HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ingenieurinformatik

Einführung in die Programmiersprache C

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Das Modul „Ingenieurinformatik“ wird in den Bachelorstudiengängen Maschinenbau, Fahrzeugtechnik, Luft- und Raumfahrt angeboten

- **Teil 1: Grundlagen der Programmierung**
  - Einführung in die Grundlagen der Programmiersprache C
  - Umfang insgesamt: drei Semesterwochenstunden (3 SWS), davon 2 SWS seminaristischer Unterricht („Vorlesung“) und 1 SWS Rechnerübung („Praktikum“, 6 Termine)
  - Die regelmäßige Bearbeitung von Programmieraufgaben am eigenen Rechner zu Hause ist unbedingt notwendig, um den Stoff zu verstehen und die Prüfung mit Erfolg zu bestehen.
  - Schriftliche (Teil-)Prüfung (60 Minuten)
- **Teil 2: Numerik für Ingenieure**
  - Einführung in die Arbeit mit MATLAB und Simulink
  - Umfang insgesamt: zwei Semesterwochenstunden (2 SWS), davon 1 SWS seminaristischer Unterricht („Vorlesung“) und 1 SWS Rechnerübung („Praktikum“, 6 Termine)
  - Schriftliche (Teil-)Prüfung (60 Minuten)

Achtung: Abhängig davon, wann Sie Ihr Studium begonnen haben, müssen Sie sich entweder zu den Teilprüfungen „Programmierung“ und „Numerik für Ingenieure“ separat anmelden (Variante 1) oder für die Gesamtprüfung „Ingenieurinformatik“ (Variante 2).

Variante 1 (Teilprüfungen „Programmierung“ und „Numerik“):

- Jede Teilprüfung muss mindestens mit der Note 4.0 bestanden werden.
- Die Teilprüfungen dürfen in unterschiedlichen Semestern geschrieben werden.
- Die Endnote ergibt sich aus den Noten der Teilprüfungen „Programmierung“ und „Numerik für Ingenieure“ im Verhältnis 60:40 gewichtet.

Variante 2 (Gesamtprüfung „Ingenieurinformatik“):

- Beide Prüfungsteile müssen im selben Semester geschrieben werden.
- Aus den Punkten, die in „Programmierung“ und „Numerik für Ingenieure“ erreicht wurden wird eine Gesamtpunktzahl ermittelt. Dabei werden die beiden Prüfungsteile im Verhältnis 60:40 gewichtet. Die Endnote ergibt sich aus der Gesamtpunktzahl.
- Es muss also nicht jeder Prüfungsteil separat „bestanden“ werden.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Wozu Ingenieurinformatik? Ich studiere doch Maschinenbau, Fahrzeugtechnik oder Luft- und Raumfahrt!

Der Rechner als Werkzeug

Beispiele: Textverarbeitung, CAD-Systeme, Simulation, Datenbanken, Internet, FEM-Berechnungen, Projektplanung
→ Der Ingenieur nutzt Informationstechnik

Der Rechner als Projektbestandteil

Beispiele: Automatisierung von Maschinen und Anlagen, Fahrzeugelektronik, Leittechnik
→ Der Ingenieur definiert Anforderungen an die Informationstechnik („Auftraggeber“)

Der Rechner als Entwicklungsobjekt

Beispiele: Entwicklung intelligenter Maschinen (z. B. für die Produktion aber auch als Produkte), Entwicklung von Messtechnik (z. B. zur Versuchsauswertung)
→ Der Ingenieur entwickelt Informationstechnik

Einführung in die Programmiersprache C

1. **Einleitung**
2. **Zahlensysteme, Darstellung von Informationen**
   Dezimal-, Dual-, Hexadezimalsystem, Integer- und Fließkommazahlen
3. **Mein erstes C-Programm**
   Compiler und Linker, Beispiele einfacher C-Programme
4. **Kontrollstrukturen**
   Befehlssequenzen, Verzweigungen, Schleifen, Sprunganweisungen
5. **Namen, Datentypen, Operatoren**
   Namen, Schlüsselwörter, Datentypen, Konstanten, Operatoren
6. **Funktionen**
   Deklaration und Definition von Funktionen, lokale/globale Variablen
7. **Vektoren und Zeichenketten**
   Vektoren und Matrizen, einfache Sortierverfahren, Zeichenketten
8. **Adressen und Zeiger**
   Anwendung von Zeigern und Zeigerarithmetik

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Lehrbücher zur Programmiersprache C

- Gerd Küveler, Dietrich Schwoch
  Informatik für Ingenieure und Naturwissenschaftler (Bd. 1 und 2)
  Vieweg-Verlag (siehe auch: www.springerlink.de)
- Brian Kernighan, Dennis Ritchie
  Programmieren in C
  Hanser Fachbuch, 2. Ausgabe, 1990
- **Bevor Sie ein teures Buch kaufen:**
  Gemeinsam mit Ihnen lernen Millionen Studierende überall auf der Welt dieselben Informatik-Grundlagen. Sie finden im Internet (Google, Wikipedia…) alles, was Sie zum Lernen benötigen. Suchen Sie auch nach englischen Begriffen!

Skripte (Fachschaft)

- Vorlesungsfolien, alte Klausuren

Kapitel 1
Einleitung

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Was ist Informatik?

Informatik ist die **Wissenschaft von der systematischen Verarbeitung von Informationen**, insbesondere der automatischen Verarbeitung mit Hilfe von Rechenanlagen. Historisch hat sich die Informatik als **Wissenschaft aus der Mathematik entwickelt**, während die Entwicklung der ersten Rechenanlagen ihre Ursprünge in der Elektrotechnik und Nachrichtentechnik hat.

Dennoch stellen **Computer nur ein Werkzeug** und Medium der Informatik dar, um die theoretischen Konzepte praktisch umzusetzen. Der niederländische Informatiker Edsger Wybe Dijkstra formulierte: „In der Informatik geht es genauso wenig um Computer wie in der Astronomie um Teleskope."

(Quelle: Wikipedia)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

1. Teilgebiet: Theoretische Informatik

Formale Sprachen und Automatentheorie (Eigenschaften eines abstrakten Rechners), Algorithmen (Sortieren), Berechenbarkeit, Kryptographie

2. Teilgebiet: Technische Informatik

Aufbau und Struktur der Computerhardware, Rechnerarchitektur, Mikroprozessoren, Rechnernetzwerke und Rechnerkommunikation

3. Teilgebiet: Praktische Informatik

Programmiersprachen, Übersetzer (Compiler), Softwareengineering, Betriebssysteme, verteilte Systeme, Datenbanken

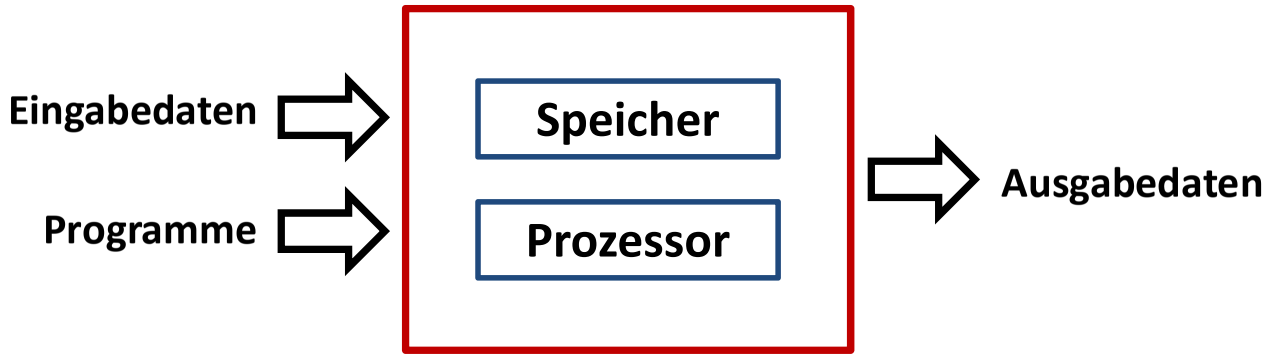
4. Teilgebiet: Angewandte Informatik

Anwendung der Informatik in anderen Wissenschaften,
CAx (Computer Aided) Systeme , z. B. CAD (Design), CAE (Engineering),
Wirtschaftsinformatik, Medizininformatik...

Kapitel 2
Zahlensysteme, Darstellung von Informationen

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Ein Computer speichert und verarbeitet mehr oder weniger große Informationsmengen, je nach Anwendung und Leistungsfähigkeit.



Die Programmierung derartiger Rechner setzt Kenntnisse über die Darstellung der Informationen voraus!

Wegen der in Computern verwendeten Bauelemente werden alle Informationen binär (dual) dargestellt.

Informationselement	Realisierung
Schalter	offen / geschlossen
Festplatte	Magnetfeld „ein“ / „aus“
Speicher (RAM)	Spannung ein / aus
CD-ROM, DVD	Reflexion / keine Refl.

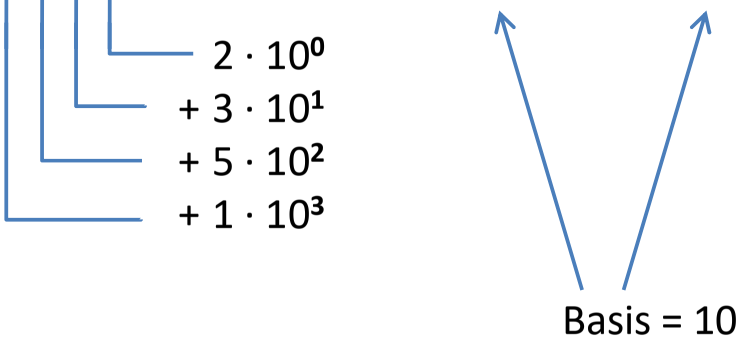
Kapitel 2 – Zahlensysteme

Zahlen werden im täglichen Leben im **Dezimalsystem** dargestellt. Das gilt erst recht für ganze Zahlen. Das Dezimalsystem ist ein **Stellenwertsystem** (auch „**polyadisches Zahlensystem**“).

Beispiel:

$$1532 = 2\cdot 1 + 3\cdot 10 + 5\cdot 100 + 1\cdot 1000$$

$$= 2\cdot 10^0 + 3\cdot 10^1 + 5\cdot 10^2 + 1\cdot 10^3$$



Für die Darstellung ganzer Zahlen in Stellenwertsystemen gilt ganz allgemein (z. B. im Dezimalsystem mit der Basis b = 10):

$$X = a_0\cdot b^0 + a_1\cdot b^1 + a_2\cdot b^2 + \ldots + a_n\cdot b^n$$

Oder kürzer:

$$X = \sum a_i\cdot b^i \qquad i = 0, 1, \ldots$$

$$a_i \in \{0, 1, 2, \ldots, b\text{-}1\}$$

Zur Erinnerung: $b^0 = 1$

Im Zusammenhang mit Digitalrechnern sind drei Stellenwert-systeme besonders relevant:

- **Das Dezimalsystem**
- **Das Dualsystem (Binärsystem)**
- **Das Hexadezimalsystem**

Die Bedeutungen der ersten beiden Systeme liegen auf der Hand. Das Hexadezimalsystem wird oft für eine verkürzte Darstellung von Binärzahlen genutzt, weil sich jeweils vier benachbarte Binärziffern zu einer einzelnen Hexadezimalziffer zusammenfassen lassen.

Mit der Darstellung von Binär- und Hexadezimalzahlen muss der Programmierer ebenso vertraut sein, wie mit der Umrechnung von Zahlen zwischen diesen Stellenwertsystemen.

Das **Dualsystem (Binärsystem)** besitzt folgende Eigenschaften:

Basis	2
Menge der Ziffern	{0, 1}
Stellenwerte	Potenzen von 2 2^0 2^1 2^2 2^3 2^4 … 1_{10} 2_{10} 4_{10} 8_{10} 16_{10} …

Beispiel: $1011_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$

$= 1_{10} + 2_{10} + 0_{10} + 8_{10}$

$= 11_{10}$

Da in nur einem Bit keine sinnvolle Informationsmenge gespeichert werden kann, arbeitet man mit Gruppen von mehreren/vielen Bits:

- Eine Gruppe von 8 Bits nennt man ein **Byte**
- Eine Gruppe von 2 Bytes nennt man ein **Wort (Word)**
- Eine Gruppe von 4 Bytes nennt man ein **Doppelwort (Longword)**
- 1024 Bytes nennt man ein **Kilobyte** (1 KB, mit großem „K“)
- 1024 Kilobytes nennt man ein **Megabyte**
- 1024 Megabyte nennt man ein **Gigabyte**
- 1024 Gigabyte nennt man ein **Terabyte**

- 1 KB = 2^{10} Bytes = 1 024 Bytes
- 1 MB = 2^{20} Bytes = 1 048 576 Bytes
- 1 GB = 2^{30} Bytes = 1 073 741 824 Bytes
- 1 TB = 2^{40} Bytes = 1 099 511 627 776 Bytes

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Mit **8 Bits** (oder **einem Byte**), von denen jedes Bit zwei Zustände (1/0) annehmen kann, lassen sich **256 verschiedene Kombinationen** bilden:

0000 0000
0000 0001
0000 0010
0000 0011
0000 0100
0000 0101
0000 0110
. . .
1111 1110
1111 1111

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiele:

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Das **Hexadezimalsystem** besitzt folgende Eigenschaften:

Basis	16
Menge der Ziffern	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} A bis F besitzen die dezimalen Nennwerte zehn bis fünfzehn!
Stellenwerte	Potenzen von 16 16^0 16^1 16^2 16^3 … 1_{10} 16_{10} 256_{10} 4096_{10} …

Beispiel: $AFFE_{16} = 14 \cdot 16^0 + 15 \cdot 16^1 + 15 \cdot 16^2 + 10 \cdot 16^3$

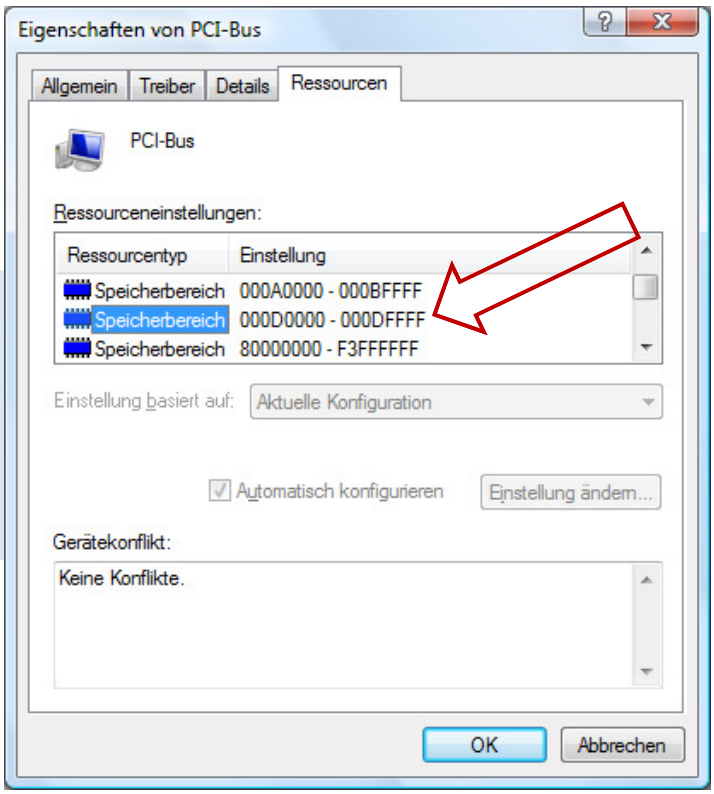
$= 14_{10} + 240_{10} + 3840_{10} + 40960_{10}$

$= 45054_{10}$

Beispiel:

Speicherbelegung am PCI-Bus bei Intel-PC, 32-Bit-Adresse

Windows Vista:
- **Rechtsklick auf „Computer“,**
- **Eigenschaften,**
- **Geräte-Manager,**
- **Systemgeräte,**
- **PCI-Bus**



$000D\ FFFF_{16}$ =

0000 0000 0000 1101 1111 1111 1111 1111_2

Kapitel 2 – Zahlensysteme

2.2. Umrechnung zwischen Zahlensystemen

Zwischen den hier relevanten Stellenwertsystemen Dezimal, Binär und Hexadezimal gibt es **sechs Umrechnungsverfahren**, die auf den folgenden Folien vorgestellt werden:

a) Binär → Dezimal
b) Hexadezimal → Dezimal
c) Dezimal → Binär
d) Dezimal → Hexadezimal
e) Binär → Hexadezimal
f) Hexadezimal → Binär

2.2.1. Umrechnung Binär → Dezimal

Methode: **Summe der Zweierpotenzen bilden, dabei am besten „von rechts“ beginnen**

Beispiel 1: $10111_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4$
$= 1 + 2 + 4 + 16$
$= 23_{10}$

Beispiel 2: $110011_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5$
$= 1 + 2 + 16 + 32$
$= 51_{10}$

2.2.1. Umrechnung Binär → Dezimal

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Hilfstabelle: Zweierpotenzen

n	2^n	n	2^n
0	1	11	2 048
1	2	12	4 096
2	4	13	8 192
3	8	14	16 384
4	16	15	32 768
5	32	16	65 536
6	64	17	131 072
7	128	18	262 144
8	256	20	1 048 576
9	512	24	16 777 216
10	1 024	32	4 294 967 296

2.2.2. Umrechnung Hexadezimal → Dezimal

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Methode: Summe der Sechzehnerpotenzen bilden, dabei am besten „von rechts“ beginnen

Beispiel 1: $1E3_{16}$

$$= 3 \cdot 16^0 + 14 \cdot 16^1 + 1 \cdot 16^2$$
$$= 3 + 224 + 256$$
$$= 483_{10}$$

Beispiel 2: $FFFF_{16}$

$$= 15 \cdot 16^0 + 15 \cdot 16^1 + 15 \cdot 16^2 + 15 \cdot 16^3$$
$$= 15 + 240 + 3840 + 61440$$
$$= 65535_{10}$$

2.2.2. Umrechnung Hexadezimal → Dezimal

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Hilfstabelle: Sechzehnerpotenzen

n	16^n
0	1
1	16
2	256
3	4 096
4	65 536
5	1 048 576
6	16 777 216
7	268 435 456
8	4 294 967 296

2.2.3. Umrechnung Dezimal → Binär

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Methode: Fortgesetzte Division durch zwei

Die umzuwandelnde Dezimalzahl wird fortlaufend durch zwei dividiert, bis null erreicht wird. Die dabei auftretenden Divisionsreste – in umgekehrter Reihenfolge – ergeben die gesuchte Binärzahl.

Beispiel:

```
223 : 2 = 111  Rest 1
111 : 2 = 55   Rest 1
55  : 2 = 27   Rest 1
27  : 2 = 13   Rest 1
13  : 2 = 6    Rest 1
6   : 2 = 3    Rest 0
3   : 2 = 1    Rest 1
1   : 2 = 0    Rest 1
```

Die entsprechende Binärzahl lautet: 1 1 0 1 1 1 1 1

2.2.4. Umrechnung Dezimal → Hexadezimal

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Methode: Fortgesetzte Division durch 16

Die Dezimalzahl wird fortlaufend durch 16 dividiert, bis null erreicht wird. Die dabei auftretenden Divisionsreste – in umgekehrter Reihenfolge – ergeben die gesuchte Hexadezimalzahl.

Beispiel:

```
443 : 16 =  27   Rest 11
 27 : 16 =   1   Rest 11
  1 : 16 =   0   Rest 1
```

Die entsprechende Hexadezimalzahl lautet: 1 B B

Hilfstabelle: Ziffern A…F im Hexadezimalsystem

A	B	C	D	E	F
10	11	12	13	14	15

2.2.4. Umrechnung Dezimal → Hexadezimal

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Hilfstabelle: Vielfache von 16

n	n · 16	n	n · 16
1	16	9	144
2	32	10	160
3	48	11	176
4	64	12	192
5	80	13	208
6	96	14	224
7	112	15	240
8	128	16	256

2.2.5. Umrechnung Binär→ Hexadezimal

Methode: 4er-Gruppen von Binärzahlen bilden

Die umzuwandelnde Binärzahl wird von rechts nach links in 4er-Bündel von Binärziffern gruppiert. Anschließend werden die Bündel in die entsprechenden Hexadezimalziffern umgewandelt.

Beispiel:

1010	1111	1111	1110
A	F	F	E

Hilfstabelle: 4er-Bündel von Binärziffern und Hexadezimalziffern

0000	0001	0010	0011	0100	0101	0110	0111
0	1	2	3	4	5	6	7
1000	1001	1010	1011	1100	1101	1110	1111
8	9	A	B	C	D	E	F

2.2.6. Umrechnung Hexadezimal → Binär

Methode: Hexadezimal in 4er-Bündel von Binärziffern „auflösen“

Die umzuwandelnde Hexadezimalzahl wird Ziffer für Ziffer in 4er-Bündel von Binärziffern „aufgelöst“.

Beispiel:

3	7	A	F
0011	0111	1010	1111

Führende Nullen zu Beginn der Binärzahl können weggelassen werden:

$$37AF_{16} = 11\ 0111\ 1010\ 1111_2$$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 2 – Zahlensysteme

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Addition im Dezimal- und im Binärsystem

- Was muss man wissen, um eine Addition im Dezimalsystem durführen zu können?
- Welche Vorteile bietet das Binärsystem gegenüber dem Dezimalsystem bei der Durchführung von Berechnungen?

Addition im Dezimalsystem:

```
   1.234.567
+  2.345.678
     11 11
-----------
   3.580.245
```

Addition im Binärsystem:

```
  1100 0101 1100
+    1001 1011
       1 1
-----------------
  1100 1111 0111
```

Kontrolle: $3164_{10} + 155_{10} = 3319_{10}$ ✓

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Additionstafel für Dezimalzahlen:

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1		2	3	4	5	6	7	8	9	10
2			4	5	6	7	8	9	10	11
3				6	7	8	9	10	11	12
4					8	9	10	11	12	13
5						10	11	12	13	14
6							12	13	14	15
7								14	15	16
8									16	17
9										18

55 Regeln!

Dualzahlen:

+	0	1
0	0	1
1		10

Nur 3 Regeln!

