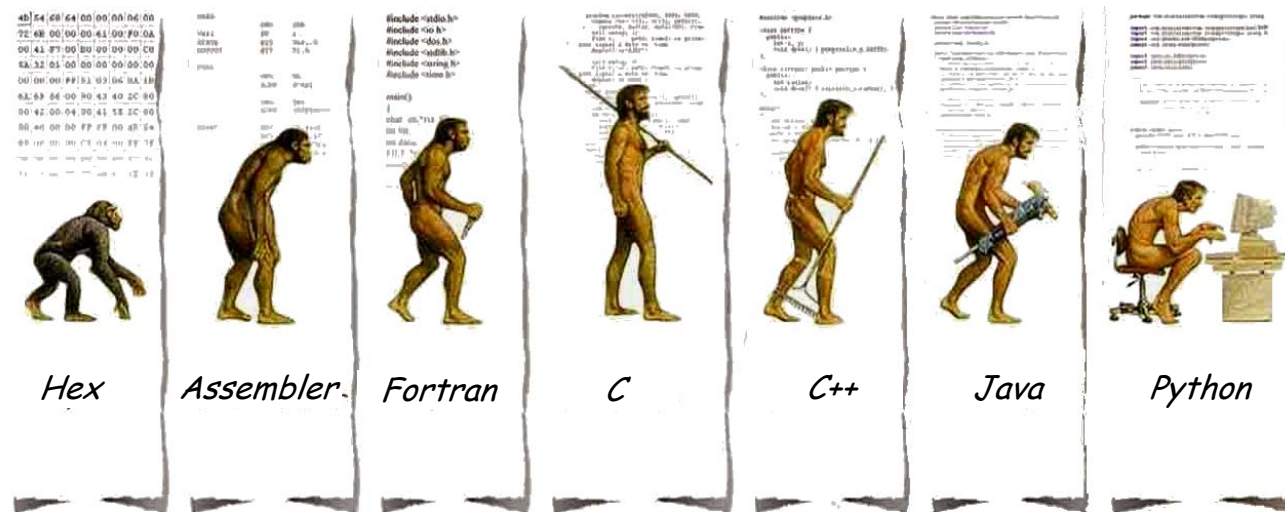


Die Programmiersprache C



- Geschichte von C
- Beispiel: Temperaturumrechnung
- Variablen, Zahlen, Ausdrücke
- Zahlensysteme
- Datentypen für Zahlen: int, float, double
- Programmierstil

Geschichte von C und Erweiterung

Was ist C ?

■ Prozedurale Programmiersprache (nicht objektorientiert)

- Keine Klassen sondern Funktionen (oder Prozeduren)
- Ein C-Programm besteht im Wesentlichen aus einer Sammlung von Funktionen

■ Compiler-Sprache (C, C++, Java, C#, Pascal, Fortran)

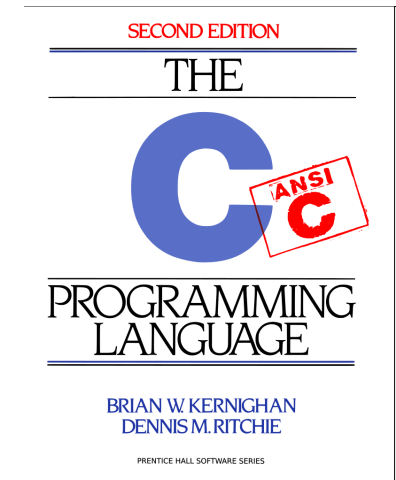
- Quelltext wird mit einem Text-Editor oder in der Entwicklungsumgebung erstellt
- Quelltext wird übersetzt (**Compiler**) -> Zwischenformat (".obj File")
 - *Zwischenformat ist standardisiert, so dass auch andere Sprachen (z.B. ASM) in dieses Format übersetzt werden können*
- Übersetzte Dateien werden zu ausführbarem Programm gebunden (**Linker**)
- Vorteil: der Compiler/Linker findet syntaktische und einige logische Fehler

■ Interpreter/Script-Sprachen (Ruby, Python, JavaScript)

