 2;
```



Alle vier Zeilen haben die gleiche Wirkung!

Und nochmal Schritt für Schritt (4)

```
int b[8];  
int *c = &b[0];
```

```
b[2] = 4;  
*(b+2) = 4;  
  
c[2] = 4;  
*(c+2) = 4;
```




Alle vier Zeilen haben die gleiche Wirkung!

Operanden * und [] in je zwei Varianten

- Operand * erwartet einen Zeiger und gibt die Speicherstelle, auf der der Zeiger zeigt, aus
- Operand * erwartet einen Vektor und gibt das erste Element des Vektors aus
- Operand [] erwartet einen Vektor und gibt das Element im Vektor aus, welches durch die Zahl in den Klammern angegeben ist
- Operand [] erwartet einen Zeiger. Die Zahl in den Klammern wird auf den Zeiger „addiert“ und die resultierende Speicherstelle ausgegeben
- Auch die Addition vor dem Zugriff mit dem Operand * kann mit Zeigern und Vektoren arbeiten (siehe vorige Slide)

Wofür ist das gut?

- Beispiel 1:



Was wird
ausgegeben?

```
char Text [] = "links rechts";  
printf("Text: %s \n" , Text +6);
```

```
Text: rechts
```

Vektoren als Funktionsparameter

- Auch Vektoren können als Parameter einer Funktion übergeben werden
- Unterschied: Vektoren werden nicht als Kopie übergeben, sondern immer als Referenzparameter (Call-by-Reference)
- Auch die Anzahl der Elemente kann entfallen
- Beispiel für Deklaration:

```
void Init(int Note[]);
```

Im Hintergrund werden Zeiger genutzt

```
void Init(int* Note);
```

Auch ein Zeiger als Funktionsparameter ist möglich

Vektoren als Funktionsparameter (II)

- Beispiel:

```
//void Init(int Note[6]); // Vektor als Funktionsparameter
void Init(int Note[]); // Anzahl der Elemente kann weggelassen werden

int main()
{
    int Note_Sem_1[6];

    Init(Note_Sem_1);

    return 0;
}

//void Init(int Note[6])
void Init(int Note[]) // Anzahl der Elemente kann weggelassen werden
{
    int i;

    for (i = 0; i < 6; i++) Note[i] = 0;
}
```

Im Hintergrund wird ein Zeiger auf das erste Element im Array übergeben

Größe eines Vektors

- Die Speichergröße eines Vektors kann mit der Funktion *sizeof()* bestimmt werden

```
printf("Speichergroesse von FtoC: %u\n", sizeof(FtoC));
```

- Es lässt sich auch die Anzahl der Elemente ermitteln

```
printf("Elemente of FtoC: %u\n", sizeof(FtoC)/sizeof(float));
```

- Aber:
 - Die Größe eines Vektors ist nur in der Funktion bekannt, in der der Vektor definiert wurde
 - Und nicht in Funktionen, die den Vektor als Parameter übergeben bekommen haben
 - Häufig wird deswegen die Länge (1.Dimension) als Funktionsparameter übergeben

```
void InitTemperatur_v2(float FtoC[][2], unsigned len);
```


ZEIGERVEKTOREN

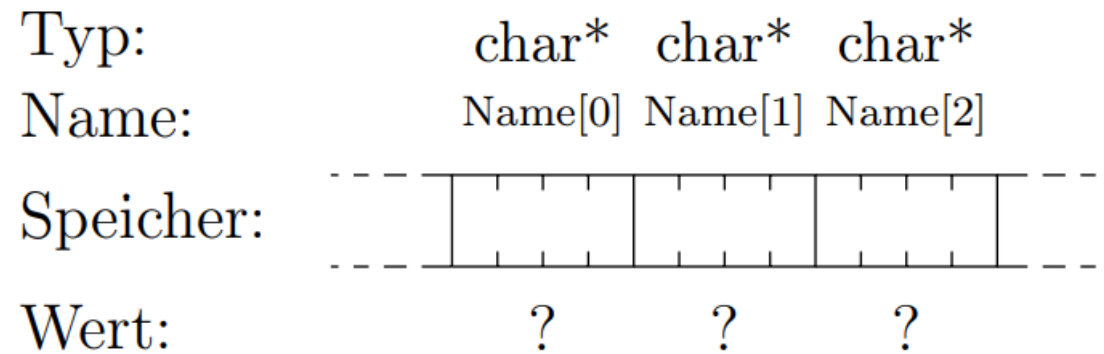
Zeigervektoren

- Wie alle anderen Datentypen können auch Zeiger zu Vektoren verkettet werden
- Ein *Zeigervektor* ist ein *Vektor*, dessen Elemente *Zeiger* sind
- Definition `<Datentyp> *<Variablenname>[<Anzahl>];`
- Beispiel: `double *pd[5];`

Typ:	double*	double*	double*	double*	double*
Name:	pd[0]	pd[1]	pd[2]	pd[3]	pd[4]
Speicher:	<div><div></div><div></div><div></div><div></div><div></div></div>				
Wert:	?	?	?	?	?

Zeigervektoren

- Beispiel 2: `char *Name[3];`



- Wie viel Speicher benötigen die Zeigervektoren?

Zeigervektoren – Initialisierung (3 Wege)

Beispiel:

