Einführung in die Programmierung

Hello World, Typen & Variablen, Arithmetik

Arvid Terzibaschian

Ablaufplan

- wöchentliche Vorlesung + Übung

- Prüfung am Ende des Semester (siehe Abstimmung)
  - http://www.doodle.com/vtcqm9k8b7q57bx8
  - Achtung: Abstimmung auch für BRB-Termin! (Wer beides schreibt, sollte auch mindestens zwei möglich Termine angeben)
    - ENDE der Abstimmung 9.12. um 12 Uhr
- Klausur 90 min
  - auch für Studenten des gesamten Kurses (+BRB)

Inhalte

- Inhalte der Vorlesung
  - Elemente Konzepte Programmiersprachen
  - Code, Compiler und Linker
  - Datentypen und Variablen
  - Pointer und Arrays
  - Arithmetik, Kontrollstrukturen, Blöcke und Funktionen
  - Strukturen und Typdefinitionen
  - Eingabe/Ausgabe
  - Dateien
  - …

„Hello World!“, Programmierung.

Programmierung

- Computer führen Anweisungen aus
- Anweisungen in Maschinensprache formuliert
  - binäre Anweisungen (Wörter) bestimmter „Bitbreite“ z.B. 32-Bit:
    - 1000 1010 0001 0100 1000 1010 1100 1101
  - Binärprogramm: Folge von binären Anweisungen

- Lösungen in Binärsprache zu formulieren für Menschen zu komplex
  - Lösung: **Programmiersprachen**

Programmiersprachen

- Eine **Programmiersprache** ist eine definierte *Sprache* zur Formulierung von Datenstrukturen und *Rechenvorschriften*, die von einem *Computer* ausgeführt werden können. (Quelle: Duden Informatik)
- Sie setzen sich aus Anweisungen nach einem definierten Muster (Syntax) zusammen.

  - **Maschinensprache** (Binärcode/ASM) drückt Programm direkt durch Maschinenbefehle aus
    - niedrige Abstraktionsebene
    - schwer verständlich für Menschen

  - **Programmiersprachen (Hochsprachen** C, Java, **…)**
    - werden vor oder während der Ausführung übersetzt (kompiliert)
    - sind für Menschen einfacher verständlich

Arten von Programmiersprachen

(eher) deklarative Programmiersprachen

- definieren Ergebnis:

```
A := "alle Autos aus
Washington mit blauem Lack
und ´als gestohlen gemeldet´"
```

- Beispiele:
  - SQL, LINQ, Xpath, HTML, XML,Prolog, Haskell…
  - Mathematik

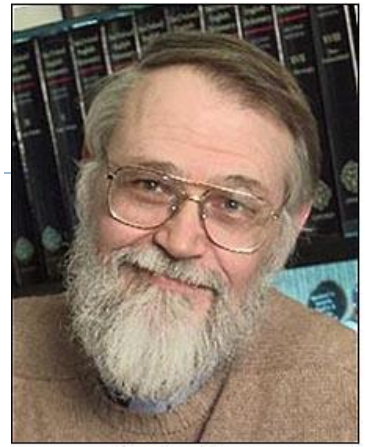
(eher) imperative Programmiersprachen

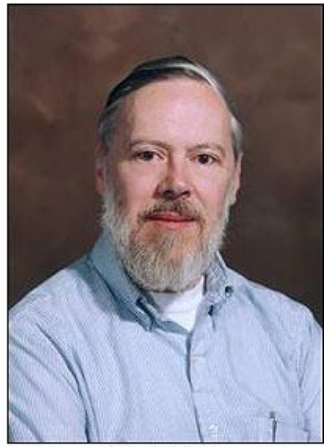
- definieren Rechenweg:

```
A = leere Liste;
für i := 1 bis AnzahlAutos
   Wenn: Auto(i)
      aus Washington,
      blau,
      als gestohlen gemeldet,
   Dann:
      füge Auto(i) in Liste A ein;
```

