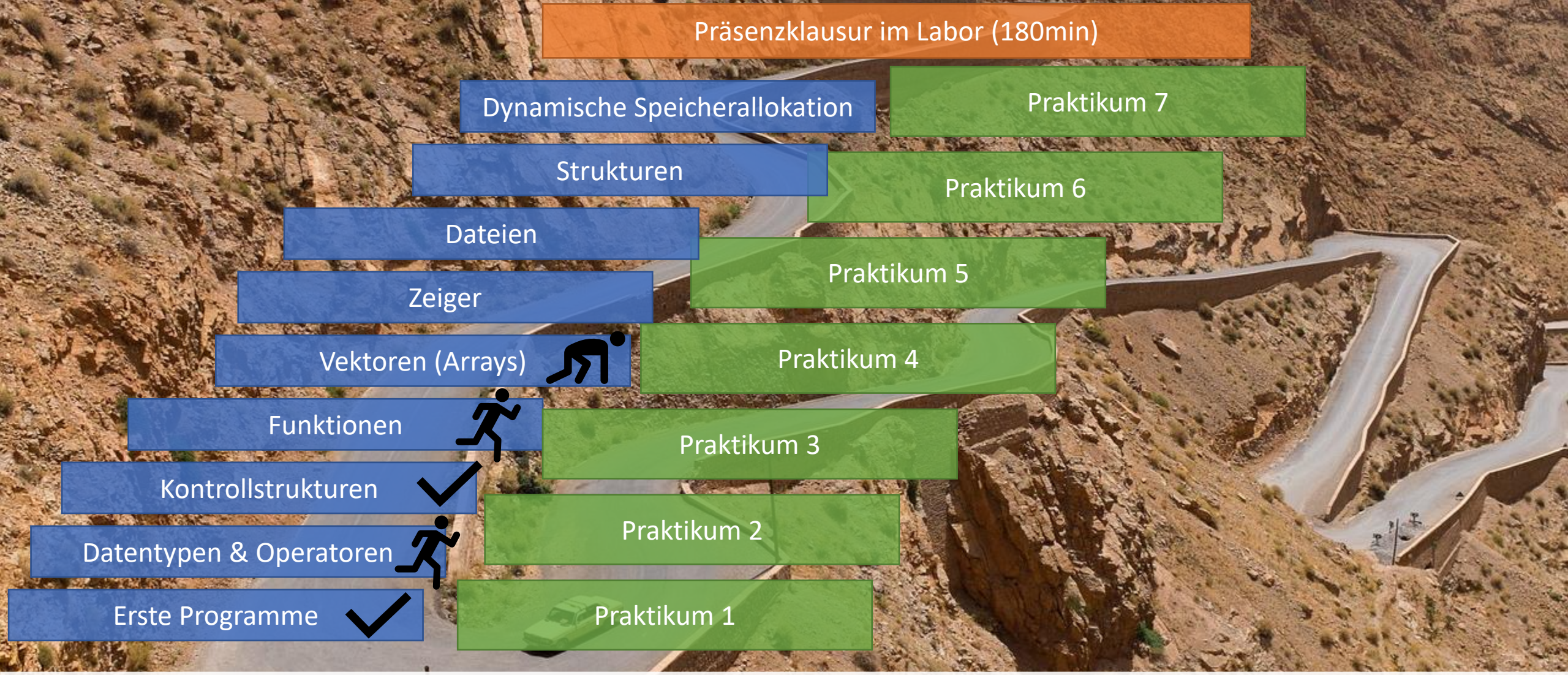
PROGRAMMIEREN I

WS 2022

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

HAW HAMBURG

Check-In



Unser Weg durch das Semester

FUNKTIONEN

HAW
HAMBURG

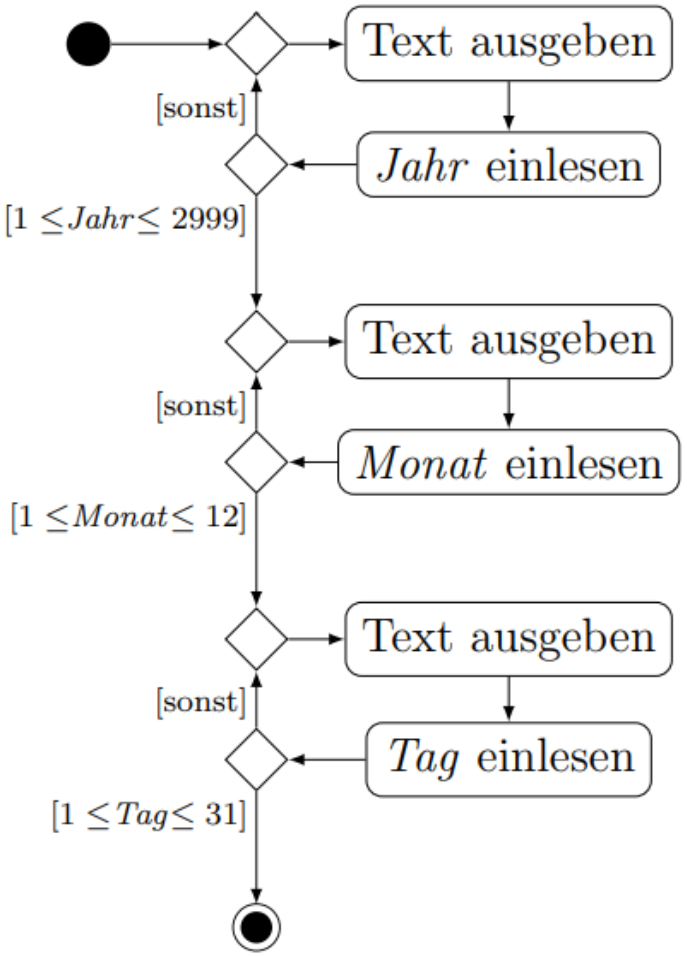
Funktionen (Wiederholung)

- Was sind Vorteile von Funktionen?
- Was steht im Funktionskopf?
- Was steht im Funktionsrumpf?
- Was ist der Unterschied zwischen Funktions-Deklaration und Definition?
- Wo im Quellcode steht die Funktions-Deklaration?
- Was ist *void*?

Beispiel: Benutzereingabe prüfen

- Ein Benutzer soll ein Datum eingeben
- Die Werte für Jahr, Monat und Tag werden nacheinander eingelesen
- Bei der Eingabe sollen die Werte auf Plausibilität geprüft werden (siehe Aktivitätendiagramm)

- Die Prüfung soll in einer Funktion implementiert werden



Beispiel: Benutzereingabe prüfen (II)

Funktions-definition:

```c
int getInt(char Text[], int min, int max)
{
    int Zahl;

    do {
        printf("%s (%d-%d): ", Text, min, max);
        scanf("%d", &Zahl);

        // mit der folgenden Zeile wird der Eingabe-Puffer geleert
        while (getchar() != '\n') {}

    } while (Zahl<min || Zahl>max);

    return Zahl;
}
```

Aufruf:

```c
    Jahr = getInt("Geben Sie das Jahr ein", 1, 2999);
    Monat = getInt("Geben Sie den Monat ein", 1, 12);
    Tag = getInt("Geben Sie den Tag ein", 1, 31);
    printf("Datum: %d.%d.%d\n", Tag, Monat, Jahr);
```

Übung

- Es soll eine Funktion mit dem Namen *fill* erstellt werden, mit der ein beliebiges Zeichen beliebig oft auf dem Bildschirm ausgegeben werden soll.
- Mit dieser Funktion können z.B. horizontale Trennstriche in einer Anwendung erzeugt werden.
- Um z.B. fünfzig Sterne auszugeben soll der Aufruf der Funktion folgendermaßen aussehen:

```
fill('*', 50);
```