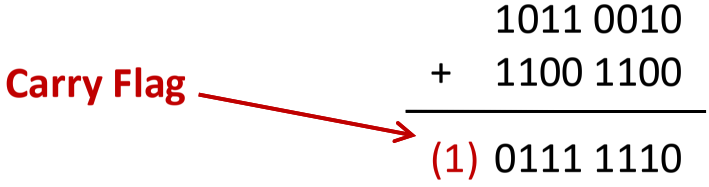
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Überlauf und „Carry Flag“

Bei der Addition kann es zum **Überlauf** kommen, falls für das Ergebnis ein begrenzter Speicherplatz zur Verfügung steht!

Der Prozessor merkt sich dieses Ereignis durch Setzen eines speziellen Status-Bits („**Carry Flag**“). Dieses Bit kann im weiteren Programmverlauf abgefragt werden, falls auf den Überlauf reagiert werden muss.

Beispiel: **Addition von zwei 8-Bit-Zahlen, für das Ergebnis steht ebenfalls ein Speicherplatz von 8 Bit zur Verfügung**



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 2 – Zahlensysteme

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Mit den verfügbaren Bits (zum Beispiel einem Register im Prozessor) müssen neben dem **Betrag der Zahl** auch das **Vorzeichen** abgespeichert werden.

- Annahme: 4-Bit-Register

.	.	.	.
3	2	1	0

Erster Ansatz:

- Bit 3 ist das Vorzeichen-Bit: Bit 3 = 0 entspricht +
  Bit 3 = 1 entspricht –
- Bit 0 - Bit 2 speichern den Betrag der Zahl
- Beispiel:

+2	0	0	1	0
-2	1	0	1	0

Welche Nachteile besitzt dieses Verfahren?

Zweiter Ansatz:

Dezimalzahl	2er-Komplement	Dezimalzahl	2er-Komplement
0	0 0 0 0		
1	0 0 0 1	-1	1 1 1 1
2	0 0 1 0	-2	1 1 1 0
3	0 0 1 1	-3	1 1 0 1
4	0 1 0 0	-4	1 1 0 0
5	0 1 0 1	-5	1 0 1 1
6	0 1 1 0	-6	1 0 1 0
7	0 1 1 1	-7	1 0 0 1
		-8	1 0 0 0

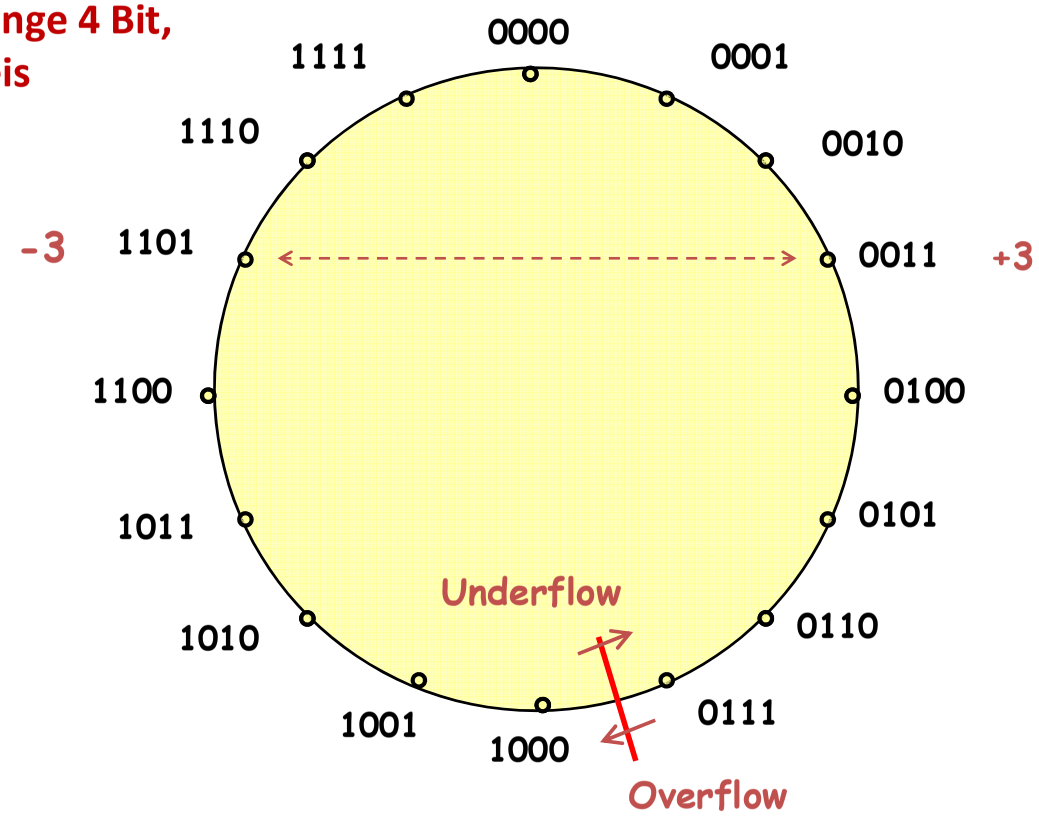
Darstellbarer Zahlenbereich beim sog. **„2er-Komplement“**: -8 ... +7
Fast alle Prozessoren verwenden diese Darstellung – mit mehr als 4 Bits!

Eigenschaften der 2er-Komplementdarstellung

a) Positive Zahlen: höchstwertiges Bit = **0** hier Bit 3 (typisch Bit 7, 15...)
   Negative Zahlen: höchstwertiges Bit = **1**
b) Wie wird für eine positive Zahl die Darstellung der zugehörigen negativen Zahl berechnet? **Antwort: 2er-Komplement**
   1.) Bilde bitweises Komplement (0→1 und 1→0) – „Einerkomplement“
   2.) Addiere anschließend 1 zum Einerkomplement
c) Umkehrung: Wie wird aus einer negativen Zahl die zugehörige positive Zahl berechnet? **Antwort: Ebenfalls mit dem 2er-Komplement!**
d) Bekannte Rechenregeln aus dem Dualsystem bleiben erhalten
e) z + (-z) = 0 – Übertrag (Carry Flag) wird ignoriert
f) Subtraktion wird auf Addition zurückgeführt – siehe d)
g) Mögliche Bereichsüberschreitung beachten!

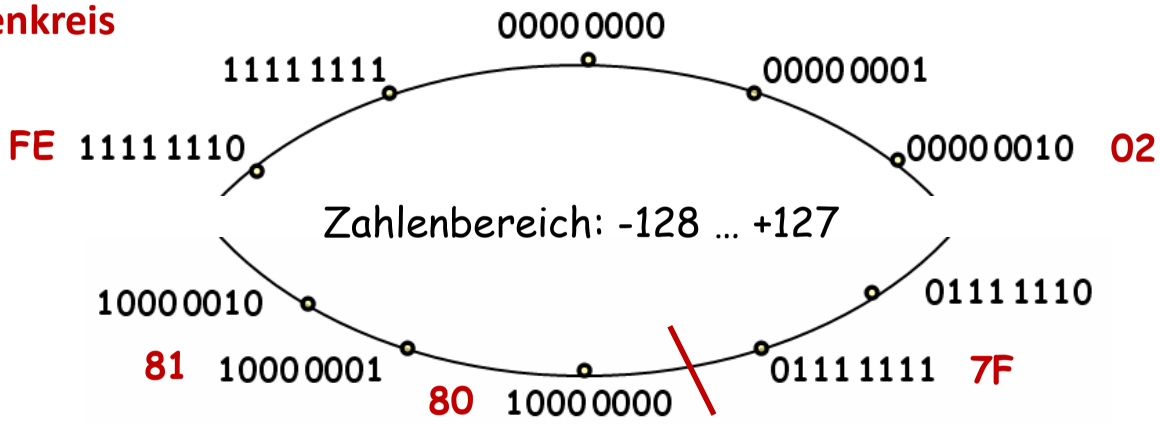
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Registerlänge 4 Bit, Zahlenkreis



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Registerlänge 8 Bit, Zahlenkreis



Allgemein (für eine Registerlänge von n Bit): $-2^{n-1} \ldots +2^{n-1} - 1$

n = 16:	-32 768	...	+32 767
n = 32:	-2 147 483 648	...	+2 147 483 647
n = 64:	$-9{,}22 * 10^{18}$	...	$+9{,}22 * 10^{18}$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

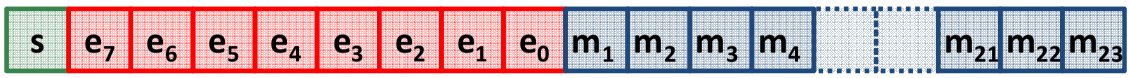
Kapitel 2 – Zahlensysteme

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Zahlen mit Nachkommastellen werden in einem Rechner völlig anders dargestellt als ganze Zahlen und auch völlig anders verarbeitet! Die Zahlen werden zuerst in eine Standardform (halblogarithmisch) gebracht:

$$37{,}625 = +0{,}37625 \times 10^{2} = s \times m \times B^{ex}$$
$$(+100101{,}101)_2 = (+1{,}00101101 \times 2^{101})_2$$

Im Register werden **Vorzeichen**, **Mantisse** und **Exponent** abgespeichert. Die Basis ist stets 2 und wird deshalb nicht gespeichert.



Vorzeichen: Es ist nur ein Bit notwendig, 0 ↔ positiv und 1 ↔ negativ. Die verbleibenden Bits im Register müssen sich **Mantisse** und **Exponent** teilen. Ist die Mantisse zu lang, dann wird sie abgeschnitten.

Bei Zahlen kleiner als 1 möchte man negative Exponenten vermeiden.

$$(+0{,}375)_{10} = (+0{,}011)_2 = (+1{,}1 \times 2^{-10})_2$$

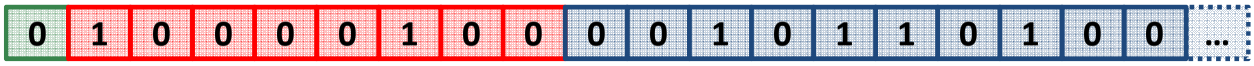
Deshalb addiert man zum Exponenten stets eine feste Zahl („bias“).

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

$(+100101{,}101)_2 = (+1{,}00101101 \times 2^{101})_2$

32 Bit:
Exponent 8 Bits (bias = 127)
und **Mantisse 23 Bits**

101
\+ 0111 1111 (bias)
1000 0100



64 Bit: **Exponent 11 Bits** (bias = 1023) und **Mantisse 52 Bits**

Konsequenzen der Gleitkommadarstellung:

\+ großer Zahlenbereich darstellbar (sehr kleine/große Zahlen)

\- Zahlen werden meist nicht exakt dargestellt, da die Mantisse abgeschnitten wird (bei 32 Bit werden nur die ersten 7 führenden Dezimalstellen gespeichert, bei 64 Bit die ersten 15) → **Rundungsfehler!**

\- Rechenoperationen dauern länger als Operationen mit ganzen Zahlen.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beim **Programmieren** wird durch den **Typ einer Variablen** festgelegt, ob eine Zahl als Gleitkommazahl, als ganze Zahl mit VZ oder als ganze Zahl ohne VZ gespeichert wird. Beispiele in der Programmiersprache C:

- Ganze Zahlen mit Vorzeichen

```
short n;          16-Bit  (Bereich:  -32768 ... +32767)
int i;            32-Bit
long long j;      64-Bit
```

- Ganze Zahlen ohne Vorzeichen

```
unsigned short u;   16-Bit  (Bereich:  0 ... +65535)
```

- Gleitkommazahlen

```
float x;          32-Bit  („einfache Genauigkeit“)
double y;         64-Bit  („doppelte Genauigkeit“)
```

Vergleichbare Variablentypen gibt es auch in anderen Programmiersprachen wie Java oder Visual Basic.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 2 – Zahlensysteme

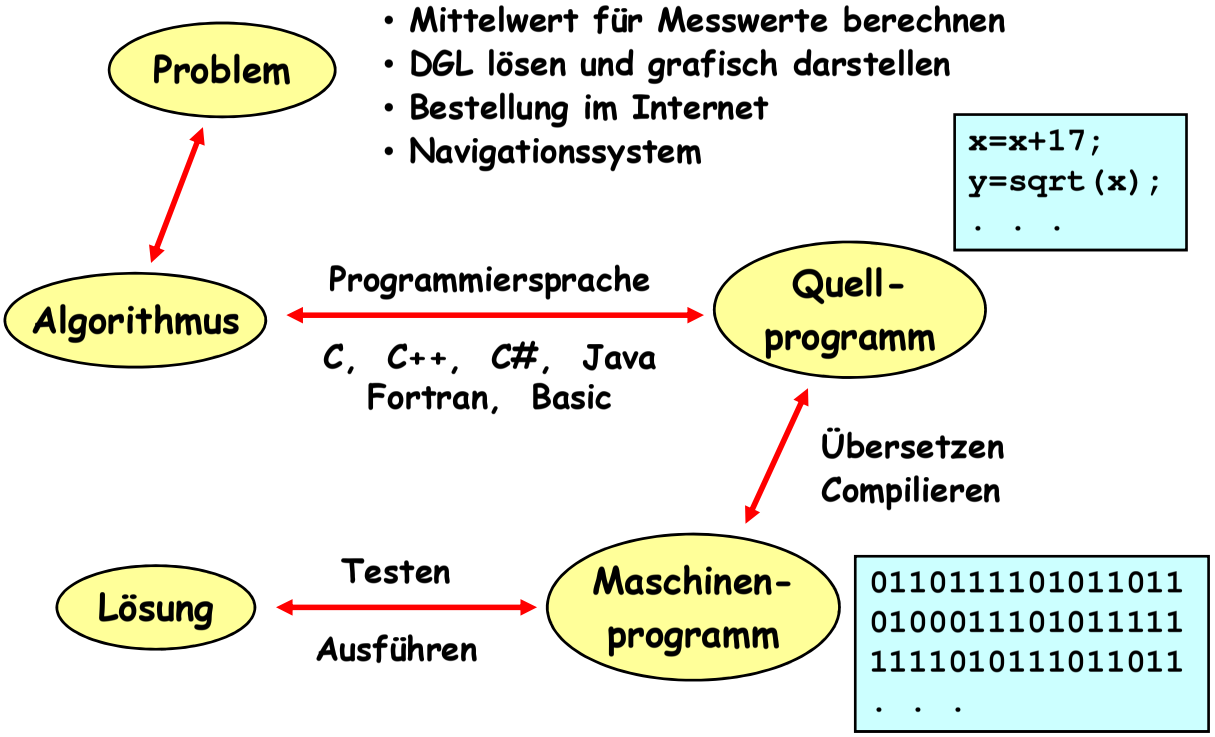
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

1. Welchen Dezimalzahlen entsprechen die folgenden vier Zahlen
   10101100_2 $E3_{16}$ $F7_{16}$ 38_{16}
   wobei es sich jeweils um 2er-Komplementdarstellungen mit 8 Bit handelt? ($E3_{16}$ ist die Hexdarstellung einer Binärzahl: $E3_{16} = 1110\ 0011_2$)
2. Wie lautet die 2er-Komplementarstellung (dual, hex) der Dezimalzahlen -13 und -43 , wenn 8- bzw. 16-Bit-Register verwendet werden?
3. Eine Gleitkommazahl wird in der 32-Bit-Darstellung abgespeichert. Wie groß ist die betragsmäßig größte bzw. kleinste Gleitkommazahl, die dargestellt werden kann? (Alle Zahlen, die kleiner sind, werden als 0.0 gespeichert.)
4. Wie viele führende Dezimalstellen werden maximal abgespeichert? Überlegen Sie sich hierzu, wie die Zahl 1.0 abgespeichert wird. Wie lautet die kleinste Zahl größer als 1.0, die noch gespeichert werden kann? Hieraus ergibt sich die Anzahl der führenden Stellen, die mit 32 Bit gespeichert werden können.

Kapitel 3
„Mein erstes C-Programm“

Gliederung

Kapitel 3 – „Mein erstes C-Programm“



3. „Mein erstes C-Programm“

Algorithmus, Lösungsweg:

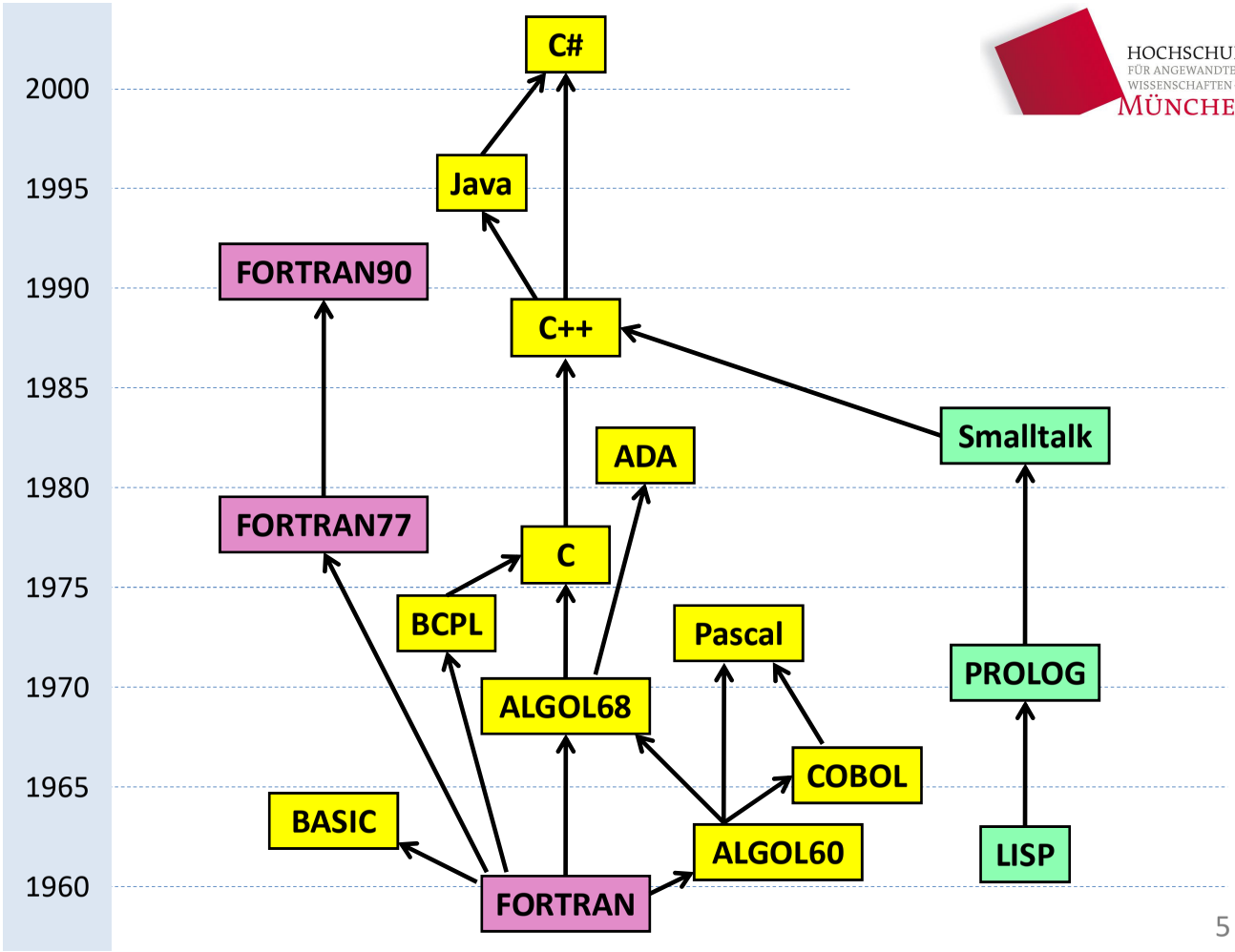
- Methode (Kochrezept) zur Lösung eines Problems, die **eindeutig** ist, aus **elementar ausführbaren Schritten** besteht und die **endlich** ist.
- Eine Verarbeitungsvorschrift, die so präzise formuliert ist, dass sie von einem mechanisch oder elektronisch arbeitenden Gerät ausgeführt werden kann. (Informatik-Duden)

Programm:

- Eine Folge von Anweisungen, durch die die Verarbeitung von Daten in einem Computer gesteuert wird.
- Formulierung eines Algorithmus und der zugehörigen Daten.

Programmiersprache:

- Eine Sprache zur Formulierung von Algorithmen und Datenstrukturen (d. h. Programmen) in einer für den Computer geeigneten Form (Schnitt-stelle zwischen Benutzer und Computer).
- Es müssen **Syntax** und **Semantik** einer Programmiersprache eindeutig definiert sein, damit man prüfen kann, welche Zeichenfolgen als Programm zugelassen sind (Syntax) und was sie auf dem Rechner bewirken (Semantik).



3.1. Einleitung

Dennis Ritchie (standing) and Ken Thompson begin porting UNIX to the PDP-11

Dennis Ritchie
"Erfinder" von C

(* 9. September 1941 in Bronxville, New York; † am 12. Oktober 2011)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Warum Programmierung in C…?

C wird in der Praxis sehr häufig verwendet:

- Programmierung von Steuergeräten (Embedded Systems) zum Beispiel in Fahrzeugen, Handys, Robotern
- Systemprogrammierung

C ermöglicht das Erstellen sehr effizienter Programme:

- Wenig Speicherplatzverbrauch
- Schnell – viele Anwendungen in Steuergeräten sind zeitkritisch

C unterstützt eine Vielzahl von Mikroprozessoren und Steuergeräten:

- Es gibt C-Compiler für die entsprechende Hardware und eine Vielzahl von Hilfsmittel, um die Programmierung zu unterstützen, zum Beispiel:
- Programmbibliotheken (fertige Programme für DGL, Matrizen, Statistik, FEM)
- Codegenerierung mit MATLAB für verschiedenste Steuergeräte

Betriebssysteme werden häufig in C programmiert, zum Beispiel Linux.

