C - Grundlagen und Konzepte
C-Datenstrukturen

Andreas Gadelmaier

2. Juli 2014

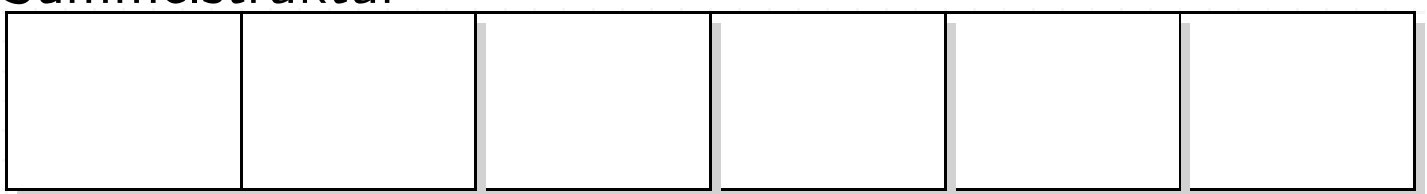
Inhaltsverzeichnis

Warum brauchen wir Datenstrukturen?

- Essenziell zum Programmieren
- Effizient

Struct

- Sammelstruktur



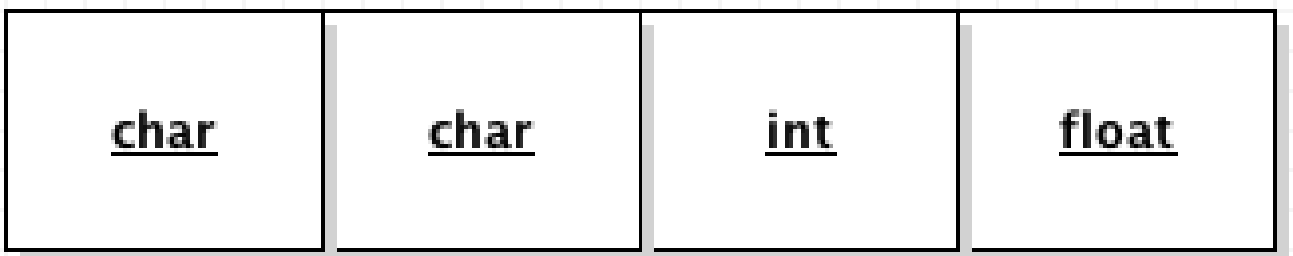
- Fassen verschiedene Komponenten zusammen
- Diese Komponenten sind von unterschiedlichen Typen
- Anzahl der Komponenten ist unbegrenzt

Syntax

- Deklaration von Name und Inhalt:

```
1 struct BankKunde{
2     char vorname[20];
3     char nachname[20];
4     int kontonummer;
5     float kontostand;
6 }Kunde;
```

BankKunde

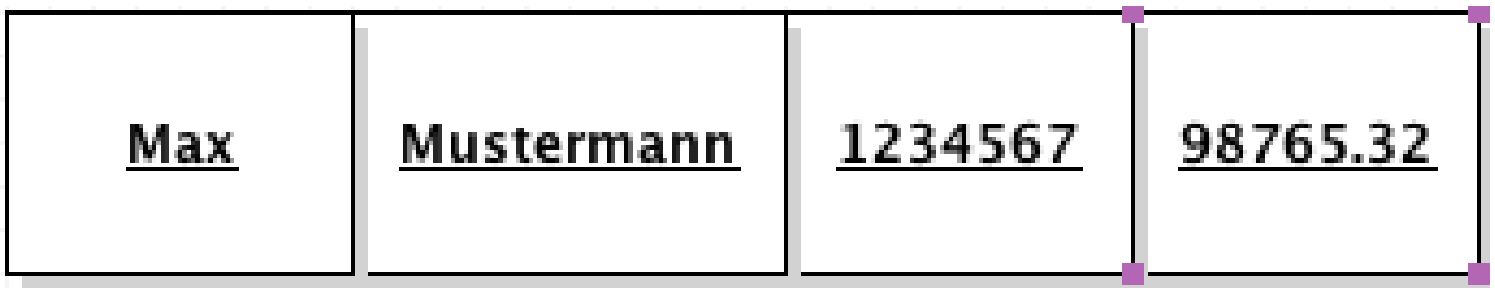


Beispiel

- Kontoinhaber

```
1  struct BankKunde Kunde
2
3    Kunde.vorname = "Max";
4    Kunde.nachname= "Mustermann";
5    Kunde.kontonummer = 1234567;
6    Kunde.kontostand = 98765.32;
```