GLOBALE UND LOKALE VARIABLEN

HAW HAMBURG

Lokale Variablen

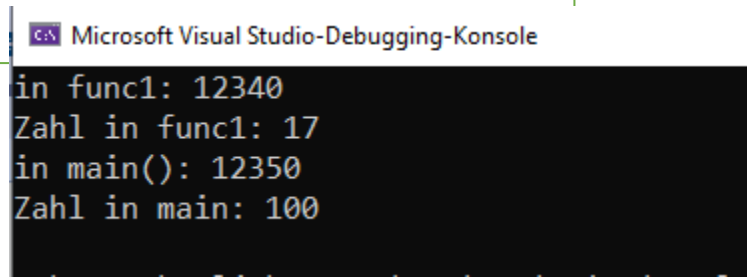
- In den bisherigen Beispielen wurden Variablen nur innerhalb einer Funktion definiert (denn auch *main()* ist eine Funktion)
- Eine solche Variable ist nur innerhalb der jeweiligen Funktion bekannt
- Wenn Sie außerhalb einer Funktion auf eine lokale Variable zugreifen wollen, so wird der Compiler einen Fehler melden
- Da lokale Variablen nur in der eigenen Funktion bekannt sind, können Namen in unterschiedlichen Funktionen auch doppelt verwendet werden.
  - Der Compiler wird sie als unterschiedliche Variablen behandeln
  - Vorteil: Wenn sie eine Funktion fertiggestellt haben, werden lokale Variablen verworfen und Sie müssen sich nicht weiter kümmern

```c
#include <stdio.h>

void func1(void);

int main(void)
{
    int a=5;
    func1();
    return 0;
}

void func1(void) {
    int a=-5;
    /* ... */
}
```

Globale Variablen

- Werden Variablen außerhalb von Funktionen definiert, so kann in allen Funktionen diese Variable genutzt werden ➔ globale Variable
- Wird in einer Funktion eine lokale Variable mit dem gleichen Namen definiert, so erstellt der Compiler zwei getrennte Variablen
- Innerhalb dieser Funktion, steht der Name für die lokale Variable
- Außerhalb für die globale Variable
- **Vermeiden Sie globale Variablen!**
  - Da es schwerer nachvollziehbar ist, wo sich eine globale Variable geändert hat
  - Fehleranfälliger mit gleichnamigen Variablen

```c
#include <stdio.h>

void func1(void);

int global = 12345;
int Zahl = 100;

int main(void)
{
    int a=5;
    func1();
    printf("in main(): %d\n", a + global);
    printf("Zahl in main: %d\n", Zahl);

    return 0;
}

void func1(void) {
    int a=-5;
    int Zahl;
    /* ... */
    Zahl = 17;
    printf("in func1: %d\n", a+global);
    printf("Zahl in func1: %d\n", Zahl);
}
```



Blöcke {}

- Sowohl in Funktionen als auch in Schleifen haben wir geschweifte Klammern {} verwendet
- Hiermit werden Blöcke definiert
- Blöcke lassen sich verschachteln (siehe Schleifen)
- Lokale Variablen beziehen sich auf Blöcke und werden beim Aufruf des Blockes angelegt und beim Verlassen wieder ungültig
- Variablen in inneren Blöcken sind nach außen hin nicht sichtbar
- Auch hier können in unterschiedlichen Blöcken die gleichen Namen für Variablen verwendet werden (**bitte vermeiden – fehleranfällig!**)

Beispiel:

```
int i = 1;
/* .. */
for (int i = 0; i < 10; i++) {
    /* do something */
    printf("%d\n", i);
}
printf("%d\n", i);
```

Die Zählvariable *i* wird erst in der for-Schleife definiert und ist auch nur innerhalb der for-Schleife gültig
(das sieht man häufiger!)

Falls auch außerhalb der for-Schleife die Variable mit Namen *i* definiert worden ist, kann es verwirrend werden..
(das bitte vermeiden!)

