Vom Problem zum Programm

- Programmentwicklung I
  - Daten- und Ablaufstrukturierung
- Einfache Standard-Datentypen
  - Integer – Real – Character – Aufzählungstyp

Programmentwicklung

Aus einem Kochbuch das folgende Rezept zur Herstellung eines so genannten Pharisäers:

Zutaten:

- ½ l heißer Kaffee
- ¼ l Sahne
- 2 Essl. Zucker
- 4 Schnapsgläser 54%iger Rum (8cl)

Zubereitung:

Den Kaffee aufbrühen und warm halten. 4 Tassen mit heißem Wasser vorwärmen. Inzwischen die Sahne steif schlagen. Das Wasser aus den Tassen gießen., die Tassen abtrocknen und in jede Tasse 1-2 Teelöffel Zucker geben. Je 1 Schnapsglas Rum darüber gießen und mit dem Kaffee auffüllen. Die Schlagsahne als Haube auf jeder Tasse Pharisäer setzen.

Programmentwicklung

- Das Rezept gliedert sich in zwei Teile.
- Im 1. Teil werden die erforderlichen Zutaten genannt.
- Im 2. Teil wird ein Verfahren beschrieben, nach dem man aus den Zutaten das gewünschte Getränk herstellen kann.
- Beide Teile sind grundverschieden, gehören aber untrennbar zusammen.
  - Ohne Zutaten ist keine Zubereitung möglich
  - Ohne Zubereitung bleiben die Zutaten ungenießbar.
- Man beachte, die verwendete Fachsprache des Autors (Essl., cl., Sahne steif schlagen, aufbrühen)
  - Ohne Fachsprache wäre die Anleitung ausführlicher, umfangreicher und sogar missverständlich.
- Die Verwendung setzt voraus, dass sich Autor und Leser zuvor auf eine gemeinsame Terminologie geeinigt haben.

Übertrag in die Datenverarbeitung

- Die Zutaten sind die Daten bzw. **Datenstrukturen**, die wir verarbeiten wollen.
- Die Zubereitungsvorschrift ist ein **Algorithmus**, der festlegt, wie die Daten zu verarbeiten sind.
- Das Rezept ist ein **Programm**, das alle Datenstrukturen und Algorithmen zum Lösen der gestellten Aufgabe enthält.
- Die gemeinsame Terminologie, in der sich Autor und Leser verständigen, ist eine **Programmiersprache**, in der das Programm geschrieben ist.
  - Die Programmiersprache muss dabei in der Lage sein, alle bzgl. Zutaten und Zubereitung bedeutsamen Informationen zweifelsfrei zu übermitteln.

Übertrag in die Datenverarbeitung

- Die Küche ist die technische Infrastruktur zur Umsetzung von Rezepten in schmackhafte Gerichte und ist vergleichbar mit einem **Computer**; seinem **Betriebssystem** und den benötigten **Entwicklungswerkzeugen**.
- Der Koch übersetzt das Rezept in einzelne Arbeitsschritte in der Küche. Üblicherweise geht ein Koch in zwei Schritten vor. Im 1. Schritt bereitet er die Zutaten einzeln und unabhängig voneinander vor, um die Einzelteile dann in einem 2. Schritt zusammenzufügen und abzuschmecken. In der Datenverarbeitung sprechen wir in diesem Zusammenhang von **Compiler** und **Linker.**
- Das fertige Gericht ist das **lauffähige Programm**, das vom Benutzer angewandt werden kann.

Voraussetzungen

- Der Programmierer muss die Programmiersprache beherrschen.
- Er muss einen Überblick über die üblicherweise verwendeten Datenstrukturen, deren Eigenschaften und Verarbeitungs-möglichkeiten haben.
- Er muss einen Vorrat an Standard-Verarbeitungsverfahren abrufbereit im Kopf haben.
- Er muss wissen, welche Datenstrukturen mit welchen Verfahren harmonisieren und welche nicht.
- Er muss wissen, was üblicherweise in einem Computer an Hilfsmitteln vorhanden ist und wie bzw. wozu diese Hilfsmittel verwendet werden.

