Prozedurale Programmierung
Datentypen und Variablen

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

Prof. Dr. F.J. Schmitt

Rinderbeinscheiben in Rotweinsauce

Hochschule **Rosenheim**
University of Applied Sciences

```
t_gemuese      kar, sel, kr;          // Karotten, Sellerie, Kohlrabi
t_fleisch      fl;                    // Rinderbeinscheiben
t_fluessigkeit oel, wein, bruehe;     // Öl, Rotwein, Brühe
t_pulver       staerke;               // Speisestärke
t_gewuerze     sl, pf;                // Salz, Pfeffer
```

Datentyp **Variablenbezeichner** **Kommentar**

Anmerkung:
die Datentypen dieses Beispiels sind bereits zusammengesetzte komplexe Datenstrukturen

ELEMENTARE DATENTYPEN

Motivation

- Algorithmus wird immer auf Daten angewendet
- in Programmiersprache ist festgelegt, welche Daten verarbeitet werden können

- Datentyp
  - Definitionsmenge von Daten plus aller Operatoren und Funktionen, die auf dieser Menge definiert sind

- Definitionsmenge eines Datentyps kann aufgrund der begrenzten Rechenkapazität eines Computers nur endlich sein

Datentypen in C

- Unterscheidung zwischen
  - elementare Datentypen
  - abgeleitete Datentypen → später

Elementare Datentypen

- Drei wesentliche Typen:
  - (1) Ganze Zahlen — -11000, 0, 1, 2, 23000
  - (2) Gleitpunktzahlen — -12.345, 0.0, 2.5, 3.0
  - (3) Zeichen — ‘A‘, ‘B‘, ‘f‘, ‘g‘, ‘5‘, ‘&‘, ‘*‘
    - Existieren nur indirekt
    - Abbildung auf ganze Zahlen

Zeichen**ketten** (Strings) sind in C kein elementarer Datentyp

Ganze Zahlen (1)

- Datentyp lautet: **`int (integer)`**

- Qualifizierer können dem Datentyp `int` vorangestellt werden
  - Verändern den Wertebereich
  - `short` und `long` legen fest wie viele Bits für den Datentyp verwendet werden
    - Bsp: 32 Bit-Rechner-Architektur: `short int` 16 Bit;
      `long int` 32 Bit (Speicherverbrauch)
  - `signed` und `unsigned` geben an, ob der Datentyp sowohl positive als auch negative oder nur positive Werte annehmen kann
  - Kombination der Qualifizierer möglich

Ganze Zahlen (2)

- Fortsetzung Qualifizierer
  - ist `short` oder `long` nicht angegeben
    - 32 Bit-Architektur: `int` wird oft wie `long` abgebildet
    - 16 Bit-Architektur: `int` wird oft wie `short` abgebildet
  - ist `signed` und `unsigned` nicht angegeben
    - default: `signed` wird angenommen
    - `int` entspricht `signed int`
- Bei Verwendung von Qualifizierern kann `int` weggelassen werden

```
long int  grosseZahl;
long      grosseZahl;

short int kurzeZahl;
short     kurzeZahl;
```

Ganze Zahlen (3)

- Wertebereich ganzzahliger Datentypen (Windows / VStudio)

Typ	Größe	Wertebereich		
short int	16 Bit	-32768	bis	32767
unsigned short int	16 Bit	0	bis	65.535
unsigned int	32 Bit	0	bis	4.294.967.295
int	32 Bit	-2.147.483.648	bis	2.147.483.647
unsigned long	32 Bit	0	bis	4.294.967.295
long	32 Bit	-2.147.483.648	bis	2.147.483.647

Achtung: architekturabhängig!
Bsp: 32 Bit für long ist nur eine Mindestanforderung – kann größer sein,
auf manchen Systemen (manche DSPs in eingebetteten Systemen z.B. 40 Bit)

Operatoren auf ganzen Zahlen

- C bietet eine Fülle von Operatoren für den Datentyp `int`
- Verschiedene Klassen:
  - Arithmetische Operatoren
  - Logische Operatoren
  - Bit-Operatoren

Arithmetische Operatoren (1)

