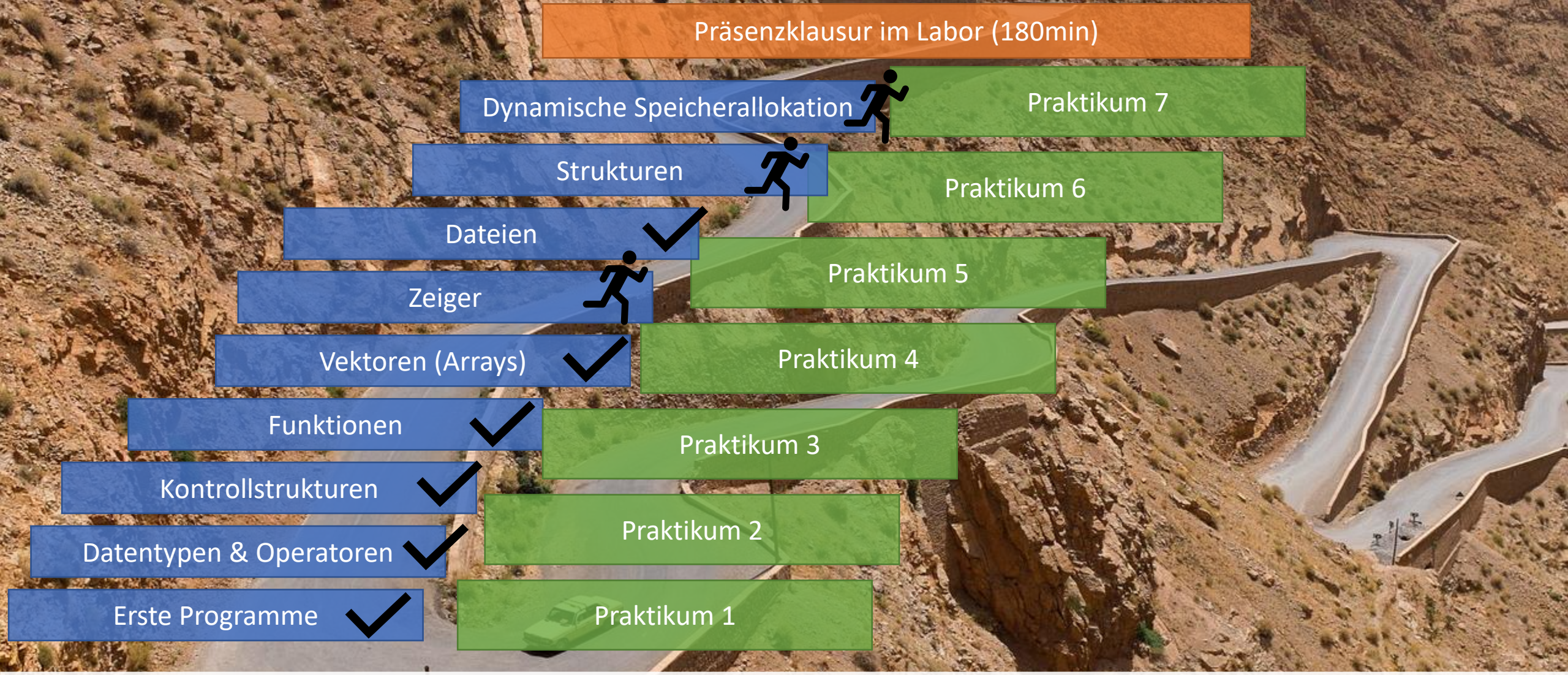
PROGRAMMIEREN I

WS 2022

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

HAW HAMBURG



Unser Weg durch das Semester

„Restprogramm“

- Heute ist die vorletzte Vorlesung
- Letzte Vorlesung am 13.1.:
  - Evtl. „Reste“
  - Wiederholung
  - Ausblick
- Praktikum 7:
  - Aufgabe erst im Termin --> „Übungsklausur“
  - Zusatzaufgabe als Vorbereitung

Bereiten Sie Fragen vor!
(am besten vorab per E-Mail schicken)

Welche Themen sollen vor allem wiederholt werden? Wünsche?

Wie sieht die Klausur aus?