Voraussetzungen

- Bei anspruchsvolleren Programmen muss er wissen, in welcher Reihenfolge und mit welchem Timing die Einzelteile zu verarbeiten sind und wie die einzelnen Aufgaben verteilt werden müssen, damit das gewünschte Ergebnis präsentiert werden kann.
- Er bzw. der "Programm-Entwerfer" muss auch wissen, worauf ein potentieller, späterer Anwender Wert legt und worauf nicht. Dies ist wichtig, wenn Programme für eine ganz bestimmte Problemstellung erstellt werden.

Einführung in die Programmierung

- Damit ein Problem durch ein Softwaresystem gelöst werden kann, muss es zunächst einmal erkannt, abgegrenzt und adäquat beschreiben werden.

  => Benutzeranforderung = User Requirement Specification

  => Systemanalyse = Design Spezifikation = Functional Req. Spec.

  => Systementwurf = Feindesign Spezifikation = Func. Req. Spec.

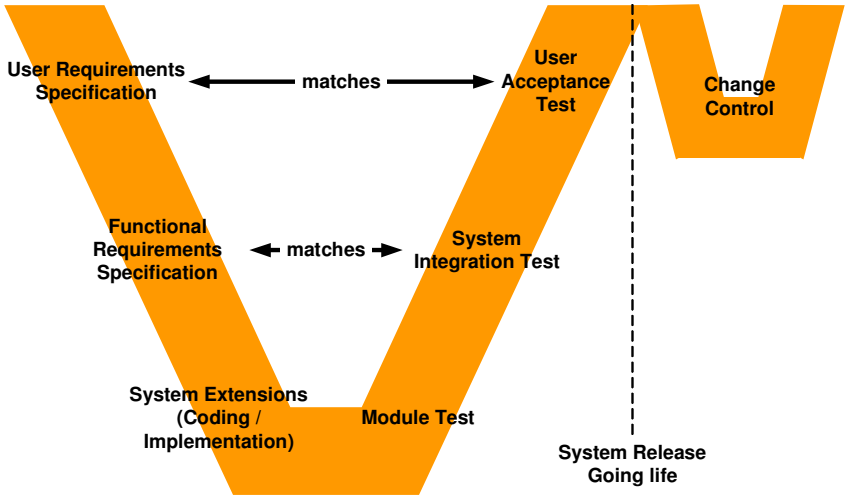
  => Realisierung, Implementierung = System Extension = Coding

  => Test, Qualitätssicherung (ISO 9000)

- Vorgehensmodell = V-Modell = Validierungsmodell

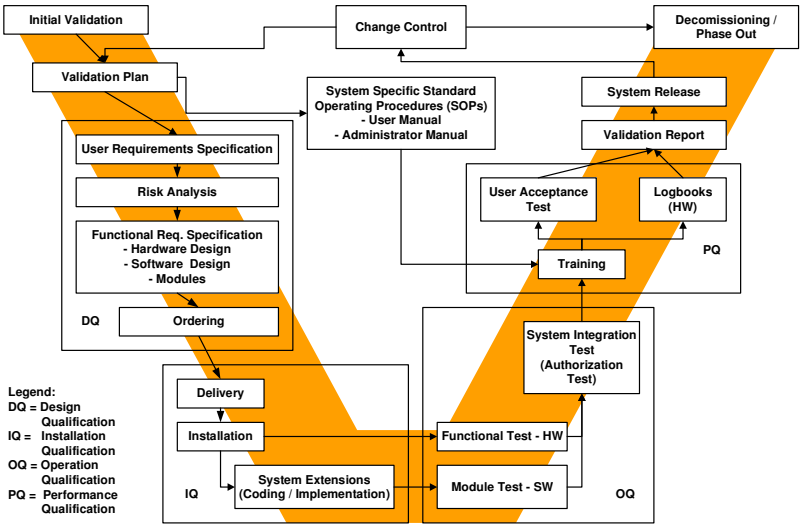
Vorgehensmodell (I)

System Development Life Cycle Model - simplified



Vorgehensmodell (II)

System Development Life Cycle Model



Programme: Analyse, Entwurf, Codierung

- Ein Programm soll in der Regel ein Problem lösen oder eine Aufgabe erfüllen. Als erstes muss darum das Problem oder die Aufgabe analysiert werden. Als nächstes überlegt man sich, wie die Aufgabe gelöst werden soll: man entwirft ein allgemeines Vorgehen zur Lösung und ein Programmkonzept. Schließlich muss noch der Quellcode geschrieben werden: das Programm wird codiert, oder wie man auch sagt implementiert.