```
char a[] = "Hans";  
char b[] = "Jutta";  
char c[] = "Lu";
```

1

```
char *Name[3];
```

```
Name[0] = &a[0];  
Name[1] = &b[0];  
Name[2] = &c[0];
```

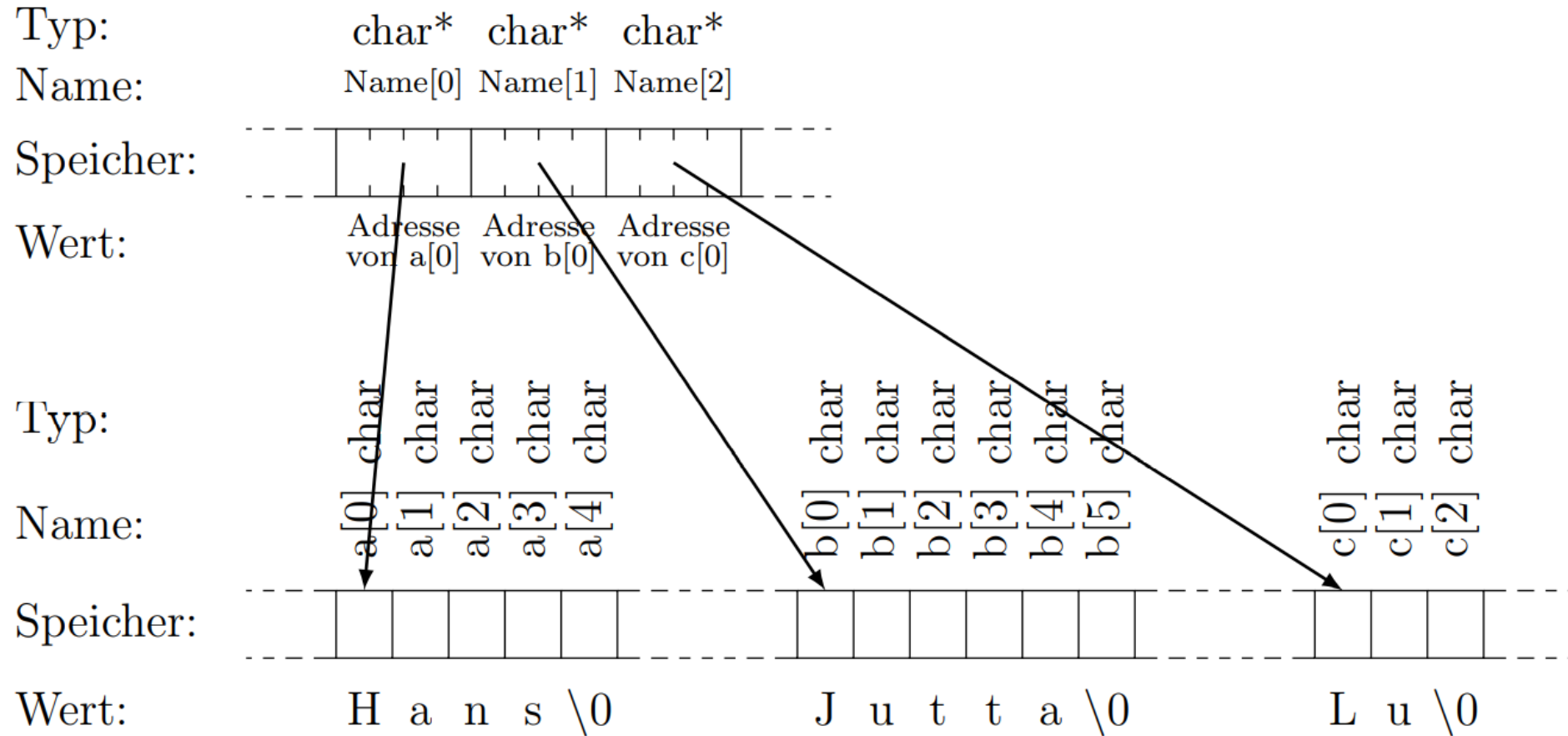
2

