# Einführung in C

Alexander Batoulis

Fakutltät IV
Technische Universität Berlin

5. Mai 2014

# Inhaltsverzeichnis

# Überblick

# Beispielprogramm in Java

```
1 public class Example
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Völlig isoliert:
              E-techniker sucht Kontakte");
6     }
7 }
```

# Beispielprogramm in C

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("Völlig isoliert: E-techniker
          sucht Kontakte \n");
5     return 0;
6 }
```

# Java ist objektorientiert

```
1 public class Car{
2     public String Name;
3     public void drive() {...}
4 }
5 public static void main(String[] args) {
6     Car lancia = new Car();
7     lancia.drive();
8 }
```

**C ist *nicht* objektorientiert**

Es ist nicht möglich, aus eigens geschriebenen Klassen Objekte zu erzeugen

# Java ist objektorientiert

```
1 public class Car{
2     public String Name;
3     public void drive() {...}
4 }
5 public static void main(String[] args) {
6     Car lancia = new Car();
7     lancia.drive();
8 }
```

**C ist *nicht* objektorientiert**

Es ist nicht möglich, aus eigens geschriebenen Klassen Objekte zu erzeugen

# Umwandlungszeichen

```
1 int main()
2 {
3     printf("%i plus %i ist gleich %s.", 3,
          2, "Fünf");
4     return 0;
5 }
```

**Ausgabe**

3 plus 2 ist gleich Fünf

# Umwandlungszeichen

Beispiele für Umwandlungszeichen:

| Zeichen | Umwandlung |
|---|---|
| %d oder %i | int |
| %c | einzelnes Zeichen |
| %f | double |
| %s | Zeichenkette ausgeben |
| %p | Adresse eines Pointers |

# Überblick

# Zeiger in C

- speichert Adresse einer anderen Variable

→ repräsentiert Adresse und nicht wie eine Variable einen Wert

- Zeiger wird erstellt, indem bei Deklaration Name mit dem Symbol * beginnt

# Zeiger in C

- speichert Adresse einer anderen Variable

→ repräsentiert Adresse und nicht wie eine Variable einen Wert

- Zeiger wird erstellt, indem bei Deklaration Name mit dem Symbol * beginnt
- Allgemein liefert Kaufmanns-Und & die Adresse einer Variable

# Zeiger in C

- speichert Adresse einer anderen Variable

→ repräsentiert Adresse und nicht wie eine Variable einen Wert

- Zeiger wird erstellt, indem bei Deklaration Name mit dem Symbol * beginnt
- Allgemein liefert Kaufmanns-Und & die Adresse einer Variable
- * vor dem Zeiger-Variablennamen gibt den Wert der Adresse zurück auf die der Zeiger zeigt

# Zeiger in C

- speichert Adresse einer anderen Variable

→ repräsentiert Adresse und nicht wie eine Variable einen Wert

- Zeiger wird erstellt, indem bei Deklaration Name mit dem Symbol * beginnt
- Allgemein liefert Kaufmanns-Und & die Adresse einer Variable
- * vor dem Zeiger-Variablennamen gibt den Wert der Adresse zurück auf die der Zeiger zeigt
- *(& pointer) = ?

# Zeiger in C

- speichert Adresse einer anderen Variable

→ repräsentiert Adresse und nicht wie eine Variable einen Wert

- Zeiger wird erstellt, indem bei Deklaration Name mit dem Symbol * beginnt
- Allgemein liefert Kaufmanns-Und & die Adresse einer Variable
- * vor dem Zeiger-Variablennamen gibt den Wert der Adresse zurück auf die der Zeiger zeigt
- *(& pointer) = ?

# Zeiger in C

- speichert Adresse einer anderen Variable

→ repräsentiert Adresse und nicht wie eine Variable einen Wert

- Zeiger wird erstellt, indem bei Deklaration Name mit dem Symbol * beginnt
- Allgemein liefert Kaufmanns-Und & die Adresse einer Variable
- * vor dem Zeiger-Variablennamen gibt den Wert der Adresse zurück auf die der Zeiger zeigt
- *(& pointer) = pointer
- sollte selben Variablentyp haben wie die Variable auf die er zeigt

# Zeiger in C

- speichert Adresse einer anderen Variable

→ repräsentiert Adresse und nicht wie eine Variable einen Wert

- Zeiger wird erstellt, indem bei Deklaration Name mit dem Symbol * beginnt
- Allgemein liefert Kaufmanns-Und & die Adresse einer Variable
- * vor dem Zeiger-Variablennamen gibt den Wert der Adresse zurück auf die der Zeiger zeigt
- *(& pointer) = pointer
- sollte selben Variablentyp haben wie die Variable auf die er zeigt

# Beispiel Zeiger