Viele Elemente von C finden sich auch in anderen Programmiersprachen, zum Beispiel in C++, Java, Visual Basic und C#.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 3 – „Mein erstes C-Programm“

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

Schreibe ein C-Programm, das die zwei Zahlen 3 und 4 addiert und das das Ergebnis am Bildschirm ausgibt.

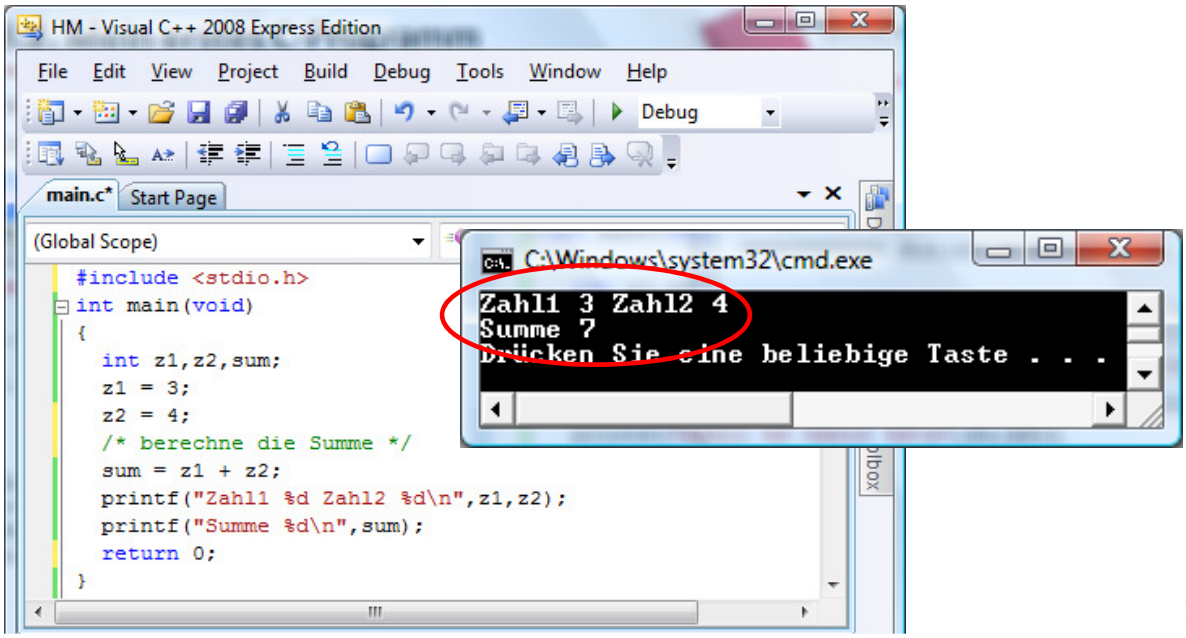
Ziel:

- Welche Schritte müssen am Rechner durchgeführt werden, um ein Programm zu erstellen, zu übersetzen und auszuführen?
- Wie sieht das Programm für dieses Problem aus?
- Wie sieht ein typisches C-Programm aus?
- Danach soll das Programm noch verbessert werden.

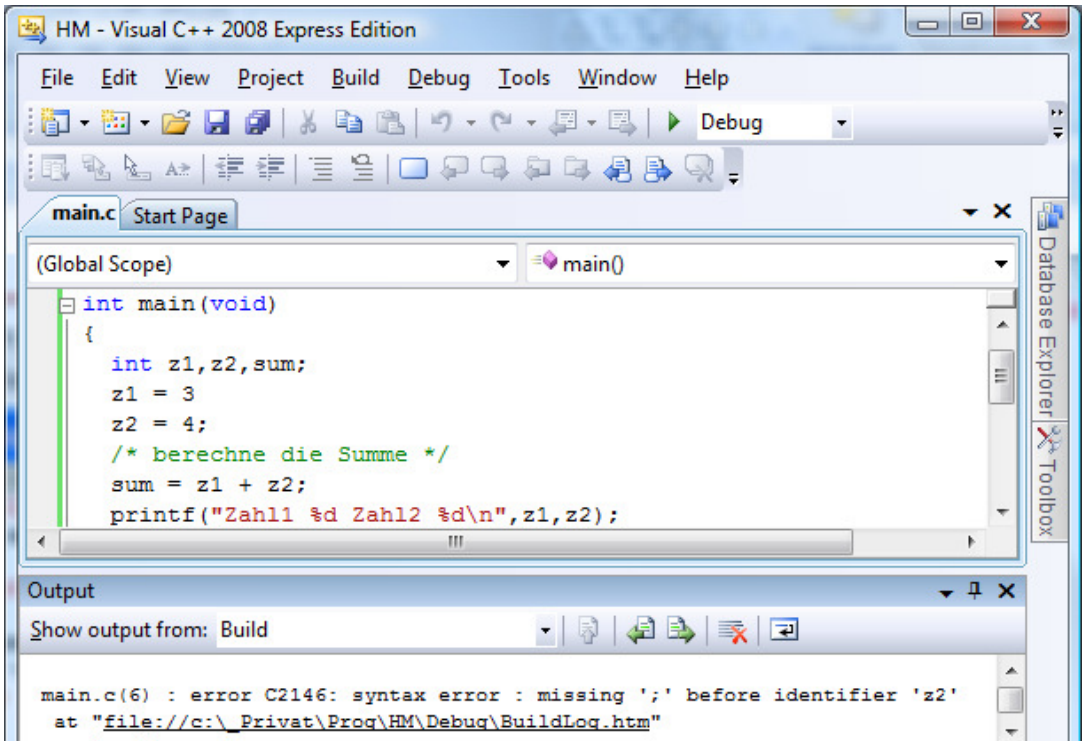
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Es müssen 3 Schritte durchgeführt werden:

1. C-Quelltext mittels Editor eingeben und speichern (**`addition.c`**).
2. Das Programm in Maschinesprache übersetzen (**`addition.exe`**).
3. Jetzt kann das Programm gestartet werden.



Eine **Programmierumgebung** ist ein Hilfsmittel, um Programme zu erstellen, zu verwalten, zu testen und auszuführen.



Bekannte Programmierumgebungen:

Qt Creator:

- http://www.qt.io
- Kostenloser Download im Internet (Lizenz: GPL)
- Für Windows, Mac OS X und Linux verfügbar
- Wird im Praktikum verwendet, zu Hause installieren!

Microsoft Visual Studio:

- Kostenlose Community-Edition
- Für Microsoft Windows verfügbar
- https://www.visualstudio.com/de/vs/community/

C-Programme, die mit einer dieser Programmierumgebungen erstellt werden, laufen auch unter anderen Umgebungen, wenn man sich an die im ISO/IEC 9899-Standard beschriebenen Regeln hält. In speziellen Umgebungen, z. B. Programmierung von Steuergeräten, stehen nicht alle Funktionen zur Verfügung. Beispiel: Bildschirm-Ausgabebefehle…

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#include <stdio.h>

int main(void)

{

  int z1,z2,sum;

  z1 = 3;

  z2 = 4;

  /* berechne die Summe */

  sum = z1 + z2;

  printf("Zahl1 %d Zahl2 %d\n",z1,z2);

  printf("Summe %d\n",sum);

  return 0;

}
```

Variablendefinition
z1, z2 und sum heißen Variablen
- Speicherplatz reservieren
- Datentyp festlegen (int - Integer)

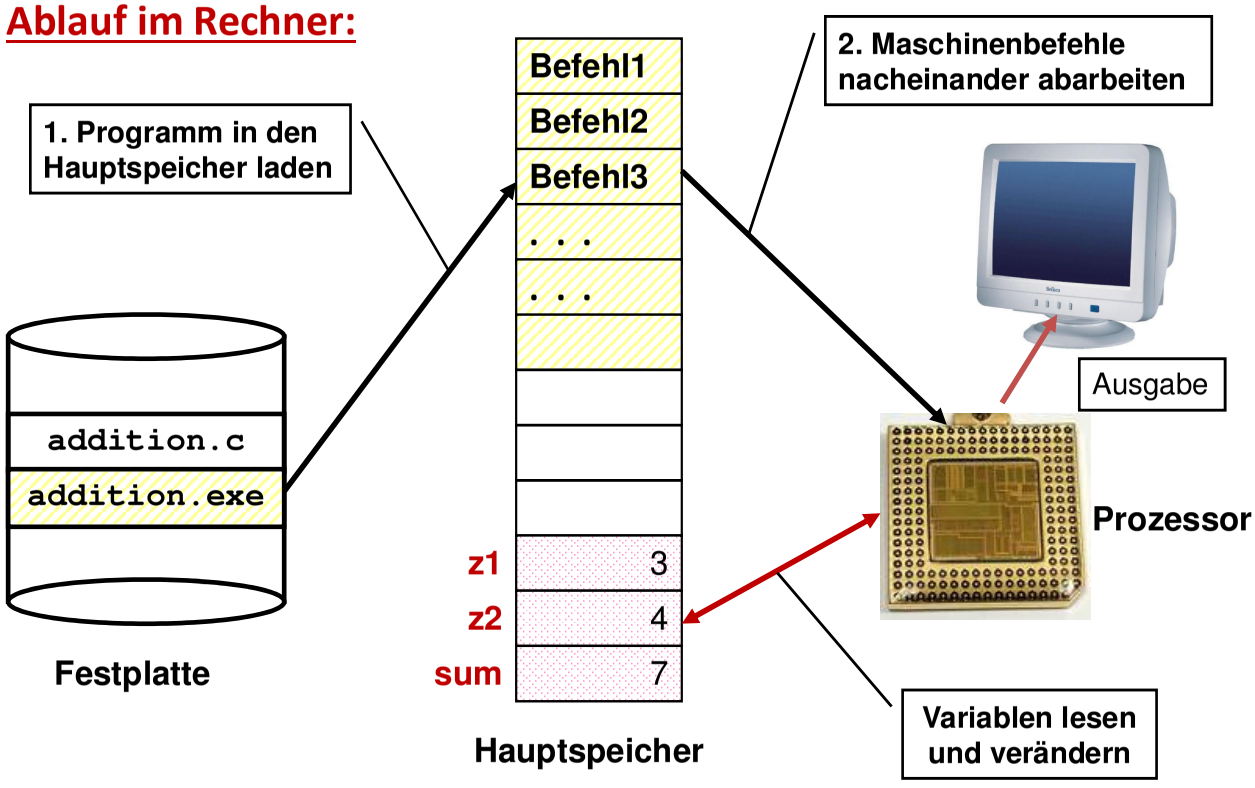
Wertzuweisung
Wert in Speicherzelle schreiben

Arithmetischer Ausdruck
Wert der rechten Seite berechnen und dann der linken Seite zuweisen

printf : Ausgabe am Bildschirm
Inhalt von " . . . " ausgeben
%d : Formatelement - dezimale Ausgabe
\n : Steuerzeichen - neue Zeile beginnen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ablauf im Rechner:



3.2. Mein erstes C-Programm

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#include <stdio.h>     /* k2addit1.c */
int main (void)
{
  int z1, z2, summe;
  printf("Erste Zahl eingeben:\n");
  scanf("%d",&z1);
  printf("Zweite Zahl eingeben:\n");
  scanf("%d",&z2);
  /* berechne die Summe */
  summe = z1 + z2;
  printf("Zahl1 %d Zahl2 %d\n",z1,z2);
  printf("Summe : %d \n",summe);
  return 0;
}
```

scanf
das Programm wartet solange, bis eine Eingabe erfolgt ist, interpretiert die Eingabe als ganze Zahl (%d) und speichert das Ergebnis in der Speicherzelle für z1 (&z1) ab.

3.2. Mein erstes C-Programm

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#include <stdio.h>    /* k2addit2.c */
int main(void)
{
  float z1, z2, sum;
  printf("Erste Zahl eingeben :\n");
  scanf("%f", &z1);
  printf("Zweite Zahl eingeben :\n");
  scanf("%f", &z2);
  sum = z1 + z2;
  printf("Zahl1:%f Zahl2:%f\n",z1,z2);
  if(sum < 0.0)
  {
    printf("Ergebnis ist negativ!");
  }
  else
  {
    printf("Summe : %f \n",sum);
    printf("Ergebnis ist positiv!");
  }
  return 0;
}
```

float
Speicherplatz für eine Fließkommazahl reservieren

%f
Formatelement für eine Fließkommanzahl

sum < 0
Bedingung – entweder wahr (erfüllt) oder falsch

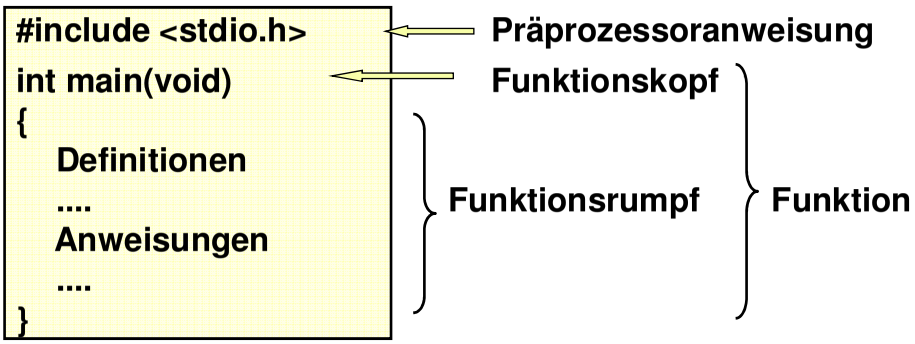
if - else - Kontrollstruktur
wenn die Bedingung erfüllt ist, dann nur den ersten Teil ausführen ansonsten nur den zweiten Teil (Alternative)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 3 – „Mein erstes C-Programm“

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

a) Struktur eines einfachen C-Programms



Ein C-Programm besteht mindestens aus der Funktion **main** (Hauptprogramm).

Eine **Funktion** besteht aus:

- Funktionskopf – **main** ist der Name der Funktion
- Funktionsrumpf

Der **Funktionsrumpf** beginnt mit **{** und endet mit **}** und enthält:

- Definition von Variablen
- Anweisungen – zum Beispiel arithmetische Ausdrücke, Wertzuweisungen
- Kommentare – beginnen mit **/*** und enden mit ***/**

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

b) Definition von Variablen

int z; Ganze Zahl mit Vorzeichen
float x, y; Fließkommazahl

Die Anweisung
int z;
definiert eine Variable vom Typ Integer (ganze Zahl mit Vorzeichen).

- Bei der Variablendefinition wird ein Datentyp wird festgelegt, hier: Integer
- Speicherplatz wird reserviert, um den Wert einer ganzen Zahl speichern zu können (für eine Integer-Variable typischerweise 4 Bytes)
- z kann nur ganze Zahlen speichern, keine Gleitkommazahlen
- z ist der Name der Variablen
- Der Variablenname ist frei wählbar, muss aber eindeutig sein
- Inhalt der Speicherzelle (Wert) ist unter dem Namen z ansprechbar

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

c) Anweisungen

```
z = 17;
sum = z1 + z2;
```

Beide Anweisungen sind **Wertzuweisungen**. Der Operator = ist der **Zuweisungsoperator** (= ist kein mathematisches Gleichheitszeichen !!!).

Bedeutung von =
1. Berechne zuerst den Wert der rechten Seite
2. Weise danach das Ergebnis der linken Seite zu

Beispiel:

```
z = z + 5;
```

Erklärung:
1. Hole den aktuellen Wert aus der Speicherzelle z („hole den aktuellen Wert der Variablen z“)
2. Addiere zu diesem Wert 5 hinzu
3. Speichere danach das Ergebnis in der Speicherzelle z

Der alte Wert von z wird überschrieben und ist damit verloren.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

d) Operatoren

+ - * / %	Arithmetische Operatoren
< >	Relationale Operatoren
=	Zuweisungsoperator
&	Adressoperator

e) Schlüsselwörter – reservierte Wörter

int , **float** , **if**, **else** sind Schlüsselwörter der Sprache C. Schlüsselwörter haben eine spezielle Bedeutung in einer Programmiersprache und dürfen nicht als Namen für Variablen oder Funktionen verwendet werden.

f) Kontrollstruktur

```
if ( Bedingung )
{
   /* Anweisungen, falls Bedingung wahr */
}
else
{
   /* Anweisungen, falls Bedingung falsch */
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

g) Ein- und Ausgabe

Ausgabe am Bildschirm mit printf:

printf("...");	Text " ... " am Bildschirm ausgeben
printf("...%d...", z);	Text " ... " und aktuellen Wert von z ausgeben

Einlesen von der Tastatur mit scanf:

scanf("%d", &z);

Formatierungszeichen („Platzhalter“ für Variablen):

%d	:	Ein-/Ausgabe ganzer Zahlen in der Dezimaldarstellung
%f	:	Ein-/Ausgabe von Gleitkommazahlen

Steuerzeichen:

\n	:	Zeilenumbruch („beginne eine neue Zeile“)

h) Sonstiges

- C unterscheidet zwischen **Groß- und Kleinschreibung**
  zahl1 ≠ **ZAHL1** ≠ **Zahl1** (drei verschiedene Namen!)
- Semikolon **;** kennzeichnet das Ende von Definitionen und Anweisungen (eine neue Zeile alleine bedeutet nicht das Ende einer Anweisung!)

```
#include  < stdio.h   >
int main ( void   )
{
  int z1,
      z2, sum          ;
 z1 = 3 ;   z2 = 4 ;
        /* berechne die Summe
                            */
 sum =   z1
             + z2        ;
printf("Zahl1:%d Zahl2:%d\n",
             z1,
    z2  )    ;
     printf(
  "Summe: %d\n"     , sum) ;
  return    0   ;
      }
```

```
#include <stdio.h>
int main(void)
{
 int z1,z2,sum;
 z1 = 3;
 z2 = 4;
 /* berechne die Summe */
 sum = z1 + z2;
 printf("Zahl1:%d Zahl2:%d\n",
        z1,z2);
 printf("Summe: %d\n",sum);
 return 0;
}
```

Kapitel 4
Strukturierte Programmierung und Kontrollstrukturen

Gliederung

Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

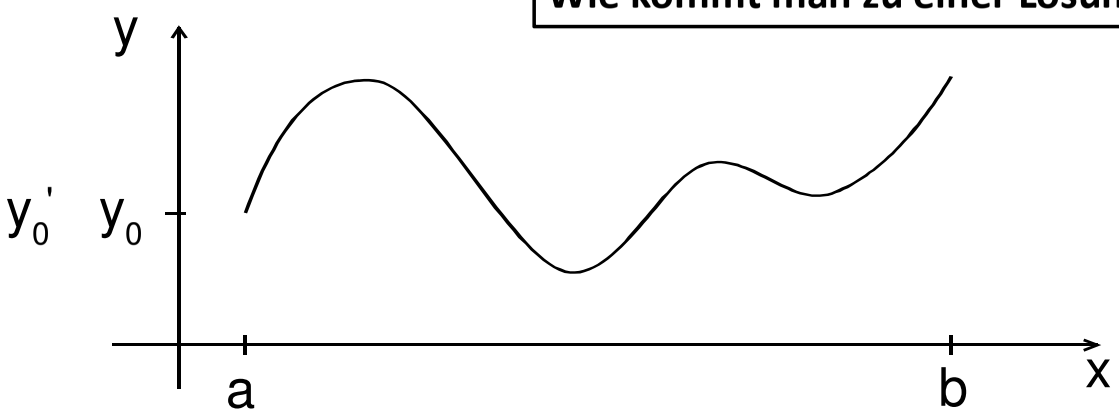
Problem:

- DGL und Anfangswerte gegeben

$$y'' + \omega^2 \cdot y = x^2 \cdot y + \lambda \cdot y^2$$

- Die Lösung der DGL soll grafisch dargestellt werden.

Wie kommt man zu einer Lösung?



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Lösungsmethode: Strukturierte Programmierung

- Schrittweise Verfeinerung des Problems
- Nur 3 einfache Kontrollstrukturen

Bei der **schrittweisen Verfeinerung** wird eine Aufgabe solange in kleinere/einfachere Teilaufgaben zerlegt, bis diese so einfach sind, dass eine Codierung erfolgen kann.

Bei der grafischen Darstellung dieser Methode durch Struktogramme wird jede Teilaufgabe durch einen sog. Strukturblock dargestellt. Ein Strukturblock besitzt folgende Eigenschaften:

- Er erfüllt eine klar definierte Aufgabe
- Jeder Strukturblock besitzt einen Eingang (von oben) und einen Ausgang (nach unten)

Die Reihenfolge, in der die einzelnen Teilaufgaben bearbeitet werden, wird durch Kontrollstrukturen festgelegt.

Bei der strukturierten Programmierung werden nur drei einfache Kontrollstrukturen zugelassen:

- **Folge** (Sequenz)
- **Alternative** (Verzweigung, Auswahl oder Selektion)
- **Wiederholung** (Schleife, Iteration)

Hat man ein Problem schrittweise verfeinert und sich auf die drei genannten Kontrollstrukturen beschränkt, dann ist die Codierung einfach und die entstehenden Programme sind erfahrungsgemäß gut lesbar, wartungsfreundlich, leicht zu testen und wenig fehleranfällig.

Zur grafischen Darstellung der Methode der strukturierten Programmierung werden sogenannte **Nassi-Shneidermann-Diagramme** oder **Struktogramme** (DIN 66261) verwendet.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Schreibe ein Programm, das zwei ganze Zahlen einliest. Das Programm soll die Summe der eingegebenen Zahlen berechnen. Die Zahlen sollen am Bildschirm ausgegeben werden, ebenso der Wert der Summe.