```
char *Name[3] = { &a[0], &b[0], &c[0] };
```

3

```
char *Name[] = { a, b, c };
```

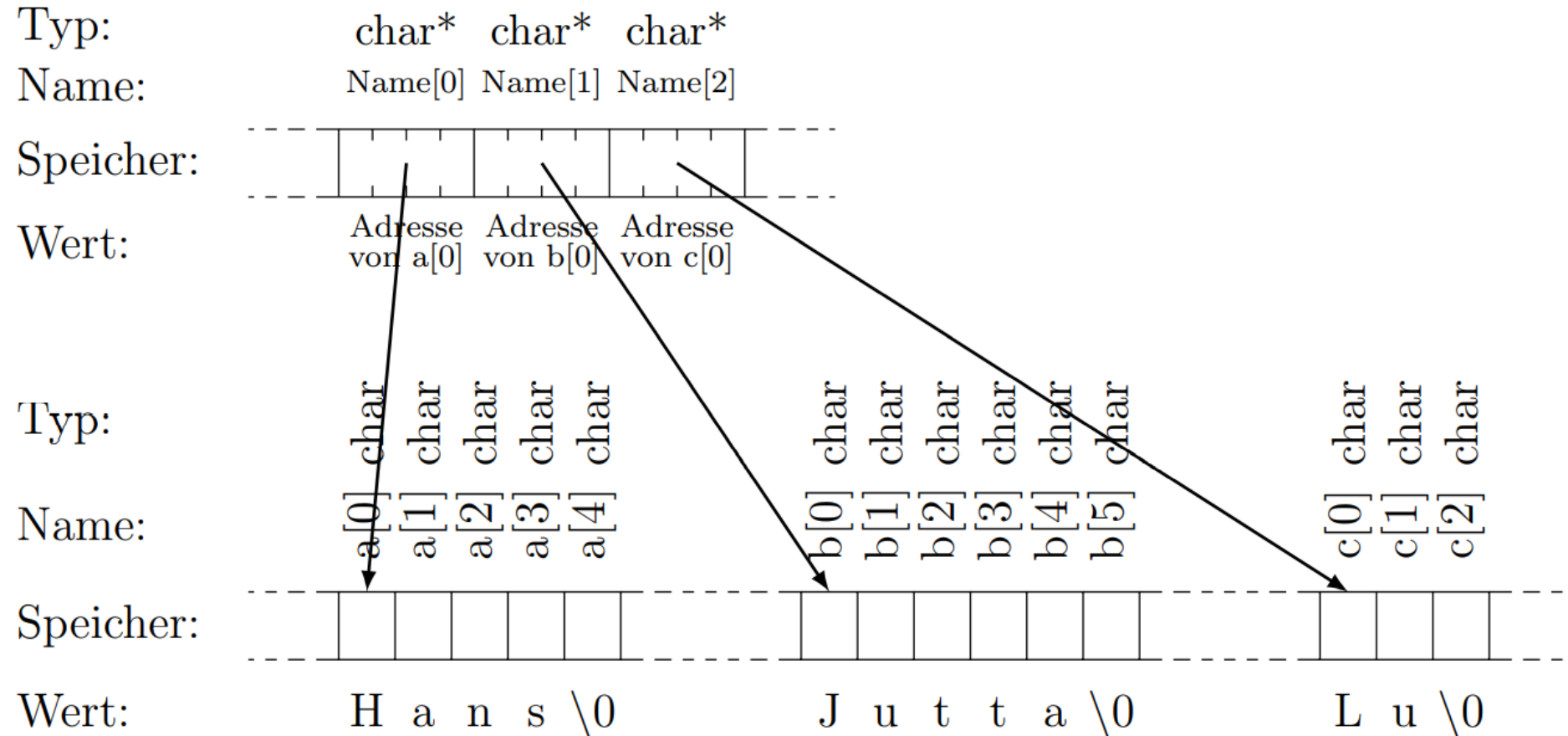
Zeigervektoren - Initialisierung



Was passiert bei

`*Name[0] = 'G';`

?



Arbeiten mit Zeigervektoren

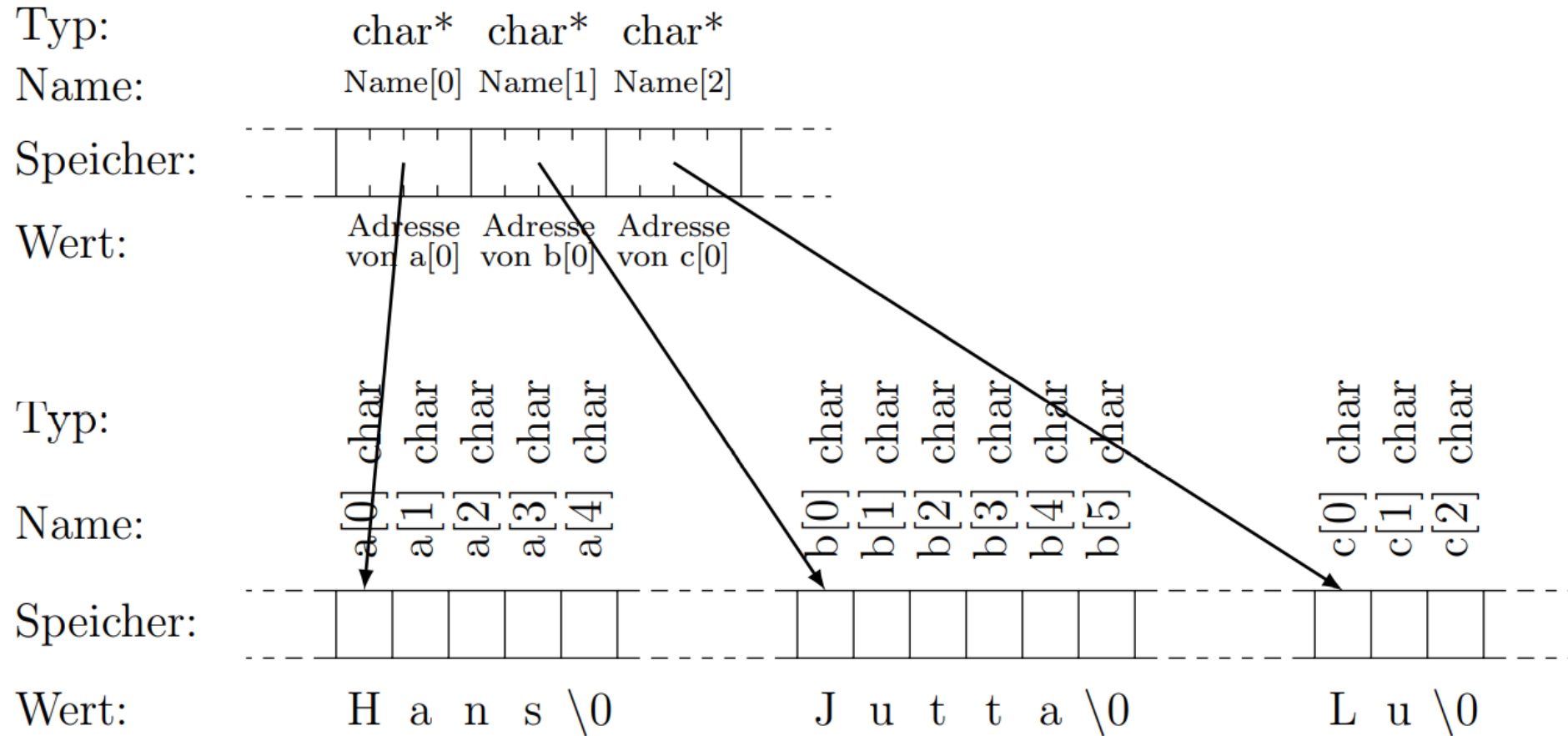
- Was passiert bei folgender Zeile?