- Präsenzklausur im Labor (kurzfristige Änderungen wegen Corona nicht ausgeschlossen!)
- Programmieraufgabe (180min)
  - Am Labor-PC (Visual Studio in Englisch!)
  - Hilfsmittel:
    - Slides zur Vorlesung, Skripte für PR1 und PR2 von Prof. R. Heß (werden digital auf der Prüfungsumgebung bereitgestellt)
    - Online-Hilfe von Visual Studio (https://docs.microsoft.com/)
    - Was dürfen Sie mitbringen? **1 DIN A4-Seite** mit **handgeschriebenen** Notizen (auch doppelseitig beschriftet)

Was machen wir heute?

Dyn. Speicher

- Wiederholung

Spez. Datentypen

- Strukturen
- Eigene Datentypen
- Aufzählungen

DYNAMISCHE SPEICHERVERWALTUNG

HAW HAMBURG

Speicherverwaltung - Wiederholung

- Wie funktioniert eine **statische** Speicherverwaltung?
- Wann benötigen wir eine **dynamische** Speicherverwaltung?
- Wie wird dynamischer Speicher angelegt?
- Welche weiteren Schritte sind wichtig?
- Welche typischen Fehler können auftreten?

Finde den Fehler!

1

```
int p = malloc(sizeof(int));
```

2

```
double* p2 = malloc(sizeof(float));
*p2 = 7.5;
```

3

```
long* p31 = calloc(10, sizeof(long));
long* p32 = calloc(10, sizeof(long));
p31 = p32;
```

4

```
char* s = calloc(100, sizeof(char));
// ..
free(*s);
```

Was macht der folgende Code?

Was fehlt oder ist nicht gut gelöst?

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    int* value;
    int size, i = 0;

    printf("Wie viele Werte benoetigen Sie : ");
    scanf("%d", &size);

    value = malloc(size * sizeof(int));
    if (NULL == value) {
        printf("Fehler bei malloc....\n");
        return -1;
    }
    while (i < size) {
        printf("Wert fuer value[%d] eingeben : ", i);
        scanf("%d", &value[i]);
        i++;
    }
    printf("Hier Ihre Werte\n");
    for (i = 0; i < size; i++)
        printf("value[%d] = %d\n", i, value[i]);

    return 0;
}
```

SPEZIELLE DATENTYPEN

→ Strukturen
→ Aufzählungen
→ Union (wird in PR1 nicht im Detail behandelt!)
→ Eigene Datentypen

HAW HAMBURG

Wiederholung – Strukturen in C

- Was ist eine Struktur?
- Nennen Sie mehrere Beispiele für die Verwendung einer Struktur?
- Mit welchem Schlüsselwort lassen sich Strukturen in C deklarieren?
- Was ist der Unterschied zwischen einer Deklaration und einer Definition?

→ Visual Studio

```c
#include <stdio.h>

// erste Struktur deklarieren
struct sDatum {
    int Tag;
    char Monat[10];
    int Jahr;
};

void main() {
    // Variable von neuer Struktur definieren
    struct sDatum Datum = { 9, "November", 2021 };

    // Ausgabe struct -> structName.ElementName
    printf("Datum: %d.%s %d\n", Datum.Tag, Datum.Monat, Datum.Jahr);

    // Werte können auch später zugewiesen werden
    struct sDatum Datum2;

    Datum2.Tag = 9;
    strcpy(Datum2.Monat, "November");
    Datum2.Jahr = 2021;

    printf("Datum(2): %d.%s %d\n", Datum2.Tag, Datum2.Monat, Datum2.Jahr);

    …
}
```

Arbeiten mit Strukturen

```
Datum.Tag = 11;
```

Um auf ein Element in der Struktur zuzugreifen, wird ein Punkt verwendet