- Betrachten wir als Beispiel die Umwandlung von Temperaturwerten von Grad Fahrenheit in Grad Celsius.

Analyse

- Am Anfang der Analyse wird die Problemstellung präzise definiert:
- Das Programm soll eine ganze oder gebrochene negative oder positive Zahl einlesen, sie als Gradangabe in Fahrenheit interpretieren und den eingegebenen Wert und den entsprechenden Wert in Celsius ausgeben.
- Als nächstes macht man sich mit dem Problemfeld vertraut, indem man sich die zur Lösung notwendigen Informationen besorgt. In unserem Fall sagt uns ein elementares Physikbuch wie Fahrenheit in Celsius umgewandelt wird:
  tCelsius = (tFahrenheit – 32) · 5 / 9

Entwurf

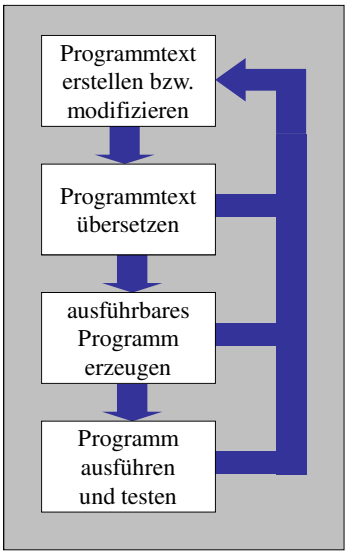
- Nachdem in der Analyse festgestellt wurde, was zu tun ist und die zur Lösung notwendigen Informationen besorgt sind, kann man sich jetzt dem wie zuwenden: dem Entwurf.
- Im Entwurf wird festgelegt, wie das Programm seine Aufgabe konkret lösen soll. Bei einer Berechnungsaufgabe wie in unserem Beispiel wird hier der Algorithmus festgelegt: Algorithmus zur Umwandlung von Fahrenheit in Grad:

  1. Grad–Fahrenheit in Variable fahrenheit einlesen
  2. celsius mit dem Wert (Wert(fahrenheit) – 32) · 5 /9 belegen
  3. Wert von fahrenheit und celsius ausgeben

Codierung

- Bei der Codierung (Implementierung) wird der Algorithmus aus dem Entwurf mit einem Editor in der korrekten C++–Syntax niedergeschrieben, mit einem Compiler übersetzt und mit einem Linker zu einem ausführbaren Programm gebunden.

Programmierumgebung



- Unterstützung des Programmierers durch
  - **Editor (vi, notepad, …)**
    - Header-Dateien (source.h)
    - Quellcode-Dateien (source.cpp, source.cc, source.c)
  - **Compiler (g++, cc, …),**
    - Objectcode (source.obj, source.o)
  - **Linker (g++, cc, …),**
    - Ausführbares Programm (source, source.exe)
  - **Debugger (gdb, dbx, …)**

Editor (to edit = einen Text erstellen oder überarbeiten)

- Programm wird in einer Textdatei erstellt und abgespeichert (z.B. Notepad). Der Programmtext (Quell- oder Sourcecode) wird mit einem so genannten Editor erstellt.
- Meistens Bestandteil einer Entwicklungsumgebung.

Compiler (to compile = zusammenstellen)

- Ein Programm in einer höheren Programmiersprache ist auf einem Rechner nicht unmittelbar lauffähig. Es muss durch einen Compiler in die Maschinensprache übersetzt werden.
- Der Compiler übersetzt Sourcecode (die *.cpp Dateien) in den so genannten Objectcode (*.obj Dateien).
- Er führt dabei verschiedene Prüfungen über die Korrektheit des übergebenen Quellcodes durch.
- Alle Verstöße gegen die Regeln der Programmiersprache werden durch gezielte Fehlermeldungen unter Angabe der Zeile angezeigt.
- Nur ein vollständig fehlerfreies Programm kann in Objectcode übersetzt werden.

Compiler (to compile = zusammenstellen)

- Viele Compiler mahnen auch formal korrekte, aber möglicherweise problematische Anweisungen durch Warnungen an (float ⇔double).
- Bei der Fehlerbeseitigung in der Reihenfolge des Auftretens vorgehen (Folgefehler).
- Der vom Compiler erzeugte Objectcode ist noch nicht das ausführbare Programm, da Programme in der Regel aus mehreren Sourcefiles, deren Objectfiles noch in geeigneter Weise kombiniert werden müssen.