```
*Name[0] = 'G';
```

- Zwei Operatoren werden nacheinander angewandt
- `[]` greift auf das erste Element des Zeigervektors zu
- `*` greift auf den Inhalt des Zeigers zu
- Reihenfolge ist definiert durch Rangfolge der Operatoren

Rang	Operatoren	Reihenfolge
1	() [] -> .	von links nach rechts
2	! ~ ++ -- + - * & (type) sizeof	<i>von rechts nach links</i>
3	* / %	von links nach rechts
4	+ -	von links nach rechts
5	<< >>	von links nach rechts
6	< <= > >=	von links nach rechts
7	== !=	von links nach rechts
8	&	von links nach rechts
9	^	von links nach rechts
10		von links nach rechts
11	&&	von links nach rechts
12		von links nach rechts
13	?:	<i>von rechts nach links</i>
14	= += -= *= /= %= &= ^= = <<= >>=	<i>von rechts nach links</i>
15	,	von links nach rechts

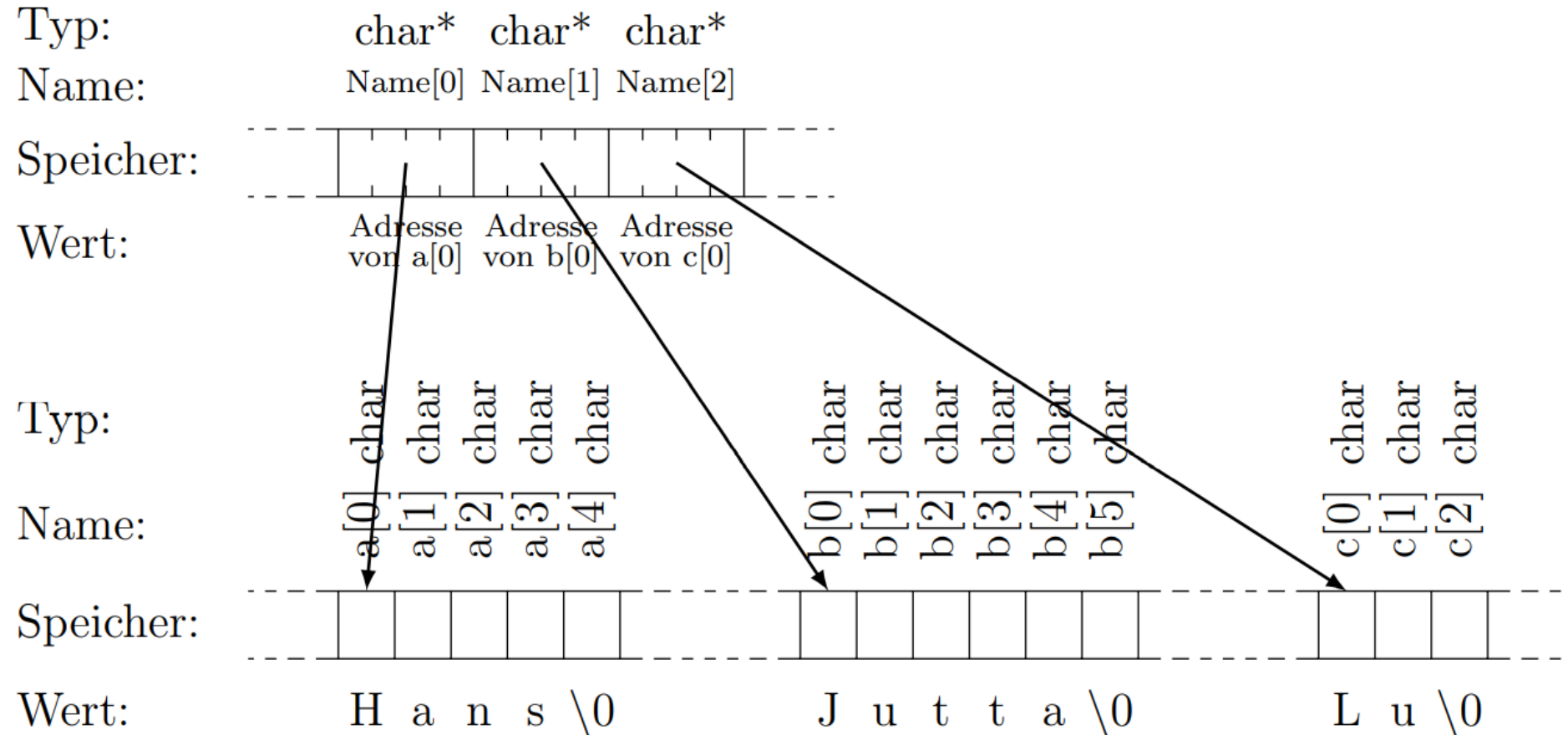
Was passiert bei `*(Name[2] + 1) = 'a';` ?



Was passiert bei

Name[2][1] = 'a';

?



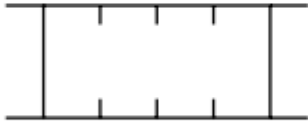

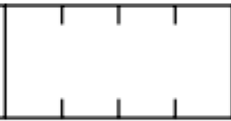

Beispiel: Auswertung der Kommandozeilenparameter

- Beim Aufruf eines Programms können Parameter übergeben werden
 - Auf der Kommandozeile werden die Parameter nach dem Kommando aufgelistet
 - Auch mit Betriebssystemen mit grafischer Oberfläche möglich
 - Beispiel: Verschieben Sie eine Textdatei auf einen Texteditor!
- Wie wertet ein Programm die Parameter aus?
 - In C im Hauptprogramm `main` über die Argumente `argc` und `argv`