```
Datum.Tag = Datum.Tag-9;
```

Elemente können wie einzelne Variablen verwendet werden

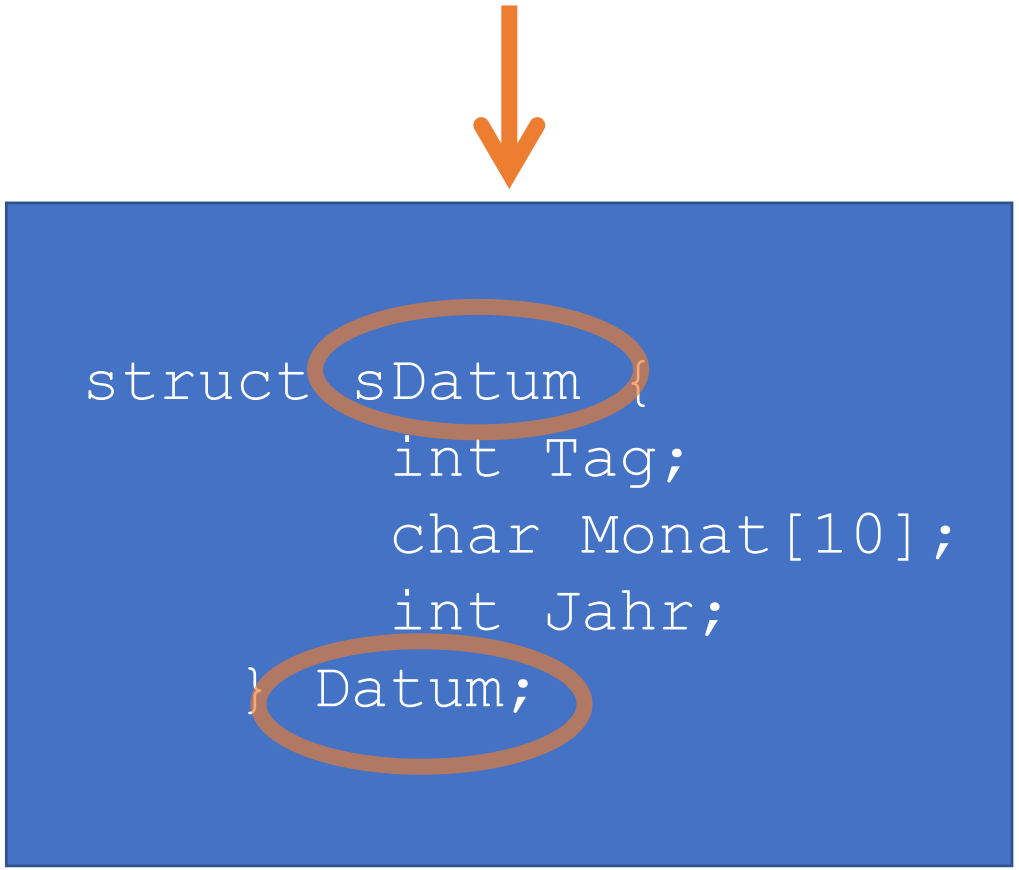
```
Kopie = Datum;
```

Strukturen können als ganzes kopiert werden (auch wenn sie Vektoren als Elemente enthalten)

```
(*pDatum).Tag = 17; //1.W
pDatum->Tag = 17; //2.Weg
```

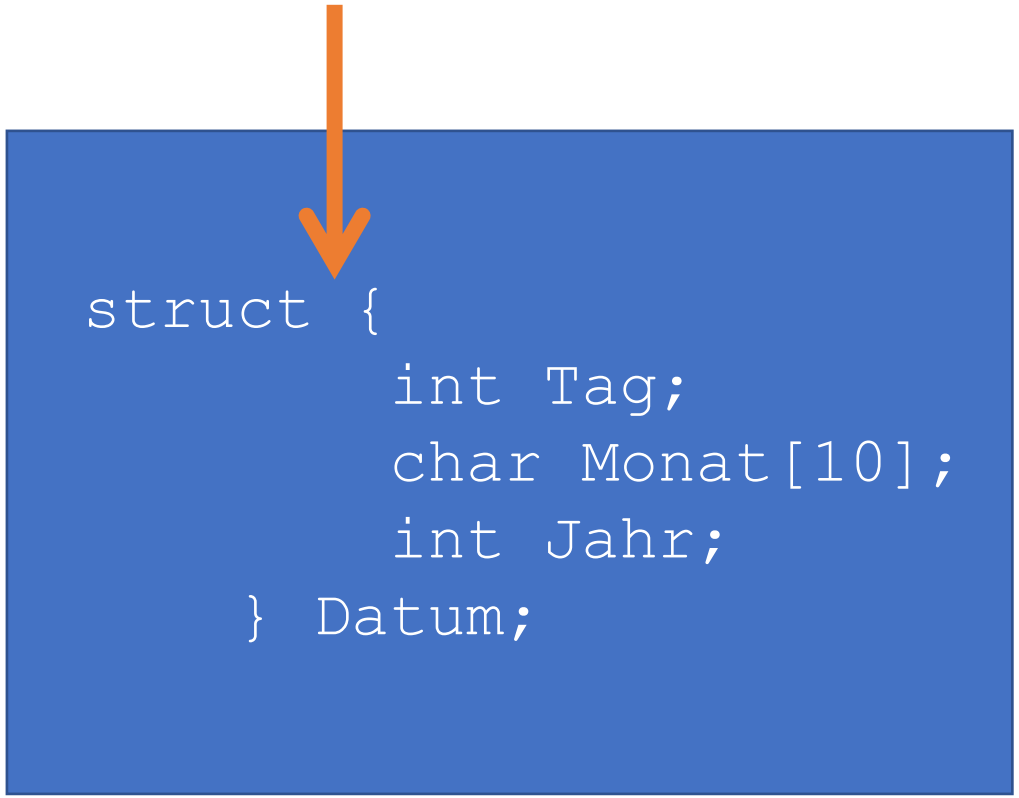
2 Möglichkeiten um mit Zeigern auf Strukturen zu arbeiten

Deklaration & Definition kann kombiniert werden