- Übersetzung und Ausführung in einem Schritt
- Vorteil: sehr schnelle Entwicklungszyklen, portabel
- Nachteil: Fehler werden erst zur Laufzeit erkannt -> Testen wird sehr zeitaufwendig

Geschichte C

- 1967-1973 Entwicklung von C als Ersatz für Assembler in Unix Betriebssystem durch Brian Kernighan & Dennis Ritchie bei Bell Labs



- 1978 K&R-Standard
Quasi-Standard durch das Buch "The C Programming Language"

- 1988 ANSI C (C89)
Standardisierung durch das ANSI-Komitee (ein paar Änderungen gegenüber K&R)

- 1999 C99
Überarbeitete Version, von den meisten Compilern aber nur teilweise unterstützt (weitgehende Unterstützung im GNU C-Compiler)
Richard Stallmann



Was ist C++ ?

- Objektorientierte Erweiterung von C
- Ist "echte Obermenge" von C ("++" ist der Postincrement Operator)
 - jedes C Programm kann mit C++ Compiler übersetzt werden
 - Joke: Viele C++ Programmierer verwenden eine Untermenge von "C++" nämlich "C++ - -"
- Sowohl prozedurale als auch objektorientierte Programmierung möglich
- Methodenaufruf: `john->drive("Corvette", "Mary's House")`
- Hello World

```
#include <iostream>

int main() {
    std::cout << "Hello, world!\n";
}
```

Geschichte C++

- 1979 Start der Entwicklung als "C mit Klassen"
B. Stroustrup bei AT&T
- 1985 Erste freigegebene C++ Version
- 1987 GNU C++
- 1990 Herausgabe von "The Annotated C++ Reference Manual" (ARM)
- 1998 ANSI/ISO C++: Verabschiedung von C++ als ANSI/ISO-Standard
- 2003 C++03 Standardisierung von ISO/IEC 14882:2003
- 2011 C++11 ISO/IEC 14882:2011 Lambda Ausdrücke (Anonyme Funktionen)
- 2014 C++14 ISO/IEC 14882:2014 Erweitert const expr
- 2020 C++20 Module



Was ist Objective C ?

- Erweiterung von C
- Viele Anlehnungen an Smalltalk (erste OO Sprache überhaupt)
- Methodenaufruf: [john driveCar:@"Corvette" toDestination:@"Mary's House"]
 - <https://en.wikipedia.org/wiki/Objective-C>
- Hello World

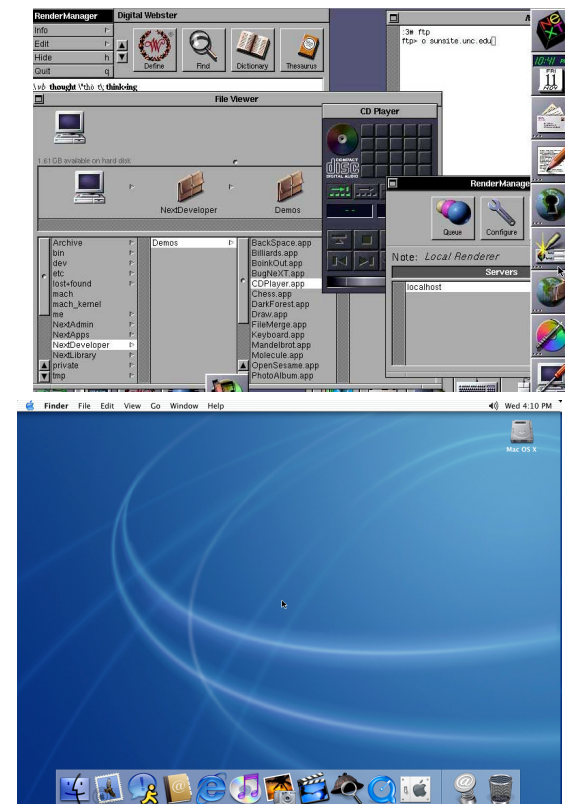
```
// First program example

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    NSLog (@"Hello, World!");
    [pool drain];
    return 0;
}
```