```
int main(int argc , char *argv [])
```

- `argc` („argument count“) – Anzahl der übergebenen Parameter (Programmname wird mitgezählt → Anzahl immer größer/gleich 1)
- `argv` („argument values“) – Werte der Parameter als Zeigervektor vom Typ `char`

Auswertung der Kommandozeilenparameter (II)

Typ:	int	char*	char*	char*	...
Name:	argc	argv[0]	argv[1]	argv[2]	...
Speicher:					...
Wert:	Anzahl Parameter	Adresse 1. Par.	Adresse 2. Par.	Adresse 3. Par.	...

Auswertung der Kommandozeilenparameter (III)

→ Beispiel in Visual Studio

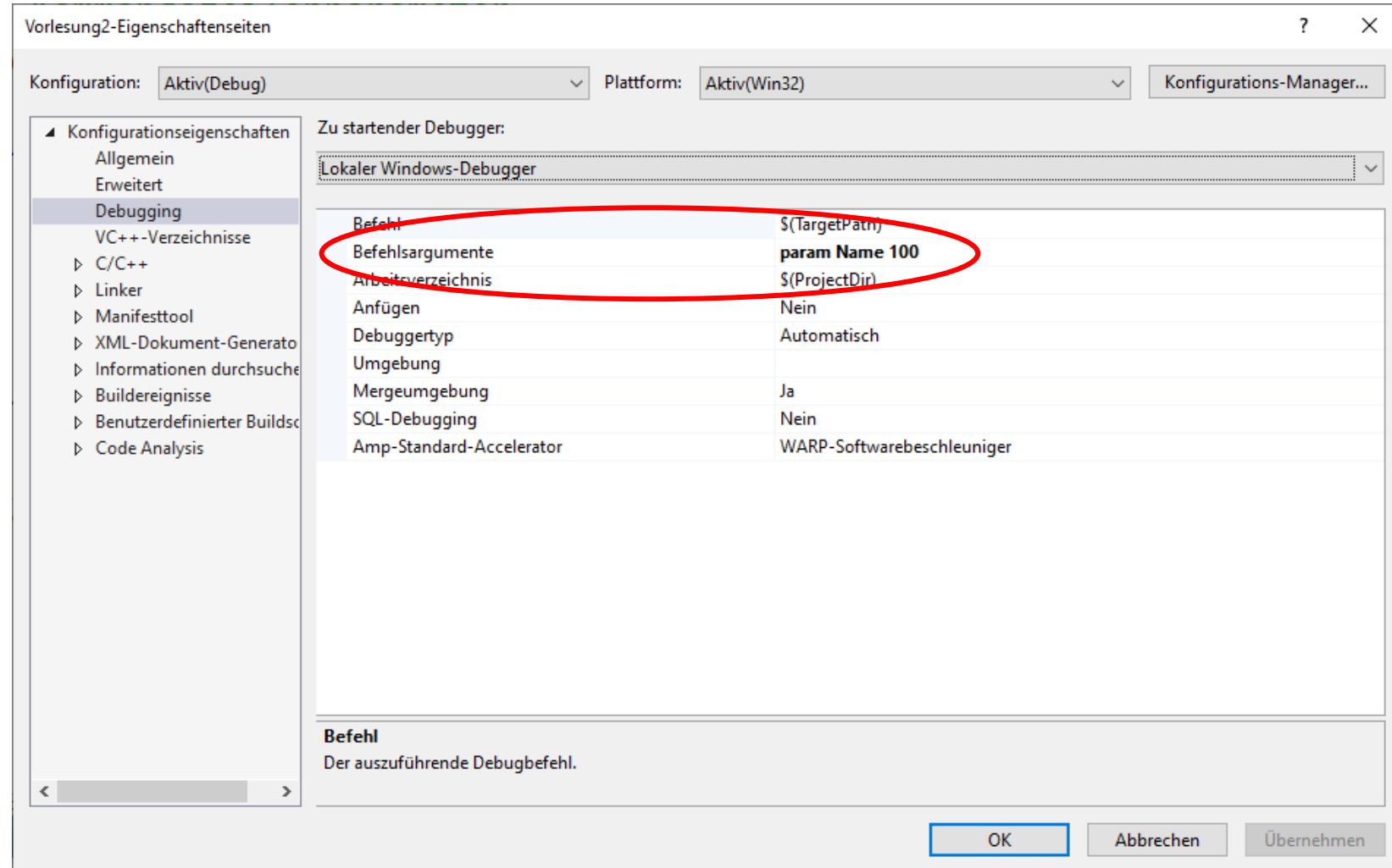
```
#include <stdio.h>

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

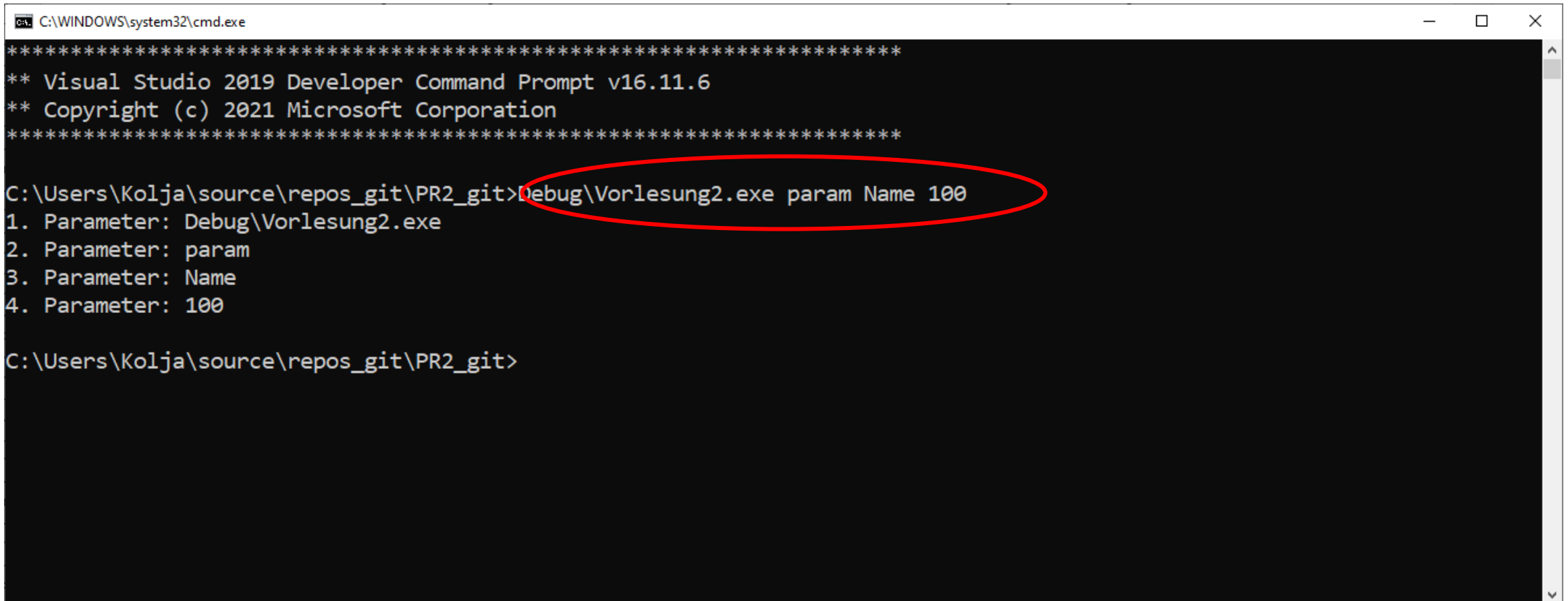
    for (i=0; i<argc; i++)
        printf("%d. Parameter: %s\n", i+1, argv[i]);
}
```


Kommandozeilenparameter in Visual Studio setzen

Projekt → Eigenschaften



Kommandozeilenparameter in Eingabeaufforderung setzen