- Beispiele:
  - C, Java, PHP, C#, Python,…

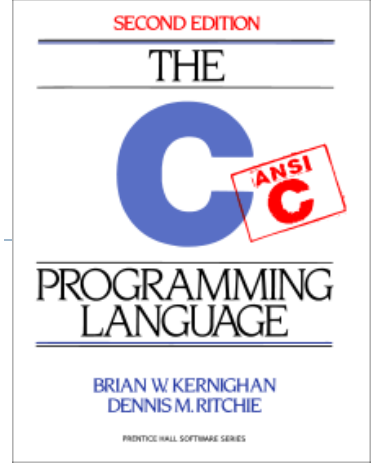
C Programmiersprache


Brian Kernighan


Dennis Ritchie

- 1970 von Kernighan und Richie für UNIX entwickelt
  - Brian W. Kernighan, Dennis M. Ritchie: *The C Programming Language/ Programmieren in C*
- C-Compiler auf allen UNIX-Systemen verfügbar
- Eigenschaften:
  - imperative Programmiersprache
  - klare, einfache Syntax durch wenig Schlüsselworte
  - direktes Ansprechen von Hardware
  - Portabilität
- viele moderne Sprachen nutzen ähnliche Syntax wie C

Entwicklung mit C

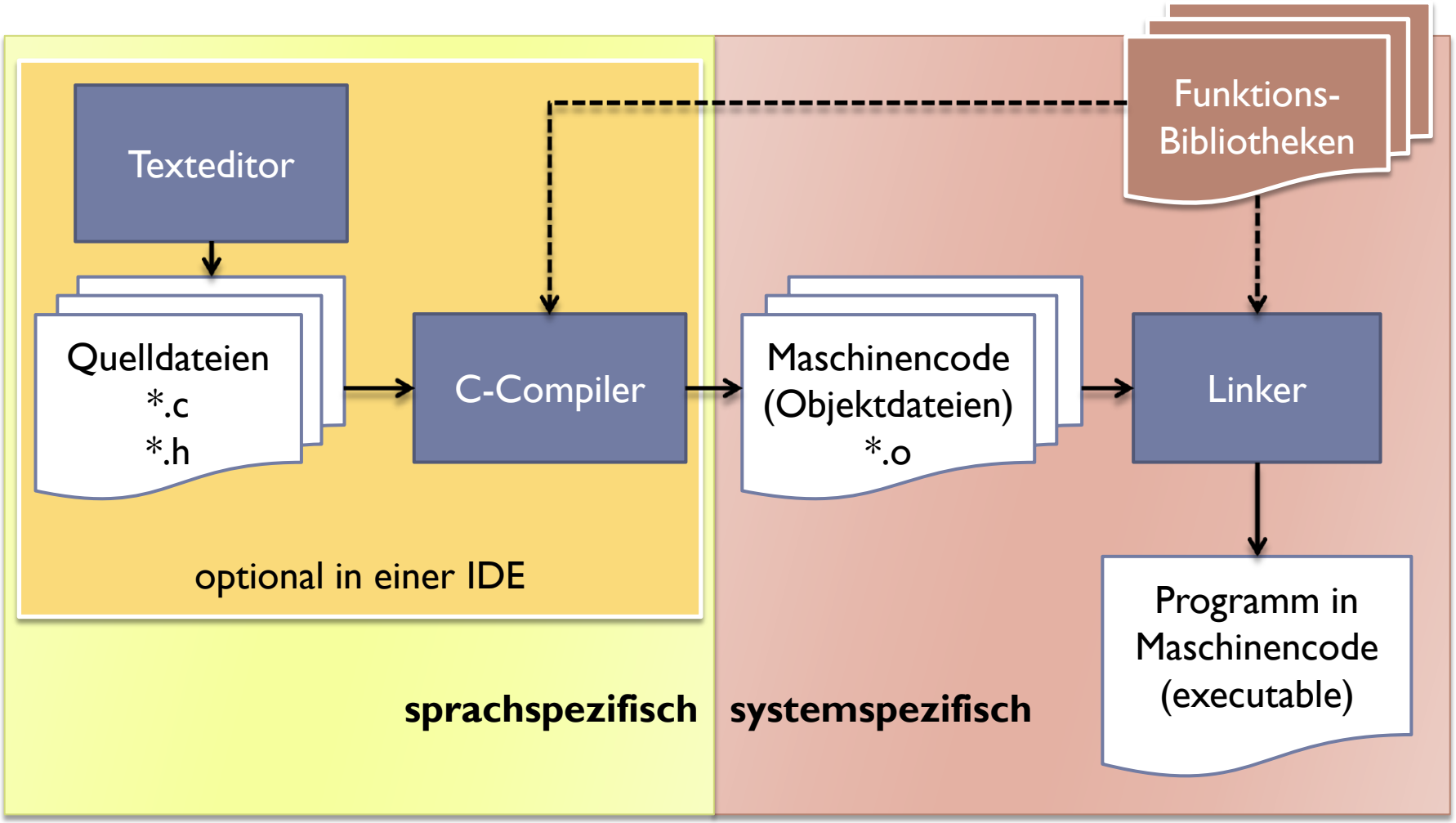


- Programm besteht aus C-Quellcodedateien
  - typischerweise mit Endungen: .c und .h
- C-Compiler übersetzt C-Quellcode in Maschinensprache