Geschichte Objective C

- 1981 Entwickelt durch Brad Cox als Erweiterung von C bei ITT
- 1988 NeXT Software (Steve Jobs) lizenziert Objective C für NeXTSTEP
- 1996 Apple kauft NeXT; S.J. wird CEO von Apple und verwendet es als Basis für OS X
- 2007 Basissprache für iOS (iPhone)
- 2014 Abgelöst durch Swift



Beispiel: Temperaturumrechnung

Beispiel

```
/**
 * Nach Anforderung einer Temperatur in Fahrenheit wird die Temperatur
 * in Celcius berechnet und ausgegeben.
 * @author Gerrit Burkert
 * @version 17-jul-2012
 */
#include <stdio.h>

int main (void) {
    float celcius, fahrenheit;

    // Eingabe in Fahrenheit
    printf("Temperatur in Fahrenheit eingeben: ");
    scanf("%f", &fahrenheit);

    // Ausgabe in Celcius
    celcius = 5.0f * (fahrenheit - 32.0f) / 9.0f;
    printf("Temperatur in Celcius: %f\n", celcius);
;
    return 0;
}
```

Kommentare

```
/*  
    Ein Kommentar, der sich über mehrere Zeilen erstrecken  
    kann  
*/  
  
// Dieser Kommentar wird durch das Zeilenende begrenzt
```

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

- Zu jedem Programm und zu jeder Funktion gehört ein beschreibender Kommentar
- Ausserdem werden alle Teile innerhalb von Funktionen, deren Zweck sich nicht unmittelbar erschliesst, mit Kommentaren beschrieben
- Werden bei der Kommentierung bestimmte Konventionen eingehalten, können die Kommentare zur Generierung einer Systemdokumentation verwendet werden
- Zu einer Variablenvereinbarung in jedem (nicht auf ein paar Zeilen beschränkten) Programm gehört ebenfalls ein beschreibender Kommentar
- Auf den Folien wird aus Platzgründen gelegentlich darauf verzichtet, im Programmcode sind die Kommentare aber erforderlich

Comment - Your program should read like an essay!

- A working but uncommented program is a time bomb waiting to explode



Programmdetails

- `celcius` und `fahrenheit` sind Variablen, die mit dem Datentyp `float` (Fließkommazahl) definiert werden
- Mit `printf` kann etwas auf der Konsole ausgegeben werden, mit `scanf` werden Werte eingelesen
- Werte vom Datentyp `float` werden mit angehängtem `f` angegeben, zum Beispiel `5.0f`

Variablen, Zahlen, Ausdrücke

- Im Speicher sind Bits abgelegt, zusammengefasst zu Bytes oder aus mehreren Bytes bestehenden Wörtern
- Es muss aber festgelegt sein, wie diese Bitfolgen zu interpretieren sind:
 - Sind es Texte oder Zahlen?
 - Wenn es Zahlen sind, wie viele Bytes bilden eine Zahl und wie sind diese zu interpretieren?
- Datentyp *int*: Anzahl Bytes (z.B. 4) bilden eine ganze Zahl
- Datentyp *float*: Anzahl Bytes (z.B. 4) bilden eine Fließkommazahl

Datentypen



2E 78 6D 6C
.
x m l
.xml

6C6D782E₁₆
1'819'113'518₁₀

6C6D782E₁₆
1.148331807 * 10²⁷

ASCII
Text

int

float

2F2368	00 14 00 06	00 08 00 00	00 21 00 32	3C BD 3E FB	00 00 00 E2	01 00 00 13	!!!!!!!2< > !!!
2F2380	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 5B 43 6F	6E 74 65 6E	!!!!!![Conten
2F2398	74 5F 54 79	70 65 73 5D	2E 78 6D 6C	50 4B 01 02	2D 00 14 00	06 00 08 00	t_Types].xmlPK
2F23B0	00 00 21 00	AA 8B 5D 00	03 00 00 00	0F 01 00 00	0B 00 00 00	00 00 00 00	!!!]% !!!
2F23C8	00 00 00 00	00 00 2C 01	00 00 5F 72	65 6C 73 2F	2E 72 65 6C	73 50 4B 01	!!!!!,!!!_rels/.relsPK
2F23E0	02 2D 00 14	00 06 00 08	00 00 00 21	00 57 C2 8A	D7 22 04 00	00 2E 0A 00	!W "!!!.f
2F23F8	00 10 00 00	00 00 00 00	00 00 00 00	00 00 00 28	02 00 00 64	72 73 2F 73	!!!!!!(\dors/s
2F2410	60 64 70 65	70 6D 6C 75	70 6D 6C 50	4B 04 02 2D	00 14 00 06	00 00 00 00	!!!!!!