```
1 int zahl = 7;
2 int *pointer = &zahl;
3
4 printf("Adresse Zahl: %p \n", &zahl);
5 printf("Wert Zahl: %d \n", zahl);
6
7 printf("Adresse pointer: %p \n", &pointer);
8 printf("Wert pointer: %p \n", pointer);
9 printf("Wert von Adresse %p %d \n",
    pointer, *pointer);
```

```
1 printf("Adresse Zahl: %p \n", &zahl);
2 printf("Wert Zahl: %d \n", zahl);
```

```
Adresse Zahl: 0028FEEC
Wert Zahl: 7
```

```
1 printf("Adresse Zahl: %p \n", &zahl);
2 printf("Wert Zahl: %d \n", zahl);
```

Adresse Zahl: 0028FEEC
Wert Zahl: 7

```
1 printf("Adresse pointer: %p \n", &pointer);
2 printf("Wert pointer: %p \n", pointer);
3 printf("Wert von Adresse %p %d \n",
    pointer, *pointer);
```

```
1 printf("Adresse Zahl: %p \n", &zahl);
2 printf("Wert Zahl: %d \n", zahl);
```

```
Adresse Zahl: 0028FEEC
Wert Zahl: 7
```

```
1 printf("Adresse pointer: %p \n", &pointer);
2 printf("Wert pointer: %p \n", pointer);
3 printf("Wert von Adresse %p %d \n",
    pointer, *pointer);
```

```
Adresse pointer: 0028FEE8
Wert pointer: 0028FEEC
Wert von Adresse 0028FEEC 7
```

```
1 printf("Adresse Zahl: %p \n", &zahl);
2 printf("Wert Zahl: %d \n", zahl);
```

```
Adresse Zahl: 0028FEEC
Wert Zahl: 7
```

```
1 printf("Adresse pointer: %p \n", &pointer);
2 printf("Wert pointer: %p \n", pointer);
3 printf("Wert von Adresse %p %d \n",
    pointer, *pointer);
```

```
Adresse pointer: 0028FEE8
Wert pointer: 0028FEEC
Wert von Adresse 0028FEEC 7
```

# Überblick

# Die primitiven Datentypen

## Datentypen

- int
- float
- double
- char

# Die primitiven Datentypen

## Datentypen

- int
- float
- double
- char

## Nicht alle Datentypen aus Java vorhanden

In C existiert kein *boolean* und kein *String*

## Modifizierer

- short
- long
- unsigned
- const
- static

## Datentypen genauer Bitbreite

- int8_t
- int16_t
- int32_t

# Arrays in C

- werden genau wie in Java behandelt
- auch hier mehrdimensional möglich

```
1 int messwerte[3];
2 messwerte[0] = 12;
3 messwerte[1] = 22;
4 messwerte[2] = 4;
```

# Arrays in C

- werden genau wie in Java behandelt
- auch hier mehrdimensional möglich

```
1 int messwerte[3];
2 messwerte[0] = 12;
3 messwerte[1] = 22;
4 messwerte[2] = 4;
```

**Deklaration**

Anders als bei Java muss schon bei der Deklaration des Arrays die Größe festgelegt werden

# Arrays in C

- werden genau wie in Java behandelt
- auch hier mehrdimensional möglich

