Hochschule **Rosenheim**
University of Applied Sciences

Prozedurale Programmierung

Einfache Datenstrukturen

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

Prof. Dr. F.J. Schmitt

Rinderbeinscheiben in Rotweinsauce

Hochschule **Rosenheim**
University of Applied Sciences

```
// belege Variablen mit Werten
oel.art = sonnenblumen;
oel.vol_ml = 45;

wein.art = rot;
wein.rebsorte = spaetburgunder;
wein.geschmacksgrad = trocken;
wein.vol_ml = 250;
wein.alk_prozent_min = -1;  // -1 = egal
wein.alk_prozent_max = -1;
wein.jahrgang = -1;
wein.winzer = -1;
wein.land = D;

bruehe.art = rind;
bruehe.vol_ml = 250;
```

Motivation

- Verarbeitung unterschiedlicher Arten von Informationen
  - Abstraktion und Beschreibung von realen Objekten, Abläufen etc.
  - Beispiel: Person
    - Name, Alter, Wohnort, Sozialversicherungsnummer, Größe, Gewicht, Fähigkeiten, Einkommen, etc.
  - je nach Anwendung sind unterschiedliche Attribute wichtig

- Zielsetzung: Aufbau von problemspezifischen Datentypen

Strukturen

- Zusammengehörige Daten können zu einer Einheit zusammengefasst werden
- Kombination von mehreren Elementen vorhandener Datentypen
- Definition eines neuen, komplexen Datentyps
- Können beliebig komplex sein und aus unterschiedlichen Datentypen bestehen
- Struktur ist ein Verbunddatentyp
- Aufbau geeigneter Datenstrukturen ist eine extrem wichtige Aufgabe eines Programmierers

Einfache Strukturen (1)

Definition des Datentyps:

```
struct Strukturname
{
   Typ1 Attribut1;
   Typ2 Attribut2;
   //...
};
```

Komponenten oder Attribute

- Es wird noch kein Speicherplatz reserviert
- Es erfolgt lediglich eine Beschreibung des neuen Datentyps

Einfache Strukturen (2)

- Definition von Variablen des neuen Datentyps:

```
struct Strukturname Variablenliste;
```

  - Variablendefinition mit selbstdefinierten Typen ist syntaktisch identisch zu der von elementaren Datentypen, mit Ausnahme, dass das Schlüsselwort `struct` verwendet werden muss

Einfache Strukturen (3)

- Beispiel: Geometrischer Punkt

```
struct Punkt_s
{
  double x;
  double y;
};
```

Definition einer Struktur

```
struct Punkt_s p1, p2;
```

Definition von Variablen

```
struct Punkt_s
{
  double x;
  double y;
} p1, p2;
```

Mischung von Typ- und Variablendefinition

(nicht empfohlen – vermeiden)

Trennung von Typ- und Variablendefinitionen

- Typdefinitionen werden meist getrennt in Header-Dateien gesammelt

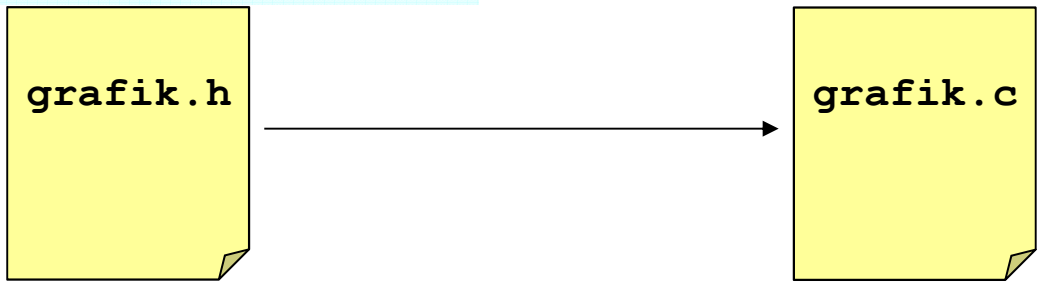
```
// Datei grafik.h

struct Punkt_s
{
   double x;
   double y;
};
```

```
// Datei grafik.c

#include "grafik.h"
//...

struct Punkt_s p1, p2;
```



Initialisierung und Zugriff

- Initialisierung von Strukturen

```
struct Punkt_s p1 = {4, 3};
```

  - bei nicht vollständiger Initialisierung werden fehlende Einträge auf 0 gesetzt
  - Achtung: geht so nur direkt in der Variablendefinition!

- Zugriff auf Attribute mit dem Punkt-Operator (Selektionsoperator)

```
p1.x = 3.6;
p1.y = 4.3;
```

Beispiel

Berechnung des Abstands zweier Punkte

```
#include <math.h>

…

double dist, dx, dy;
struct Punkt_s p1, p2;

dx = p2.x - p1.x;
dy = p2.y - p1.y;

dist = sqrt(dx * dx + dy * dy);

…
```

Zuweisungen

- Strukturen können als ganzes einander zugewiesen werden
- geht auch, wenn die Struktur Felder (→ später) enthält

```
struct Punkt_s p1, p2;

p1.x = 1;
p1.y = 2;

p2 = p1;
```

Aufgabe (1)

- Definieren Sie einen neuen (Verbund-)Datentyp zur Beschreibung einer Stadt. Folgende Informationen sollen gespeichert werden können:
  - Stadtname
  - Abkürzung für das Land als einzelner Buchstabe (Bsp: D oder A)
  - Anzahl der Einwohner

Verschachtelte Strukturen

```
struct Rechteck_s
{
  struct Punkt_s pmin;
  struct Punkt_s pmax;
};
```

Definition der Struktur
Rechteck_s

```
struct Rechteck_s r =
{
  {4, 3},
  {15,8}
};
```

Variablendefinition und
Initialisierung

Flächenberechnung:

```
double flaeche;
flaeche = (r.pmax.x - r.pmin.x) * (r.pmax.y - r.pmin.y);
```

Funktionen und Strukturen (1)

- Struktur-Variablen können ähnlich wie Variablen von vorhandenen (elementaren) Datentypen verwendet werden
  - Strukturen können als Ganzes an Funktionen übergeben werden
  - Strukturen können als Funktionswert zurückgegeben werden

Funktionen und Strukturen (2)

```
struct Rechteck_s neuesRechteck(     struct Punkt_s p1,
                                     struct Punkt_s p2 )
{
  struct Rechteck_s rect;

  rect.pmin = p1;
  rect.pmax = p2;

  return rect;
}
//...
struct Punkt_s punkt1 = {1.0, 1.0};
struct Punkt_s punkt2 = {3.0, 4.0};
struct Rechteck_s myRecht = neuesRechteck(punkt1, punkt2);
```

Parameter werden als Kopien übergeben (call by value)

⇒ Bei großen Strukturen problematisch
(Kopieren von großen Datenmengen) Lösung → später

Zusammenfassung

- komplexer Datentyp, aufgebaut aus elementaren Typen
- wird verwendet wie elementare Typen, mit zusätzlichem Schlüsselwort „struct“
- Zugriff auf Elemente mit „.“ (und später: mit „->“)
- Übergabe an Funktionen / Rückgabe wie elementare Datentypen