```
struct sDatum {
        int Tag;
        char Monat[10];
        int Jahr;
    } Datum;
```

Name für die Deklaration kann weggelassen werden, wenn nur an einer Stelle gebraucht



```
struct {
        int Tag;
        char Monat[10];
        int Jahr;
    } Datum;
```

→ Visual Studio

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>

void main() {

    // Deklaration & Definition einer Struktur in einem Schritt
    struct sDatum {
       int Tag;
       char Monat[10];
       int Jahr;
    } Datum;

    Datum.Tag = 9;
    strcpy(Datum.Monat,"November" );
    Datum.Jahr = 2021;

    // Ausgabe struct
    printf("Datum: %d.%s %d\n", Datum.Tag, Datum.Monat, Datum.Jahr);


    // Struktur für einmalige Verwendung der Variablen
    struct {
        int Tag;
        char Monat[10];
    } DatumOhneJahr;

    DatumOhneJahr.Tag = 13;
    strcpy(DatumOhneJahr.Monat, "November");

    printf("Datum: %d.%s\n", DatumOhneJahr.Tag, DatumOhneJahr.Monat);
}
```

Verschachtelte Strukturen

- Strukturen können in anderen Strukturen verwendet werden
- Beispiel →

```
struct sDatum {
    int Tag, Monat, Jahr;
};

struct sPerson {
    char Nachname[31];
    char Vorname[31];
    struct sDatum Geburt;
};

struct sSchulKlasse {
    char KlassenName[11];
    int AnzahlSchueler;
    struct sPerson Schueler[50];
};
```

Verschachtelte Strukturen

- Elemente einer verschachtelten Struktur werden entsprechend verschachtelt angesprochen

```
struct sSchulKlasse Klasse;
Klasse.Schueler[3].Geburt.Jahr = 1995;
strcpy(Klasse.Schueler[5].Nachname, "Mustermann");
Klasse.AnzahlSchueler = 23;
```

Verschachtelte Strukturen

- Strukturen können auch innerhalb einer anderen Struktur deklariert werden

```
struct sPerson {
    char Nachname[31];
    char Vorname[31];
    struct sDatum { int Tag, Monat, Jahr; } Geburt;
};
```

- Falls die Struktur sDatum nicht weiter verwendet wird, kann der Name auch weggelassen werden

Bitweise Strukturen

- Bisher benötigen alle Variablen aus den Beispielen ein Vielfaches von einem Byte als Speicher
  - `char` → 1 Byte
  - `int` → 4 Byte
  - `double` → 8 Byte

- Wie kann ich beliebige Größen definieren?
  - Z.B. für Tag Werte von 1 bis 31
  - Und für Monat Werte von 1 bis 12

Wie viele Bits notwendig?

Bitweise Strukturen

- Anzahl der Bits (nicht Bytes) kann mit Doppelpunkt definiert werden
- Elemente der Struktur werden nacheinander im Speicher abgelegt
- Elemente werden wie gewohnt angesprochen mit (Punkt „.“ oder Pfeil „->“)
- Verwendung in der HW-nahen Programmierung, z.B. zur Ansteuerung von Bauteilen

```
Typ Elementname : Breite ;
```

```
struct sDatum {
        unsigned Tag:5;
        unsigned Monat:4;
        unsigned Jahr:12;
        int FreierTag:1;
        int Sommer:1;
        int Reserviert:9;
    };
```

SPEZIELLE DATENTYPEN

→ Strukturen
→ Aufzählungen
→ Union (wird in PR1 nicht im Detail behandelt!)
→ Eigene Datentypen

HAW HAMBURG

Eigene Datentypen definieren mit `typedef`

- Mit dem Schlüsselwort `typedef` können neue Bezeichner für einen Datentypen definiert werden
- Syntax: `typedef Typendefinition Bezeichner;`
- Beispiel:

```
typedef unsigned char tByte;
tByte Byte;
Byte = 192;
```