- Variablen haben in der Programmierung eine andere Bedeutung als in der Mathematik
- Eine Variable in C repräsentiert eine *Speicherstelle*
- Diese Speicherstelle kann über den *Variablennamen* angesprochen werden
- Der Ausdruck

fahrenheit – 32

holt den Wert der Variablen *fahrenheit* aus dem Speicher und subtrahiert 32 von diesem Wert

Variablenvereinbarung

- Um eine Variable zu nutzen, muss sie zunächst *vereinbart* werden
- Im Beispiel verwenden wir zwei Variablen *fahrenheit* und *celcius*, die als Variablen vom Datentyp *float* eingeführt wurden:

```
float celcius, fahrenheit;
```

- Oder:

```
float celcius;  
float fahrenheit;
```

Zahlen, Zahlenliterale (Konstanten)

- Neben Variablen können Zahlen auch direkt in einem Programm verwendet werden:

```
5.0f * (fahrenheit - 32.0f) / 9.0f
```

- Im Gegensatz zu *Variablen* spricht man hier von *Literalen (unveränderlichen Werten)*
- Das angehängte **f** kennzeichnet diese Literale als Zahlen vom Typ *float*
- Zahl mit "." ohne f ist automatisch *double*
 - double ist der genauere Fließkommatentyp

Stringliterale

- Texte, die direkt in einem Programm vorkommen, werden als *Stringliterale* bezeichnet
- Stringliterale werden in C in Anführungszeichen "..." eingeschlossen
- Sie werden als Folge von ASCII-Zeichen im Speicher abgelegt
- Beispiel

```
"Temperatur in Fahrenheit eingeben: "
```

- Mit Hilfe von Operatoren zusammengefügte Elemente (zum Beispiel Variablen oder Literale) bilden *Ausdrücke*
- Solche Ausdrücke können berechnet (bestimmt) werden und liefern einen Wert
- Beispiel:

```
5.0f * (fahrenheit - 32.0f) / 9.0f
```

- Wenn der Wert der Variablen *fahrenheit* bekannt ist, kann der Ausdruck berechnet werden

Anweisungen

- Auszuführende Befehle nennt man *Anweisungen*
- Sie werden in C mit einem Semikolon abgeschlossen
- Beispiele
- Der Wert des Ausdrucks wird berechnet und das Ergebnis der Variablen *celcius* zugewiesen:

```
celcius = 5.0f * (fahrenheit - 32.0f) / 9.0f;
```

- Das Stringliteral wird auf der Konsole ausgegeben:

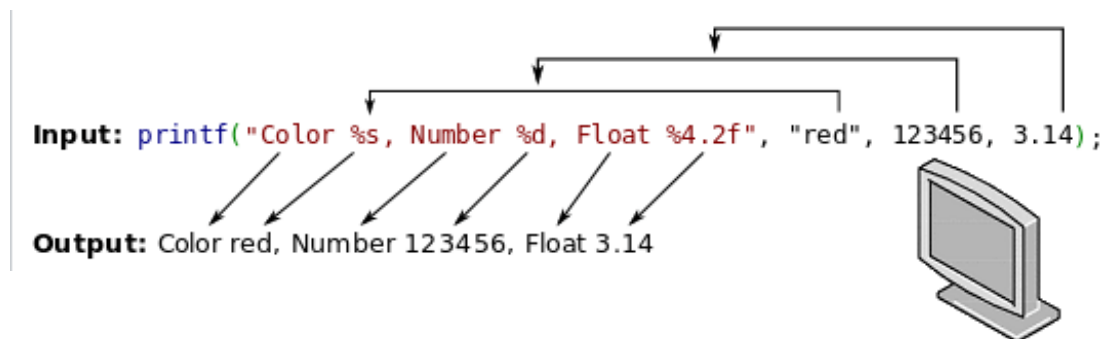
```
printf("Temperatur in Fahrenheit eingeben: ");
```

Anweisung printf

■ Argumente und Funktionsweise

- Zeichenkette mit Text und Formatierungshinweisen
- einem oder mehreren Werten.
- Die Zeichenkette wird ausgegeben und dabei die Werte in der entsprechenden Formatierung (Schreibweise) eingefügt.