```
C:\WINDOWS\system32\cmd.exe

*****
** Visual Studio 2019 Developer Command Prompt v16.11.6
** Copyright (c) 2021 Microsoft Corporation
*****

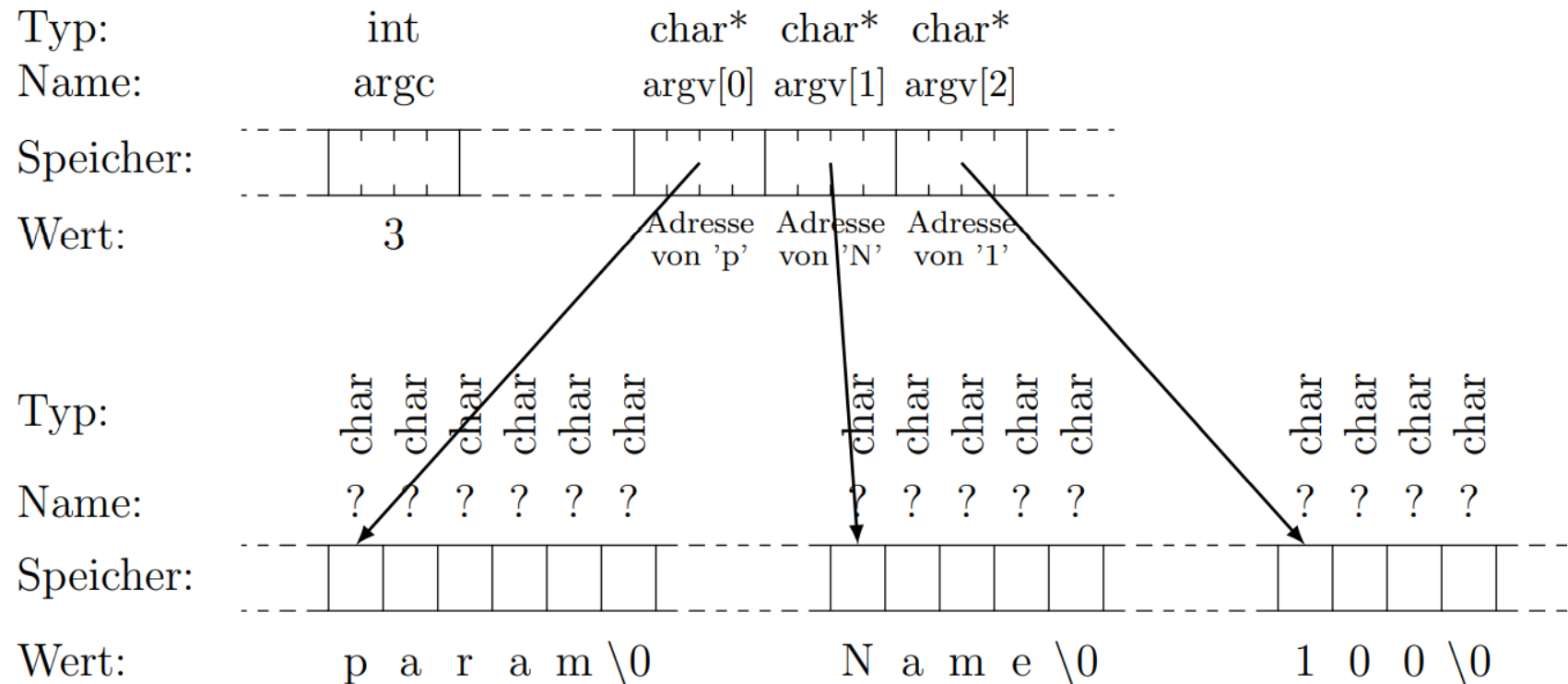
C:\Users\Kolja\source\repos_git\PR2_git>Debug\Vorlesung2.exe param Name 100
1. Parameter: Debug\Vorlesung2.exe
2. Parameter: param
3. Parameter: Name
4. Parameter: 100