Verwendete Variablen:

z1 : erste eingelesene Zahl
z2 : zweite eingelesene Zahl
sum : Summe der eingelesenen Zahlen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>
int main(void)
{
  int z1, z2, sum;
  printf("Erste Zahl eingeben:\n");
  scanf("%d", &z1);
  printf("Zweite Zahl eingeben:\n");
  scanf("%d", &z2);
  /* berechne die Summe */
  sum = z1 + z2;
  printf("Zahl 1: %d Zahl2: %d\n", z1, z2);
  printf("Summe: %d \n", sum);
  return 0;
}
```

Die Lösung eines Teilproblems (z. B. eine einzelne Anweisung) wird im Struktogramm durch ein Rechteck grafisch dargestellt:

Teilproblem

Folge, Sequenz:

Teilproblem1
Teilproblem2
. . . .

Bedeutung:

- Die einzelnen **Strukturblöcke** (Teilprobleme) werden nacheinander abgearbeitet
- Im C-Programm: Anweisungen werden nacheinander geschrieben

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ein Struktogramm kann in beliebige Sprachen übersetzt werden!

Problem:

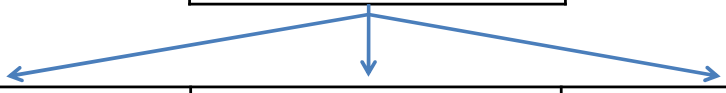
Es soll eine Zahl eingelesen und der Wert zur Kontrolle wieder ausgegeben werden.

Verwendete Variable:

z1: eingelesene Zahl

Struktogramm:

Zahl z1 einlesen
Zahl z1 ausgeben



UNIX-SHELL	C	PASCAL
read z1	scanf("%d", &z1);	readln(z1);
echo "Zahl = " $z1	printf("Zahl=%d",z1);	writeln('Zahl=', z1);

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Bei komplexeren Problemen:

- Ein Strukturblock entspricht vielen Anweisungen
- Strukturblock wiederum in Teilprobleme zerlegen
- Hierarchie von Struktogrammen

Beispiel:

10 Messwerte einlesen und sortiert wieder ausgeben

Übersichts-Struktogramm (Grobstruktur):

10 Zahlen einlesen – TP1
Zahlen der Größe nach sortieren – TP2
10 Zahlen ausgeben – TP3

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Verfeinerung (Feinstruktur):

Struktogramme für einzelne Teilprobleme

Struktogramm für TP1:

Einlesealgorithmus

. . . .
. . . .

Struktogramm für TP2:

Sortieralgorithmus

. . . .
. . . .

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Schreibe ein Programm, das zwei Gleitkommazahlen einliest. Das Programm soll die Summe der eingegebenen Zahlen berechnen. Die Zahlen sollen am Bildschirm ausgegeben werden, ebenso der Wert der Summe aber nur, wenn dieser positiv oder null ist. Es soll eine Meldung ausgegeben werden, ob das Ergebnis positiv oder negativ ist. Am Ende soll eine Meldung erscheinen, dass das Programm beendet wurde.

Variablen:

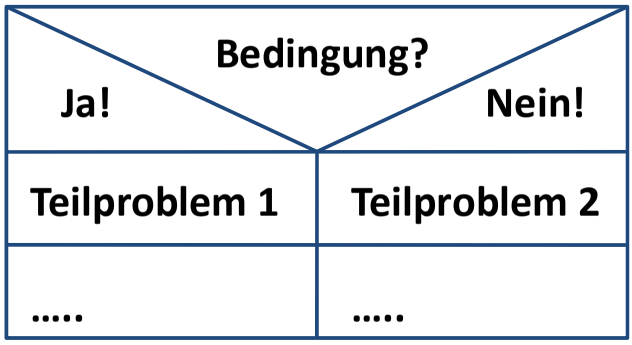
z1, z2 und sum - Typ float

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>
int main(void)
{
   float z1, z2, sum;
   printf("Erste Zahl eingeben:\n");
   scanf("%f", &z1);
   printf("Zweite Zahl eingeben:\n");
   scanf("%f", &z2);
   sum = z1 + z2;
   printf("Zahl1: %f Zahl2: %f\n", z1, z2);
   if(sum < 0.0)
   {
       printf("Ergebnis ist negativ!");
   }
   else
   {
       printf("Summe: %f\n", sum);
       printf("Ergebnis ist positiv oder null!");
   }
   printf("Ende des Programms");
   return 0;
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kontrollstruktur: einfache Verzweigung



Bedeutung:

- Ist die Bedingung erfüllt, wird Teilproblem1 bearbeitet, andernfalls Teilproblem2
- Genau eines der beiden Teilprobleme wird bearbeitet

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Umsetzung einer einfachen Verzweigung in C:

```
if ( Bedingung )
    Anweisungsblock_1
else
    Anweisungsblock_2
```

Bedingung

- Die Bedingung muss so formuliert sein, dass sie entweder erfüllt („wahr“) oder nicht erfüllt („falsch“) ist
- Beispiele für Bedingungen sind: (x > 1), (2*x < 10), (x == y), (x != y)
- Achtung: (x = 5) ist keine Bedingung, sondern eine Zuweisung!

Anweisungsblock:

- Sequenz von Anweisungen, eingeschlossen von { }
- Hinter den einzelnen Anweisungen im Anweisungsblock stehen Strichpunkte, nicht aber hinter den geschweiften Klammern { }
- Enthält ein Anweisungsblock lediglich eine einzelne Anweisung, können die geschweiften Klammern { } entfallen

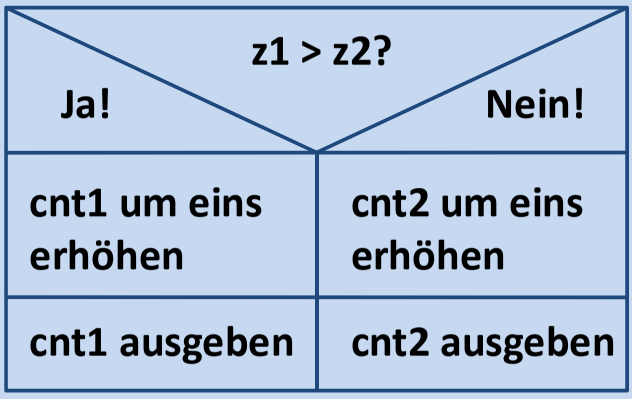
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 1:

- Es soll das Maximum von zwei Zahlen bestimmt werden (Variablen: z1, z2 und max)
- Teilaufgabe A: Struktogramm erstellen
- Teilaufgabe B: C-Quelltext erstellen

Beispiel 2:

- C-Quelltext zum Struktogramm erstellen
- Variablen: z1, z2, cnt1 und cnt2



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 3:

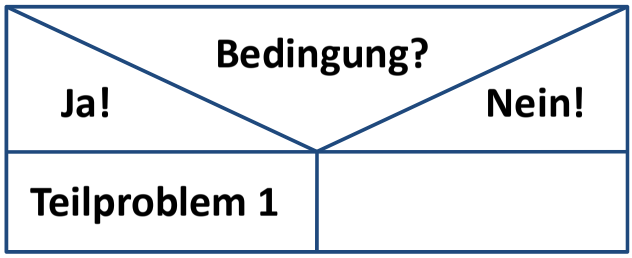
- Es soll das Maximum von drei Zahlen bestimmt werden (Variablen: z1, z2, z3 und max)
- Teilaufgabe A: Struktogramm erstellen
- Teilaufgabe B: C-Quelltext erstellen

Tipp:

- Sie können das Problem mit „verschachtelten“ if-else-Anweisungen lösen:
- Fragen Sie mit einer if-else-Anweisung zunächst die erste Bedingung ab; falls diese (nicht) erfüllt ist, fragen Sie mit einer zweiten – verschachtelten – if-else-Anweisung die zweite Bedingung ab.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kontrollstruktur: bedingte Anweisung



Bedeutung:

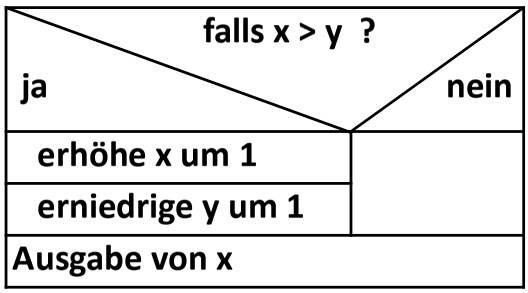
- Ist die Bedingung erfüllt, wird Teilproblem1 bearbeitet
- Andernfalls erfolgt „keine Anweisung“ (leere Anweisung)
- Entspricht einer if-Anweisung ohne „else“

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Umsetzung einer bedingten Anweisung in C:

```
if ( Bedingung )
     Anweisungsblock
```

Beispiel:



Umsetzung nach C:

```
if ( x > y )
{
   x = x + 1;
   y = y - 1;
}
printf("x = %d", x);
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 4 („Body Mass Index“):

- Nach Eingabe von Körpergröße l (in Metern) und Körpermasse m (in Kilogramm) soll der „Body Mass Index“ berechnet und ausgegeben werden: $bmi = m / l^2$
- Zusätzlich soll ermittelt werden, in welche der folgenden vier Kategorien der berechnete BMI fällt:

  BMI ≤ 19 → Untergewicht, ansonsten:
  BMI ≤ 25 → Normalgewicht, ansonsten:
  BMI ≤ 30 → leichtes Übergewicht, ansonsten:
  BMI > 30 → Übergewicht.
- Erstellen Sie zunächst ein Struktogramm und dann den C-Quelltext des Programms!

```c
#include <stdio.h>        /* Beispiel 4a: BODY MASS INDEX */
int main(void)
{
    float l, m, bmi;
    printf("Groesse in Metern:  "); scanf("%f", &l);
    printf("Koerpermasse in kg: "); scanf("%f", &m);
    bmi = m / (l*l);
    printf("BMI: %f\n", bmi);
    if(bmi <= 19)
    {
        printf("Untergewicht\n");
    }
    else
    {
        if(bmi <= 25)
        {
            printf("Normalgewicht\n");
        }
        else
        {
            if(bmi <= 30)
                printf("Leichtes Uebergewicht\n");
            else
                printf("Uebergewicht\n");
        }
    }
    return 0;
}
```

4.3. Verzweigung - Alternative

Geschachtelte if-else-Anweisungen sind oft unübersichtlich, wenn viele Bedingungen/Fälle überprüft werden müssen – **Ausweg: Mehrfachalternative**

```
if ( Bedingung_1 )
    Anweisungsblock_1
else if  ( Bedingung_2 )
    Anweisungsblock_2
else if
      . . . .
else if  ( Bedingung_n )
    Anweisungsblock_n
else
    Anweisungsblock_x
```

Bedeutung :

- Bedingungen werden nacheinander geprüft
- Diejenige Anweisung, bei der die Bedingung **erstmals erfüllt** ist, wird ausgeführt – alle anderen nicht
- Ist keine Bedingung erfüllt, wird die Anweisung nach letztem else ausgeführt
- Letztes else kann aber auch entfallen – entspricht der leeren Anweisung

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>      /* Beispiel 4b: BODY MASS INDEX */
int main(void)
{
    float l, m, bmi;
    printf("Groesse in Metern:  "); scanf("%f", &l);
    printf("Koerpermasse in kg: "); scanf("%f", &m);

    bmi = m / (l*l);
    printf("BMI: %f\n", bmi);

    if(bmi <= 19)
        printf("Untergewicht\n");
    else if(bmi <= 25)
        printf("Normalgewicht\n");
    else if(bmi <= 30)
        printf("Leichtes Uebergewicht\n");
    else
        printf("Uebergewicht\n");

    return 0;
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Operatoren nach Funktionen geordnet:

Funktion	Operatoren
arithmetisch	+ - * / % ++ --
relational	> >= < <= == !=
logisch	&& || !
bitorientiert	& | ^ ~ << >>
zeigerorientiert	& * ->
zuweisend	= += -= *= /= %= &= ^= |= <<= >>=
sonstige	, (type) ( ) [ ] sizeof ?: .

Bedingungen (zum Beispiel in if-else-Anweisungen) werden mit **relationalen Operatoren** gebildet und mit **logischen Operatoren** verknüpft.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiele für Verknüpfungen von logischen Bedingungen:

1. Maximum von 3 Zahlen z1, z2 und z3 bestimmen und der Variablen max zuweisen.
2. Bereichsprüfung: Liegt z im Bereich $1000 \leq z \leq 2000$?
3. Bereichsprüfung: Liegt z im Bereich $1 < z < 9$ und ist zugleich $z \neq 5$?
4. Body Mass Index: Ist $19 \leq BMI < 25$ (Normalgewicht)?
5. Finden Sie eine einfachere Darstellung für: „if (!(z1 <= z2))“

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Es soll die Summe der Quadrate von 1 bis n berechnet werden. (Dazu existiert zwar auch eine geschlossene Formel, diese soll allerdings nicht benutzt werden!)

Berechne:

$$sum = 1^2 + 2^2 + 3^2 + \ldots + n^2 = \sum_{i=1}^{n} i^2$$

Der Rechner kennt allerdings weder ein Summensymbol, noch ein „…“. Man muss dem Rechner daher genau vorschreiben, was er berechnen soll – jeden einzelnen Schritt!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

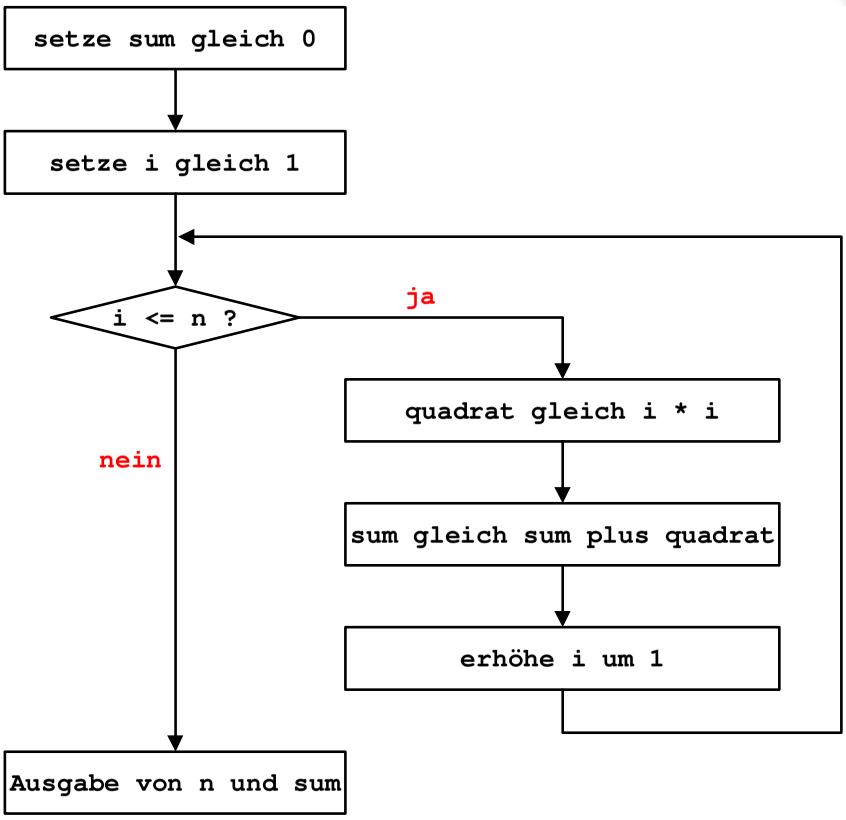
Algorithmus:

- Variable sum auf 0 setzen
- Zählvariable i auf 1 setzen
- Berechne quadrat = i * i
- Addiere quadrat zu sum
- Erhöhe i um 1
- Prüfe, ob i kleiner gleich n ist; falls ja, Verfahren wiederholen

Variablen:

i	:	Zählervariable
n	:	Endwert
quadrat	:	Berechnet in jedem Durchlauf das Quadrat von i
sum	:	Die berechnete Quadratsumme

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

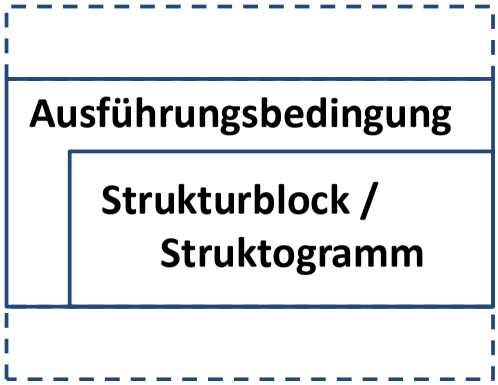


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>          /* Version 1:              */
int main(void)              /* Abweisende Schleife */
{
    int i, n, quadrat, sum;
    n   = 5;
    sum = 0;
    i   = 1;
    while(i <= n)
    {
        quadrat = i * i;
        sum = sum + quadrat;
        i = i + 1;
    }
    printf("Quadratsumme bis %d = %d\n", n, sum);
    return 0;
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kontrollstruktur: abweisende Schleife



Bedeutung:

Ausführungsbedingung prüfen

- Ist Bedingung **erfüllt**, dann:
  - Strukturblock innerhalb der Schleife ausführen
  - Danach erneut prüfen
- Ist Bedingung **nicht erfüllt**, dann:
  - Schleife verlassen
  - Nächsten Strukturblock nach der Schleife ausführen

Umsetzung in C:

```
while ( Ausführungsbedingung )
    Anweisungsblock
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Nichtabweisende Schleife:

Bei einer **abweisenden Schleife** ist die Bedingungsprüfung am Anfang. Daher ist es ggf. möglich, dass die Schleife überhaupt nicht durchlaufen wird – die Schleife wird komplett übersprungen.

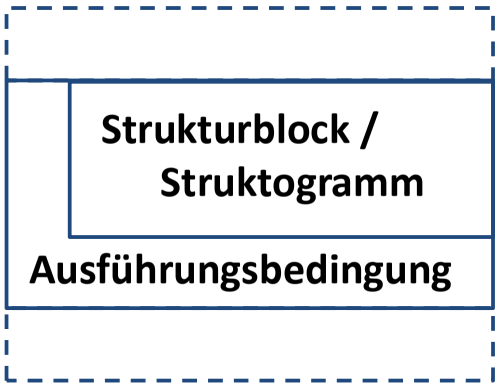
Bei manchen Problemen ist es zweckmäßiger, die Bedingungsprüfung erst am Ende eines Durchlaufs zu machen. Die Schleife wird dann auch bei einer nicht erfüllten Bedingung **wenigstens einmal** durchlaufen → nachfolgende Bedingungsprüfung oder **nichtabweisende Schleife.**

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>      /* Version 2:                    */
int main(void)          /* Nichtabweisende Schleife */
{
    int i, n, quadrat, sum;
    n = 5;
    sum = 0;
    i = 1;
    do
    {
        quadrat = i * i;
        sum = sum + quadrat;
        i = i + 1;
    }
    while(i <= n);
    printf("Quadratsumme bis %d = %d\n", n, sum);
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kontrollstruktur: nichtabweisende Schleife



Bedeutung:

1) Strukturblock ausführen
2) Ausführungsbedingung prüfen:
   - Ist Bedingung erfüllt, Struktur-block erneut ausführen (weiter mit Schritt 1)
   - Ist Bedingung nicht erfüllt, Schleife verlassen und mit erstem Struktur-block nach der Schleife fortsetzen

Umsetzung in C:

```
do
    Anweisungsblock
while ( Ausführungsbedingung );
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Es soll eine ganze Zahl x eingelesen werden. Die Eingabe ist so lange zu wiederholen, bis die eingegebene Zahl im Zahlenbereich $50 \leq x < 100$ liegt.

1) Erstellen Sie ein Struktogramm zur Lösung dieser Aufgabe!
2) Erstellen Sie ein entsprechendes C-Programm!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beobachtung:

In Zählschleifen finden sich oft die folgenden drei Komponenten:
Initialisierung, **Ausführungsbedingung**, **Veränderungsschritt**

```
#include <stdio.h>
int main(void)
{
    int i;
    i = 1;
    while(i <= 10)
    {
        printf("%d\n", i);
        i = i + 1;
    }
    return 0;
}
```

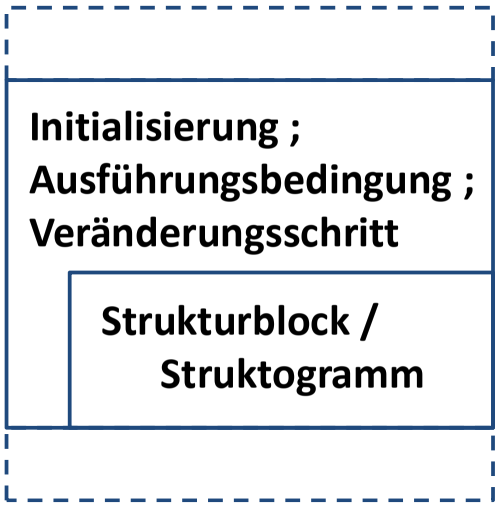
Initialisierung
(einmalig, vor Beginn der Schleife)

Ausführungsbedingung
(wird vor jedem Durchlauf der Schleife geprüft)

Veränderungsschritt
(wird nach jedem Durchlauf ausgeführt)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kontrollstruktur: Zählschleife



Bedeutung:

- Die Zählschleife ist eine Sonderform der abweisenden Schleife
- Sie bietet eine kompaktere Schreibweise für die Einzelkomponenten **Initialisierung**, **Ausführungsbedingung** und **Veränderungsschritt**

Umsetzung in C:

```
for (initialisierung; bedingung; veränderung)
    Anweisungsblock
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Berechne die folgende Summe für ein gegebenes n:

$$sum = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

Umsetzung in C:

```
float sum; int i; int n = 5;
sum = 0.0;
for(i = 1; i <= n; i = i + 1)
{
    sum = sum + 1.0 / i ;
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Schreibe ein Programm, das einen Radius r einliest und wahlweise den Kreisumfang, die Kreisfläche oder das Kugelvolumen berechnet und ausgibt.