■ Die Ausgabe auf der Standardausgabe (Kommandozeile)



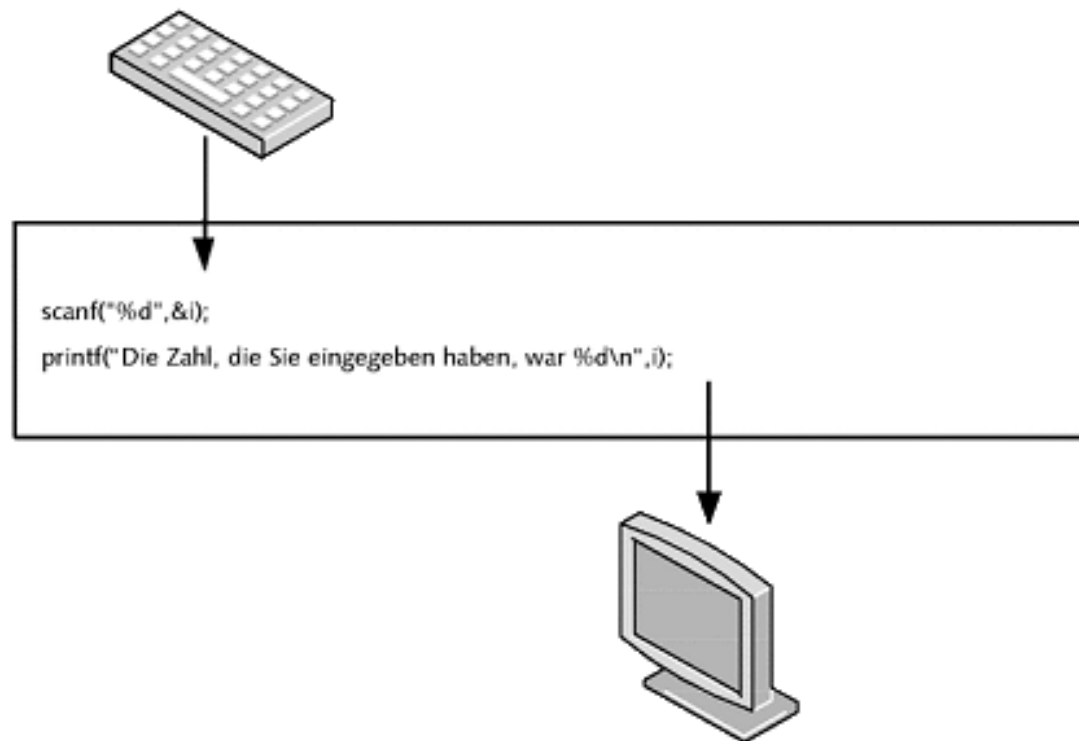
Typ	Buchstaben
int	%d
long	%ld
float/double	%f
char	%c
char* (string)	%s
(void *)	%p
hexadezimal	%x

- Ausgabe wird gebuffert (bis Zeilenumbruch); mit `fflush(stdout);` kann die Ausgabe forciert werden

Anweisung scanf

■ Argumente und Funktionsweise

- Zeichenkette mit Text und Formatierungshinweisen
- einer oder mehreren Variablen mit & vorangestellt (oder Pointer - später)
- die Werte werden von der Konsole eingelesen und in die Variablen gespeichert



Typ	Buchstaben
int	%d
long	%ld
float	%f
double	%lf
char	%c
char* (string)	%s

Ganz- Zahlensysteme

Ganze Zahlen: int

- Ein möglicher Ganzzahltyp
- Ein Ganzzahlliteral, das nur aus den Ziffern 0..9 und optional einem Vorzeichen besteht, wird als Zahl vom Typ *int* aufgefasst
- Beispiele: 5731, 0, -100
- In C ist nicht fest definiert, wie viele Bit eine Zahl vom Typ *int* umfasst
- Häufig umfasst ein *int* 4 Bytes (32 Bit)
Wertebereich in diesem Fall: -2'147'483'648..2'147'483'647
- Ganze Zahlen sind über den ganzen Wertebereich gleichmässig verteilt

