

Rechtlicher Hinweis

Diese Präsentation ist urheberrechtlich geschützt und darf nur im Rahmen von Lehrveranstaltungen der Friedrich-Schiller-Universität Jena verwendet werden. Eine Nutzung durch Verbreitung oder Veröffentlichung dieses Materials - auch in Auszügen - ist strengstens untersagt und wird die Geltendmachung von Unterlassungs- und Schadenersatzansprüchen durch die Friedrich-Schiller-Universität Jena zur Folge haben.

Legal notice

These slides are protected by copyright and may only be used as part of courses at the Friedrich Schiller University Jena. Any use through the dissemination or publication of this material - even in extracts - is strictly prohibited and will result in the assertion of injunctive relief and claims for compensation by the Friedrich Schiller University Jena.

Informatik I (B.Sc. Physik)

Strukturen (records)

Dr. Paul Bodesheim
(Paul.Bodesheim@uni-jena.de)



**FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA**

**Fakultät für Mathematik und Informatik
Lehrstuhl für Digitale Bildverarbeitung**

SoSe 2020

- 1 Motivation und Begriffe
- 2 Definition und Beispiele
- 3 Initialisierung und Elementzugriff
- 4 Objektnamen und Default-Werte
- 5 Verschachtelungen und Reihungen
- 6 Strukturen und Funktionen

Inhalt

- 1 **Motivation und Begriffe**
- 2 Definition und Beispiele
- 3 Initialisierung und Elementzugriff
- 4 Objektnamen und Default-Werte
- 5 Verschachtelungen und Reihungen
- 6 Strukturen und Funktionen

Motivation

- Eine weitere Form für komplexe Datentypen
- Gruppierung von Elementen unterschiedlicher (Basis-) Datentypen
- Eine Variable kann Werte von unterschiedlichen Typen besitzen, Beispiel: Name und Alter einer Person
- Mit Strukturen lassen sich Datentypen selber definieren (zusammensetzen)

Begriffe

- Schlüsselwort: **struct**

- Engl.: *structure*

- Struktur
- Konstruktion
- Gebilde
- ...

: 结构
结构体
施工
结构体

...
结构或数据结构 (数据结构)
备用名称: 记录
记录
文件

- Strukturen bzw. Datenstrukturen (*data structures*)

几种元素/属性的组合

- Alternative Bezeichnung: *record*

- Datensatz
- Akte

- Zusammenfassung von mehreren Elementen / Attributen

Inhalt

- 1 Motivation und Begriffe
- 2 Definition und Beispiele**
- 3 Initialisierung und Elementzugriff
- 4 Objektnamen und Default-Werte
- 5 Verschachtelungen und Reihungen
- 6 Strukturen und Funktionen

Definition

```

struct <TYPNAME> {
    <DATENTYP_1> <ELEMENTNAME_1>;
    <DATENTYP_2> <ELEMENTNAME_2>;
    ...
    <DATENTYP_N> <ELEMENTNAME_N>;
};

```

: 新数据类型的唯一名称 (标识符)
 hDATENTYP 1i, ..., <数据类型Ni: 个人的已知数据类型
 元素
 hELEMENTNAME 1i, ..., <ELEMENTNAME Ni: 的名称 (标识符)
 个别元素
 不要忘记在代码块末尾的分号 (类型定义) !
 使用惯例 (建议, 而非规则) 输入带大写字母的名称
 首字母

- <TYPNAME>: eindeutiger Name (Bezeichner) für den neuen Datentyp
- <DATENTYP_1>, ..., <DATENTYP_N>: bekannte Datentypen für die einzelnen Elemente
- <ELEMENTNAME_1>, ..., <ELEMENTNAME_N>: Name (Bezeichner) für die einzelnen Elemente
- Semikolon am Ende des Blocks nicht vergessen (Typdefinition)!
- Übliche Konvention (Empfehlung, keine Regel): Typname mit großem Anfangsbuchstaben
- Definition außerhalb und vor den Funktionen, in denen sie verwendet wird