Wie ist die Ausgabe des folgenden Programms?

```c
#include <stdio.h>

void Funktion1();
void Funktion2();

int Zahl;

int main()
{
    Zahl = 25;

    printf("Main:      Zahl=%d\n", Zahl);
    Funktion1();
    Funktion2();
    Funktion1();
    printf("Main:      Zahl=%d\n", Zahl);

    return 0;
}
```

```c
void Funktion1()
{
    printf("Funktion1: Zahl=%d\n", Zahl);
}

void Funktion2()
{
    int Zahl;

    Zahl = 17;
    printf("Funktion2: Zahl=%d\n", Zahl);
}
```

VEKTOREN (ARRAYS)

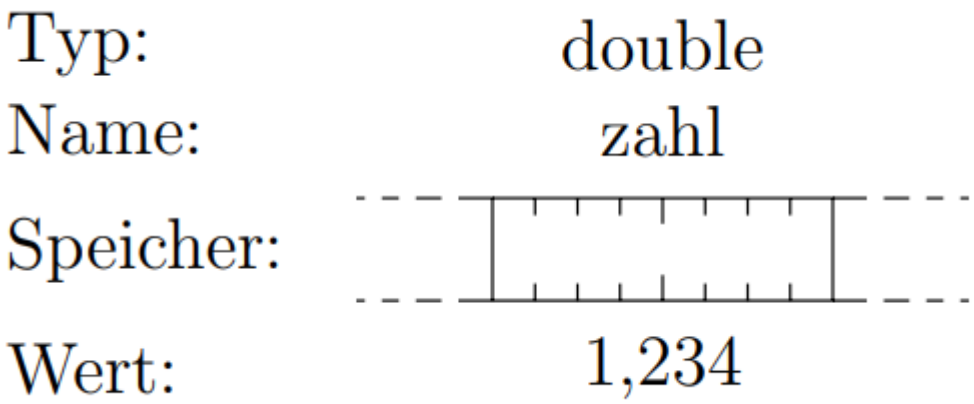
HAW HAMBURG

Eine Variable hat 4 Eigenschaften

Wiederholung

- 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;
```



Vektoren (Arrays)

- Häufig benötigt man mehrere verkettete Variablen vom gleichen Typ
- Beispiele:
  - Messreihe (Messwerte über die Zeit)
  - Zeichenketten
  - Darstellung einer mathematischen Funktion durch eine Liste
- Mehrdimensionale Vektoren
  - Pixel eines Bildes (2D)
  - Temperaturverteilung in einem Raum (3D)
  - Temperaturverteilung in einem Raum über die Zeit (4D)

➔ In C wird eine Verkettung von Variablen mittels Vektoren (Arrays) implementiert

Eindimensionale Vektoren

- **Allgemein** lautet die Syntax für Vektoren

```
<Datentyp> <Variablenname>[<Anzahl>]
```

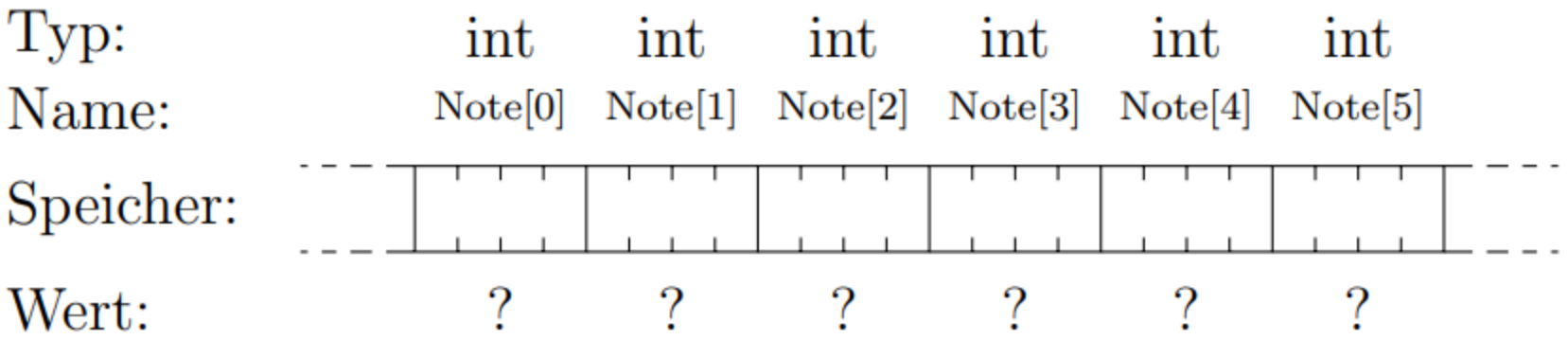
Beispiel: Um einen Notenspiegel zu berechnen, definieren Sie einen Vektor mit der Anzahl der Einsen, Anzahl der Zweien, .. und der Sechsen