Positive und negative Zahlen

- Vorzeichenbehaftete ganze Zahlen werden im **Zweierkomplement-Format** gespeichert
- Die Regel zur Bildung des 2er-Komplements lautet:
 - Alle Bits der entsprechenden positiven Zahl werden zuerst **invertiert**, d.h. aus einem 1 Bit wird ein 0 Bit und umgekehrt.
 - Zu der so erhaltenen 'Zahl' wird der **Wert 1 hinzuaddiert**. Ein eventuell auftretender Überlauf wird verworfen
- Vorteil: dieselbe HW funktioniert mit und ohne Vorzeichen
- Was passiert wenn Sie 0111 1111 (127) eine 1 addieren (1 Byte Arithmetik)?

Binäre Darstellung der *char*-Zahl 2:

2 ist in binär	0000 0010	
alle Bits invertieren	1111 1101	
und 1 hinzuaddieren	1111 1110	(entspricht -2)

Überprüfung durch Addition $2 + (-2)$

2 ist binär	0000 0010
-2 ist binär	1111 1110
Überträge	1] 1111 1100
Ergebnis	1] 0000 0000

Als weitere Methode kann man die Zahl, wenn sie negativ ist, einfach von der Zahl direkt jenseits des Wertebereichs subtrahieren. Beispielsweise überdecken vorzeichenlose 8-Bit-Zahlen den Wertebereich 0–255, die direkt folgende Zahl ist die 256. Eine –1 muss man nur von 256 abziehen und erhält den Wert 255 (= $1111\ 1111_2$), wie gewünscht. Analog führt eine –128 zum Wert 128.

Datentypen für Zahlen: float, double

Fließkommazahlen: float - "halbe Genauigkeit"

- Ein möglicher Fließkommatyp
- Ein Fließkommaliteral vom Typ *float* kann einen Dezimalpunkt sowie eine mit e (oder E) angehängte Zehnerpotenz enthalten, ein abschliessendes f (oder F) dient zur Kennzeichnung als *float*
- Beispiele: 0.5f, 345.12e23f, -0.000126f, 5e-9f
- In C ist es im Prinzip nicht fest definiert, wie viele Bit eine Zahl vom Typ *float* umfasst
- I.d.R. umfasst ein *float* jedoch 4 Bytes (32 Bit)
Wertebereich in diesem Fall: ca. -1.5E-45..3.4E38
Genauigkeit: 7-8 Stellen

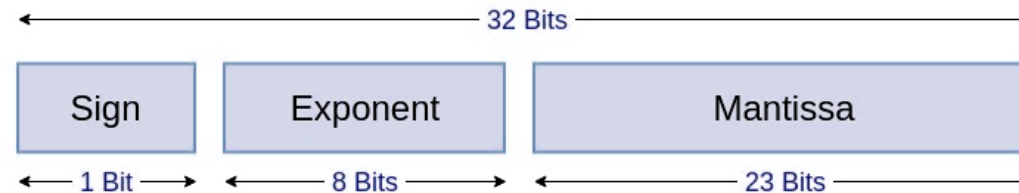
Fließkommazahlen: double

- Der **bessere** Fließskommatyp
- Ein Fließkommaliteral vom Typ *double* kann einen Dezimalpunkt sowie eine mit e (oder E) angehängte Zehnerpotenz enthalten
- Beispiele: 0.5, 345.12e23, -0.000126, 5e-9
- I.d.R. umfasst ein *double* 8 Bytes (64 Bit)
Wertebereich in diesem Fall: ca. +/-4,9E-324 ... +/-1,7E+308
Genauigkeit: 15 Stellen.
- Hinweis: bei scanf muss "lf" als Muster angegeben werden