Unix: gcc **Mac: gcc, clang** **Win: VC++, Cygwin**

  - Sprache systemübergreifend standardisiert
    - Quellcode von System zu System übertragbar („portierbar“)
- Entwicklungsumgebung ist Texteditor + Compiler
  - verfügbare IDEs:
    - Eclipse CDT (Windows + Linux)
    - Visual Studio (Windows)
    - K-Develop

Entwicklung mit C



Hello World!


```
/*    Datei hello.c
*/

#include <stdio.h>    // binde Ein- und Ausgabefunktionen ein

int main() {
       printf("Hello World!");       // Ausgabe
       return 0;                     // Alles ok;
}


/*
       Erstes, elementares C-Programm:
       Gibt „Hello World!“ auf der Kommandozeile aus.
*/
```

Hello World

- Übersetzen mit Compiler gcc:
  - `gcc –o hallo hallo.c`
  - Übersetzt die Datei hallo.c und erstellt ausführbare Datei „hallo“
- Ausführen in der Kommandozeile:
  - ./hallo
- Ergebnis:

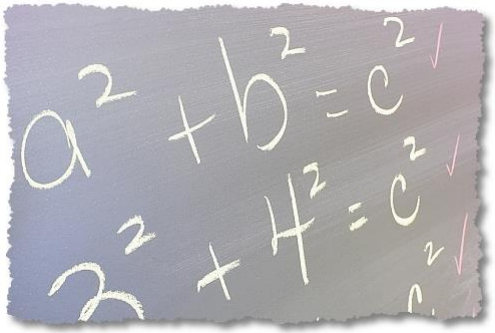
Datei hallo.c

```
#include <stdio.h>
int main() {
     printf(“Hello World!“);
     return 0;
}
```

```
$ gcc –o hallo hallo.c
$ ./hallo
Hello World!
$
```

Variablen und Datentypen

Variablen und Datentypen



- Variablen sind Datenspeicher
- werden mit speziellen Anweisungen
  - erschaffen (deklariert) und initialisiert,
  - gelesen.
  - verändert.
- existieren maximal bis Ende des Programmes, meist wesentlich kürzer, an eindeutiger Speicherstelle
- Variablen in C haben *immer* einen festgelegten Datentyp
  - streng typisierte Sprache
- In C gibt es *elementare* und *komplexe* Datentypen

Variablen deklarieren

- Anweisung, die Compiler über Variable „informiert“

```
#include <stdio.h>
int main() {
      int myVar;
      return 0;
}
```

- definiert Variable namens „myVar“ mit Datentyp „int“
  - der C-Datentyp **int** kann auf 32-/64-Bit-Systemen Zahlen zwischen -2^31 und 2^31 speichern

Elementare Datentypen für Variablen in C

Name	Wertebereich (64-Bit Architektur)	Beschreibung
int	-2^31 bis 2^31	ganze Zahlen (32 Bit)
long (int)	-2^63 bis 2^63	ganze Zahlen (64 Bit)
short(int)	-2^15 bis 2^15	ganze Zahlen (16 Bit)
char	-128 bis 127	einzelnen Zeichen, ein Byte
float	-2^128 bis 2^127 mit ~7 Dezimalstellen	reelle Zahlen
double	-2^1023 bis 2^1023 mit ~16 Dezimalstellen	reelle Zahlen, doppelte Anzahl von Nachkommastellen
	exakte Größe Compilerabhängig!	