BankKunde max



Zugriff

```
1 printf("Kontodaten:␣\n
2 Vorname␣=␣%s␣\n
3 Nachname␣=␣%s␣\n
4 Kontonummer=␣%d␣\n
5 Kontostand␣=␣%f"
6 ,max.vorname
7 ,max.nachname
8 ,max.kontonummer
9 ,max.kontostand);
```

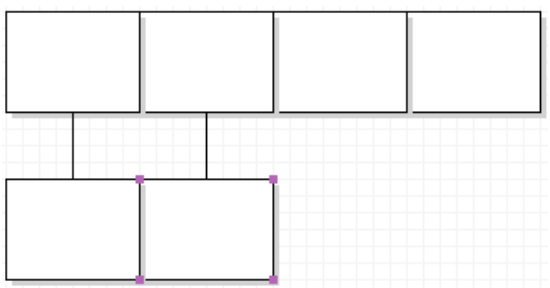
Ausgabe

Kontodaten:
Vorname = Max
Nachname = Mustermann
Kontonummer = 1234567
Kontostand = 98765.32

Syntax

- Setzt sich aus verschiedenen Komponenten zusammen
- Definition von Variablen im selben Speicherbereich
- Der Speicherbedarf wird durch die größte Komponente bestimmt

```
1 union Kundendaten {
2 int kundennummer;
3 short filiale;
4 }Daten;
```



Beispiel

- Anwendungsbereiche dürfen sich nicht überschneiden

```
1 Daten max
2 max.kundennummer = 123456;
3 printf("%d", max.kundennummer);
```

- Ausgabe: 123456

Beispiel

```
1 Daten max
2 max.kundennummer = 123456;
3 printf("%d\n", max.kundennummer);
4 max.filiale = 226;
5 printf("%d",max.filiale);
```

- Ausgabe für Kundennummer: 123456
- Ausgabe für Filiale : 226

Gefahr

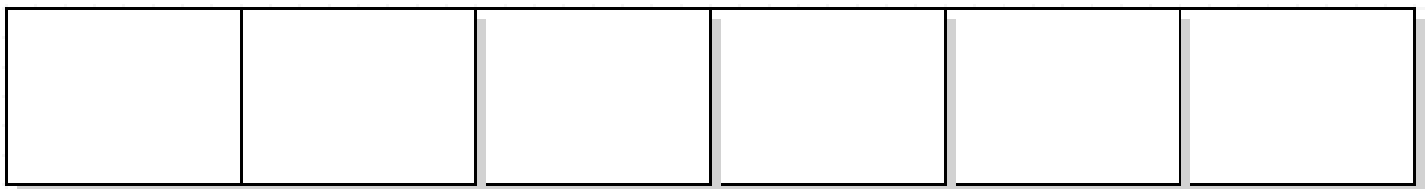
- Variablen teilen sich einen Speicherbereich
- Fehler die daraus resultieren sind schwer zu finden

```
1 Daten max
2 max.kundennummer = 123456;
3 max.filiale = 226;
4 printf("%d\n", max.kundennummer);
5 printf("%d",max.filiale);
```

- Ausgabe für Kundennummer: Falsch
- Ausgabe für Filiale: 226

Syntax

- Sammelstruktur



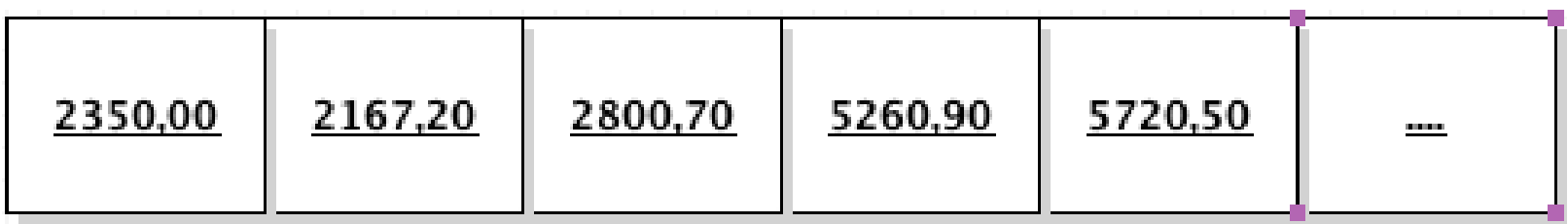
- Deklaration von Typ und Größe
- Gleiche Typen
- Möglichkeit 1

```
1 float monatlicherKontostand[12];
2
3 monatlicherKontostand[0]= 2350.00;
4 monatlicherKontostand[1]= 2167.20;
5 monatlicherKontostand[2]= 2800.70;
6 monatlicherKontostand[3]= 5260.90;
7 monatlicherKontostand[4]= 5720.50;
8 ....
```