Variablen:

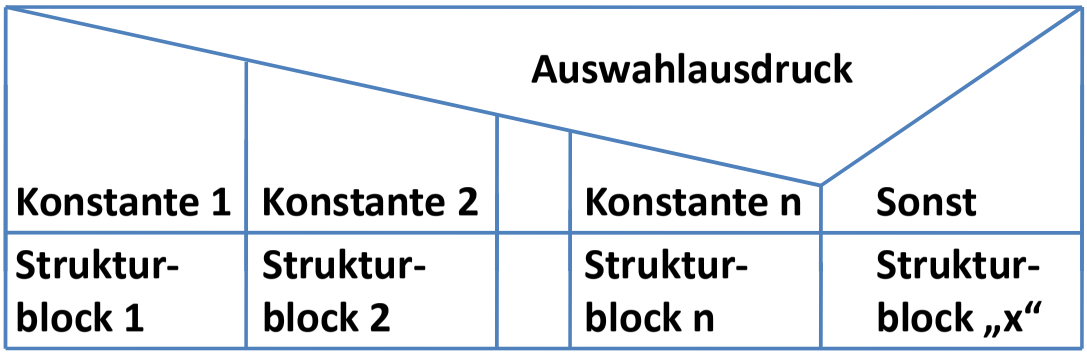
auswahl	:	1 = Kreisumfang	Typ int
	:	2 = Kreisfläche	
	:	3 = Kugelvolumen	
		andere Werte sind ungültig	
r	:	Radius	Typ float
erg	:	Ergebnis	Typ float

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>     /* Mehrfachauswahl:  Kreis, Kugel berechnen */
int main(void)
{
    int wahl; float r, erg;
    printf("Umfang(1)-Flaeche(2)-Volumen(3)?"); scanf("%d", &wahl);
    printf("Radius ?\n"); scanf("%f", &r);
    switch(wahl)
    {
    case 1:  erg = 2.0*3.14*r;
             printf("Umfang = %f", erg);
             break;
    case 2:  erg = 3.14*r*r;
             printf("Flaeche = %f", erg);
             break;
    case 3:  erg = 4.0/3.0*3.14*r*r*r;
             printf("Volumen = %f", erg);
             break;
    default: printf("Auswahl falsch");
    }
    printf("\nProgrammende!");
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kontrollstruktur: Mehrfachauswahl



Bedeutung:

1) Auswahlausdruck (Fallabfrage) auswerten – ergibt einen Integerwert
2) Strukturblock ausführen, bei dem die Konstante mit dem Wert des Auswahlausdrucks übereinstimmt
3) Tritt keiner der aufgeführten Fälle ein, „Sonst-Block“ ausführen

Der „Sonst-Block“ kann entfallen; dann erfolgt keine Aktion, wenn der Auswahlausdruck mit keiner Konstanten übereinstimmt

Achtung: Konstante 1, Konstante 2... müssen Integer-Konstanten sein, also keine Gleitkommazahlen und auch keine Variablen!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Achtung:

An dieser Stelle sind lediglich Integer-Konstanten erlaubt (keine Variablen, auch keine Bereiche)!

Umsetzung in C:

```
switch ( Auswahlausdruck )
{
      case Konstante_1 : Anweisungen_1
                         break;
      case Konstante_2 : Anweisungen_2
                         break;

      . . . .

      case Konstante_n : Anweisungen_n
                         break;
      default          : Anweisungen_x
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Sprunganweisungen bewirken, dass das Programm an einer anderen Stelle fortgesetzt wird, sie unterbrechen den linearen Ablauf de Programms.

Ein Beispiel für eine Sprunganweisung, die **break-Anweisung**, findet sich im Beispiel zur Mehrfachauswahl („Kreis, Kugel berechnen“)

- Die break-Anweisung verlässt den Bereich der switch-case-Anweisung und setzt die Programmusführung mit der ersten nachfolgenden Anweisung fort.
- Wird break vergessen, dann werden die nachfolgenden case-Blöcke ebenfalls ausgeführt!

Die break-Anweisung wird aber nicht nur in switch-case-Anweisungen eingesetzt, sondern auch bei der Programmierung von Schleifen:

Wirkung der break-Anweisung:

- Die aktuelle for-, while-, do-while-Schleife bzw. switch-Anweisung wird sofort verlassen
- Die Programmausführung wird unmittelbar nach dem verlassenen Block (Schleife/Anweisung) fortgesetzt
- Beachte: Nur die innerste Schleife wird verlassen!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
int i, j, k;
Anweisungen ...
for(j = 0; j < 100; j++)
{
    Anweisungen ...
    if(j == (k + 23))
        break;
    Anweisungen ...
    for(i = 0; i < 50; i++)
    {
        Anweisungen ...
        if(i == k)
            break;
        Anweisungen ...
    }
    Anweisungen ...
}
Anweisungen ...
```

Beispiel:

Verschachtelte Schleifen und break

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

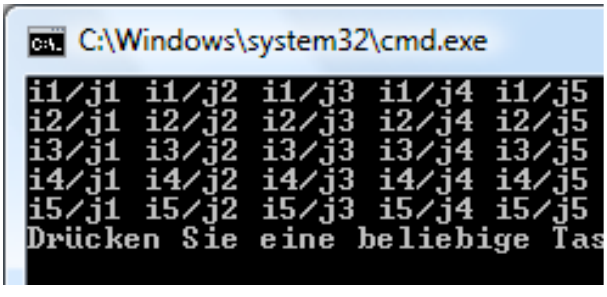
Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 1:

- Wenn eine (äußere) Schleife wiederum eine (innere) Schleife enthält, spricht man von verschachtelten Schleifen.
- In jedem einzelnen Durchlauf der äußeren Schleife wird die komplette innere Schleife abgearbeitet.

```
#include <stdio.h>
int main(void)
{
  int i, j;
  for(i = 1; i < 6; i = i + 1)
  {
    for(j = 1; j < 6; j = j + 1)
    {
      printf("i%d/j%d \t", i, j);
    }
    printf("\n");
  }
  return 0;
}
```



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 2:

- Ermitteln Sie, ob die Ziffernsumme (Quersumme) einer ganzen Zahl gerade oder ungerade ist.
- Fordern Sie den Anwender zur Eingabe der Zahl auf. Geben Sie die Zahl zur Kontrolle aus. Geben Sie die Zwischenwerte der Quersummenbildung aus.
- Nachdem Sie ausgegeben haben, ob die Quersumme gerade oder ungerade ist, verlangen Sie nach einer neuen Zahl. Durch Eingabe einer 0 soll Ihr Programm abgebrochen werden.
- Erstellen Sie zunächst ein Struktogramm. Schreiben Sie anschließend das zugehörige C-Programm.

Anwendung zum Beispiel bei Prüfsummen: *Prüfen, ob eine vierstellige Artikelnummer gültig ist oder nicht. Dazu wird eine zusätzliche Stelle hinzugefügt, so dass die Quersumme durch 10 teilbar ist.*

12331 42716 42718

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Teilprobleme:

- Zahl einlesen und wieder ausgeben
- Quersumme bilden
- Prüfen, ob Quersumme gerade oder ungerade ist
- Ergebnis ausgeben
- Neue Zahl einlesen und wieder ausgeben
- Abbruchbedingung prüfen → Abbruch bzw. Wiederholung

Quersumme bilden: 4613 → 14

alle Ziffern addieren – aber wie bekommt man die Ziffern?

letzte Ziffer: Rest bei Division durch 10 (Modulo-Operator!)

neue Zahl: alte Zahl / 10 (Integer-Division, Rest abschneiden!)

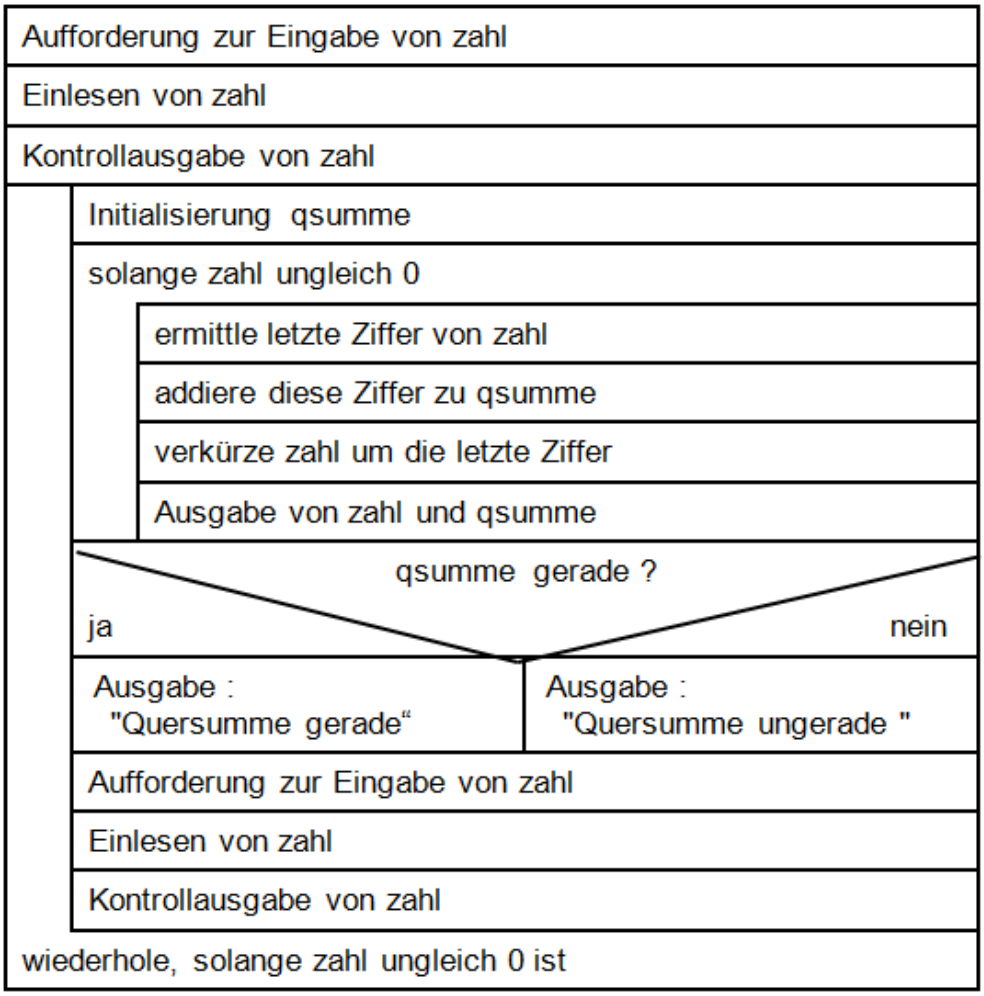
Quersumme gerade oder ungerade:

quersumme % 2 == 0, gerade == 1, ungerade

Variablen:

zahl (eingelesene Ziffer), qsumme (Quersumme), lziffer (letzte Ziffer)

Struktogramm



Kapitel 5
Datentypen und Operatoren

Gliederung

Kapitel 5 – Datentypen und Operatoren

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Elementare Datentypen (int, float usw.) sind bereits in der Sprache vorgegeben und es gibt Operationen für diese Datentypen.

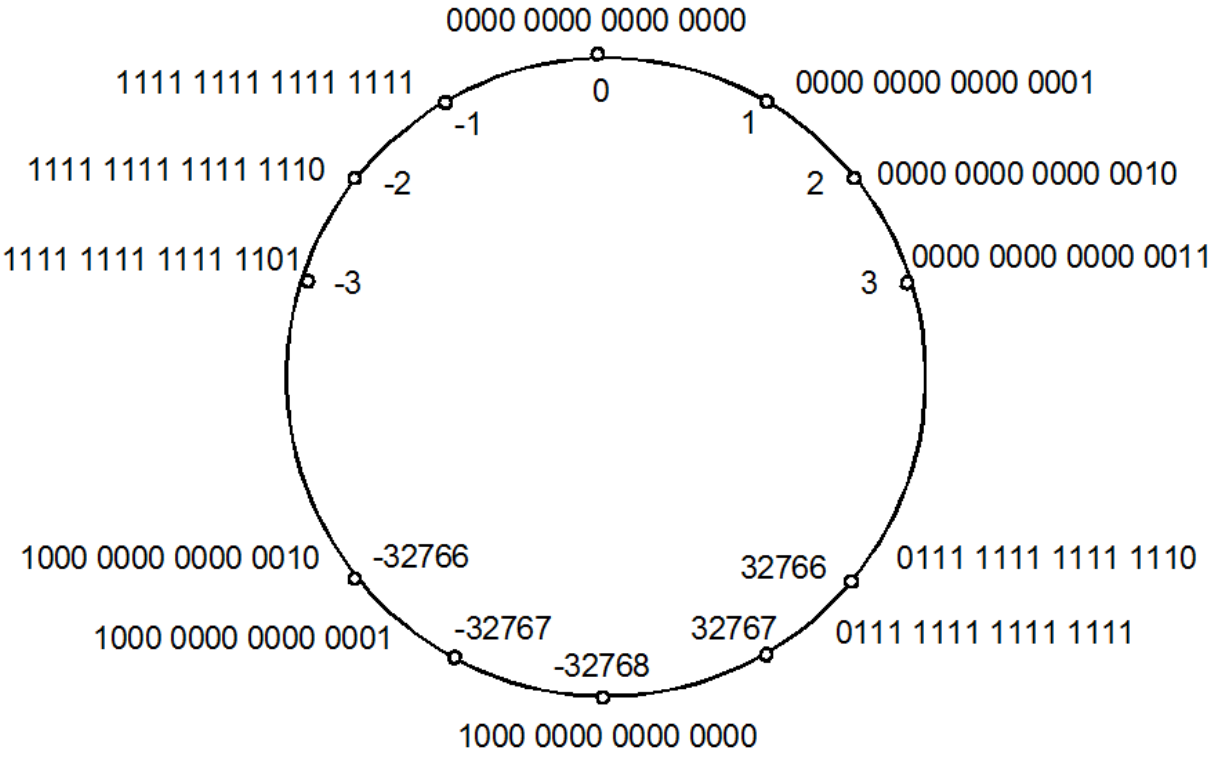
Zusammengesetzte Datentypen werden vom Programmierer definiert, zum Beispiel:
- um komplexe Zahlen darzustellen
  (Kombination zweier float-Zahlen → Addition ist komplizierter),
- um einen Studenten zu beschreiben
  (setzt sich ebenfalls aus elementaren Datentypen zusammen).

Eigenschaften eines Datentyps:

- Wertebereich, Nachkommastellen (bei Gleitkommazahlen)
- Speicherplatzbedarf
- Interne Darstellung / Speicherung
- Gültige Operationen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ganze Zahlen mit Vorzeichen (a):



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ganze Zahlen mit Vorzeichen (b):

Neben dem Datentyp int gibt es weitere Typen zur Darstellung vorzeichenbehafteter ganze Zahlen. Diese decken unterschiedliche Wertebereiche ab (Kompromiss: Speicherplatz/Wertebereich/ Rechenzeit):

short	`short i;`
int	`int i;`
long	`long i;`
long long	`long long i;`

Zahlenbereich bei n Bit: $(-2^{n-1}) \ldots (+2^{n-1}-1)$

n = 16, 32, 64 - 2, 4, 8 Bytes Speicherplatz
positive Zahlen : höchstwertiges Bit 0
negative Zahlen : höchstwertiges Bit 1

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

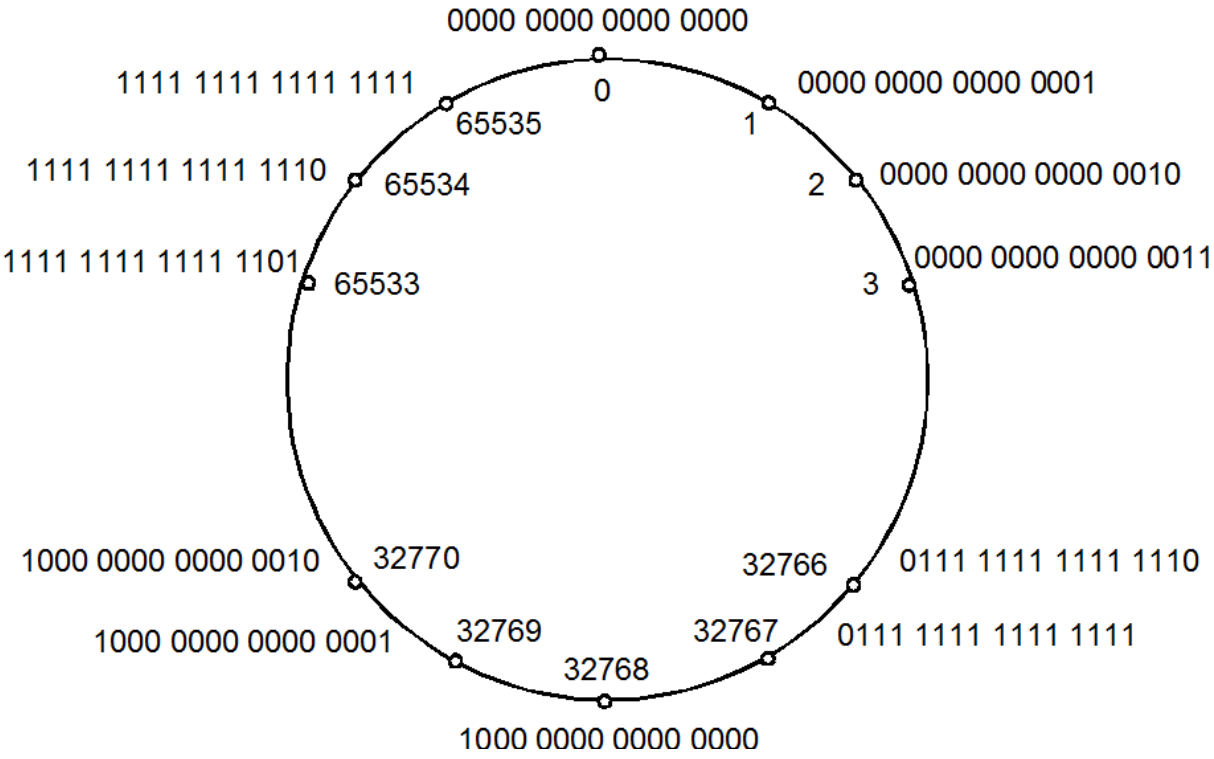
Ganze Zahlen mit Vorzeichen (c):

Größe in Bytes	Linux/GCC	LCC-WIN32	Visual C++
short	2	2	2
int	4	4	4
long	4	4	4
long long	8	8	8

Größe in Bytes	Wertebereich
2 Byte / 16 Bit	-32.768 ... +32.767
4 Byte / 32 Bit	-2.147.483.648 ... +2.147.483.647
8 Byte / 64 Bit	$-9{,}2 \times 10^{18}$ ... $+9{,}2 \times 10^{18}$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Vorzeichenlose ganze Zahlen (a):



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Vorzeichenlose ganze Zahlen (b):

unsigned short	`unsigned short i;`
unsigned int	`unsigned int i;`
unsigned long	`unsigned long i;`
unsigned long long	`unsigned long long i;`

```
#include <stdio.h>
int main(void)
{
        unsigned short      s = -1;
        unsigned int        i = -1;
        unsigned long long  l = -1;
        printf("%d, %x  \n", sizeof(s), s);
        printf("%d, %x  \n", sizeof(i), i);
        printf("%d, %llx\n", sizeof(l), l);
        return 0;
}
```

Wie lautet die Ausgabe dieses Programms…?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Vorzeichenlose ganze Zahlen (c):

Größe in Bytes	Linux/GCC	LCC-WIN32	Visual C++
unsigned short	2	2	2
unsigned int	4	4	4
unsigned long	4	4	4
unsigned long long	8	8	8

Größe in Bytes	Wertebereich
2 Byte / 16 Bit	0 … 65.535
4 Byte / 32 Bit	0 … 4.294.967.295
8 Byte / 64 Bit	0 … $18{,}4 \times 10^{18}$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Fließkommazahlen, Gleitkommazahlen (a):

Fließkommazahlen werden benötigt für:

- Sehr große und sehr kleine Zahlen
- Zahlen mit Nachkommastellen

Fließkommazahlen werden **völlig anders behandelt** als ganze Zahlen. Für jede Fließkommazahl werden 32 oder 64 Bit Speicherplatz benötigt. Eine Fließkommazahl wird dabei als Kombination von **Vorzeichen, Exponent und Mantisse** gespeichert.

Typen:

`float` einfache Genauigkeit (4 Byte / 32 Bit)
Zahlenbereich (jeweils pos./neg.) $1{,}2 \times 10^{-38}$ … $3{,}4 \times 10^{+38}$

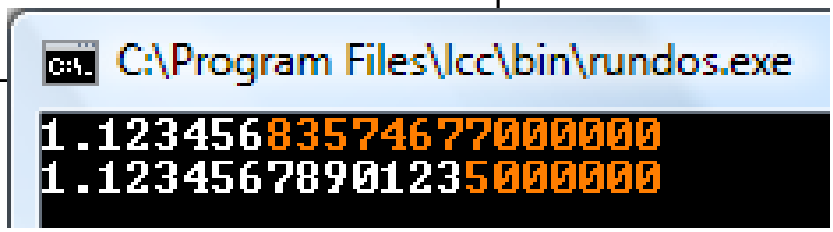
`double` doppelte Genauigkeit (8 Byte / 64 Bit)
Zahlenbereich (jeweils pos./neg.) $2{,}2 \times 10^{-308}$ … $1{,}8 \times 10^{+308}$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Fließkommazahlen, Gleitkommazahlen (b):

Bei der Arbeit mit Fließkommazahlen ist deren **begrenzte Genauigkeit** zu beachten! Beim Datentyp float werden 6-7 **signifikante Stellen** berücksichtigt, bei double sind es 15-16 Stellen.

```
#include <stdio.h>
int main(void)
{
      float  f = 1.12345678901234567890;
      double d = 1.12345678901234567890;
      printf("%.20f\n%.20f\n", f, d);
      return 0;
}
```

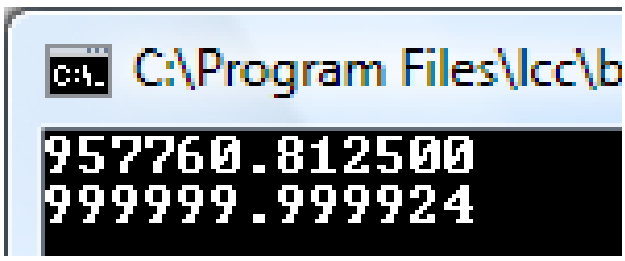


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Fließkommazahlen, Gleitkommazahlen (c):

Achtung: Rundungsfehler können sich aufsummieren!

```
/* 7.000.000 x 1/7 berechnen */
#include <stdio.h>
int main(void)
{
  float  f = 1.0 / 7.0, fsum = 0;
  double d = 1.0 / 7.0, dsum = 0;
  long l;
  for(l = 1; l <= 7000000; l = l + 1)
  {
    fsum = fsum + f;
    dsum = dsum + d;
  }
  printf("%f\n", fsum);
  printf("%f\n", dsum);
  return 0;
}
```



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Buchstaben, Ziffern, Sonderzeichen… (a):

Um einzelne Zeichen zu speichern gibt es einen eigenen Datentyp **char** mit einer Größe von 8 Bit (1 Byte). Zur Bearbeitung von Buchstaben, Ziffern und Sonderzeichen wird dabei in der Regel ein erweiterter ASCII-Code benutzt (z. B. ISO-Latin1).

```
#include <stdio.h>
int main(void)
{
  char c;
  c = 'C';
  c = c - 2;
  printf("%c hat ASCII-Code %d\n", c, c);
  return 0;
}
```

Definition einer char-Variablen

Zeichenkonstanten werden mit einfachen Anführungszeichen geschrieben

Mit Zeichen (bzw. deren ASCII-Code) kann man auch rechnen

Das Formatelement („Platzhalter“) zur Ausgabe eines Zeichens ist %c

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Buchstaben, Ziffern, Sonderzeichen… (b):

```
char y = 5;   /* ASCII-Code Steuerzeichen "ENQ" */
printf("y = %d", y);
printf("y = %c", y);

y = '5';   /* y bekommt den Wert 53 */
printf("y = %d", y);
printf("y = %c", y);
```

Ausgabe: y = 5
y =

Ausgabe: y = 53
y = 5

Beispiele:

1) Prüfen, ob char-Variable gleich 'j' ist
2) Prüfen, ob char-Variable gleich 'j' oder 'J' ist
3) Prüfen, ob char-Variable Großbuchstaben enthält

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 5 – Datentypen und Operatoren

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Symbolische Konstanten sind Konstanten, die einen Namen besitzen. Meist werden sie am Programmanfang definiert; dies ist allerdings nicht zwingend erforderlich.