Mögliche Namen für Variablen

- Variablen beginnen immer mit Buchstaben oder _

```
int a;   int _a;
```

- Groß- und Kleinschreibung wird unterschieden

```
int a; int A;       // zwei verschiedene Variablen
```

- andere zulässige Zeichen sind:
  - alle Ziffern:
  - Unterstrich:

```
int a123;
int a_123;
```

- maximale Länge compilerabhängig
  - oft 255 Zeichen
  - Standard garantiert mindestens 63-Zeichen maximale Länge

Variablen initialisieren

- Anweisung, die Compiler über Variable „informiert“

```
#include <stdio.h>
int main() {
     int myVar = 23;
     return 0;
}
```

- initialisiert „myVar“ mit dem Wert 23

Variablen verändern

- Anweisung, die Compiler über Variable „informiert“

```
#include <stdio.h>
int main() {
      int myVar = 23;
      return 0;
}
```

- = ist der Zuweisungsoperator
  - setzt den Wert einer Variablen für nachfolgende Anweisungen
  - Operator ist **nicht** symmetrisch:

```
224 = myVar; //  ist keine gültige Anweisung
```

Variablen auslesen

- Variablen werden über ihren Bezeichner ausgelesen

```
#include <stdio.h>
int main() {
      int myVar = 23;
      // Auslesen von myVar:
      return myVar;
}
```

- Programm gibt bei Aufruf den Wert „23“ an Betriebssystem zurück

Variablen auslesen

```
#include <stdio.h>
int main() {
     int myVar = 23;
     int myVar2;
     myVar2 = myVar;
}
```

- Computer liest Wert von Variable myVar aus und weist diesen Variable myVar2 zu

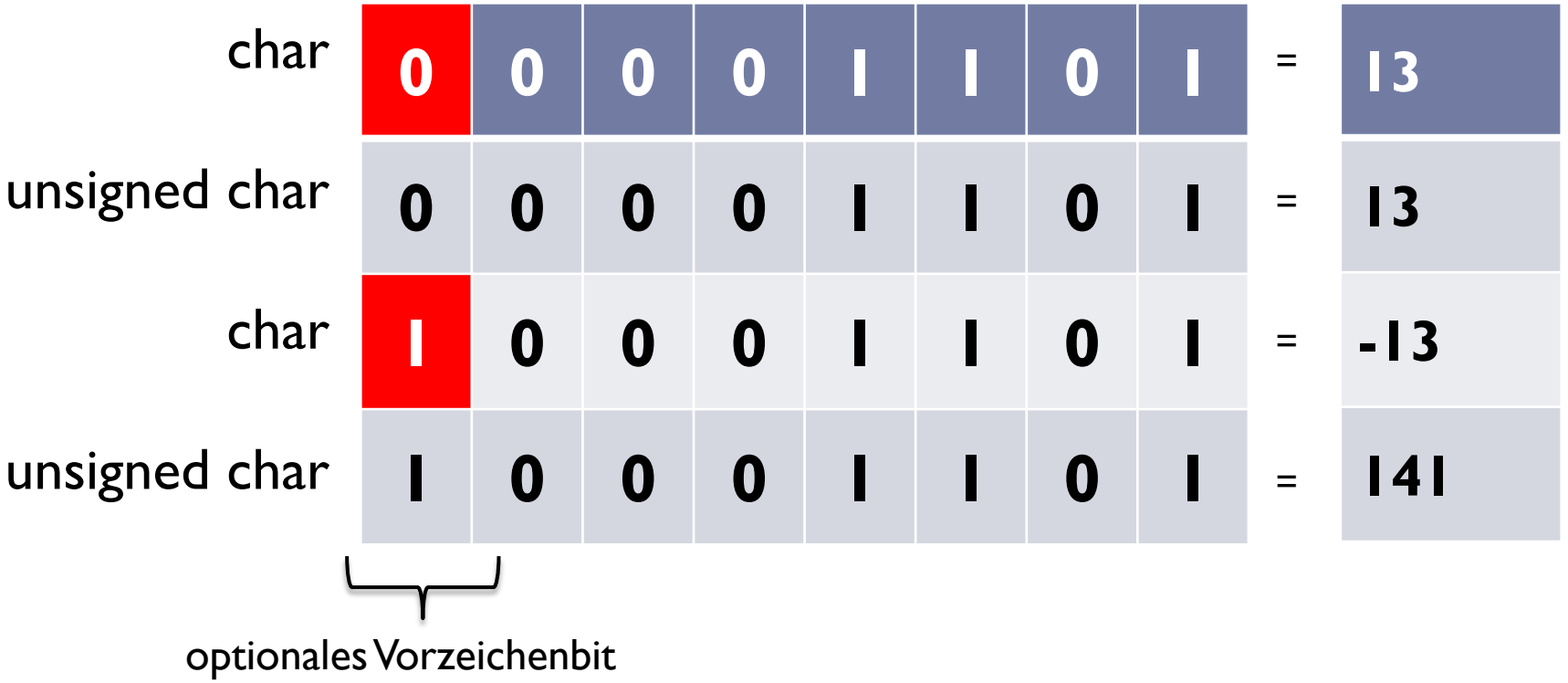
Schlüsselwort unsigned

- Natürliche Zahlen werden über das Schlüsselwort „unsigned“ vereinbart:
  - **unsigned int myVar;**
- unsigned gibt es für die Datentypen
- **short, int, long, char**