```
int Noten[6];
```

Vektoren – Beispiel: Notenspiegel

int Noten[6];

- Im Arbeitsspeicher wird Platz für sechs int-Variablen geschaffen (allokiert)
- Alle Variablenwerte sind noch undefiniert



→ Beispiel in Visual Studio

```
int main(void) {
	// 1) Definition eines Arrays
	int Noten[6];

	// 5) Option 1
	int Noten_v1[6] = { 0,0,0,0,0,0 };// Initialisierung direkt bei
	der Definition
	// Option 2
	int Noten_v2[] = { 0,0,0,0,0,0 };
	// Option 3
	int Noten_v3[6] = { 0 };
	// Option 4
	int Noten_v4[6] = { 0, 1, 2 };

	// 2) Initialisierung nach Definition
	Noten[0] = 0;// Zugriff auf Elemente des Vektors mit eckigen
	Klammern und Index
	Noten[1] = 0;// Achtung: Index startet mit null! Und endet eins
	vor der Anzahl der Elemente
	Noten[2] = 0;
	Noten[3] = 0;
	Noten[4] = 0;
	Noten[5] = 0;
	//...
	//...
	// 3) Zugriff ausserhalb der Grenzen des Arrays führt zu
	schweren Fehlern (insbes. Laufzeit)
	Noten[6] = 0;// schwerer Fehler - nur Warnung!

	// 4) oder mithilfe einer Schleife
	for (int i = 0; i < 6; i++)
		Noten[i] = 0;

	// Arbeiten mit Vektoren
	Noten[2]++;// Zugriff mit eckigen Klammern und Index.
	Hier: Erhöhung der Dreien um Eins

	int Bestanden = Noten[0] + Noten[1] + Noten[2] +
	Noten[3];// Anzahl aller bestandenen Studenten

	int Zahl = 3;
	if (Zahl >= 1 && Zahl <= 6)
		Noten[Zahl - 1]++;// Index kann Ergebnis eines
	Ausdruck seins

	return 0;
}
```

Initialisierung eines Vektors – Weg 1: Nach der Definition

- Zugriff auf Elemente mit eckigen Klammern und Index
- Achtung: Index startet mit null und endet eins vor der Anzahl der Elemente
- Beispiel:

```
// Initialisierung nach Definition
Noten[0] = 0
Noten[1] = 0
Noten[2] = 0;
Noten[3] = 0;
Noten[4] = 0;
Noten[5] = 0;
```

Initialisierung eines Vektors – Weg 2: Direkt bei der Definition

- Werte können in geschweiften Klammern bei der Definition angegeben werden
- Elemente werden durch Komma getrennt
- Anzahl der Elemente muss nicht in eckigen Klammern angegeben werden, sondern kann über die Anzahl der Elemente in geschweiften Klammern abgeleitet werden
- Beispiel:

```
int Noten_v2[] = { 0,0,0,0,0,0 };
```

- Falls Anzahl der Elemente in eckigen Klammern angegeben ist und die Initialisierung nicht alle Elemente enthält, werden alle anderen Elemente auf null gesetzt
- Beispiel:

```
int Noten_v3[6] = { 0 };
```

Arbeiten mit Vektoren

- Zugriff auf Elemente mit eckigen Klammern und Index
- Achten Sie auf die Grenzen des Arrays!
- Beispiele:

```
Noten[2]++;
```

```
int Bestanden = Noten[0] + Noten[1] + Noten[2] + Noten[3];
```

```
int Zahl = 3;
if (Zahl >= 1 && Zahl <= 6)
        Noten[Zahl - 1]++;
```

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[]);
```