```
1 int messwerte[3];
2 messwerte[0] = 12;
3 messwerte[1] = 22;
4 messwerte[2] = 4;
```

**Deklaration**

Anders als bei Java muss schon bei der Deklaration des Arrays die Größe festgelegt werden

# #include < stdlib.h >

- Datentyp size_t vom Typ long int

→ wird für Angabe einer Datengröße in Byte verwendet

- Funktion sizeof(Datentyp) gibt bei Übergabe eines Datentyps (int, double, char,...) benötigte Speichergröße in Byte an

→ Rückgabewert ist vom Datentyp size_t

# #include < stdlib.h >

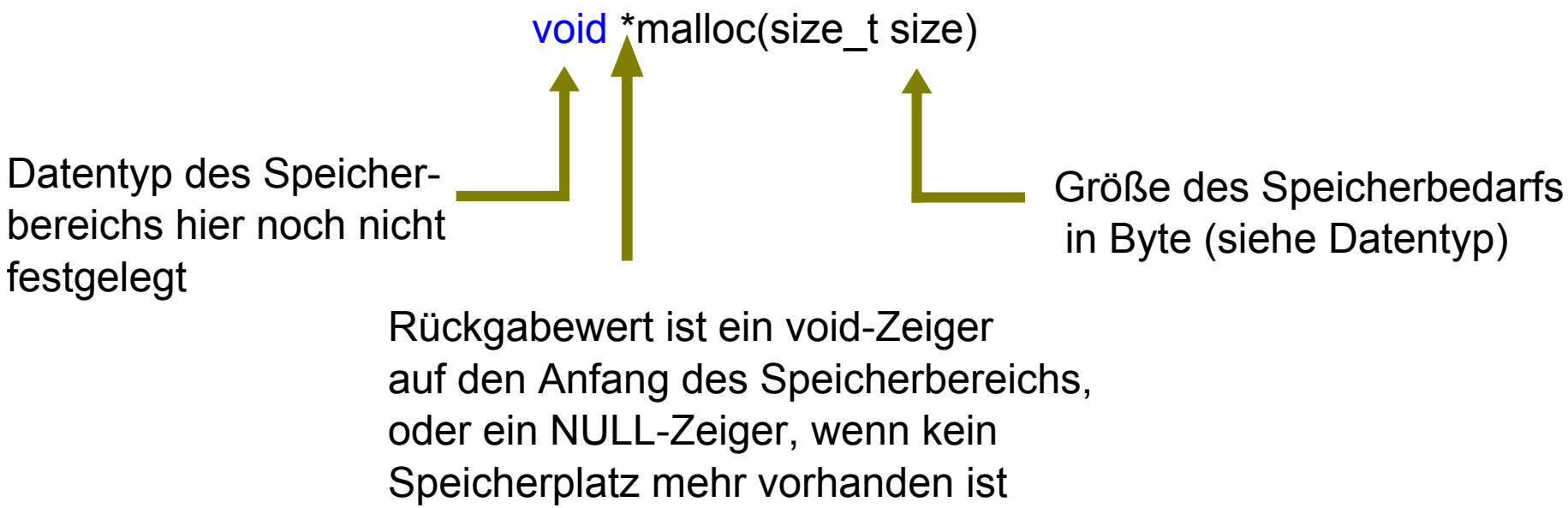
- Datentyp size_t vom Typ long int

→ wird für Angabe einer Datengröße in Byte verwendet

- Funktion sizeof(Datentyp) gibt bei Übergabe eines Datentyps (int, double, char,...) benötigte Speichergröße in Byte an

→ Rückgabewert ist vom Datentyp size_t

# Memory Allocation



```c
int size=0;
int *array;

printf("Array-Groesse eingeben: ");
scanf("%d", &size);

// Speicher reservieren
array = (int *) malloc(size * sizeof(int));

if(array != NULL)
    printf("\nSpeicher ist reserviert\n");
else
    printf("\nKein freier Speicher
        vorhanden.\n");
```

# calloc

- reservierter Speicherinhalt wird von malloc zunächst nicht verändert

→ dort stehen “alte“ Werte

⇒ calloc (Cleared Memory Allocation) macht das selbe wie malloc, es werden jedoch alle Speicherstellen mit 0 initialisiert

# calloc

- reservierter Speicherinhalt wird von malloc zunächst nicht verändert

$\rightarrow$ dort stehen “alte“ Werte

$\Rightarrow$ calloc (Cleared Memory Allocation) macht das selbe wie malloc, es werden jedoch alle Speicherstellen mit 0 initialisiert

# Aufzählungen

- Aufzählung einzeln möglich

```
1 const int mo = 0,di = 1,mi = 2,do = 3,fr =
    4,sa = 5,so = 6;
2 int projektlabor = do;
```

- mit enum können die Variablen direkt zusammengefasst werden

```
1 enum {mo, di, mi, do, fr, sa, so};
2 int projektlabor = do;
```

# Aufzählungen

- Aufzählung einzeln möglich

```
1 const int mo = 0,di = 1,mi = 2,do = 3,fr =
    4,sa = 5,so = 6;
2 int projektlabor = do;
```

- mit enum können die Variablen direkt zusammengefasst werden

```
1 enum {mo, di, mi, do, fr, sa, so};
2 int projektlabor = do;
```

# Strukturen

## struct

Zunächst zusammenhanglose Variablen können mittels der Funktion struct strukturiert werden

```
1 struct datum {
2     int tag;
3     char monat[10]
4     int jahr;
5     };
6 struct datum ferien;
7 ferien.jahr = 2014;
8 struct datum geburtstag = (17, "Dezember",
      1999);
```

# Strukturen

**struct**

Zunächst zusammenhanglose Variablen können mittels der Funktion struct strukturiert werden

```
1 struct datum {
2     int tag;
3     char monat[10]
4     int jahr;
5     };
6 struct datum ferien;
7 ferien.jahr = 2014;
8 struct datum geburtstag = (17, "Dezember",
      1999);
```

# Überblick



# Verzweigungen

- if-Verzweigung
- switch case
- funktionieren in C genau so wie in Java

# if-Verzweigung

```
1    if(geld <= 50)
2        Geldabheben();
3    else
4        Geldausgeben();
```

```
1    if(aussage != 0)
2        wahreAussage();
3    else
4        falscheAussage();
```

- In C werden boolsche Abfragen negiert formuliert

# if-Verzweigung