- Folgende arithmetische Grundrechenarten sind implementiert:

Operator	Erklärung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo

Arithmetische Operatoren (2)

- Division (Ganzzahlendivision)
  - Quotient zweier Zahlen wird berechnet
  - Ergebnis ist eine **ganze** Zahl!
  - Bsp.: 2/3 = 0, 5/4 = 1

- Modulo
  - Berechnet den Rest der Ganzzahlendivision
  - Oft verwendet: herausfinden, ob eine Zahl durch andere teilbar ist (Ergebnis = 0)

```
5   %  2   =  1
-5  %  2   = -1
5   % -2   =  1
-5  % -2   = -1
```

Gleitpunktzahlen

- Implementierungen in C:
  - Einfach genaues Zahlenformat → Datentyp **`float`**
  - Doppelt genaues Zahlenformat → Datentyp **`double`**
  - **`long double`**: unterschiedliche Genauigkeit, da maschinenabhängig verschiedene Umsetzungen

Datentyp	Bits
`float`	32
`double`	64
`long double`	>= 64

```
Beispiele:
3.1415
0.147 oder
.147 oder
147e-3
```

Operatoren für Gleitpunktzahlen

- Folgende arithmetische Grundrechenarten sind implementiert:

Operator	Erklärung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division

- Modulo-Operator existiert nicht (wäre sinnlos…)
- Divisionsoperator liefert eine „reelle“ Zahl (Gleitpunktzahl)

Mathematische Funktionen

- Standard-Mathematik-Bibliothek stellt verschiedenste mathematische Funktionen zur Verfügung
  - Potenzfunktion, Quadratwurzel, Logarithmische Funktionen, Exponentialfunktion
  - Rundungsfunktionen und Betrag
  - Trigonometrische Funktionen (sin(x), cos(x), tan(x), …) → rad!
  - …
- Parameter und Rückgabewerte sind vom Typ `double`
- für `float` muss ein „f“ an den Funktionsnamen angehängt werden
- Header-Datei **`math.h`** muss inkludiert werden

Gleitpunktzahlen – Wertebereiche

- Prinzip

dezimal:	Vorzeichen	Mantisse	Exponent
-12.3 ➔	-	0.123	+2

- Die interne Darstellung ist nicht dezimal, sondern binär
- Wertebereiche:

TYP	Byte	Mantisse (Stellen / Bit)	Exponent (Werte / Bit)
float	4	8 / 23	10^{-38} - 10^{38} / 8
double	8	11 / 52	10^{-308}- 10^{308} / 11
long double	8 (10)	18 / 64	10^{-4932}-10^{4932} / 15

Zeichen (1)

- Zeichen existieren in C nur indirekt
- Speichern eines Zeichens ⇒ zugehöriger ASCII-Code wird abgespeichert
- Ausgabe eines Zeichens (bspw. printf) ⇒ ASCII-Code wird an die Ausgabefunktion übergeben

- Datentyp: **`char`**
  - 8 Bit lang
  - Kann Zahlen von -128 bis 127, bzw. 0 bis 255 annehmen
  - Speichert eigentlich Zahlenwerte
  - Bsp: 'A' = 65; 'B' = 66; …
  - Sollte nur für Buchstaben verwendet werden

Zeichen (2)

- Angabe durch **einfache** Hochkommata
- Steuerzeichen können ebenfalls angegeben werden
- Zwei Zeichen finden in einem `char` nicht Platz

```
Beispiele:

char c = 'A';

char newline = '\n';
```

Dies ist ein einzelnes Steuerzeichen!

Operatoren auf Zeichen

- Es können die selben Operatoren wie für ganze Zahlen verwendet werden
- i.A. aber nur selten notwendig, Zeichen mit Operatoren zu manipulieren

Funktionen für Zeichen (1)

- C-Standard-Bibliothek bietet einige Funktionen
- Header-Datei **`ctype.h`** muss inkludiert werden

- Einige Beispiele
  - Funktionen zur Klassifikation von Zeichen
    - isalpha (c) → 0 wenn c kein Buchstabe, sonst nicht 0
    - islower (c) → 0, wenn c kein Kleinbuchstabe, sonst nicht 0
    - isupper (c) → 0, wenn c kein Großbuchstabe, sonst nicht 0

```
char c = 'A';
long i;
i = isupper(c);
```