Beispiele für Definitionen

```
struct Person {  
    string name;  
    int alter;  
    double groesse;  
};
```

```
struct Quader {  
    double laenge, breite, hoehe;  
};
```

```
struct Person2 {  
    int id; // z.B. Personal-Nr., Kunden-Nr., Patienten-Nr.  
    string vorname, nachname, geburtsdatum, geschlecht;  
    bool verheiratet;  
    int alter, gewicht, anzahlGeschwister, anzahlKinder;  
    string adresse, email_adresse;  
    double groesse;  
    string augenfarbe, haarfarbe;  
};
```

Inhalt

- 1 Motivation und Begriffe
- 2 Definition und Beispiele
- 3 Initialisierung und Elementzugriff**
- 4 Objektnamen und Default-Werte
- 5 Verschachtelungen und Reihungen
- 6 Strukturen und Funktionen

Initialisierung und Elementzugriff

- Variablen anlegen wie bei anderen Datentypen auch
- Initialisierung mit geschweiften Klammern oder über Elementzugriff
- Elementzugriff mit Operator "." via $\langle \text{VARIABLENNAME} \rangle . \langle \text{ELEMENTNAME} \rangle$

```
struct Person {  
    string name;  
    int alter;  
    double groesse;  
};  
  
int main()  
{  
    Person p1 = {"Hans", 32, 1.96};  
    Person p2;  
    p2.name = "Eva";  
    p2.alter = 27;  
    p2.groesse = 1.74;  
    ...  
}
```

Beispiel: Reihenfolge beachten

```
// Definition einer
// grossen Struktur
struct Person2 {
    int id;
    string vorname;
    string nachname;
    string geburtsdatum;
    string geschlecht;
    bool verheiratet;
    int alter;
    int gewicht;
    int anzahlGeschwister;
    int anzahlKinder;
    string adresse;
    string email_adresse;
    double groesse;
    string augenfarbe;
    string haarfarbe;
};
```

```
int main()
{
    Person2 p =
        {15, // ID
        "Max", // Vorname
        "Mustermann", // Nachname
        "29. Februar 1976", // Geb.-datum
        "maennlich", // Geschlecht
        true, // verheiratet
        43, // alter
        71, // gewicht
        1, // Geschwister
        2, // Kinder
        "Weg 1a, 01234 Stadt", // adresse
        "max.muster@mail.com", // email
        1.84, // groesse
        "braun", // augenfarbe
        "blond" // haarfarbe
        };
    ...
}
```

Lesender und schreibender Elementzugriff

```
int main()
{
    Person p1 = {"Hans", 32, 1.96};

    Person p2;
    p2.name = "Eva";
    p2.groesse = 1.74;
    p2.alter = 27;

    Person p3 = p2; // Zuweisung: Kopie aller Werte der Elemente
    p3.name = "Hugo";
    p3.alter = p1.alter;

    cout << p3.name << " (" << p3.alter << " Jahre, ";
    cout << p3.groesse << " m)" << endl;

    return 0;
}
```

Ausgabe: Hugo (32 Jahre, 1.74 m)

Inhalt

- 1 Motivation und Begriffe
- 2 Definition und Beispiele
- 3 Initialisierung und Elementzugriff
- 4 Objektnamen und Default-Werte**
- 5 Verschachtelungen und Reihungen
- 6 Strukturen und Funktionen

Erweiterte Definition um Objektnamen

```

struct <TYPNAME> {
    <DATENTYP_1> <ELEMENTNAME_1>;
    <DATENTYP_2> <ELEMENTNAME_2>;
    ...
    <DATENTYP_N> <ELEMENTNAME_n>;
} <OBJEKTNAMEN>;