Syntax

- Möglichkeit 2

```
1 float monatlicherKontostand[12]= {2350.00,
2 2167.20,
3 2800.70,
4 5260.90,
5 5720.50,
6 ....};
```



Zugriff

- Bei nicht Inizialisierung 0

```
1 int i;
2 for (i=0; i<12;i++){
3 printf ("Kontostand: \n
4 %f", kontostand[i];)
5 } ;
6 put(" ");
```

Ausgabe: 2350.00 2167.20 2800.70 5260.90 5720.50

Mehrdimensonale arrays

```
1 int terminkallender [365][3] ={
2 {1, 123, 456},
3 {2, 789, 101},
4 {3, 112, 131},
5 {...}
6 };
```

string

- Strings sind Zeichenketten
- Char Arrays zum Speichern von Zeichenketten
- Funktionieren genau wie int Arrays

```
1 char name = 'a';
2 char name[20]='m','a','x','\0';
3 char name[20]="max";
```

charr Array

- Char Arrays werden oft dynamisch erstellt

```
1 char string[] = ("Konto ist leer!");
```

Mehrdimensonale char Arrays

```
1 char Kundenliste [3][20] = {
2 "Mueller"
3 ,"Krause"
4 ,"Schaefer"
5 };
```

enum

- Enumeration ist eine Aufzählung
- Quelltext wird lesbarer
- MagicNumbers : x?

```
1 float druckeZinsen (int x){
2 if(x < 6) printf("Zinssatz bei 1,5%");
3 else(x >= 6) printf("Zinssatz bei 2,7%")
4 };
```

Syntax

- Elemente im enum sind Nummeriert

```
1  typedef enum{
2  januar, februar, maerz,
3  april, mai, juni,
4  juli, august, september,
5  oktober, november, dezember
6  } monate;
7  float druckeZinsen (monat m){
8  if(m < 6) printf("Zinssatz bei 1,5%");
9  else(m => 6) printf("Zinssatz bei 2,7%")
10 };
```

Beispiel

```
1 int main(){
2 druckeZinsen(4);
3 return 0;
4 }
```

Beispiel

```
1 int main(){
2 druckeZinsen(mai);
3 return 0;
4 }
```

typedef

- Datenvereinbarung
- Name für bestehende Struktur
- keine Unterscheidung zur Variablendeklaration

Beispiel

- Mit typedef:

```
1 typedef char character
2 character buchstabe = 'A' ;
```

- Ohne typedef:

```
1 char buchstabe = 'A' ;
```

Beispiel

- Mit typedef:

```
1 typedef struct BankKunde{
2     char vorname[20];
3     char nachname[20];
4     int kontonummer;
5     float kontostand;
6 }Kunde;
```

```
1 int main ( ){
2 BankKunde max;
3 return 0;
4 }
```

Beispiel

- Ohne typedef:

```
1 int main ( ){
2 struct BankKunde max;
3 return 0;
4 }
```

Quellen

http://www.proggen.org/doku.php?id=c:tutorial:struct
http://www2.informatik.uni-halle.de/lehre/c/c_struct.html
http://www2.informatik.uni-halle.de/lehre/c/c_union.html
http://www2.informatik.uni-halle.de/lehre/c/c_typdef.html
http://home.fhtw-berlin.de/~junghans/cref/CONCEPT/arrays.html
http://www.c-programmieren.com/C-Lernen.html#Einleitung
http://www.youtube.com/watch?v=E7dwfDpzLKo
http://www.youtube.com/watch?v=0_pKyx7QpU8
http://www.youtube.com/watch?v=xHn3p3mVsMw
http://www.proggen.org/doku.php?id=c:type:enum
C Programmieren von Anfang an von Helmut Erlenk