```
1    if(geld <= 50)
2        Geldabheben();
3    else
4        Geldausgeben();
```

```
1    if(aussage != 0)
2        wahreAussage();
3    else
4        falscheAussage();
```

- In C werden boolsche Abfragen negiert formuliert

# ternärer Auswahloperator

```
1 const char *x;
2 //if-else-Konstrukt
3 if (zahl == 5)
4     x = "Zahl gleich 5";
5 else
6     x = "Zahl ungleich 5";
7
8 // Mit Auswahloperator:
9 x = (zahl == 5) ? "Zahl gleich 5" : "Zahl
      ungleich 5";
```

# Die switch case Verzweigung

```
1    switch(voltage)
2    {
3        case 0: i = 0; break;
4        case 1: i = 1; break;
5        case 2; i = 2; break;
6        default: i = 21;
7    }
```

**switch case**

Bei der switch case Verzweigung unbedingt darauf achten, das break zu setzen.

# Die switch case Verzweigung

```
1    switch(voltage)
2    {
3        case 0: i = 0; break;
4        case 1: i = 1; break;
5        case 2; i = 2; break;
6        default: i = 21;
7    }
```

**switch case**

Bei der switch case Verzweigung unbedingt darauf achten, das break zu setzen.

# while Schleife

```
1 int i = 1;
2 while(i <= 100) {
3     printf("Zahl %d \n", i);
4     i++;
5 }
```

# do while-Schleife

```
1 do {
2     /* do something */
3 }   while(Bedingung == wahr);
```

Es wird immer zu erst die Anweisung ausgeführt und im Anschluss die Bedingung geprüft

# for-Schleife

```
1 int i;
2 for(i = 0; i < 10; i++) {
3     /* do something */
4 }
```

In C darf die Laufvariable nicht im Kopf der for-Schleife deklariert werden

# Überblick

# Allgemeines zu Funktionen

```
1 int addiere(int summmand1, int summand2) {
2     return (summand1 + summand2);
3 }
```

- werden wie in Java geschrieben
- jede Funktion muss über der main-Funktion stehen oder mittels Funktions-Prototyp deklariert werden

# Allgemeines zu Funktionen

```
1 int addiere(int summmand1, int summand2) {
2     return (summand1 + summand2);
3 }
```

- werden wie in Java geschrieben
- jede Funktion muss über der main-Funktion stehen oder mittels Funktions-Prototyp deklariert werden

# Beispiel Funktions-Prototyp

```
1  #include<stdio.h>
2
3  float eingabeZahl();
4  float multi(float zahl1, float zahl2);
5  void ausgabeErgebnis(float ergebnis);
6
7  int main(){...}
8  // Funktionen
9  float eingabeZahl() {...}
10 float multi(float zahl1, float zahl2) {...}
11 void ausgabeErgebnis(float ergebnis) {...}
```

# Einrückung und Absätze nicht vergessen

```
1 #include<stdio.h>
2 float eingabeZahl(){
3 float eingabe;
4 scanf("%f", &eingabe);
5 return eingabe
6 }
7 float multipliziere(float zahl1, float zahl2){
8 return (zahl1*zahl2);
9 }
10 int main(){
11 float faktor1, faktor2, ergebnis;
12 //Eingabe
13 faktor1 = eingabeZahl();
14 faktor2 = eingabeZahl();
15 //Ergebnis
16 ergebnis = multipliziere(faktor1, faktor2);
17 return 0
18 }
```

```c
#include<stdio.h>

float eingabeZahl(){
    float eingabe;
    scanf("%f", &eingabe);

    return eingabe
}

float multipliziere(float zahl1, float zahl2){
    return (zahl1*zahl2);
}

int main(){
    float faktor1, faktor2, ergebnis;

    //Eingabe
    faktor1 = eingabeZahl();
    faktor2 = eingabeZahl();

    //Ergebnis
    ergebnis = multipliziere(faktor1, faktor2);

    return 0
}
```

# Aussagekräftige Variablen verwenden und kommentieren

```
1 #include<stdio.h>
2
3 int main(){
4     float var0 = 0.19
5     float var1 = 350;
6     float var2 = var1*var2;
7
8     return 0;
9 }
```

# Leicht übertriebenes Beispiel

```
1 #include<stdio.h>
2
3 //berechnet den Mehrwertsteueranteil von einem Betrag
4 int main(){
5     float mwstFaktor = 0.19    //aktueller
          Mehrwertsteuer-Faktor
6     float betrag = 350;        //Betrag von dem die MWST
          berechnet werden soll
7     float mwst = var1*var2;    //Mehrwertsteueranteil
8
9     return 0;
10 }
```

# Überblick

# Quellen

http://de.wikipedia.org/
http://www.c-howto.de/
http://de.wikibooks.org/wiki/C-Programmierung