Linker (to link = verbinden)

- Der Linker verbindet die einzelnen Objectfiles zu einem fertigen Programm.
- Der Linker nimmt dabei die noch ausstehenden Prüfungen, wobei auch noch Fehler auftreten können.
  - z.B. kann der Linker feststellen, dass in einem Sourcefile versucht wird, eine Funktion zu verwenden, die es nirgendwo gibt.
- Der Linker erstellt das ausführbare Programm.

Debugger (to debug = entwanzen)

- Der Debugger dient zum Testen von Programmen.
- Mit dem Debugger können die erstellten Programme bei ihrer Ausführung beobachtet werden.
- Durch manuelles Ändern von Variablenwerten kann das laufende Programm verändert werden.
- Der Debugger ist nicht nur zur Lokalisierung von Programmierfehlern, sondern auch zur Analyse eines Programms durch Nachvollzug des Programmablaufs hilfreich.

Form eines C-Programms

- Schlüsselwörter in ANSI-C
  - Basisdatentypen — char, int, float, double, void
  - Ergänzungen dazu — short, long, signed, unsigned
  - Datentypen — enum, struct, union, typedef
  - Speicherklassen — auto, extern, register, static
  - Qualifizierer — const, volatile
  - Kontrollstrukturen — if, else, while, for, do, switch, case, default
  - Sprunganweisungen — break, continue, return, goto
  - Operatoren — sizeof, new, delete, operator, this
- Erweiterung für C++
  - Basisdatentyp — bool
  - Klasseneinbindung — string

Form eines C-Programms

- Es gibt folgende Basisdatentypen
  - Zeichen — char(acter)
  - Ganze Zahlen — int(eger)
  - Gleitkommazahlen — float
  - Gleitkommazahlen doppelte Genauigkeit — double
  - Ohne Wert — void
  - Logischer Wert (true, false) — bool
- Größe und Bereich dieser Datentypen können abhängig vom Prozessor variieren.
- Allerdings ist ein char immer 1 Byte groß.
- Durch Modifier (short, long, signed, unsigned) können die Grundtypen angepasst werden.

Allgemeine Form eines C-Programms

- Alle C-Programme bestehen aus einer oder mehreren Funktionen (Prozedur => prozedural).
- Die einzige Funktion, die vorhanden sein muss, heißt main().
- Sie ist die erste Funktion, die beim Start aufgerufen wird.

Ausgewählte Sprachelemente von C

- Programmrahmen

```
// Programm Umwandlung
#include <iostream>
using namespace std;

int main(parameter)
{
  . . .
  . . .
  . . .

  . . .
  . . .
  . . .
  system ("pause");
  return 1;
}
```

Am Anfang stehen die **Variablendefinitionen.**

Dann folgt der **Programmcode.**

Erstes Beispiel: Hello_World

```
/* Programm Hello World */
#include <iostream>
using namespace std;

int main()
{
  printf("eigenes hello world test\n");
  cout << "noch eine Ausgabe" <<endl;
  return 0;
}
```

- Es wird von Library Funktionen Gebrauch gemacht, die beim Linken der kompilierten Sourcen/Quellen dazu gebunden werden.

Syntaktische Fehler
Verstöße gegen die Regeln der Programmiersprache

- In der Regel wird man es nicht schaffen, ein Programm gleich beim ersten Versuch korrekt einzutippen. Bei Programmen, in denen nicht penibel alle Regeln der Sprache C++ eingehalten werden, verweigert der Compiler die Übersetzung.
- Vergisst man auch nur ein Komma, oder schreibt ein Komma an eine Stelle, an der ein Semikolon stehen muss, wird das Programm nicht vom Compiler übersetzt. Statt dessen reagiert er mit einer Fehlermeldung.
- Nehmen wir an, dass die sechste Zeile in unserem kleinen Beispiel nicht – wie es richtig wäre – mit einem Semikolon, sondern mit einem Komma beendet wird:

Syntaktischer-Fehler am Beispiel Hello_World

```
/* Programm Hello World */
#include <iostream>
using namespace std;

int main()
{
  printf("eigenes hello world test\n").
  cout << "noch eine Ausgabe" << endl;
  return 0;
}
```