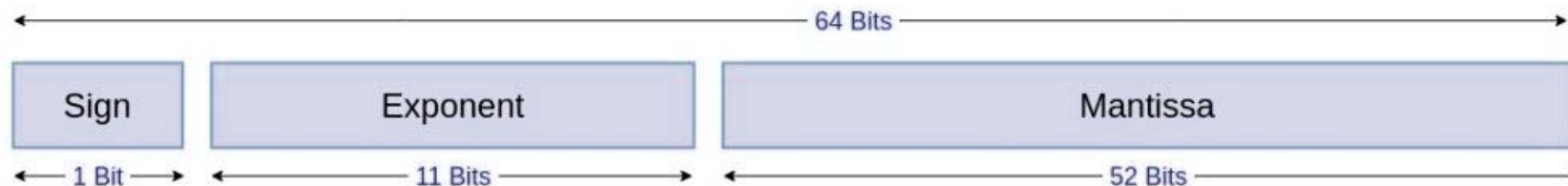
Wenn es nicht **sehr** gute Gründe gibt, *float* zu nehmen, dann lieber *double* verwenden

Fließkommazahlen: double

■ Darstellung im Speicher gemäss IEEE 754



Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

- Fließkommazahlen werden in der Regel so normalisiert, dass sie als eine Eins mit einer Folge von Nachkommastellen multipliziert mit einer Zweierpotenz dargestellt werden (die erste Eins kann dann i.d.R. weggelassen werden)

- Beispiel:

25.88

-> 1.10011110000101000111101 * 2⁴

= 1 + 1*2⁻¹ + 0*2⁻² + 0*2⁻³ + 1 * 2⁻⁴ + ...

- Allgemein:
$$\left(1 + \sum_{n=1}^{p-1} bit_n \times 2^{-n} \right) \times 2^{exp}$$

- Weitere Informationen:

https://de.wikipedia.org/wiki/Doppelte_Genauigkeit

<https://de.wikipedia.org/wiki/Gleitkommazahl>

Fliesskommazahlen

- Damit ist klar, dass Fliesskommazahlen, die dezimal genau angegeben werden, binär **meist nur angenähert** werden können

- Beispiel: 0.5 lässt sich exakt darstellen

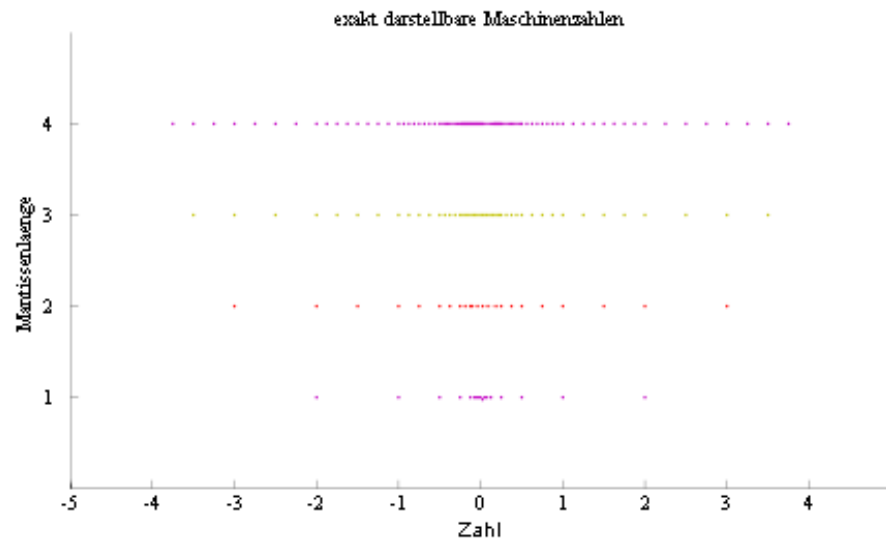
- $0.5 = 1.000000000000000000000000 * 2^{-1}$

- Beispiel: 0.3 lässt sich **nicht exakt** darstellen

- 0.3
-> $1.00110011001100110011010 * 2^{-2}$
-> 0.3000000119

Fliesskommazahlen - Verteilung

- Der Datentyp *double* kann zwischen 0 und 1 eine grosse Menge von Zahlen darstellen
- Ebenso lassen sich mit *double* sehr viel grössere Zahlen als mit *int* darstellen
- Fliesskommazahlen sind nicht gleichmässig auf dem Zahlenstrahl verteilt.
- Sie sind exakter bei kleinen Werten und liegen weiter auseinander bei grossen Werten