Funktionen für Zeichen (2)

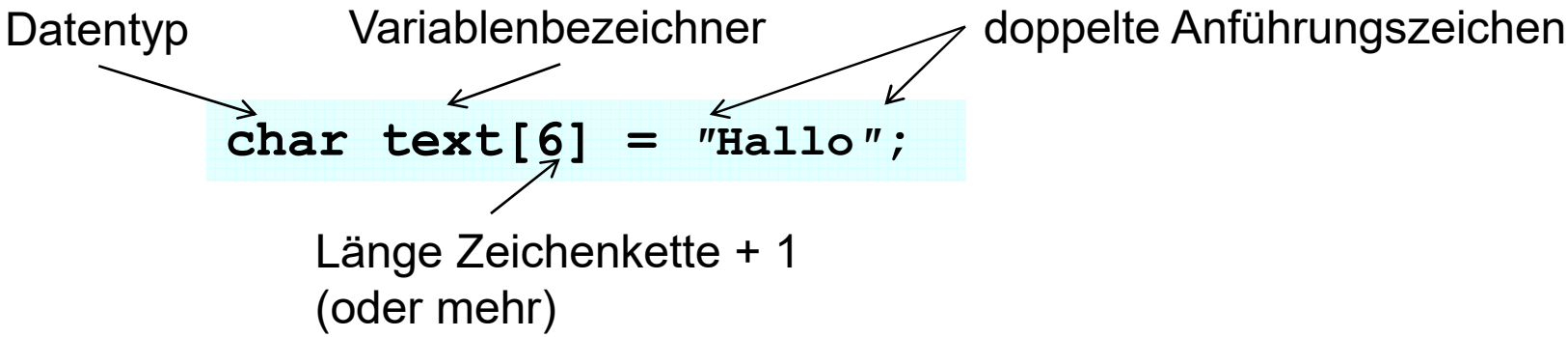
- Einige Beispiele
  - Funktionen zur Umwandlung von Zeichen
    - tolower (c) → liefert das Zeichen c umgewandelt in einen Kleinbuchstaben (keine Umlaute)
    - toupper (c) → liefert das Zeichen c umgewandelt in einen Großbuchstaben (keine Umlaute)

```
char c = 'A';
// ...
c = tolower(c);
```

Zeichenketten

- sind in C kein elementarer Datentyp (Details → später)
- sondern eine Folge hintereinander abgespeicherter Zeichen
  - Typ: char

- Definition mit Initialisierung



Achtung: Zuweisung mit = geht nur direkt bei der Initialisierung

Zeichenketten

- folgendes geht **nicht**:

```
char text[6];
text = "Hallo";
```

- **richtig** wäre:

```
char text[6];
strcpy(text, "Hallo");
```

Achtung:

- es ist immens wichtig, dass die Längenangaben stimmen
- sonst: Programmabstürze / seltsames Verhalten
- generell ist strcpy eine unsichere Funktion (Buffer Overflow) genau wie viele andere Stringfunktionen

Elementare Funktionen für Zeichenketten (1)

- C-Standard-Bibliothek stellt eine Vielzahl von Funktionen zur Manipulation von Zeichenketten zur Verfügung
- Header-Datei `string.h` muss inkludiert werden
- Auswahl enthält nächste Folie, wobei Parameter
  - `s` und `t` Zeichenketten sind
  - `n` eine ganze Zahl

Elementare Funktionen für Zeichenketten (2)

Funktion	Beschreibung
`strcat(s,t)`	hängt t an s an
`strncat(s,t,n)`	hängt die ersten n Zeichen von t an s an
`strcmp(s,t)`	vergleicht s und t zeichenweise und liefert negative Zahl (s<t), 0 (s==t) oder positive Zahl (s>t)
`strncmp(s,t,n)`	wie strcmp, vergleicht jedoch nur die ersten n Zeichen
`strcpy(s,t)`	kopiert t nach s
`strncpy(s,t,n)`	wie strcpy, kopiert jedoch nur die ersten n Zeichen
`strlen(s)`	liefert die Länge von s ohne Null-Byte