- Dann kommt prompt die Quittung in Form einer Fehlermeldung:
  `Links von ".cout" muss sich eine Klasse/Struktur/Union befinden.`

Syntaktischer-Fehler am Beispiel Hello_World

- Die wesentliche Information dieser Fehlermeldung ist, dass der Compiler vor der geschweiften Klammer in Zeile 7 des Programms Probleme hat, die er parse error nennt.
- Damit meint er, dass die Analyse des Textes (engl. To parse = zergliedern, grammatikalisch analysieren) gezeigt hat, dass der Text nicht den syntaktischen Regeln der Sprache C++ entspricht. Er enthält einen Syntaxfehler.

Semantische Fehler
Inhaltliche Fehler in formal korrekten Programmen

- Manchmal wird der von uns eingegebene Quellcode vom Compiler ohne Fehlermeldung übersetzt, aber das erzeugte Programm verhält sich nicht so wie erwartet.
- Beispielsweise wollen wir, dass “hello” ausgegeben wird, tatsächlich erscheint aber “hellp” auf dem Bildschirm.
- Man sagt, das Programm enthält einen semantischen (inhaltlichen) Fehler. Es tut etwas, aber nicht das was wir von ihm erwarten. In diesem Fall ist die Ursache schnell festgestellt. Wir haben uns sicher einfach nur vertippt und das Programm sieht folgendermaßen aus:

Semantischer-Fehler am Beispiel Hello_World

```
/* Programm Hello World */
#include <iostream>
using namespace std;

int main()
{
  printf("eigenes hellp world test\n");
  cout << "noch eine Ausgabe" << endl;
  return 0;
}
```

Semantischer-Fehler am Beispiel Hello_World

- Im Gegensatz zu vorhin ist hier das Programm auch mit dem Tippfehler syntaktisch korrekt und kann übersetzt werden. Der Compiler kann ja nicht wissen, was wir eigentlich ausgeben wollten. Er hält sich an das was im Quellprogramm steht.
- Leider sind die semantischen Fehler meist nicht so trivial, wie in diesem Beispiel. Ein semantischer Fehler wird in der Regel nicht durch Tippfehler, sondern durch “Denkfehler” verursacht.

Semantischer-Fehler am Beispiel Hello_World

- Man glaubt das Programm sei korrekt und mache das, was es machen soll, in Wirklichkeit macht es zwar etwas, aber nicht das, was wir wollen, sondern das was wir aufgeschrieben haben und von dem wir nur dachten, es sei das, was wir wollen.
- Semantische Fehler sind die ernsten, die richtigen Fehler. Man lässt darum meist die Kennzeichnung “semantisch” weg und spricht einfach von “Fehler”. Nur wenn ausdrücklich betont werden soll, dass es sich um etwas so triviales wie einen syntaktischen Fehler handelt, spricht man von einem “Syntax–Fehler”.

Beispiel Maßeinheiten-Umwandlung

```
// Programm Maßeinheiten-Umwandlung
#include <iostream>
using namespace std;

int main ()
{
  double fahrenheit; // Variable fuer Fahrenheit-Werte
  double celsius;    // Variable fuer Celsius-Werte
  // Grad in Fahrenheit einlesen:
  cout << "Bitte Grad in F : ";
  cin >> fahrenheit;
  // Grad in Celsius berechnen:
  celsius = ((fahrenheit - 32) * 5) / 9;
  // Grad in F. und C. ausgeben:
  cout << fahrenheit << " Grad Fahrenheit ist ";
  cout << celsius << " Grad Celsius " << endl;
  system("pause");
  return 0;
}
```


Erläuterung

- Das Programm beginnt mit der Zeile
  `#include <iostream>`
  Dies ist eine Anweisung an den Präprozessor, an dieser Stelle den Inhalt des Textfiles iostream einzufügen. Es enthält Deklarationen aller I/O Funktionen.
- Mit der Zeile
  `using namespace std;`
  wird der Zugriff auf die I/O Funktionen erleichtert. Man könnte auch weiter unten statt der kompakten cout-Zeilen folgendes schreiben.
  `std::cout << std::endl << "Zahl...";`

Erläuterung

- Dem folgt die so genannte main-Funktion, die in jedem Programm als Startpunkt vorhanden sein muss:

```
int main ()
{ . . .
  return 0;
}
```

In ihrem Codeblock stehen die Anweisungen, die ausgeführt werden sollen. Dabei ist alles hinter // ein Kommentar. Kommentare werden vom Compiler ignoriert. Sie dienen lediglich dem Verständnis menschlicher Leser.