- Rechnungen mit **ganzen Zahlen** sind genau (so lange kein Überlauf aus dem Wertebereich erfolgt)

- Überlauf wird in C nicht überprüft, sondern führt bei ganzen Zahlen zur Umkehr des Vorzeichens -> hat schon mehrmals zu Katastrophen geführt



- Rechnungen mit **Fliesskommazahlen** sind in der Regel mit Ungenauigkeit verbunden

- Es ist nicht sinnvoll, Fließkommazahlen auf Gleichheit zu überprüfen:
Der Ausdruck $0.2 + 0.1$ hat i.d.R. **nicht** genau den Wert 0.3
- Stattdessen: $\text{EPS} = 0.000001$ (d.h. genügend kleiner Wert - abhängig von den Anforderungen) definieren und $|\text{Wert} - 0.3| < \text{EPS}$ vergleichen



Programmierstil

Ein Programm muss gut lesbar und verständlich sein, sonst ist es praktisch wertlos

- sprechende **Variablennamen** und **Funktionsnamen** wählen (object, item, thing, foo, bar sind tabu)
- **Kommentare**, um die Verständlichkeit zu verbessern
- Konsequente und konsistente **Einrückung** - nicht mehr so wichtig da es "Beautififier" gibt
e.g. (*Source*-> *Format* in NetBeans)

Programmierstil - Einrückungsregel 1

- Das "{" auf derselben Zeile wie die Funktion
- Funktionen und Includes ohne Einrückung
- die Anweisungen selber sind durch ein tab eingerückt
- Das "}" auf einer neuen Zeile nicht eingerückt

```
#include <iostream>

int main() {
    std::cout << "Hello, world!\n";
}
```

Abschreckendes Gegenbeispiel

```
int m=1711276033,N=1,t[1<<25]={2},a,*p,i,e=39717691,s,c,U=1;g(d,h){for(i=s;i<1<<24;i*=2)d=d*1LL*d%m;for(p=t;p<t+N;p+=s)for(i=s,c=1;i;i--)a=p[s]*(h?c:1LL)%m,p[s]=(m*1U+*p-a)*(h?1LL:c)%m,*p=(a*1U+*p)%m,p++,c=c*1LL*d%m;}main(){while(e/=2){N*=2;U=U*1LL*(m+1)/2%m;for(s=N;s/=2;)g(40,0);for(p=t;p<t+N;p++)*p=*p*1LL**p%m*U%m;for(s=1;s<N;s*=2)g(983983719,1);for(a=0,p=t;p<t+N;)a+=*p<<(e&1),*p++=a%10,a/=10;}while(!*--p);for(t[0]--;p>=t;)putchar(48+*p--);}
```

- Das obige C Programm kann übersetzt werden, es **läuft** auch (liefert die Primzahl: $2^{57885161}-1$) **es ist aber kaum lesbar!**
- Sobald ein Programm über eine triviale Grösse hinausgeht, wird man immer wieder bestehenden Code lesen und verstehen müssen
- Zentrales Problem dann: Überblick nicht zu verlieren
- Ohne sauber strukturierten Code ist das hoffnungslos !
- OO ist ein gutes Mittel um umfangreichen Code zu strukturieren - später

■ The amount of time spent on maintenance is skyrocketing

- A Program is read 10-100x more often than written

■ Clean Code

- <https://dzone.com/articles/clean-code-robert-c-martins-way>
- Meaningful Names
- Don't be clever - Make your program as clear, concise, and simple as possible
- Don't optimize to early



■ 12 Practices That Make You a Good Programmer

- 1. As computer scientist Niklaus Wirth said in his well-known statement: "Data Structures + Algorithms = Programs"
- 2. Studying Algorithms
- 3. Studying Design Patterns
- 4. Using Unit Tests
- ...

Noch Fragen?



Übersetzen und Ausführen - Kommandozeile

```
c:> gcc -o temperatur temperatur.c  
c:> temperatur  
Temperatur in Fahrenheit eingeben: 89.0  
Temperatur in Celcius: 31.666666  
c:>
```