C:\Users\Kolja\source\repos_git\PR2_git>
```

The screenshot shows a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.exe". It displays the output of the Visual Studio 2019 Developer Command Prompt v16.11.6, including copyright information. The user has entered the command `Debug\Vorlesung2.exe param Name 100` at the prompt `C:\Users\Kolja\source\repos_git\PR2_git>`. This command line is circled in red. The output shows a list of parameters: `1. Parameter: Debug\Vorlesung2.exe`, `2. Parameter: param`, `3. Parameter: Name`, and `4. Parameter: 100`. The prompt then returns to `C:\Users\Kolja\source\repos_git\PR2_git>`.

Auswertung der Kommandozeilenparameter (IV)

- Im Falle vom Programmaufruf `param Name 100`



Auswertung der Kommandozeilenparameter (V)

- Merke:
 - Prüfe die Anzahl der Parameter vor dem Zugriff auf das Element im Zeigervektor
 - Alle Parameter werden als `char[]` übergeben (auch Zahlen) → Konvertiere den Parameter zum entsprechenden Typ, z.B.
 - `atoi` - wandelt eine ASCII-Zeichenkette in eine ganze Zahl um
 - `atof` - wandelt eine ASCII-Zeichenkette in eine Gleitkommazahl um
 - `sscanf` - wertet eine ASCII-Zeichenkette gemäß einer Formatanweisung um, und speichert die Ergebnisse in eine variable Anzahl an Variablen
 - `strcpy` – kopiert eine Zeichenkette in eine andere
 - z.B. `strcpy(NameEingabedatei, argv[1]);`

ZEIGER AUF ZEIGER (DOUBLE POINTER)

Zeiger auf Zeiger

- Ein einfacher Zeiger zeigt auf Datentypen wie `int` oder `float`
 - Zeiger können auch verschachtelt werden
- es ist möglich einen Zeiger zu definieren, welcher wieder auf einen Zeiger zeigt

Definition eines Doppelzeigers

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

- Beispiel: `int **ppInt;`
- Kann auch als `int* *ppInt` interpretiert werden
- `ppInt` ist ein Zeiger auf den Typ `int*`
- Anzahl der `*` gibt die Verschachtelungstiefe an

Beispiel: Doppelzeiger

→ Visual Studio

```
#include <stdio.h>

int main()
{
    double e = 2.71828;
    double* pe;
    double** ppe;

    pe = &e;
    ppe = &pe;

    printf("Wert der Variable ppe: %p\n", ppe);
    printf("Wert der Variable pe: %p\n", *ppe);
    printf("Wert der Variable e: %lf\n", **ppe);
}
```



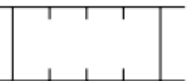

Initialisierung eines Doppelzeigers

- Mittels Adressoperator &
- Beispiel:

```
double e=2.71828;  
double *pe;  
double **ppe;
```

```
pe = &e;  
ppe=&pe;
```

Nach welcher Zeile ergibt sich folgendes Abbild?

Typ:	double	double*	double**
Name:	e	pe	ppe
Speicher:			
Wert:	2,71828	?	?

Initialisierung eines Doppelzeigers

- Beispiel:

```
double e=2.71828;
```

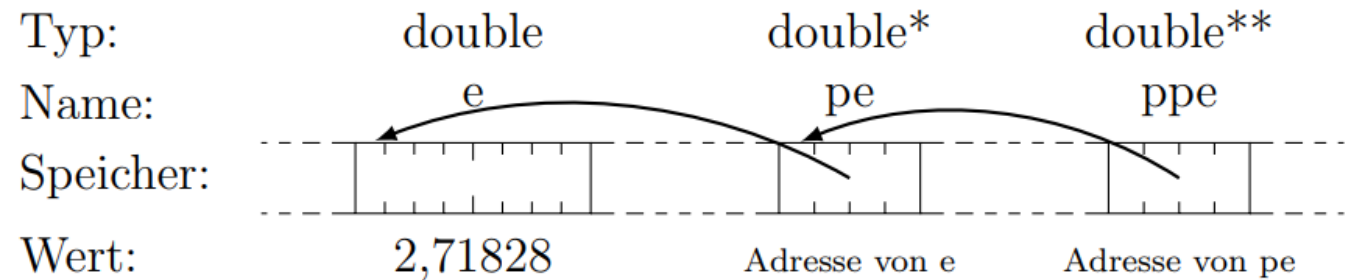
```
double *pe;
```

```
double **ppe;
```

```
pe = &e;
```

```
ppe=&pe;
```

Und jetzt?



Initialisierung eines Doppelzeigers (III)

- Definition und Initialisierung können auch in einem Schritt erfolgen

```
double e=2.71828;  
double *pe=&e;  
double **ppe=&pe;
```

Arbeiten mit Doppelzeigern