```

定义现有对象的变量名称的可选列表
 直接定义新数据类型的对象
 仍然可以创建其他变量
 初始化列表中对象的元素

- <OBJEKTNAMEN>: optionale Liste von Variablennamen zur Definition von existierenden Objekten
- Direktes Festlegen von Objekten des neuen Datentyps
- Anlegen zusätzlicher Variablen weiterhin möglich
- Initialisierung der Elemente eines Objekts in der Liste erlaubt

Beispiel: Objektnamen

```
struct Quader {  
    double laenge, breite, hoehe;  
} q1,q2,q3; // Objektnamen
```

```
int main()  
{  
    q1.laenge = 2.0;  
    q1.breite = 2.0;  
    q1.hoehe = 2.0;  
  
    ...  
  
    Quader q4;  
    q4.laenge = 1.98;  
    q4.breite = 2.73;  
    q4.hoehe = 1.74;  
  
    ...  
}
```


Beispiel: Objektnamen mit Initialisierung

```
struct Quader {  
    double laenge, breite, hoehe;  
} q1, q2 {1.2, 3.4, 5.6}, q3 {9.1, 8.2, 7.3};  
  
int main()  
{  
    q1.laenge = 2.0;  
    q1.breite = 2.0;  
    q1.hoehe = 2.0;  
  
    cout << q2.laenge; // 1.2  
    cout << q3.breite; // 8.2  
  
    Quader q4;  
    q4.laenge = 1.98;  
    q4.breite = 2.73;  
    q4.hoehe = 1.74;  
  
    ...  
}
```

Default-Werte

- Standardwerte von Elementen festlegen
- Auch nur teilweise möglich 设置元素的默认值
也只有部分可能

```
struct Quader {
    double laenge=1.5, breite=2.5, hoehe=3.5;
} q1, q2={1.2, 3.4, 5.6};

int main()
{
    Quader q3;

    cout << "q1: " << q1.laenge << " x " << q1.breite;
    cout << " x " << q1.hoehe << endl; // q1: 1.5 x 2.5 x 3.5

    cout << "q2: " << q2.laenge << " x " << q2.breite;
    cout << " x " << q2.hoehe << endl; // q2: 1.2 x 3.4 x 5.6

    cout << "q3: " << q3.laenge << " x " << q3.breite;
    cout << " x " << q3.hoehe << endl; // q3: 1.5 x 2.5 x 3.5

    ...
}
```

Default-Werte (2)

```
struct Person
{
    string name = "Hans";
    int alter = 18;
    double groesse = 1.92;
} hans, marie = {"Marie", 21, 1.74};

int main()
{
    cout << hans.alter << endl; // 18
    cout << marie.groesse << endl; // 1.74

    ...

    return 0;
}
```

Inhalt

- 1 Motivation und Begriffe
- 2 Definition und Beispiele
- 3 Initialisierung und Elementzugriff
- 4 Objektnamen und Default-Werte
- 5 Verschachtelungen und Reihungen**
- 6 Strukturen und Funktionen

Verschachtelungen

- Elemente einer Struktur können vom Typ einer anderen Struktur sein

```

struct Adresse {
    string strasse , hausnummer , plz , ort ;
};

struct Kunde {
    int kundennummer;
    string name;
    Adresse rechnungsadresse , versandadresse ;
};

int main()
{
    Kunde k1 = {123,"Max Mustermann",
                {"Weg", "1a","02468","Dorf"},
                {"Strasse", "2","13579","Stadt"}};
    k1.versandadresse.ort = "Gemeinde";

    cout << k1.name; // Max Mustermann
    cout << k1.rechnungsadresse.plz; // 02468
    ...
}

```

Reihungen

- Struktur kann Elementtyp eines Vektors sein

```
struct Zeitreihe {  
    string beginn, ende;  
    vector<double> messwerte;  
};
```

```
struct Experiment {  
    int id;  
    string datum, name;  
    vector<Zeitreihe> messungen; 结构可以是向量的元素类型  
};
```