→ Beispiel in Visual Studio

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;
}
```

Zeichenketten

- Auch Zeichen (*char*) können zu Vektoren verkettet werden und man erhält eine Zeichenkette
- Beispiel:

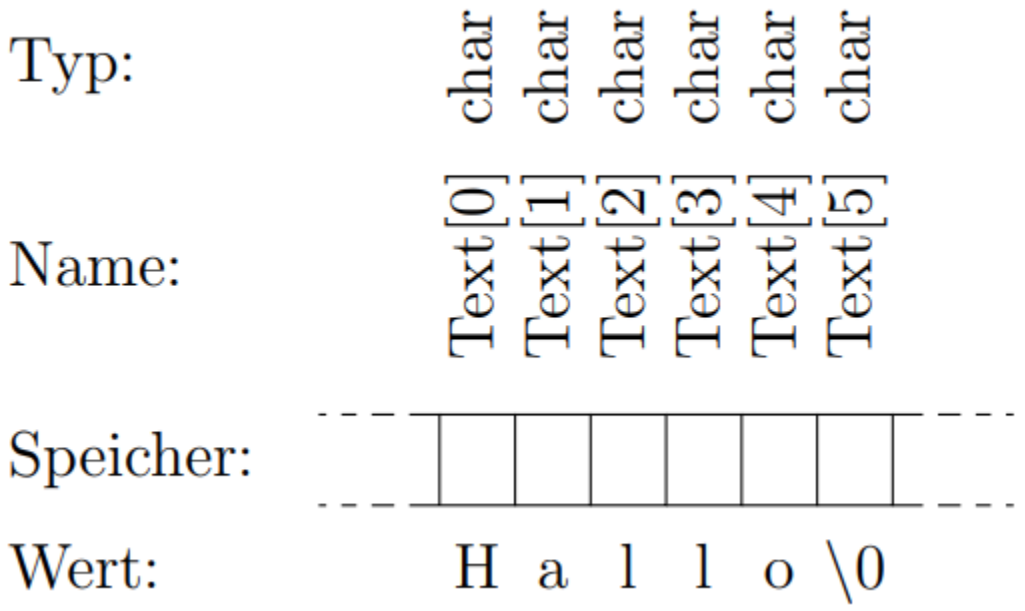
```
char Text[] = { 'H', 'a', 'l', 'l', 'o', '\0' };
```

- Für Zeichenketten kann auch eine vereinfachte Schreibweise zur Initialisierung genutzt werden

```
char Text[] = "Hallo";
```

Zeichenketten (II)

- Zeichenketten werden mit `'\0'` terminiert
- Hierfür ist ein zusätzliches Element notwendig
- Auch als Funktionsparameter können Zeichenketten übergeben werden
- Das Ende wird am Nullzeichen erkannt (`'\0'`)



Übung

- Bei uns werden Notenpunkte vergeben (siehe rechts)
- Erstellen Sie ein Array für die Notenpunkte
- Berechnen Sie den Notendurchschnitt für folgende Verteilung und geben sie ihn aus

1x 15 Punkte	4x 8 Punkte
3x 14 Punkte	3x 7 Punkte
1x 13 Punkte	4x 6 Punkte
0x 12 Punkte	2x 5Punkte
7x 11 Punkte	1x 4 Punkte
3x 10 Punkte	1x 3 Punkte
5x 9 Punkte	1x 0 Punkte

Zusatzaufgabe: Geben Sie zu dem Notendurchschnitt auch die Benotung in Worten aus (z.B. „gut“).

Notenpunkte	Dezimalzahlen-bewertung		Note (Benotung)
15	0.7	=	ausgezeichnet
14 und 13	1.0 und 1.3	=	sehr gut
12, 11 und 10	1.7, 2.0 und 2.3	=	gut
9, 8 und 7	2.7, 3.0 und 3.3	=	befriedigend
6 und 5	3.7 und 4.0	=	ausreichend
4 bis 0 4 3 2 bis 0	4.3 bis 5.0 4.3 4.7 5.0		nicht ausreichend

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!

HAW HAMBURG