```
printf("Wert der Variable ppe: %p\n", ppe);  
printf("Wert der Variable pe: %p\n", *ppe);  
printf("Wert der Variable e: %lf\n", **ppe);
```

- In der ersten Zeile wird die Adresse, die in der Variable `ppe` gespeichert ist, direkt ausgegeben
- In der zweiten Zeile wird mit dem Verweisoperator auf die Variable `pe` zugegriffen und dessen Inhalt ausgegeben
- In der dritten Zeile greift zunächst der rechte Stern mit der Variable `ppe` indirekt auf die Variable `pe` zu. Danach ermittelt der linke Stern mit dem Ergebnis des rechten Sterns den Inhalt der Variable `e`.

Wann sind Doppelzeiger sinnvoll?

- Wenn sie bereits mit Zeigern arbeiten und z.B. eine Funktion mit *Call by Reference* implementieren wollen!
- Beispiel: Dateien
 - Rückgabewerte beim Öffnen einer Datei ist `FILE *`
 - Alle weiteren Funktionen für Dateien erwarten diesen Zeiger

Wie kann man eine **Funktion** implementieren, welche eine Datei öffnet und auf Fehler prüft?

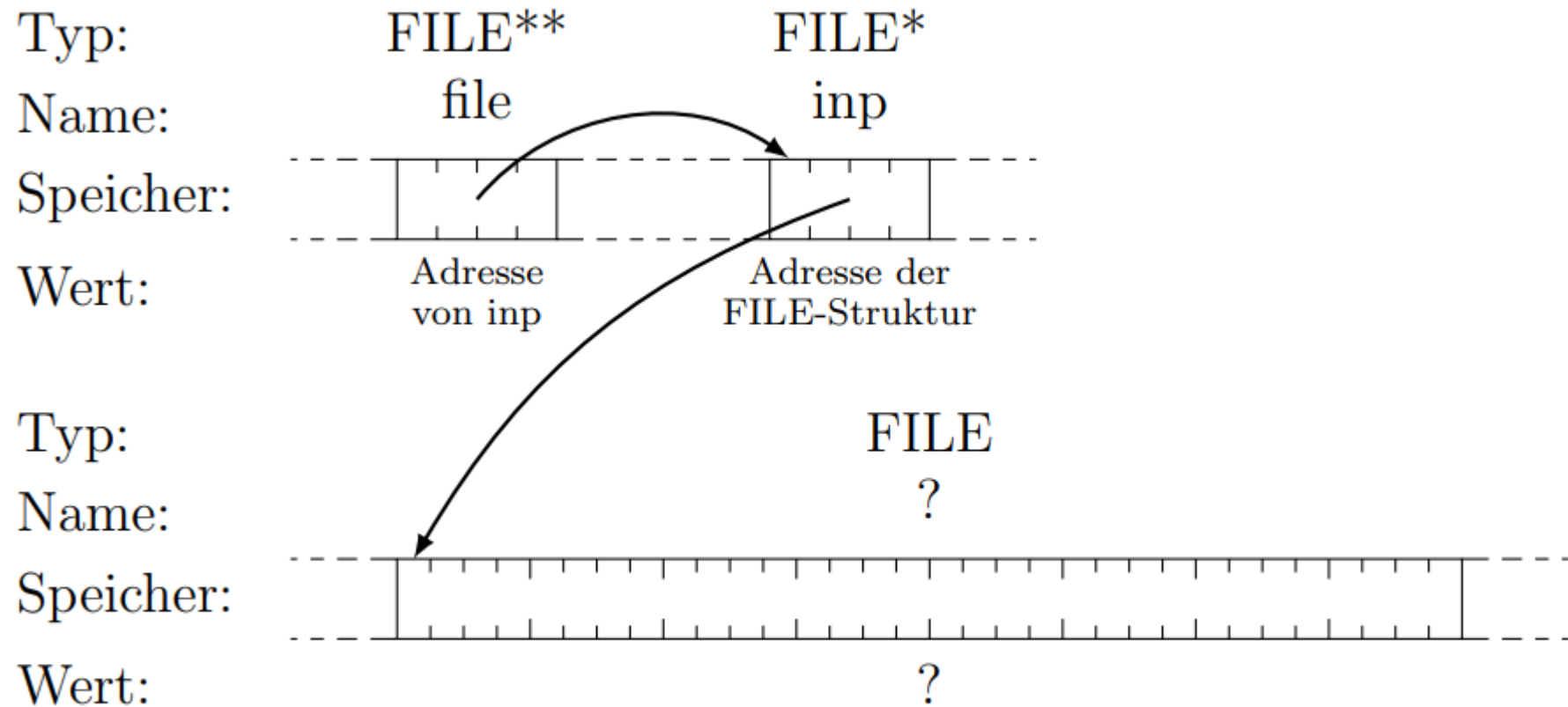
Bespiel: Funktion zum öffnen einer Datei

→ Visual Studio

```
int openFile(char fileName[], char mode[], FILE** file);

int main() {
    FILE* inp;
    if (openFile("Beispiel.txt", "rt", &inp)) return -1;
}

int openFile(char fileName[], char mode[], FILE** file) {
    *file = fopen(fileName, mode);
    if (*file == NULL) {
        printf("Fehler beim oeffnen der Datei %s\n", fileName);
        return -1;
    }
    return 0;
}
```



Damit der Zeiger `FILE *inp` innerhalb der Funktion manipuliert werden kann, muss ein Zeiger auf diesen Zeiger übergeben werden!

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!

Binärdateien – Beispiel 1: Schreiben

→ Visual Studio

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

int main() {
    int i = 27; // ganze Zahl zum Schreiben
    double e = 2.718281828; // Gleitkommazahl zum Schreiben
    int Prim[6] = {2, 3, 5, 7, 11, 13}; // Vektor zum Schreiben
    FILE* out = NULL; // Zeiger für Ausgabedatei

    // Erster Schritt: Datei öffnen
    out = fopen("Beispiel.dat", "wb");
    if (out == NULL)
        printf("Konnte Ausgabedatei nicht \224ffnen: Beispiel.dat\n");

    // Zweiter Schritt: Datei verarbeiten
    if (out) {
        fwrite(&i, sizeof(int), 1, out);
        fwrite(&e, sizeof(double), 1, out);
        fwrite(Prim, sizeof(int), 6, out);
    }

    // Dritter Schritt: Datei schließen
    if (out) fclose(out);

    // Schlussmedung
    printf("Datei erfolgreich erstellt\n");

    return 0;
}
```

Binärdateien – Beispiel 2: Lesen → Visual Studio

```
int main()
{
    int i; /* ganze Zahl zum Lesen */
    double e; /* Gleitkommazahl zum Lesen */
    int Prim[6]; /* Vektor zum Lesen */
    FILE* inp = NULL; /* Zeiger für Eingabedatei */

    /* Erster Schritt: Datei öffnen */
    inp = fopen("Beispiel.dat", "rb");
    if (inp == NULL)
        printf("Konnte Eingabedatei nicht \224ffnen:
        Beispiel.dat\n");

    /* Zweiter Schritt: Datei verarbeiten */
    if (inp) {
        fread(&i, sizeof(int), 1, inp);
        fread(&e, sizeof(double), 1, inp);
        fread(Prim, sizeof(int), 6, inp);
    }

    /* Dritter Schritt: Datei schließen */
    if (inp) fclose(inp);

    /* Ergebnisse ausgeben */
    if (inp) {
        printf("i: %d\n", i);
        printf("e: %.15g\n", e);
        printf("Prim[0]: %d\n", Prim[0]);
        printf("Prim[1]: %d\n", Prim[1]);
        printf("Prim[2]: %d\n", Prim[2]);
        printf("Prim[3]: %d\n", Prim[3]);
        printf("Prim[4]: %d\n", Prim[4]);
        printf("Prim[5]: %d\n", Prim[5]);
    }

    return 0;
}
```