- Neuer Datentyp `tByte` wurde definiert
- Anstatt den relativ langen Bezeichner `unsigned char` zu verwenden, kann jetzt mit dem neuen Typ `tByte` gearbeitet werden

Eigene Datentypen – Beispiel: `size_t`

- Die Funktion `sizeof()`
  - gibt die Länge des Speicherbereiches (gemessen in Bytes) an
  - und hat einen Rückgabewert vom Typ `size_t`
- `size_t` ist ein Alias für einen integralen Datentyp
- Definition in `stddef.h` mit `typedef` abhängig von der Plattform, z.B.

```
#ifdef _WIN64
    typedef unsigned __int64 size_t;
#else
    typedef unsigned int     size_t;
#endif
```

- Vorteil → Durch eine gute Namensvergabe kann schon am Datentyp ersehen werden, wofür eine Variable gedacht ist.

Eigene Datentypen für Strukturen

- Auch für Strukturen, Aufzählungen etc. möglich
- Beispiel:

```
typedef struct sDatum tDatum;
```

- Vorteil → übersichtlicher & kürzer da das Schlüsselwort `struct` bei der Definition entfällt
- Deklaration der Struktur und Typendefinition auch in einem Schritt möglich

```
typedef struct sStruktur {
    int a;
    …
} tDatentyp;
```

Eigene Datentypen für Strukturen – Beispiel: FILE

```
struct _iobuf {
    char* _ptr;
    int   _cnt;
    char* _base;
    int   _flag;
    int   _file;
    int   _charbuf;
    int   _bufsiz;
    char* _tmpfname;
};
typedef struct _iobuf FILE;
```

Übung

- Erstellen Sie eine Struktur für eine Adresse aus den Elementen
  - Straße
  - Hausnummer
  - PLZ
  - Ort
- Erstellen Sie für die Struktur einen Datentyp mit Namen `tAdresse`
- Definieren Sie eine Variable vom Typ `tAdresse`
- Weisen Sie der Variablen Beispiel-Werte zu und geben Sie die Struktur aus

Weitere Vorteile mit `typedef`

- Datentypen können auf unterschiedlichen Plattformen unterschiedlich definiert sein
- Beispiel: `int`
  - In Visual C → 4 Byte
  - Z80-Compiler für Microcontroller → 2 Byte

- Wie können Sie ein Programm plattform-unabhängig entwickeln?

Plattform-unabhängige Datentypen mit `typedef`

Abhängig von ihrer Umgebung wird die entsprechende Header-Datei verwendet

```c
/* Header-Datei für Visual C */
typedef short Int2;
typedef int Int4;
```

```c
/* Header-Datei für Z80-Compiler */
typedef int Int2;
typedef long Int4;
```

```c
/* Hauptprogramm */
#include "DatenTypen.h"

int main()
{
    Int2 Klein = 12345;
    Int4 Gross = 1234567890;
}
```

Quellcode in der .c-Datei ist für beide Umgebungen gleich!

SPEZIELLE DATENTYPEN

→ Strukturen
→ Aufzählungen
→ Union (wird in PR1 nicht im Detail behandelt!)
→ Eigene Datentypen

HAW HAMBURG

Aufzählungen

- Häufig wird in einer Variablen ein Wert aus einer begrenzten Anzahl von Möglichkeiten gespeichert
- Beispiel: Jahreszeiten
  - 4 Möglichkeiten:
    - Frühling,
    - Sommer,
    - Herbst und
    - Winter
- Es handelt sich hierbei um eine Aufzählung

Mögliche Lösung mit Integer-Variable **OHNE** Aufzählung!

```c
#include <stdio.h>

#define FRUEHL 0
#define SOMMER 1
#define HERBST 2
#define WINTER 3

int main() {
    int Jahreszeit;

    Jahreszeit = HERBST;

    switch (Jahreszeit) {
        case FRUEHL: printf("Frühling\n"); break;
        case SOMMER: printf("Sommer\n"); break;
        case HERBST: printf("Herbst\n"); break;
        case WINTER: printf("Winter\n"); break;
        }
    return 0;
}
```

→ Visual Studio

C hat für Aufzählungen den spez. Datentyp `enum`

```c
#include <stdio.h>

int main()
{
    enum eJahreszeit { Fruehl, Sommer, Herbst,
Winter };
    enum eJahreszeit Jahreszeit;

    Jahreszeit = Herbst;

    switch (Jahreszeit) {
    case Fruehl: printf("Frühling\n"); break;
    case Sommer: printf("Sommer\n"); break;
    case Herbst: printf("Herbst\n"); break;
    case Winter: printf("Winter\n"); break;
    }

    return 0;
}
```