Wahrheitswerte

- Sind in C nicht typorientiert implementiert
  - erst ab C99 und C++ verfügbar

- oft wird Datentyp `int` verwendet
  - Zahl 0 bedeutet „falsch“ (logisch interpretiert)
  - Alle anderen Zahlen (auch negative) werden als „wahr“ interpretiert

- Alternative: Verwendung von „enum“ → später

Rationale Zahlen

- zur exakten Darstellung von Brüchen (z.B. 1/3) ohne Konvertierung in eine Gleitkommazahl
- in C nicht verfügbar

void

- „Typ“ `void` bedeutet in C soviel wie „nichts“
- Ist eigentlich kein Datentyp, da
  - er keinen Wertebereich und
  - keine Operatoren hat.

Zusammenfassung

- Was ist ein Datentyp
- ganze Zahlen + Operatoren
- Gleitkommazahlen + Operatoren
- Zeichen
- Wahrheitswerte

VARIABLEN UND KONSTANTEN

Rinderbeinscheiben in Rotweinsauce

Hochschule **Rosenheim**
University of Applied Sciences

```
t_gemuese      kar, sel, kr;          // Karotten, Sellerie, Kohlrabi
```

Datentyp **Variablenbezeichner** **Kommentar**

Anmerkung:
die Datentypen dieses Beispiels sind bereits zusammengesetzte komplexe Datenstrukturen

Grundlegende Eigenschaften von Variablen und Konstanten (1)

Hochschule **Rosenheim**
University of Applied Sciences

```
t_gemuese      kar, sel, kr;          // Karotten, Sellerie, Kohlrabi
```

Datentyp **Variablenbezeichner** **Kommentar**

- Name
  - auch: Bezeichner
  - eindeutige Identifizierung
- Speicherort
  - Hardware-Register oder Speicheradresse → später mehr
- Datentyp
  - welche Daten können in der Variablen abgespeichert werden
- Wert
  - Variable hat zu jeder Zeit einen Wert
    (Ort → Speicher trägt zu jeder Zeit ein Bitmuster)

Grundlegende Eigenschaften von Variablen und Konstanten (2)

Hochschule **Rosenheim**
University of Applied Sciences

- Gültigkeitszeitraum
  - Variable muss vor Gebrauch „angefordert“ bzw. „erschaffen“ werden
  - Variable kann solange verwendet werden, bis sie zerstört wird

- Sichtbarkeitsbereich
  - Existierende Variablen können in manchen Programmteilen unsichtbar sein

Definition von Variablen

- Variablen sind veränderbar
- bevor Variable verwendet werden kann, muss sie definiert werden
  - Festlegung von Datentyp, Name sowie Speicherklasse (→ später)

```
long   wert;
double ergebnis;
```

  - entsprechender Speicher wird reserviert
  - mehrere Variablen können, durch Komma getrennt, definiert werden

```
long differenzbetrag, summe;
```

  - Vorgriff: für externe globale Variablen genügt eine Deklaration

Name von Variablen

- Folge von Buchstaben (A…Za…z), Ziffern (0…9) und Unterstrich (_)
- erstes Zeichen: nur Buchstabe oder Unterstrich
  - ab dem zweiten Zeichen auch Ziffern
- Umlaute/Sonderzeichen sind nicht erlaubt
- Vorsicht: Groß- und Kleinschreibung wird unterschieden!

```
long  wert;
long  Wert;
long  wErT;
```

Drei verschiedene Variablen !

Initialisierung einer Variablen

- Setzen eines Wertes gleich bei der Definition (Anlegen) einer Variable

```
long wert = 15;
```

- (Lokale) Variablen werden in C nicht automatisch mit 0 vorbesetzt
  - der für die Variable angeforderte Speicher wird nicht extra gelöscht
  - im Speicher steht irgendein Wert (vor der Anforderung)

Programmbeispiel

```
...
int main()
{
    double  wert;
    double  zehnfacherWert;

    zehnfacherWert = 10 * wert;
    return 0;
}
```

- Variable `wert` wurde nicht initialisiert ⇒
  enthält irgendeine Zahl
- Nichtinitialisierung ist häufiger Programmierfehler!