- Die Anweisung **#define** definiert eine symbolische Konstante (gefolgt von Name und Wert der Konstanten)
- Symbolische Konstanten müssen auf jeden Fall vor der ersten Verwendung definiert werden.
- Konvention: **Namen in Großbuchstaben** schreiben

Beispiele:

```
#define PI 3.14159265
#define BAUTEIL_EINZELPREIS 10.0
#define BEARBEITER "Tom"
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#include <stdio.h>
#define PI 3.14159265
int main(void)
{
  int wahl; float r, erg;
  printf("Umfang(1)-Flaeche(2)-Volumen(3)?"); scanf("%d", &wahl);
  printf("Radius ?\n"); scanf("%f", &r);
  switch(wahl)
  {
  case 1:  erg = 2.0*PI*r;
           printf("Umfang = %f", erg);
           break;
  case 2:  erg = PI*r*r;
           printf("Flaeche = %f", erg);
           break;
  case 3:  erg = 4.0/3.0*PI*r*r*r;
           printf("Volumen = %f", erg);
           break;
  default: printf("Auswahl falsch");
  }
  return 0;
}
```

Symbolische Konstanten erhöhen die Lesbarkeit eines Programms. Sie erleichtern außerdem nachträgliche Änderungen des Quelltextes.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Symbolische Konstanten werden häufig in **Include-Dateien** (Header-Dateien) aufgelistet. Solche Dateien können leicht von anderen Programmen (weiter-)verwendet werden. Viele Include-Dateien werden schon vom Compiler-Hersteller mitgeliefert.

Beispiel: Ausschnitt aus der Include-Datei math.h

```
…
/* Useful constants. */
#define M_E        2.71828182845904523536
#define M_LN2      0.69314718055994530942
#define M_LN10     2.30258509299404568402
#define M_PI       3.14159265358979323846
#define M_PI_2     1.57079632679489661923
…
```

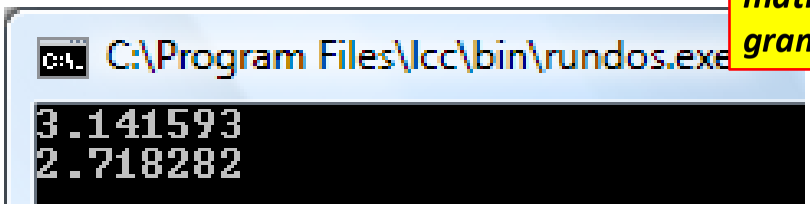
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#define _USE_MATH_DEFINES
#include <math.h>
#include <stdio.h>
int main(void)
{
    printf("%f \n", M_PI);
    printf("%f \n", M_E );
    return 0;
}
```

Fehlt diese Definition, werden bei manchen Compilern die Konstanten M_PI, M_E (usw.) nicht erkannt!

Der Inhalt der Datei math.h wird gelesen, wenn das C-Programm compiliert wird.

Alle Konstanten in der Datei math.h können jetzt im Programm verwendet werden.



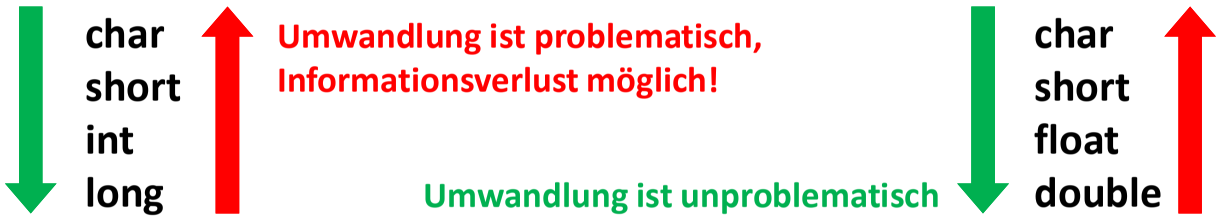
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 5 – Datentypen und Operatoren

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Im Rechner werden ganze Zahlen und Fließkommazahlen völlig unterschiedlich verarbeitet.

- Darf man einfach einer Fließkommavariablen den Wert einer Integer-Variablen zuweisen?
- Ist das Umgekehrte ebenfalls möglich?
- Was passiert mit den Nachkommastellen einer Fließkommazahl, wenn man sie einer Integer-Zahl zuweist?
- Welche anderen Probleme können auftreten?



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Implizite Typumwandlung:

Bei der impliziten Typumwandlung wird die Umwandlung vom Programmierer nicht „ausdrücklich verlangt“. Sie wird vom Compiler automatisch anhand der Datentypen von Variablen und Ausdrücken erkannt und „implizit“ durchgeführt

```
int i1, i2;
double d1, d2, d3, d4;

i1 = 3;
d1 = i1 / 4;
d2 = i1 / 4.0;
d3 = 7.6;
d4 = 3.5;
i2 = d3 + d4;
```

Welche Werte haben die Variablen jeweils nach der Zuweisung?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Explizite Typumwandlung:

Anders als bei der impliziten Typumwandlung wird die explizite Typumwandlung im Quelltext angegeben. Es gilt folgende Syntax:

```
(gewünschter Typ)Ausdruck
```

```
int i1, i2, i3;
double d1, d2, d3, d4;
i1 = 3;
d1 = (double)i1 / 4;
d2 = i1 / (double)4;
d3 = (double)(i1 / 4);
d4 = 1.75;
i2 = (int)(d4 + d4);
i3 = (int)d4 + (int)d4;
```

Welche Werte haben die Variablen jeweils nach der Zuweisung?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 5 – Datentypen und Operatoren

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Operatoren nach Funktionen geordnet:

Funktion	Operatoren
arithmetisch	+ - * / % ++ --
relational	> >= < <= == !=
logisch	&& || !
bitorientiert	& | ^ ~ << >>
zeigerorientiert	& * ->
zuweisend	= += -= *= /= %= &= ^= |= <<= >>=
sonstige	, (type) ( ) [ ] sizeof ?: .

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Arithmetische Operatoren:

+ - * /	Addition, Subtraktion, Multiplikation, Division
%	Modulo-Operator (Rest einer Integer-Division)
++ --	Inkrement-, Dekrement-Operatoren (Wert einer Variablen um 1 erhöhen, erniedrigen)

Inkrement-Operatoren können als Präfix- (++a) oder als Postfix-Operatoren (a++) angewendet werden. Welcher Unterschied besteht zwischen beiden Varianten?

```
int  x, y, zahl;
zahl = 5;
++zahl;
zahl++;
x = ++zahl;
y = zahl++;
```

Welche Werte haben die Variablen jeweils nach der Ausführung dieser Programmzeilen?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Relationale Operatoren:

> >= <= <	größer, größer/gleich, kleiner/gleich, kleiner
==	Vergleich auf Gleichheit
!=	Vergleich auf Ungleichheit

Logische Operatoren:

&& ||	Und- bzw. Oder-Verknüpfung
!	Negation

```
if( (x >= 10) && (x <= 20) )
        printf("x im Bereich 10...20");
if( !(x == y) )
        printf("x nicht gleich y");
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Zuweisende Operatoren:

=	Zuweisungsoperator (keine math. Gleichheit)
+= -=	Kurzschreibweisen für Addition, Subtraktion
*= /=	Kurzschreibweisen für Multiplikation, Division
%=	Kurzschreibweise für Modulo-Operator

```
zahl += 10          zahl = zahl + 10
zahl %= 10          zahl = zahl % 10
```

Inkrementierung einer Integer-Zahl:

```
zahl = zahl + 1     zahl += 1
zahl++              ++zahl
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Sonstige Operatoren:

,	Aufzählungsoperator zur Verkettung von Variablen oder Ausdrücken, z. B. in einer for-Anweisung
sizeof()	Dieser „Operator“ liefert die Größe (in Bytes) einer Variablen oder eines Datentyps.

```
int a, b;
for(k=0, j=2; k<=100; k++, i+=7)
          ...

i = sizeof(short);
long l;
i = sizeof(l);
```

sizeof(short) → 2
sizeof(l) → 4

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Operator
( ) [ ] -> .
! ~ ++ -- + - * & (type) sizeof
* / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
?:
= += -= *= /= %= &= ^= |= <<= >>=
,

Operatoren nach Priorität geordnet

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6
Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Funktionen werden verwendet, um

- Teilaufgaben zu bearbeiten, die häufig vorkommen,
- große Programme in kleine, übersichtliche Teile zu unterteilen.

C-Programme bestehen mindestens aus einer einzelnen Funktion, der Funktion **main**. Mit dieser Funktion beginnt die Ausführung des Programms.

Beispiel: Berechne $y = x^n$ für vorgegebene x und n

```
y = 1.0
for (i = 1; i <= n; i++)
{
    y = y * x;
}
```

Sollte diese Potenz öfters berechnet werden, müsste dieser Programmteil ständig wiederholt werden!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Auslagerung der Teilaufgabe „Potenz berechnen“ in eine Funktion:

- Eingabe: Werte von Basis (x) und Exponent (n)
- Ausgabe: Wert der n-ten Potenz

```
#include <stdio.h>
float power(float x, int k);

int main(void)
{
  float x1, x2, z, t;
  int n = 4;
  x1 = 3.0;
  z = power(x1, n);
  x2 = 3.75;
  t = power(x2, 5);
  return 0;
}
```

```
float power(float x, int k)
{
  int i;
  float y;
  y = 1.0;
  for (i = 1; i <= k; i++)
  {
     y = y * x;
  }
  return y;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

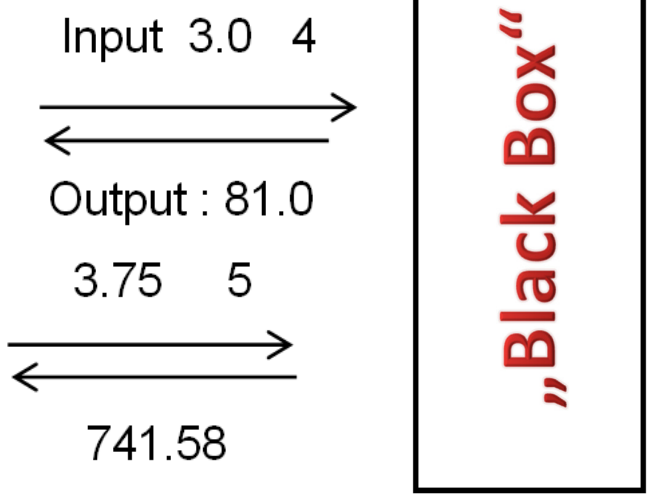
Datenaustausch zwischen Hauptprogramm und Funktion:

main : **power :**

```
x1 = 3.0;
n = 4;
z = power(x1, n);

…

t = power(3.75, 5);
```



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beim Start des Programms passiert Folgendes:

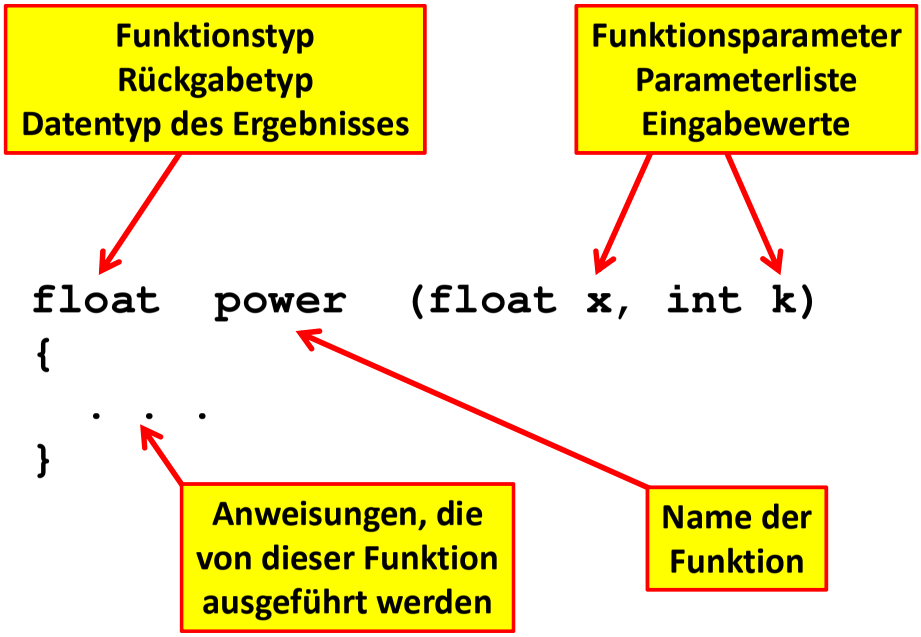
- Es wird immer zuerst die Funktion main aufgerufen.
- Die Zeilen in main werden der Reihe nach abgearbeitet.
- In der Zeile `z = power(x1,n);` wird die Funktion main verlassen. Es wird Speicherplatz für die **lokalen Variablen** x und k der Funktion power bereitgestellt; dann werden die **Werte kopiert** von x1 nach x bzw. von n nach k.
- Die Funktion power wird abgearbeitet.
- Bei der Anweisung `return y;` wird die Funktion power verlassen und der Wert von y wird der linken Seite beim Funktionsaufruf (hier z) zugewiesen.
- Schließlich wird die Funktion main fortgesetzt.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufbau einer Funktionsdefinition:



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Funktionsname:

- Es gelten die gleichen Regeln wie für Variablennamen.
- *Der Name einer Funktion sollte etwas mit der Aufgabe zu tun haben, die von der Funktion erledigt wird!*

Funktionsparameter:

- Funktionsparameter sind diejenigen Werte, die die Funktion von außen (vom aufrufenden Programm) benötigt.
- Datentypen, Namen, Reihenfolge festlegen – **formale Parameter**.
- `z = power(x1,n);` – x1 und n heißen **aktuelle Parameter**.
- Erst beim Funktionsaufruf (während der Laufzeit des Programms) wird Speicherplatz für die Parameter reserviert.
- Parameter dürfen nur innerhalb der Funktion verwendet werden (sog. **lokale Variablen**).
- **void** („nichts, leer“), falls keine Aufrufparameter vorhanden sind.

Funktionstyp, Rückgabewert der Funktion:

- Funktionstyp und Datentyp des Rückgabewerts, der in der **return-Anweisung** zurückgegeben wird, müssen übereinstimmen.
- Alle Datentypen sind erlaubt.
- Funktionstyp **void** wird für Funktionen ohne Rückgabewert verwendet.

Rücksprung, return-Anweisung:

- Mit der return-Anweisung wird die Funktion verlassen – es erfolgt ein Rücksprung zum aufrufenden Programm.
- Beispiele: `return y;` `return y*y + 3.16;`
- Mehrere return-Anweisungen in einer Funktion möglich.
- Rücksprung erfolgt ebenfalls, wenn das Funktionsende erreicht wird (nur sinnvoll, falls Funktionstyp void verwendet wird).
- In der Funktion main bewirkt die return-Anweisung das Programmende; der Wert kann vom Betriebssystem ausgewertet werden.

Deklaration:

- Jede Funktion muss vor ihrer Verwendung **deklariert** werden.
- Der sog. **Funktionsprototyp** legt die Schnittstelle zum Aufruf der Funktion fest (Name der Funktion; Datentypen, Anzahl und Reihenfolge der Übergabeparameter).
- Die Funktionsdeklaration (bzw. der Funktionsprototyp) sieht aus wie die erste Zeile der Funktionsdefinition – sie wird allerdings mit einem Semikolon abgeschlossen.
- Beispiel: `float power(float x, int k);`

Funktion main:

- Die Funktion main wird beim Start des Programms automatisch aufgerufen.
- Gültige Varianten (lt. C-Standard):

```
int main (void)
int main (int argc, char *argv[])
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 1: Minimum von 2 ganzen Zahlen bestimmen

- Aufrufparameter: zwei Integerzahlen
- Rückgabewert: Integerzahl
- Funktionsprototyp: `int imin(int a, int b);`

```
#include <stdio.h>
int imin(int a, int b);
int main(void)
{
   int i, j, k;
   i = 3;
   j = 7;
   k = imin(i, j);
   printf("Min = %d\n", k);
   k = imin(i, 234);
   printf("Min = %d\n", k);
   return 0;
}
```

Deklaration von imin (Funktionsprototyp)

Aufruf der Funktion imin

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 2: Minimum von zwei Fließkommazahlen

- Aufrufparameter: zwei Fließkommazahlen (float oder double)
- Rückgabewert: Fließkommazahl
- Funktionsprototyp: `float imin(float a, float b);`
  oder: `double imin(double a, double b);`

Beispiel 3: Minimum von 2 ganzen Zahlen, kein Rückgabewert

- Die Funktion soll das Minimum ermitteln und auch ausgeben
- Aufrufparameter: zwei Integerzahlen
- Rückgabewert: void
- Funktionsprototyp: `void imin(int a, int b);`

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:
Es soll ein einfacher Taschenrechner mit zwei Funktionen erstellt werden. Der Taschenrechner kann zwei Integerzahlen multiplizieren und eine Zahl, die dezimal eingegeben wurde, in hexadezimaler Form wieder ausgeben. Schreiben Sie ein C-Programm, das folgende Aufgaben erfüllt: Der Anwender wird aufgefordert eine Taschenrechnerfunktion auszuwählen. Nach der Ausführung dieser Funktion kann eine weitere Funktion ausgewählt oder das Programm beendet werden.