Erläuterung

- Mit der Zeile

```
system("pause");
```

wird das Programm vor dem Ende angehalten. Weiter geht es nur nach Eingabe einer beliebigen Taste.
Hinweis auf dem Bildschirm:

```
Drücken Sie eine beliebige Taste …
```

bzw.

```
Press any key …
```

Dies wird benötigt, falls das Programm im Debug-Modus ausgeführt wird.

Erläuterung

- Die main–Funktion beginnt mit Variablendeklarationen:
  ```
  double fahrenheit;
  double celsius;
  ```
  Hier werden zwei Variablen für Fließkommazahlen angelegt (definiert). Variablen sind Speicherplätze für Daten.

- In C/C++ werden die Namen von Variablen, Funktionen, Labeln und anderen benutzerdefinierten Objekten Bezeichner genannt. Das erste Zeichen des Bezeichners muss ein Buchstabe sein.

Erläuterung

- Die Ausgabeanweisung
  ```
  cout << "Bitte Grad in Fahrenheit";
  ```
  gibt den Text (die Zeichen innerhalb der Anführungszeichen) aus und die Eingabeanweisung
  ```
  cin >> fahrenheit;
  ```
  liest einen Wert in die folgende Variable (hier fahrenheit) ein.

- Mit der Zuweisung
  ```
  celsius = (( fahrenheit – 32 ) * 5 ) / 9;
  ```
  wird aus dem Wert in fahrenheit der gesuchte Wert berechnet und in celsisus abgelegt.

Zahlen

- Natürlich benötigen Programme auch konkrete Zahlenwerte.
- Man unterscheidet zwischen
  - ganzen Zahlen:
    - 123, -4711
  - Gleitkommazahlen
    - 1.234, -4.711, 12e3, 47e-11
- Wichtig: Bei Gleit**komma**zahlen wird der englischen Schreibweise entsprechend ein Dezimal**punkt** verwendet.

Variable im Computer

- In einem Computerprogramm verwendet man Variablen, um sich die Zwischen- und Endergebnisse von Berechnungen zu merken, um sie an anderer Stelle erneut verwenden zu können.
- Unter Variablen versteht man einen mit einem Namen versehenen Speicherbereich, in dem Daten eines bestimmten Typs hinterlegt werden können. Das im Speicherbereich der Variablen hinterlegte Datum bezeichnet man als Wert der Variablen.
- Zu einer Variablen gehören also:
  - ein Name,
  - ein Typ,
  - ein Speicherbereich und
  - ein Wert.
- Der Name dient dazu, die Variable im Programm eindeutig ansprechen zu können.
- Denkbare Typen sind die 5 Grundtypen, eventuell per Modifier modifiziert.
- Der Speicherbereich wird durch den Compiler/Linker festgelegt.
- Die Lebensdauer und der Sichtbarkeitsbereich von Variablen werden später behandelt werden.

Variablendefinition

- In C müssen Variablen immer vor ihrer erstmaligen Verwendung angelegt (definiert) werden. Dazu wird im Programm der Typ der Variablen gefolgt vom Variablennamen angegeben.
- Die Variablendefinition wird durch ein Semikolon abgeschlossen.
- Mehrere solcher Definitionen können aufeinander folgen.

- Eine **Variablendeklaration** macht die Variable dem Compiler bekannt.
- Eine **Variablendefinition** reserviert Speicherplatz zur Ablage eines Wertes.

Variablendefinition

```
#include <iostream>
using namespace std;

int main()
{
  int summe;
  int a;
  double b;

  . . .
  . . .
  . . .
  return 0;
}
```

Hier steht der **(Daten-)Typ** der Variablen.

Dann folgt der **Name** der Variablen**.**

Weitere **Variablendefinitionen** können folgen.