Konstanten

- Wert einer konstanten Variablen ist nicht veränderlich
- Verwendung sehr sinnvoll, wenn Unveränderbarkeit eines Datenwerts hervorgehoben werden soll
- Schlüsselwort `const`

```
const double PI = 3.1415927;
```

- Programmierkonvention: Namen von Konstanten in Großbuchstaben schreiben

Verwendung Variablen und Konstanten

- Prinzipiell keine Unterschiede

```
#include <stdio.h>
const double PI = 3.1415927;
int main(void)
{
  double radius = 10;
  double flaecheHalbkreis;
  double flaecheViertelkreis;

  flaecheHalbkreis = radius * radius * PI / 2;

  flaecheViertelkreis = flaecheHalbkreis / 2;

  printf ("Flaeche Halbkreises:   %g\n", flaecheHalbkreis);
  printf ("Flaeche Viertelkreis   %g\n", flaecheViertelkreis);
  return 0;
}
```

Definition von Variablen/Konstanten

- sollten nur am Anfang eines Blocks stehen
  - also **direkt** nach der öffnenden Klammer {
  - bevor irgendeine andere Anweisung kommt (Ausnahme: Präprozessoranweisungen/Kommentare)

Zuweisungen (1)

- Welchen Wert hat b am Ende?

```
...
int main(void)
{
    long a,b;

    a = 1;
    b = a;
    a = 2;

...
}
```

Zuweisungen (2)

- Gegeben sind zwei Variablen a und b vom Typ `long`. Tauschen Sie die Werte der Variablen.

```
...
int main(void)
{
    long a,b;

    a = 1;
    b = 2;
}
```

Datentypen

- Was sagen Sie zu folgenden Zuweisungen?

```
...
char c = 32;
long l = 5223;
double d = 3.13;

c = l; // ???
l = c; // ???
l = d; // ???
d = l; // ???
```

- Implizite Typumwandlung findet statt!

Implizite Typumwandlung

- Findet statt, wenn ein Wert eines Datentyps in einen anderen Datentyp umgewandelt werden soll
- Bsp:
  - `char` in `long` oder `double`
  - `long` in `double`
- Typumwandlung wird zur Übersetzungszeit festgestellt
- Vorsicht: Datenverlust kann auftreten!, evtl. undefiniert
- Verlustbehaftete Typumwandlung z.B.:
  - `long` in `char`
  - `double` kann in keinen anderen Datentyp umgewandelt werden

Implizite Typwandlung (2)

- Vor der Berechnung werden bei nicht-kompatiblen Typen
  - alle float nach double konvertiert
  - alle short/char nach int
  - bei dann immer noch verschiedenen Typen eines Operanden: int → unsigned → long → double

- Umwandlungen hin zu einem Typ mit kleinerem Wertebereich sollten **nie implizit** durchgeführt werden (Datenverlust)

Explizite Typumwandlung (1)

- Explizite Angabe eines Zieldatentyps
- Beispiel (hier noch implizit):

```
...
long   anzahlTage   = 3;
long   anzahlHemden = 289;
double hemdenProTag = anzahlHemden / anzahlTage;
```

- Ergebnis = 96 (Ganzzahldivision)
- Was ist notwendig um ein exaktes Ergebnis zu bekommen?

Explizite Typumwandlung (2)

- Umwandlung mindestens einer der beiden Operanden in eine Gleitpunktzahl (Casting)

```
...
long   anzahlTage   = 3;
long   anzahlHemden = 289;
double hemdenProTag = (double)anzahlHemden / anzahlTage;
```

- Verschiedene Cast-Operatoren
  = geklammerter Zieldatentyp
  - (long)
  - (double)
  - …

Explizite Typumwandlung (3)

```
int    i = 3;
float  y;

y = i / 2;                    // y = ???

y = (float) i / 2;            // y = ???

y = (float)(i / 2);           // y = ???

y = (float) i / (float)2;     // y = ???
```

Zusammenfassung

- Variablen
- Konstanten
- Zuweisungen
- Typwandlung
  - implizit
  - explizit