```
Einfacher Taschenrechner
(0) Beenden
(1) Multiplikation
(2) Hexadezimalzahl
Bitte wählen Sie: _
```

```
Multiplikation
Erste Zahl: _
Zweite Zahl: _
Produkt _ * _ = _
```

```
Hexadezimalzahl
Geben Sie eine Dezimalzahl ein: _
Dezimal _ entspricht _ Hexadezimal
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Strukturierte Programmierung

- Schrittweise Verfeinerung
- Größeres Problem in kleinere Teilprobleme zerlegen
- Teilprobleme gegebenenfalls wieder zerlegen
- Funktionen helfen, größere Probleme zu zerlegen; jede Funktion bearbeitet ein definiertes, einfaches Teilproblem
- Beschränkung auf 3 einfache Kontrollstrukturen:

1)	**Folge**	`anweisung1;` `anweisung2;`
2)	**Alternative**	`if else` `switch`
3)	**Wiederholung**	`while` `do while` `for`

Teilaufgaben

1) Eingabeaufforderung anzeigen, Funktion auswählen (1 oder 2)
   Eingegebene Zahl prüfen, gegebenenfalls Eingabe wiederholen
   → Funktion: **auswahl**
2) Zwei Zahlen einlesen
   Multiplikation durchführen, Produkt ausgeben
   → Funktion: **multiplikation**
3) Aufforderung zur Eingabe einer Zahl
   Dezimalzahl einlesen, hexadezimal ausgeben
   → Funktion: **hexdarstellung**
4) Hauptprogramm, Ablaufsteuerung
   Funktion auswahl aufrufen, gibt Integerwert zurück
   Gewählte Funktion ausführen, ggf. Programm beenden
   auswahl erneut aufrufen
   → Funktion: **main**

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Hauptprogramm des Taschenrechners:

```c
#include <stdio.h>

int auswahl(void);   /* Funktionsdeklarationen */
void hexdarstellung(void);
void multiplikation(void);

int main(void)   /* Hauptprogramm */
{
    int wahl;
    do
    {
        wahl = auswahl();
        switch(wahl)
        {
        case 1: multiplikation(); break;
        case 2: hexdarstellung(); break;
        }
    }
    while(wahl != 0);   /* Beenden durch Eingabe von 0 */
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beim Programmieren gibt es eine Reihe von Standardaufgaben, die immer wieder auftreten:

- Berechnung der Quadratwurzel
- n-te Potenz einer Zahl berechnen
- trigonometrische Funktionen
- Ein- und Ausgabe

Hierfür existieren bereits fertige **Standardfunktionen** – normiert im ANSI-Standard. Sie sind in praktisch allen Programmierumgebungen vorhanden.

Auf den folgenden Folien werden verschiedene Gruppen von Standardfunktionen vorgestellt (viele weitere Beispiele finden sich im Internet oder in der Online-Hilfe).

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Mathematische Standardfunktionen: `#include<math.h>`

Quadratwurzel $y = \sqrt{x}$	`double sqrt(double x)`
Potenzfunktion x^y	`double pow(double x,double y)`
Exponentialfunktion	`double exp(double x)`
Natürlicher Logarithmus	`double log(double x)`
Logarithmus zur Basis 10	`double log10(double x)`
Absoluter Betrag	`double fabs(double x)`
Sinus	`double sin(double x)`
Kosinus	`double cos(double x)`
Tangens	`double tan(double x)`
Arcus Sinus	`double asin(double x)`
Arcus Kosinus	`double acos(double x)`
Arcus Tangens	`double atan(double x)`
Sinus Hyperbolicus	`double sinh(double x)`
Kosinus Hyperbolicus	`double cosh(double x)`
Tangens Hyperbolicus	`double tanh(double x)`

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ausschnitt aus der Header-Datei math.h:

```
/* math.h -- Definitions for the
   math floating point package.  */
double cos(double x);
double sin(double x);
double tan(double x);
double tanh(double x);
double fabs(double x);
double acos(double x);
double asin(double x);
double cosh(double x);
double sinh(double x);
double exp(double x);
double pow(double base, double power);
double sqrt(double x);
#define M_E   2.7182818284590452354
#define M_PI  3.14159265358979323846
. . .
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Verarbeitung von Zeichen: `#include<ctype.h>`

`int isalnum(int c);`	Ist c ein Buchstabe oder eine Ziffer
`int isalpha(int c);`	Ist c ein Buchstabe?
`int iscntrl(int c);`	Ist c ein Steuerzeichen?
`int isdigit(int c);`	Ist c eine Ziffer?
`int isgraph(int c);`	Ist c ein druckbares Zeichen (außer Leerzeichen)?
`int islower(int c);`	Ist c ein Kleinbuchstabe?
`int tolower(int c);`	Umwandlung von c in einen Kleinbuchstaben
`int toupper(int c);`	Umwandlung von c in einen Großbuchstaben
`int isprint(int c);`	Ist c ein druckbares Zeichen?
`int ispunct(int c);`	Ist c ein Interpunktionszeichen?
`int isspace(int c);`	Ist c ein Leerzeichen/Tabulatur/Zeilenumbruch?
`int isupper(int c);`	Ist c ein Großbuchstabe?
`int isxdigit(int c);`	Ist c eine Hexadezimalziffer?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ein- und Ausgabe: `#include<stdio.h>`

- `printf, scanf` – Ausgabe auf Bildschirm
- `fprintf, fscanf` – Ein-/Ausgabe mit Dateien
- `getchar` – Einzelnes Zeichen einlesen

„Standard-Library“, Verschiedenes: `#include<stdlib.h>`

- `rand` – Integer-Zufallszahl zwischen 0 und RAND_MAX
- `abs, labs` – Absoluter Betrag von Integer- und Long-Variablen (vergl. `fabs` aus math.h für Double-Variablen)

Verarbeitung von Datum, Uhrzeit: `#include<time.h>`

Arbeit mit Zeichenketten: `#include<string.h>`

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Was passiert, wenn wir den Funktionsparametern und der Variablen in der Funktion **imin** die Namen i, k und j geben (Achtung: In der Funktion **main** gibt es ebenfalls Variablen i, j und k…)?

Ist das erlaubt? (→ **Ja!**) Belegen die Variablen die gleichen Speicherzellen, kommt es zu „Überschneidungen“? (→ **Nein!**)

```c
#include <stdio.h>
int imin(int i, int k);
int main(void)
{
    int i, j, k;
    i = 3; j = 7;
    k = imin(i, j);
    printf("Min = %d\n", k);
    k = imin(i, 234);
    printf("Min = %d\n", k);
    return 0;
}
```

```c
int imin(int i, int k)
{
    int j;
    if (i < k)
        j = i;
    else
        j = k;
    return j;
}
```

Lokale Variablen:

- Lokale Variablen werden im Kopf einer Funktion (wie die Variablen i und k bei der Funktion imin) oder innerhalb einer Funktion definiert (wie die Variable j in der Funktion imin oder die Variablen i, j, k und n in der Funktion main).
- Sie dürfen bzw. können nicht von anderen Funktionen verwendet werden.
- Nur während eine Funktion bearbeitet wird, ist Speicherplatz für die lokalen Variablen der Funktion reserviert; nach Beendigung der Funktion gibt es diese Variablen nicht mehr.
- Lokale Variablen, die den gleichen Namen besitzen, aber in unterschiedlichen Funktionen definiert sind, haben nichts miteinander zu tun (wie die zwei Variablen j in den Funktionen main und imin); sie haben nur zufällig den gleichen Namen.


```
#include <stdio.h>
void imin(int a, int b);
int k;

int main(void)
{
   int i = 3, j = 7;
   imin(i, j);
   printf("Min = %d\n", k);
   k = 17; imin(k, 234);
   printf("Min = %d\n", k);
   return 0;
}

void imin(int a, int b)
{
   if (a < b)
      k = a;
   else
      k = b;
}
```

Beispiel für eine globale Variable: k ist sowohl in der Funktion main als auch in imin sichtbar!

Welche Ausgabe liefert dieses Programm?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Globale Variablen:

- Globale Variablen werden außerhalb von Funktionen, meist am Programmanfang definiert.
- Sie existieren während der gesamten Programmlaufzeit, d. h. es sind während der gesamten Programmlaufzeit Speicherzellen für die globalen Variablen reserviert.
- Sie dürfen von allen Funktionen verwendet werden, außer es gibt innerhalb einer Funktion eine lokale Variable mit dem gleichen Namen – lokale Variable hat Vorrang; in diesem Fall kann die Funktion nicht auf die globale Variable zugreifen (die globale Variable ist „unsichtbar“).
- Typische Anwendung globaler Variablen: Informationsaustausch zwischen Funktionen; speziell zur **gleichzeitigen Rückgabe mehrerer Werte** aus einer Funktion.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 6 – Funktionen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

Erstellen Sie ein Programm zur Lösung quadratischer Gleichungen mit der pq-Formel. Lösen Sie die quadratische Gleichung in einer Funktion **`int qsolve(double p, double q)`** und geben Sie beide Nullstellen in den globalen Variablen x1 und x2 zurück.

Die Funktion qsolve gibt in ihrem Integer-Rückgabewert eine 1 zurück, falls die Gleichung gelöst werden konnte, andernfalls eine 0.

$$x^2 + px + q = 0$$

$$\Rightarrow \quad x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

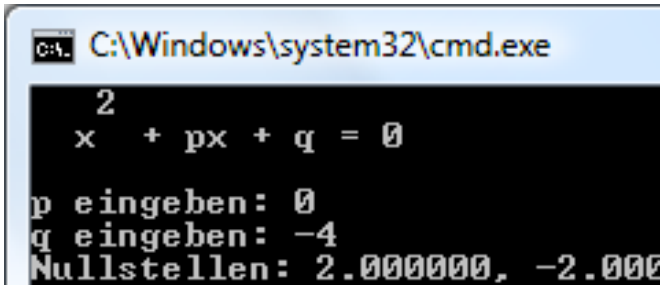
Hauptprogramm mit Aufruf der Funktion qsolve:

```
#include <stdio.h>
#include <math.h>

double x1, x2;
int qsolve(double p, double q);

int main(void)
{
    double p, q;
    printf("   2                \n");
    printf("  x  + px + q = 0\n\n");
    printf("p eingeben: "); scanf("%lf", &p);
    printf("q eingeben: "); scanf("%lf", &q);

    if(qsolve(p, q) != 0)
        printf("Nullstellen: %f, %f\n", x1, x2);
    else
        printf("Keine Nullstellen!\n");
    return 0;
}
```



Kapitel 7

Zusammengesetzte Datentypen, Vektoren, Zeichenketten

Gliederung

Kapitel 7 – Zusammengesetzte Datentypen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

Es soll ein Programm zur Versuchsauswertung erstellt werden:

- Bis zu 100 Messwerte einlesen
- Messwerte zur Kontrolle wieder ausgeben
- Mittelwert und Maximum berechnen

Es werden dazu 100 Variablen vom Typ float oder double benötigt. Es wäre möglich – aber sehr umständlich – zu diesem Zweck 100 verschiedene Variablen zu definieren.

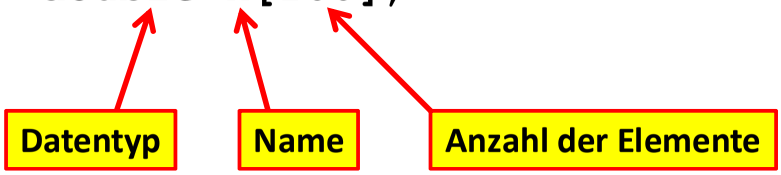
(Was würde bei einer Erweiterung auf 1000 Werte passieren?)

Einfacher: Definition eines Vektors mit 100 Elementen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Definition eines Vektors:

```
double x[100];
```



- Reserviert 100 Speicherplätze vom Typ double
- Die Anzahl der Elemente sollte **als Integer-Konstante** angegeben werden, Folgendes ist in älteren Varianten von C sowie in C++ **nicht erlaubt**: `float x[dim];`

- Die einzelnen Elemente des Vektors liegen im Hauptspeicher in aufeinanderfolgenden Speicherzellen
-
```
float x[4] = {2.0, 4.0, 0.0, 1.2};
int z[3] = {1, 5, 10};
int z[] = {1, 5, 10};
```

Zugriff auf einzelne Elemente:

`x[0] = 5.5;`	Zuweisung an Element Nr. 0
`x[1] = y;`	Zuweisung an Element Nr. 1
`x[k] = y;`	(k+1)-tes Element des Vektors, k muss Integervariable oder -konstante sein

Ein einzelnes Element x[k] des Vektors kann wie eine normale double-Variable verwendet werden:

```
x[k] = x[3];
printf("Sechster Wert im Vektor: %f", x[5]);
if (x[i] > x[i+1])
    printf("Groesser!\n");
```

Beachte:

- Index läuft von 0 bis 99 (und nicht etwa von 1 bis 100)
- Index darf nicht zu groß werden; x[391] wird vom Compiler nicht erkannt!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Teilaufgaben/Funktionen:

1) `int einlesen(void)`
   - Zunächst Anzahl der vorhandenen Messwerte abfragen
   - Messwerte der Reihe nach einlesen, Elemente x[0] bis x[zahl-1] belegen
   - Aufrufparameter: keine (void)
   - Rückgabewert: Anzahl (int)
2) `void ausgeben(int n)`
   - Aufrufparameter: n – Zahl der eingelesenen Messwerte
   - *Funktion muss wissen, wie viele Elemente von x tatsächlich Messwerte sind*
   - Rückgabewert: keiner (void)
3) `double mittelwert(int n)`
   - Aufrufparameter: n – Zahl der eingelesenen Messwerte
   - Rückgabewert: Mittelwert (double)
4) `double maximum(int n)`
   - Aufrufparameter: n – Zahl der eingelesenen Messwerte
   - Rückgabewert: Maximum (double)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#include <stdio.h>
#define SIZE 100
int einlesen(void);
void ausgeben(int n);
double mittelwert(int n);
double maximum(int n);
double x[SIZE];
int main(void)
{
   double xmit, xmax; int anzahl;
   anzahl = einlesen();          /* Messwerte eingeben    */
   ausgeben(anzahl);             /* Messwerte ausgeben    */
   xmit = mittelwert(anzahl);    /* Mittelwert berechnen  */
   xmax = maximum(anzahl);       /* Maximum berechnen     */
   printf("Mittelwert = %f\n", xmit);
   printf("Maximalwert = %f\n", xmax);
   return 0;
}
```

Hauptprogramm, Funktion `main`

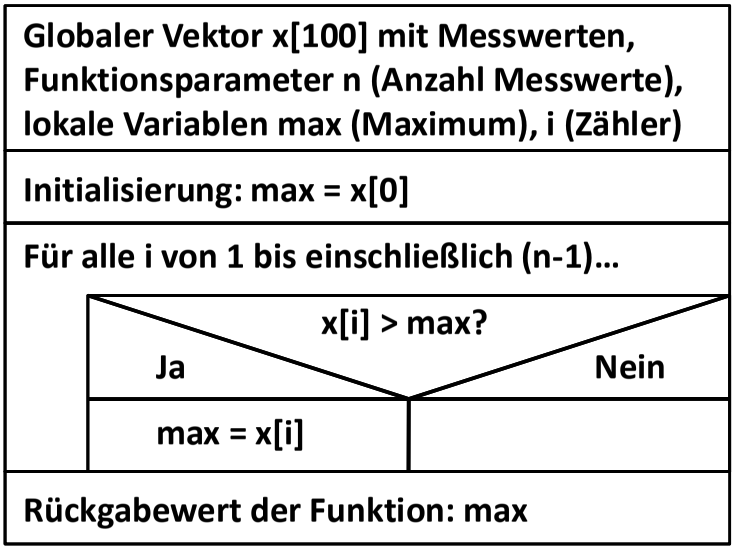
Definition eines Vektors mit 100 double-Elementen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel: Funktion **maximum** zur Ermittlung des max. Messwerts.

```
/* Es werden alle n Messwerte in dem Vektor x[]
   durchsucht; das Maximum wird zurückgegeben */
double maximum(int n)
{
  double max;
  int i;
  max = x[0];
  for(i=1; i<n; i++)
  {
    if(x[i] > max)
      max = x[i];
  }
  return max;
}
```

Was passiert bei n == 0…?



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

- Erstellen Sie die Struktogramme und C-Quelltexte der fehlenden Funktionen **`einlesen`**, **`ausgeben`** und **`mittelwert`**!
- Übersetzen Sie das Gesamtprogramm und testen Sie es!
- Was könnte noch verbessert werden? (Zum Beispiel die Eingabe der Messwerte: Nicht erst die Anzahl eingeben und danach alle Messwerte, sondern direkt alle Messwerte eingeben und durch Eingabe von -1 beenden. Das Programm kann dann die Anzahl automatisch ermitteln…)
- Geben Sie die Messwerte nicht über die Tastatur ein, sondern speichern Sie die Werte zunächst in einer Textdatei. Rufen Sie dann Ihr Programm über die DOS-Eingabeaufforderung auf und lesen Sie die Messwerte per Eingabeumlenkung ein!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 7 – Zusammengesetzte Datentypen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

Erweitern Sie das Messwertprogramm wie folgt:

Nach dem Einlesen der Messwerte sollen diese zunächst sortiert und erst danach ausgegeben werden!

- Schreiben Sie dazu eine Funktion **`void sortiere(int n)`**
- Die Elemente von x müssen so umgeordnet werden, dass gilt:

  x[0] ≤ x[1] ≤ x[2] ≤ … ≤ x[n-1] (für alle n Elemente)
- Verwenden Sie zum Sortieren keinen zweiten (Hilfs-)Vektor, sondern führen Sie die Sortierung direkt auf den Elementen des Vektors x[] durch!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Idee: Sortierung des Vektors mittels „Bubble-Sort“

Erster Durchlauf:

(**5 1** 4 2 8) → (**1 5** 4 2 8) Die ersten zwei Elemente vergleichen, tauschen
(1 **5 4** 2 8) → (1 **4 5** 2 8)
(1 4 **5 2** 8) → (1 4 **2 5** 8)
(1 4 2 **5 8**) → (1 4 2 **5 8**) Reihenfolge stimmt schon, nicht tauschen

Zweiter Durchlauf:

(**1 4** 2 5 8) → (**1 4** 2 5 8)
(1 **4 2** 5 8) → (1 **2 4** 5 8) Elemente tauschen
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

Die rot markierten Elemente werden verglichen und ggf. vertauscht

Dritter Durchlauf:

(**1 2** 4 5 8) → (**1 2** 4 5 8) Im dritten Durchlauf werden keine Elemente
(1 **2 4** 5 8) → (1 **2 4** 5 8) mehr vertauscht; der Algorithmus erkennt, dass
(1 2 **4 5** 8) → (1 2 **4 5** 8) der Vektor jetzt sortiert ist und stoppt
(1 2 4 **5 8**) → (1 2 4 **5 8**)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
void sortiere(int n) /* x[] mittels Bubble-Sort sortieren */
{
    int i; double hilf; char sortiert;

    do   /* Schleife wiederholen, bis Vektor sortiert ist */
    {
        sortiert = 'j';       /* Annahme, x[] ist sortiert */
        for(i = 1; i < n; i++)
        {
            if(x[i] < x[i-1])
            {
                hilf = x[i];      /* x[i], x[i-1] tauschen */
                x[i] = x[i-1];
                x[i-1] = hilf;
                sortiert = 'n';   /* ...war nicht sortiert */
            }
        }
    }
    while(sortiert == 'n');
}
```

Implementierung des Bubble-Sort-Algorithmus

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 7 – Zusammengesetzte Datentypen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Es soll ein C-Programm zum Rechnen mit (2-, 3-, mehrdimensionalen) Matrizen erstellt werden. Bislang sind aber nur eindimensionale Vektoren bekannt.

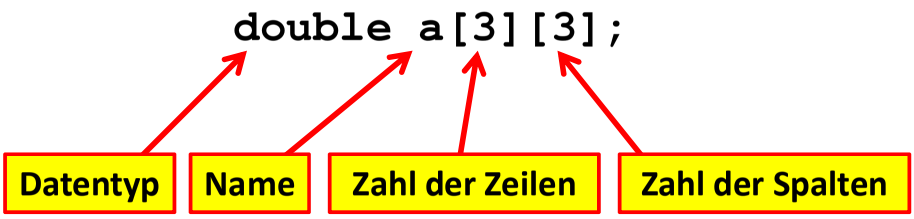
Anwendung von Matrizen: Lineare Gleichungssysteme, Finite-Elemente-Methode (sehr große Matrizen!), Drehimpuls/Trägheitsmatrix, Mathematik/ Geometrie/Computergrafik usw.

$$\begin{aligned} 7 \cdot x_1 + 8 \cdot x_2 &= 17 \\ 2 \cdot x_1 + 4 \cdot x_2 &= 11 \end{aligned} \quad \rightarrow \text{(als Matrix)} \quad \begin{bmatrix} 7 & 8 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 17 \\ 11 \end{bmatrix}$$

Eine Matrix besitzt Zeilen und Spalten → im C-Programm wird eine Datenstruktur für zwei-/mehrdimensionale Felder benötigt.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Definition eines zweidimensionalen Feldes:



Zugriff auf einzelne Elemente:

```
a[0][0] = 5.5;
a[0][1] = 1.5;
a[1][0] = 0.5;
a[i][k] = 1.7;
```

mit **0 ≤ i < 3** und **0 ≤ k <3**

(i und k sind Variablen vom Typ Integer)

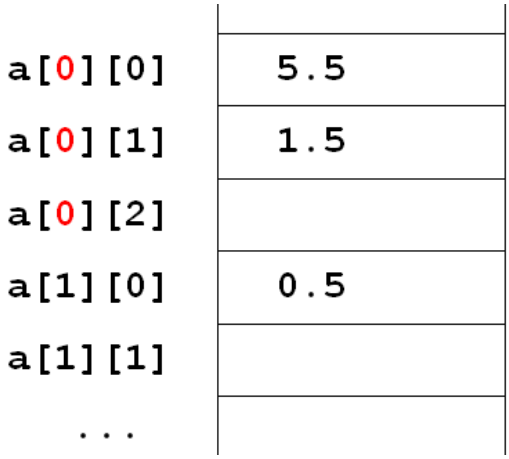
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Elemente in Matrixform:

$$a = \begin{vmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{vmatrix} = \begin{vmatrix} 5.5 & 1.5 & .. \\ 0.5 & .. & .. \\ .. & .. & .. \end{vmatrix}$$

Spalten — Zeilen

Anordnung im Hauptspeicher:



Die Matrix wird zeilenweise im Hauptspeicher abgelegt!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiele:

1) `double b[2][3];` (nicht quadratisch – 2 Zeilen, 3 Spalten)

2) Definition und gleichzeitige Initialisierung:
```
double b[2][3] = { {2.0,4.0,0.0}, {2.2,4.2,1.2} };
```

3)
```
#define   ZEILEN   1000
#define   SPALTEN   500
double d[ZEILEN][SPALTEN];
```

Zeilen- und Spaltenanzahl sollten mit Konstanten angegeben werden!