Aufzählungen mit `enum`

- Ähnlich wie bei Strukturen gibt es auch für Aufzählungen eine Deklaration und eine Definition

- Deklaration:

```
enum eName {const1, const2, .. constN};
```

- Beispiel:

```
enum eJahreszeit { Fruehl, Sommer, Herbst, Winter };
```

- Definition:

```
enum eName name;
```

- Beispiel:

```
enum eJahreszeit Jahreszeit;
```

Aufzählungen mit `enum` (Deklaration & Definition)

- Deklaration & Definition können in einem Schritt erfolgen

```
enum eJahreszeit { Fruehl, Sommer, Herbst, Winter } Jahreszeit;
```

- Name der Aufzählung kann auch entfallen falls nur einmal genutzt

```
enum {Fruehl, Sommer, Herbst, Winter } Jahreszeit;
```

- Bei der Definition kann die Variable auch initialisiert werden

```
enum eJahreszeit Jahreszeit = Herbst;
```

Aufzählungen mit `enum` (II)

- Eine Aufzählung besteht aus einem Satz von benannten Konstanten
- Diese werden im Hintergrund beginnend mit Null durchnummeriert
- Elementen kann auch explizit ein Wert zugewiesen werden
- Folge-Elemente ohne Wert werden fortlaufend durchnummeriert
- Beispiel:

```
enum eJahreszeit { Fruehl=1, Sommer, Herbst, Winter };
```

Aufzählungen mit `enum` (III)

- Zahlenwerte können auch mehrfach verwendet werden
- Beispiel:
  - Boolescher Ausdruck soll die Werte „wahr“ und „falsch“ annehmen können
  - Gleichzeitig sollen im Programm auch englische Bezeichnungen „true“ und „false“ nutzbar sein

```
enum eBool { Falsch, Wahr, False=0, True };
```

Aufzählungen mit `enum` (IV)

- Aufzählungen können mit anderen Datentypen verkettet werden und Teil einer Struktur oder als Vektor genutzt werden

```
enum eBool Logik[10];

struct sDatum {
        int Tag, Monat, Jahr;
        enum eJahreszeit Jahreszeit;
        enum eBool Feiertag;
    } Datum;
```

Aufzählungsvariable als Vektor

Aufzählung innerhalb einer Struktur

Arbeiten mit `enum`

- Eine Variable vom Typ `enum` soll nur die Werte einnehmen, die bei der Deklaration angegeben wurden
- Intern wird eine Aufzählung als `int` gespeichert und benötigt 4 Bytes
- In C können auch noch beliebige Integer-Werte zugewiesen werden.
  - Diese Vermischung von Typen sollte vermieden werden!
  - In C++ ist dies auch nicht mehr erlaubt!
- Ansonsten lassen sich Variablen vom Typ `enum` wie andere Variablen ansprechen

Übung zu `enum`

- Deklarieren Sie eine Aufzählung für die Wochentage (Montag, Dienstag, .. , Sonntag) mit dem Namen `eWochentag`
- Definieren Sie einen auch einen Datentyp `tWochentag` mit `typedef` für die Aufzählung
- Erstellen Sie eine Variable von `tWochentag` und weisen sie ihr den heutigen Wochentag zu

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!

HAW HAMBURG

```c
// Übungsaufgabe für struct & typedef
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>


#define VERSION 1

struct sAdresse {
     char strasse[100];
     short hausnummer;
     long plz;
     char ort[50];
};

typedef struct sAdresse tAdresse;

typedef struct {
     char strasse[100];
     short hausnummer;
     long plz;
     char ort[50];
} tAdresse_v2;
```

```c
void main() {
     #if VERSION == 1
     struct sAdresse addr;
     #elif VERSION == 2
     tAdresse_v1 addr;
     #elif VERSION == 3
     tAdresse_v2 addr;
     #endif

     strcpy(addr.strasse, "Berliner Tor");
     addr.hausnummer = 7;
     addr.plz = 20999;
     strcpy(addr.ort, "Hamburg");

     printf("Adresse:\n");
     printf("%s %d\n", addr.strasse, addr.hausnummer);
     printf("%d %s\n", addr.plz, addr.ort);
}
```

```
// mit enum arbeiten
enum eWochentag { Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag};
typedef enum eWochentag tWochentag;
tWochentag tag;

tag = Donnerstag;
```