  der Wertebereich der Variablen beginnt bei 0
  - negative Zahlen können nicht mehr dargestellt werden
  - **char myVar;** ganze Zahlen von -128 … 127
  - **usigned char myVar;** ganze Zahlen von 0 … 255

signed vs. unsigned: char

- Interpretation einer Binärfolge mit verschiedenen Datentypen



- Achtung: Darstellung in Maschine im 2er Komplement

Fließkommazahlen

- In C Datentypen float und double
  - Ersatz für reele Zahlen auf dem Computer
- Variablen vom Typ „double“ können
  - Zahlenbereich von 2^-1022 bis 2^1023 abdecken,
  - Zahlen zwischen 0 und 1 mit einer Genauigkeit von 16 Nachkommastellen darstellen

- Wie kann ein einzelner Datentyp so viele verschiedene Werte darstellen, ohne sehr viele Bits zu verbrauchen?

Fließkommazahlen

- **Wie kann ein einzelner Datentyp so viele verschiedene Werte darstellen, ohne sehr viele Bits zu verbrauchen?**

- **Idee**: Zahlen in Exponentialschreibweise darstellen.
  - **0,0000356 = 3.56 * 10^(-5)**
  - **356 000 000 = 3.56 * 10^(8)**
  - **3,1416 = 3.1416 * 10^0**

Mantisse **Exponent**

- Mantisse und Exponent mit wenigen Bits darstellbar
  - Exponent bestimmt grob den Zahlenbereich
  - Mantisse legt exakten Wert fest

Fließkommazahlen

Typ	Mantisse		Exponent		
	Bits	Dezimal-stellen	Bits	Bereich (Basis 2)	Bereich (Basis 10)
float	23	~7	8	-126...127	-38...38
double	52	~16	11	-1022...1023	-300...300

- exakte Darstellung abhängig von Programmiersprache
  - viele Programmiersprachen verwenden IEEE -754-Standard
  - kaum eine Sprache erfüllt den Standard vollständig
- Deklaration in C:

```
double    x = 3.1416e-4;                  // Exponentialschreibweise (e=10)
double    y = 3.14159265359e+300;         // sehr große Zahl
float     z = -32;                        // auch ganze Zahlen möglich
float     v = 1.432;                      // auch ohne Exponent möglich
```

Ausgabe von Zahlen mit printf

- printf ermöglicht formatierte Ausgabe von Variablen und Text
  - ```
    int x = 17;
    printf("Wert von x: %d",x);
    ```

Formatstring:
Text mit Platzhaltern

auszugebende
Variable(n)

  - Platzhalter %[Datentyp] im Formatstring bedeutet:
    - gib an dieser Stelle im Text den Wert einer Variablen vom Typ "Datentyp" aus
    - Datentyp des Platzhalters und Datentyp der Variablen müssen kompatibel sein, sonst ist Ausgabe fehlerhaft.
    ```

printf: Platzhalter und Datentypen

Datentyp	Platzhalter	Beispiel
(unsigned) int, short	%d (decimal)	`int n = -200;` `printf("Eine ganze Zahl: %d",n);`
(unsigned) long	%ld (long-decimal)	`unsigned long n = 12345678910;` `printf("Eine ganze Zahl: %ld",n);`

char	%c (character)	`char c = 'a';`
		`printf("Ein Zeichen: %c",c);`
		`// Wird nicht als Zahl interpretiert!`

float	%f (floating point)	`float x = 3.1416e-10;` `printf("Pi/10^10: %f",x);`
double	%lf (long floating-pt)	`double pi = 3.141592653589793;` `printf("Pi: %lf",x);`

- Es können mehrere Variablen gleichzeitig ausgegeben werden
  - Reihenfolge der Platzhalter im Formatstring und der Variablen müssen übereinstimmen

Arithmetik in C

Arithmetische Operationen

- elementare mathematische Operation in C
  - +, -, *, /, %
  - definiert für alle elementaren Datentypen (außer „%“)
  - Verwendung in C in Form von Ausdrücken mit (typisierten) Konstanten und Variablen
- Beispiele für Ausdrücke:

```
float x = 1.4142;
int a = 16;
float y;
y = x*x;            // setze y auf x^2
a = (a-10)*a;       // Berechne, dann speichere
y = y*a;            // Ausdruch mit versch. Typen …
```

Arithmetische Operationen für elementare Datentypen in C

Operator	Semantik	Beispiel	
=	Zuweisung	int **a = 3**;	// a ist 3
+	Addition	int a = **3 + 7**;	// a ist 10
-	Subtraktion	int a = **3 – 7**;	// a ist -4
*	Multiplikation	int a = **3 * 7**;	// a ist 21
/	Division	int a = **22 / 7**;	// a ist 3;
%	Modulo (Rest der Ganzzahl-Division)	int a = **22 % 7**;	// a ist 1;

+=	Addition, gefolgt von Zuweisung	int a = 3;
		a += 17*2;
			
			// berechne a = a + (17*2)
			// a ist 37

*=, /=, -=		entsprechend +=	

Operatorprioritäten

- Was ist das Ergebnis von ... ?
  - `int a = 2*5 % 3 + 27/3;  // ???`
- Bei Verschachtelung mehrerer Operationen:
  - explizite Reihenfolge durch Klammern angeben
    - `int a = ((2*5) % 3) + (27/3);  // a ist 10`
  - *implizite* Reihenfolge durch Prioritäten definiert
    - hohe Priorität: ***, /, %**
    - niedrige Priorität: **+,-**
    - Auswertung der Ausdrücke von links nach rechts

```
int a = 2*5 % 3 + 27/3;
// a ist 10, auch ohne Klammern!
```

Mathematische Funktionen der C-Standardbibliothek

- Einbinden der Mathe-Bibliothek über den Befehl
  - `#include <math.h>`
- aufrufen mathematischer Funktionen:

```
float pi = 3.1416;                  // ~ pi
float x1 = sqrt(2.0);               // Wurzel aus 2
float x2 = sin(45.0*pi/180.0);      // Sinus von 45°
float y = x1*x2*2;                  // y = ?
```

- Funktionen können in arithmetische Ausdrücke eingebaut werden
- Funktionen geben Werte entsprechend der Argumente zurück

Auszug: Mathematische Funktionen der C-Standardbibliothek (Beispiele)

Definition	
double cos(double x)	Kosinus von x
double sin(double x)	Sinus von x
double acos(double x)	Arkustangens von x
double asin(double x)	Arkussinus von x
double exp(double x)	e^x
double log(double x)	ln (x)
double log10(double x)	$\log_{10}(x)$
double ceil(double x)	Aufrunden zur nächsten Ganzzahl
double floor(double x)	Abrunden zur nächsten Ganzzahl
double pow(double y,double x)	y^x
double fabs(double x)	Betrag von x
double sqrt(double x)	Wurzel von x

Zinsrechner

- Sparkonto:
  - p = Zinssatz pro Jahr (in Prozent)
  - k0 = Spareinlage in €
  - n = Laufzeit in Jahren

  Wie groß ist die Geldeinlage „kN“ am Ende der Laufzeit?
- Formel:
  - kN = k0 * (1 + p/100.0)^n;
- C-Programmcode:

```
float p  = 4;
float k0 = 2500;
int   n  = 18;
float kN = k0 * pow(1 + p/100.0,n);
printf("kN = %f",kN);
```

Umwandlung elementarer Datentypen: Type-Casting

- verschiedene elementare Datentypen können durch arithmetischen Ausdrücke kombiniert werden
  - explizite Umwandlung durch Programmierer:
    - Programmierer legt Datentyp eines Ausdrucks fest
      - „Type-Casting“ mit Cast-Operator

```
int   i = 3;
float x = 2/(float)i;        // x ist 3
```

  - implizite Umwandlung durch Compiler
    - Compiler wandelt nach bestimmten Regeln Typen ineinander um

```
float pi = 3.1416;
int x = pi;     // x ist 3
int y = (1/sqrt(2)*1/sqrt(2))*100;   // y is ??
```

Implizite Umwandlung elementarer Datentypen in arithmetischen Ausdrücken

- Ziel: Datenverlust minimieren

Datentyp von X	Datentyp von Y	Datentyp des Ausdrucks X*Y bzw. Y*X
double	float	**double**
double	int	**double**
float	int	**float**
long	int	**long**

Vielen Dank!

- Bei Fragen einfach eine Mail an:
  - arvid@cs.uni-potsdam.de