4) `double temp[10][10][10];`
3-dimensionales Feld – zum Beispiel Temperaturfeld im Raum:
Jedem Gitterpunkt im Raum wird eine Temperatur zugeordnet

5) `double v[100][100][200][3];` (4 dimensionales Feld)
Jedem Gitterpunkt im Raum werden 3 Werte zugeordnet –
z. B. drei Geschwindigkeitskomponenten in einem Strömungsfeld:

`v[2][4][7][0]` v_x am Punkt x=2, y=4, z=7
`v[2][4][7][1]` v_y am Punkt x=2, y=4, z=7

$100 \times 100 \times 200 \times 3 = 6 \times 10^6$ Elemente
Speicherplatzbedarf $6 \times 10^6 \times 8$ Byte = 48 MB

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 7 – Zusammengesetzte Datentypen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 1: Alle Elemente der Matrix a auf 0 setzen

Beispiel 2: Zwei Matrizen addieren: c = a + b

Beispiel 3: Prüfen, ob Matrix a symmetrisch ist

(symmetrische Matrizen sind „gutartig“, sie haben reelle Eigenwerte und N linear unabhängige Eigenvektoren…)

Beispiel 4: Matrix mit Vektor multiplizieren: $y = a \times x$

Beispiel 5: Multiplikation von zwei Matrizen: $c = a \times b$

```
#define N 10
double a[N][N], b[N][N], c[N][N];
double x[N], y[N];
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 7 – Zusammengesetzte Datentypen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Frage:

Wie werden Zeichenketten („Strings“) in der Programmiersprache C abgespeichert und verarbeitet?

Erster Ansatz: char-Vektor verwenden

```
char wort[5] = { 'H','a','l','l','o' };

int i;
for(i = 0; i < 5; ++i)
     printf("%c", wort[i]);
```

Nachteile:

- Initialisierung, Ausgabe ist umständlich – viel Schreibarbeit
- Bei der Bearbeitung (z. B. Ausgabe mittels printf), muss man sich merken, wie viele Buchstaben das Wort besitzt.

Wörter werden als Zeichenketten („Strings“) abgespeichert. Strings besitzen einige Besonderheiten bezüglich Initialisierung, Länge und Endekennung, sind aber im Wesentlichen normale char-Vektoren:

```
char s[]="Hallo";
```

- Erzeugt und initialisiert den char-Vektor s
- Binäre Null als Endekennung, binäre Null ≠ ASCII-Zeichen '0'!
- Anzahl Elemente: Buchstabenzahl + 1
- Länge des Vektors wird vom Compiler berechnet – über rechte Seite bestimmt
- Binäre Null wird als Endekennung angehängt

s[0]	H	72	0100 1000
s[1]	a	97	0110 0001
s[2]	l	108	0110 1100
s[3]	l	108	0110 1100
s[4]	o	111	0110 1111
s[5]	\0	0	0000 0000

Ein String ist ein char-Vektor mit einer binären Null zur Endekennung

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 1: Zeichen 'o' in "Hallo" durch 'e' ersetzen (→ "Halle")

Beispiel 2: Ausgabe eines Strings mittels printf

Beispiel 3: Einlesen eines Strings von der Tastatur

Beispiel 4: Klein- in Großbuchstaben umwandeln

Beispiel 5: Headerdatei string.h mit vielen Funktionen zur Arbeit mit Zeichenketten

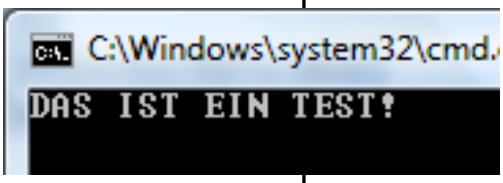
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 4: Klein- in Großbuchstaben umwandeln:

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    int i = 0;
    int diff = 'a' - 'A';

    char s[] = "Das ist ein Test!";
    while(s[i] != 0)
    {
        if('a'<=s[i] && s[i]<='z')
            s[i] = s[i] - diff;
        ++i;
    }
    printf("%s", s);
    return 0;
}
```



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 5: Headerdatei string.h

- **`int strlen(str);`**
  - Gibt die Länge eines Strings zurück, zählt Zeichen bis zur Endekennung

    `i = strlen("Hallo");` gibt 5 zurück

    (Mit binärer Null benötigt der String 6 Elemente im char-Vektor!)
- **`int strcmp(str1, str2);`**
  - Gibt 0 zurück, wenn beide Zeichenketten gleich sind
  - Ergebnis ist > 0, wenn str1 lexikalisch größer ist als str2
  - Ergebnis ist < 0, wenn str1 lexikalisch kleiner ist als str2

    `"z" > "a"  "ab" < "abc"  "ad" < "af"  "ac" > "abc"`
- **Viele weitere Funktionen**
  - Kopieren, Anhängen, Unterteilen von Zeichenketten, Suchen in Zeichenketten usw.
  - Vergl. Online-Hilfe zu string.h!

Kapitel 8
Adressen und Zeiger

Gliederung

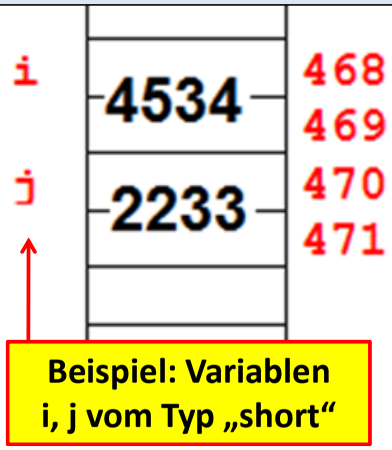
Kapitel 8 – Adressen und Zeiger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufbau des Hauptspeichers:

- Der Hauptspeicher besteht aus einer langen Sequenz von Speicherzellen (jeweils ein Byte groß)
- Jede Speicherzelle besitzt eine eindeutige Nummer/Adresse
- Eine Variable in einem C-Programm belegt entsprechend ihres Typs eine feste Anzahl von Bytes

- *Die Adressen der Variablen sind nicht absolut festgelegt – sie ändern sich zum Beispiel beim Programmneustart*
- *Der Programmierer hat keinen Einfluss darauf, in welchen Speicherzellen die Variablen liegen*

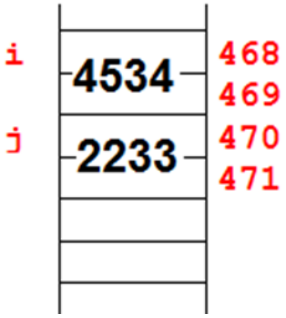


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Bisher wurde auf den Hauptspeicher (indirekt) über Variablen zugegriffen. Nun folgt der (direkte) Zugriff auf bestimmte Adressen:

- „Schreib den Wert 123 in die int-Variable an Adresse 1000!“
  „Lies den Wert der short-Variablen an Adresse 1004!“
- Nur die Kombination von Adresse + Variablentyp identifiziert die Speicherzellen einer Variablen eindeutig
- Unter der „Adresse einer Variablen i“ verstehen wir die Adresse der ersten Speicherzelle, wo der Wert von i gespeichert ist

- *Um die Variable i auf 4534 zu setzen, haben wir bisher die Zuweisung „`i = 4534;`“ verwendet*
- *Nun werden wir direkt auf den Hauptspeicher zugreifen: „**Schreib den Wert 4534 in die short-Variable an Adresse 468!**“*



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Fragen:

- *Wie erhält man die Adresse einer Variablen?*
- *Wie kann man eine Adresse zwischenspeichern?*
- *Wie liest/beschreibt man eine Variable, deren Adresse bekannt ist?*
- *Welche Vorteile bringt der direkte Zugriff auf den Hauptspeicher?*

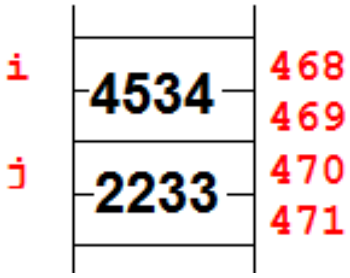
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Wie erhält man die Adresse einer Variablen?

- Ist i der Name einer Variablen, so ist &i die Adresse der Variablen
- Der Operator & ist der sog. „**Adressoperator**“
- Eine Adresse ist grundsätzlich eine ganze Zahl. Eine Adresse ist allerdings keine normale short- oder int-Zahl, man spricht vielmehr von sogenannten „**Zeigern**“

```
short i, j;
i = 4534;
j = 2233;

/*  i hat den Wert 4534       */
/*  &i hat den Wert 468       */
/*  Der Typ von i ist short   */
/*  Der Typ von &i ist "Zeiger auf short"  */
```



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Wie kann man eine Adresse zwischenspeichern?

- Zum Abspeichern der Adresse einer short-Variablen benötigt man eine Variable vom Typ „Zeiger auf short“, eine sog. **Zeigervariable**
- Definition einer Zeigervariablen (engl. „Pointer“):

```
short *z;
```

auf welchen Datentyp — *Name der Variablen*

z ist Zeigervariable

- z „zeigt auf“ eine Variable vom Typ short

- *Die Adresse der Variablen i wird in der Zeigervariablen z gespeichert:*

```
z = &i;   /* z erhält Adresse von i zugewiesen */
```

- *Direkt nach der Definition enthält z zunächst einen zufälligen Wert, nach der Zuweisung „zeigt z auf die Variable i“*
- *Zeigervariablen sind wie andere Variablen: Sie besitzen einen Namen und einen Wert und können durch Operatoren verändert werden*

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Wie liest/beschreibt man eine Variable, deren Adresse bekannt ist?

- Um auf eine Variable zuzugreifen, deren Adresse bekannt ist, verwendet man den Operator * (sog. „**Inhaltsoperator**“):

```
*z = 17;
```

→ ***Die Zeigervariable z „zeigt auf“ eine bestimmte Variable – schreib in diese Variable den Wert 17!***

```
j = *z;
```

→ ***Die Zeigervariable z „zeigt auf“ eine bestimmte Variable – lies den Wert dieser Variablen und speichere ihn in der Variablen j!***

Beachte: *Der Operator * besitzt verschiedene Bedeutungen:*

- *Multiplikationsoperator* (`x1 = 3 * y1;`)
- *Definition einer Zeigervariablen* (`int *ptr;`)
- *Inhaltsoperator* (`y = *ptr;`)

→ ***Die konkrete Bedeutung ergibt sich jeweils aus dem Zusammenhang!***

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 1: Funktion „power“ zur Berechnung von Potenzen; Rückgabe des Ergebnisses mittels Zeiger (und nicht per „return“ oder über eine globale Variable)

```
#include <stdio.h>
void power(float basis, int exp, float *pErg);

int main(void)
{
    float x1, x2, y; int n;
    x1 = 3.0; n = 4;
    power(x1, n, &y); printf("%f\n", y);
    x2 = 3.75;
    power(x2, 5, &y); printf("%f\n", y);
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 2: Funktion „qsolve“ zum Lösen quadratischer Gleichungen, Rückgabe **beider** Nullstellen mittels Zeiger (statt über globale Variablen)

```
#include <stdio.h>
#include <math.h>
int qsolve(double p, double q, double *x1, double *x2);
int main(void)
{
    double p, q, null1, null2;
    printf("p eingeben: "); scanf("%lf", &p);
    printf("q eingeben: "); scanf("%lf", &q);
    if(qsolve(p, q, &null1, &null2) != 0)
        printf("Nullstellen: %f, %f\n", null1, null2);
    else
        printf("Keine Nullstellen!\n");
    return 0;
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel 3:
Das folgende Programm kann ohne Fehler übersetzt werden. Nach dem Programmstart passiert allerdings Folgendes:

- Manchmal stürzt das Programm ab
- Manchmal arbeitet es scheinbar korrekt
- Manchmal wird ein falscher Wert ausgegeben

```
#include <stdio.h>
int main(void)
{
    int *z;
    *z = 5;
    printf("*z = %d", *z);
    return 0;
}
```

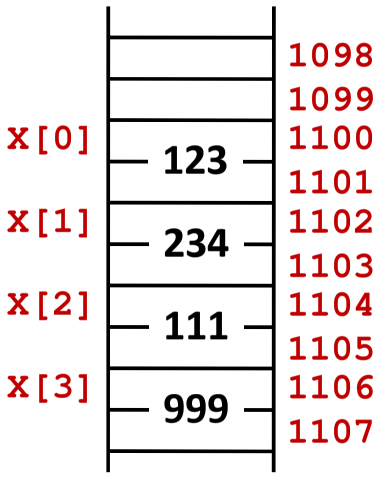
Wo liegt das Problem?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Zeigerarithmetik:
Für Zeiger gelten eigene Rechenregeln, die von den arithmetischen Rechenregeln für „normale“ ganze Zahlen abweichen.




```
short x[4];
short *z1, *z2, *z3, *z4;
int i = 3;

z1 = &x[0];  /* z1 zeigt auf x[0] */
z2 = z1 + 1; /* z2 zeigt auf x[1] */
z3 = z1 + i; /* z3 zeigt auf x[i] */

/* Nach folgender Anweisung zeigt */
/* z4 vor den Beginn des Vektors  */
/* x (Adresse 1098) - Vorsicht!!  */
z4 = z1 - 1;
```

Die Zeigerarithmetik ist bei der Arbeit mit Vektoren besonders praktisch. So lassen sich sehr einfach Vektoren „durchlaufen“:

```
int i, x[100];
int *ptr;

for(i = 0; i < 100; ++i)
{
    /* Zeiger auf i-tes Vektorelement: */
    ptr = &x[0] + i;

    /* Ausgabe des i-ten Vektorelements: */
    printf("%d\n", *ptr);
}
```

Beispiel: Im Vektor x sollen die Quadratzahlen von 0^2 bis 99^2 gespeichert werden.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

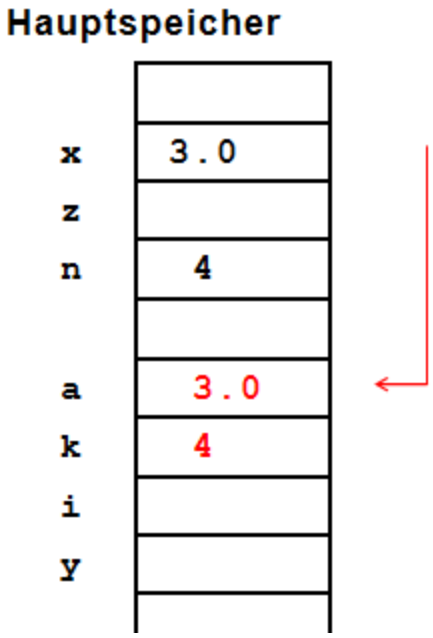
Kapitel 8 – Adressen und Zeiger

Wird eine „normale“ Variable als Parameter an eine Funktion übergeben, dann wird stets der Wert der Variablen übergeben. Der Wert der Variablen wird beim Aufruf in einen lokalen Parameter der Funktion kopiert.

```
double power(double a, int k);

int main(void)
{
  double x, z; int n;
  x = 3.0; n = 4;
  z = power(x, n);
  :
}

double power(double a, int k)
{
  int i; double y = 1.0;
  for(i = 1; i <= k; i++)
    y = y * a;
  return y;
}
```



Problem:

Die Funktion „quadrat“ soll die ersten n Elemente eines Vektors, der als Parameter übergeben wird, mit Quadratzahlen belegen

```
void quadrat(int *ptr, int n);
int main(void)
{
    int x[100];
    quadrat(&x[0], 50);
    :
}
void quadrat(int *ptr, int n)
{
    int i;
    for(i = 0; i < n; i++)
        *(ptr + i) = i * i;
}
```

Bei der Übergabe eines Vektors an eine Funktion wäre es zu aufwendig, den gesamten Vektor mit all seinen Elementen zu kopieren.

Man übergibt stattdessen einen Zeiger auf das Anfangselement. Die aufgerufene Funktion greift dann mittels Zeigerarithmetik auf den Original-Vektor zu.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

Bislang verarbeitet das Messwertprogramm eine einzelne Messreihe mit max. 100 Messwerten. Die Messreihe wird in einem globalen Vektor `double x[100]` abgespeichert.

Das Programm soll so erweitert werden, dass eine zweite Messreihe mit ebenfalls max. 100 Messwerten verwaltet werden kann.

8.5. Messwertprogramm, zwei Messreihen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

***Variante 1**: Zwei Vektoren x[100], y[100] und separate Funktionen für beide Vektoren. → Die Funktionen mittelwertx, mittelwerty unterscheiden sich nur in einem einzigen Buchstaben…!*

```
#define DIM 100
double x[DIM], y[DIM];
double mittelwertx(int n);
double mittelwerty(int n);
int main(void)
{
    double xmittel, ymittel;
    int anzahlx, anzahly;
    anzahlx = einlesenx();
    anzahly = einleseny();
    xmittel = mittelwertx(anzahlx);
    ymittel = mittelwerty(anzahly);
    return 0;
}
```

```
double mittelwertx(int n)
{
    int i; double z = 0.0;
    for(i = 0; i < n; i++)
        z = z + x[i];
    return z = z / (double)n;
}
double mittelwerty(int n)
{
    int i; double z = 0.0;
    for(i = 0; i < n; i++)
        z = z + y[i];
    return z / (double)n;
}
```


8.5. Messwertprogramm, zwei Messreihen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

***Variante 2**: Nur eine Funktion einlesen, mittelwert, sortiere usw. Der zu bearbeitende Vektor wird als Parameter an die Funktionen übergeben.*

```
int main(void)
{
  double x[SIZE], y[SIZE]; int anz1, anz2;
  printf("Messreihe 1 eingeben...\n"); anz1 = einlesen(&x[0]);
  printf("Messreihe 2 eingeben...\n"); anz2 = einlesen(&y[0]);
  sortiere(&x[0], anz1); sortiere(&y[0], anz2);
  printf("Messreihe 1...\n"); ausgeben(&x[0], anz1);
  printf("Messreihe 2...\n"); ausgeben(&y[0], anz2);
  printf("Mittelwert 1: %.2f\n", mittelwert(&x[0], anz1));
  printf("Mittelwert 2: %.2f\n", mittelwert(&y[0], anz2));
  printf("Maximum 1: %.2f\n", maximum(&x[0], anz1));
  printf("Maximum 2: %.2f\n", maximum(&y[0], anz2));
  return 0;
}
```

Durch die Übergabe als Parameter können die globalen Variablen double x[100] und y[100] vermieden werden!

***Beispiel*: Funktionen „ausgeben“ und „maximum“ mit Vektor als Funktionsparameter**

```
/* Alle n Messwerte ausgeben */
void ausgeben(double *mw, int n)
{
    int i;
    for(i = 0; i < n; ++i)
        printf("\tMesswert %d: %.2f\n", i + 1, *(mw+i));
    printf("\n");
}

/* Maximalen Messwert ermitteln und zurueckgeben */
double maximum(double *mw, int n)
{
    double max; int i;
    max = *(mw+0);
    for(i = 1; i < n; i++)
        if(*(mw+i) > max) max = *(mw+i);
    return max;
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Vektorschreibweise:

Bei der Arbeit mit Zeigervariablen, speziell bei Zeigerarithmetik im Zusammenhang mit Vektoren, können manche Operationen auch anders geschrieben werden (**ohne Änderung der Funktion!**):

```
void fill_n(int *x, int n);
int main(void)
{
    int x[100];
    fill_n(&x[0], 100);
    return 0;
}
void fill_n(int *x, int n)
{
    int i;
    for(i = 0; i < n; ++i)
        *(x+i) = i;
}
```

```
void fill_n(int x[], int n);
int main(void)
{
    int x[100];
    fill_n(x, 100);
    return 0;
}
void fill_n(int x[], int n)
{
    int i;
    for(i = 0; i < n; ++i)
        x[i] = i;
}
```


HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Zusammenfassung:

```
int x[100];
```

`&x[0]`	oder	`x`	Zeiger auf (bzw. Adresse von) x[0]
`&x[i]`	oder	`x+i`	Zeiger auf (bzw. Adresse von) x[i]
`x[0]`	oder	`*x`	Inhalt von x[0], Wert des Anfangselements
`x[i]`	oder	`*(x+i)`	Inhalt von x[i], Wert des i-ten Elements

```
void func(int *feld);
void func(int feld[]);
```

} Übergabe eines Zeigers an eine Funktion

Es handelt sich wirklich nur um verschiedene Schreibweisen mit derselben Bedeutung. Insbesondere bedeutet die Funktionsdeklaration in der letzten Zeile nicht, dass hier ein kompletter Vektor übergeben wird – auch in diesem Fall handelt es sich um die Übergabe eines Zeigers!

American Standard Code for Information Interchange (ASCII)

Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII
0	0x00	000	NUL	32	0x20	040	SP	64	0x40	100	@	96	0x60	140	`
1	0x01	001	SOH	33	0x21	041	!	65	0x41	101	A	97	0x61	141	a
2	0x02	002	STX	34	0x22	042	"	66	0x42	102	B	98	0x62	142	b
3	0x03	003	ETX	35	0x23	043	#	67	0x43	103	C	99	0x63	143	c
4	0x04	004	EOT	36	0x24	044	$	68	0x44	104	D	100	0x64	144	d
5	0x05	005	ENQ	37	0x25	045	%	69	0x45	105	E	101	0x65	145	e
6	0x06	006	ACK	38	0x26	046	&	70	0x46	106	F	102	0x66	146	f
7	0x07	007	BEL	39	0x27	047	'	71	0x47	107	G	103	0x67	147	g
8	0x08	010	BS	40	0x28	050	(	72	0x48	110	H	104	0x68	150	h
9	0x09	011	TAB	41	0x29	051	)	73	0x49	111	I	105	0x69	151	i
10	0x0A	012	LF	42	0x2A	052	*	74	0x4A	112	J	106	0x6A	152	j
11	0x0B	013	VT	43	0x2B	053	+	75	0x4B	113	K	107	0x6B	153	k
12	0x0C	014	FF	44	0x2C	054	,	76	0x4C	114	L	108	0x6C	154	l
13	0x0D	015	CR	45	0x2D	055	-	77	0x4D	115	M	109	0x6D	155	m
14	0x0E	016	SO	46	0x2E	056	.	78	0x4E	116	N	110	0x6E	156	n
15	0x0F	017	SI	47	0x2F	057	/	79	0x4F	117	O	111	0x6F	157	o
16	0x10	020	DLE	48	0x30	060	0	80	0x50	120	P	112	0x70	160	p
17	0x11	021	DC1	49	0x31	061	1	81	0x51	121	Q	113	0x71	161	q
18	0x12	022	DC2	50	0x32	062	2	82	0x52	122	R	114	0x72	162	r
19	0x13	023	DC3	51	0x33	063	3	83	0x53	123	S	115	0x73	163	s
20	0x14	024	DC4	52	0x34	064	4	84	0x54	124	T	116	0x74	164	t
21	0x15	025	NAK	53	0x35	065	5	85	0x55	125	U	117	0x75	165	u
22	0x16	026	SYN	54	0x36	066	6	86	0x56	126	V	118	0x76	166	v
23	0x17	027	ETB	55	0x37	067	7	87	0x57	127	W	119	0x77	167	w
24	0x18	030	CAN	56	0x38	070	8	88	0x58	130	X	120	0x78	170	x
25	0x19	031	EM	57	0x39	071	9	89	0x59	131	Y	121	0x79	171	y
26	0x1A	032	SUB	58	0x3A	072	:	90	0x5A	132	Z	122	0x7A	172	z
27	0x1B	033	ESC	59	0x3B	073	;	91	0x5B	133	[	123	0x7B	173	{
28	0x1C	034	FS	60	0x3C	074	<	92	0x5C	134	\	124	0x7C	174	|
29	0x1D	035	GS	61	0x3D	075	=	93	0x5D	135	]	125	0x7D	175	}
30	0x1E	036	RS	62	0x3E	076	>	94	0x5E	136	^	126	0x7E	176	~
31	0x1F	037	US	63	0x3F	077	?	95	0x5F	137	_	127	0x7F	177	DEL

Quelle: Wikipedia (Artikel: „American Standard Code for Information Interchange“)