Reihungen (2)

```
int main()
{
    Zeitreihe z1 = {"13:00 Uhr", "13:05 Uhr", {1.3, 2.1,
        1.4, 1.6, 1.7, 1.4} };
    Zeitreihe z2 = {"13:10 Uhr", "13:15 Uhr", {1.5, 1.2,
        1.8, 1.3, 1.4, 1.5} };

    Experiment e = {12, "heute", "Exp01", {z1,z2}};

    // vector<Zeitreihe> zeitreihen;
    // zeitreihen.push_back(z1);
    // zeitreihen.push_back(z2);
    // Experiment e = {12, "heute", "Exp01", zeitreihen};

    cout << e.messungen[0].beginn << endl; // 13:00 Uhr
    cout << e.messungen[0].messwerte[1] << endl; // 2.1

    ...
}
```

Reihungen (3)

```
...

Zeitreihe z3;
z3.beginn = "13:30 Uhr";
z3.ende = "13:35 Uhr";

e.messungen.push_back(z3);
for (int i=30; i<=35; i++)
{
    double d = drand48();
    e.messungen[2].messwerte.push_back(d);
}

for (int i=0; i<=e.messungen[2].messwerte.size(); i++)
    cout << e.messungen[2].messwerte[i] << endl;

...
}
```


Inhalt

- 1 Motivation und Begriffe
- 2 Definition und Beispiele
- 3 Initialisierung und Elementzugriff
- 4 Objektnamen und Default-Werte
- 5 Verschachtelungen und Reihungen
- 6 Strukturen und Funktionen**

Strukturen als Funktionsparameter

- Strukturen können auch Typen für Funktionsparameter sein
- Komplexer Datentyp \Rightarrow (konstante) Referenzparameter verwenden

结构也可以是功能参数的类型
复杂数据类型) 使用 (恒定) 参考参数

```

struct Person {
    string name;
    int alter;
    double groesse;
};

void print(const Person & p)
{
    cout << p.name << " (" << p.alter << " Jahre, ";
    cout << p.groesse << " m)" << endl;
}

int main()
{
    Person p = {"Hans", 32, 1.96};
    print(p); // Hans (32 Jahre, 1.96 m)

    ...
}

```

Funktionen in Strukturen

- Funktionen speziell für Struktur schreiben
- Aufruf wie Zugriff auf Element mit Operator "." über
 <VARIABLENNAME>.<FUNKTIONSNAME> 专门为结构编写函数
像使用运算符\一样访问元素。
- Elemente der Struktur in der Funktion verwendbar hVARIABLENNAME.i.hFUNCTIONNAME i
该结构的元素可以在函数中使用

```

struct Person {
    string name;
    int alter;
    double groesse;
    void print()
    {
        cout << name << " (" << alter << " Jahre, ";
        cout << groesse << " m)" << endl;
    }
};

int main()
{
    Person p = {"Hans", 32, 1.96};
    p.print(); // Hans (32 Jahre, 1.96 m)
    ...
}

```

Funktionen in Strukturen (2)

- “Externe” Funktionsdefinition möglich, aber Deklaration innerhalb der Struktur (Zuordnung)
- Implementierung erfordert Kennzeichnung der Zugehörigkeit zur Struktur: einfach `<TYPNAME>::` vor Funktionsnamen schreiben

```

struct Person {
    string name;
    int alter;
    double groesse;
    void print();
};

void Person::print()
{
    cout << name << " (" << alter << " Jahre, ";
    cout << groesse << " m)" << endl;
}

int main()
{
    Person p = {"Hans", 32, 1.96};
    p.print(); // Hans (32 Jahre, 1.96 m)
    ...
}

```

\外部” 函数定义可能，但结构内的声明（赋值）
 实现需要识别结构的成员身份：简单
 在函数名称前编写hTYPNAMEI ::

Gibt es Fragen?

(Es gibt keine dummen Fragen!)