Variablendefinition

- Zwei verschiedene Typen: int (integer = ganze Zahl) und double (floating point number = Gleitkommazahl)
- Alle Zahlen könnten als Typ double dargestellt werden, aus vielen Gründen ist es sinnvoll, in Programmiersprachen diese Unterscheidung zu treffen.
  - Ganze Zahlen belegen weniger Speicherplatz als Gleitkommazahlen.
  - Mit ganzen Zahlen kann schneller gerechnet werden als mit Gleitkommazahlen (ALU <==> FPU).
- Der Variablenname kann relativ frei vergeben werden und besteht aus einer Folge von Buchstaben (keine Umlaute oder ß), Ziffern und dem Zeichen „_“.
- Das erste Zeichen eines Variablennamens muss ein Buchstabe oder ein „_“ sein.
- Groß- und Kleinbuchstaben werden unterschieden.

Variablendefinition

- Die Anzahl signifikanter Zeichen ist implementierungs-abhängig (im Allgemeinen mehr als 30 Zeichen)
- Teilweise existieren zusätzliche Einschränkungen für externe Namen (Linker, Assembler, andere Sprachen, ...)
- Reservierte C++ Worte sind zur Namensgebung verboten. (z.B. if, main, while, const, class, …)

Variablendefinition

- Über diese Vorschriften hinaus, sind folgende Punkte Empfehlungen, die sich im praktischen Einsatz als nützlich (lebenswichtig) erwiesen haben:
  - Name sollen nicht mit Unterstrich "_" beginnen.
  - Namen sollten natürlichsprachig, problemnah, aussagekräftig sein:
    - nicht hugo oder emil, sondern summe oder ergebnis
  - Inhaltsleere Namen (i, j, k) sind allenfalls für lokale Schleifenzähler etc. sinnvoll.
  - Oft werden projektbezogene Konventionen vorgegeben, die den firmenüblichen Namenskonventionen Rechnung tragen.
- Namen sind **nicht** Schall und Rauch, sondern entscheidend über die Lesbarkeit von Programmen und sind somit ein **Qualitätsmerkmal** der erzeugten Software!

Operationen

- Variablen an sich sind wertlos, wenn man nicht sinnvolle Operationen mit ihnen ausführen kann. Üblicherweise denkt man dabei sofort an folgende Operationen:
  - Variablen Werte zuweisen
  - Mit Variablen und Zahlen rechnen
  - Variablen und Zahlen miteinander vergleichen
- Das gibt es natürlich auch in der Programmiersprache C.

Zuweisungsoperation

```
#include <iostream>
using namespace std;

int main()
{
  int summe;
  int a;
  double b;

  a = 123;
  b = a;
  . . .
  return 0;
}
```

Der Variablen **a** wird der Wert 123 zugewiesen**.**

Der Variablen **b** wird der Wert a (= 123) zugewiesen**.**

Operationen

- Zuweisungsoperationen:
  - Links vom Gleichheitszeichen steht der Name der zuvor definierten Variablen.
  - Dieser Variablen wird durch Zuweisung ein Wert gegeben. Als Wert kommen dabei konkrete Zahlen, Variablenwerte oder auch allgemeinere Ausdrücke (Berechnungen, Formeln) in Frage.
  - Die Wertzuweisung erfolgen in der angegebenen Reihenfolge; d.h. a hat bereits den Wert 123 bevor die Zuweisung an b erfolgt.
  - Ein ganz-zahliger Wert kann einer Gleitkommazahl zugewiesen werden (ohne Compiler-Warnung). Umgekehrt ist dies nicht möglich.
- Mit Variablen und Zahlen kann gerechnet werden, wie aus der Schulmathematik bekannt.

Rechenoperation

- Die Operatoren für einfache arithmetische Ausdrücke sind:

Addition	+
Subtraktion	–
Multiplikation	*
Division	/

- Es gilt auch hier Punktrechnung (*,/) vor Strichrechnung (+,-) wobei Klammern in der üblichen Bedeutung verwendet werden können.

```
int a;
double b;
a = 1;
b = 2 * ( a + 1.234 );
b = ( b + a * ( a – 1 ) ) / 3;
```


Rechenoperation

- Ganze Zahlen und Gleitkommazahlen können dabei gemischt verwendet werden.

- Es wird solange wie möglich im Bereich der ganzen Zahlen gerechnet. Sobald die erste Gleitkommazahl ins Spiel kommt, wird die weitere Berechnung im Bereich der Gleitkomma-zahlen durchgeführt.

Rechenoperation

```
#include <iostream>
using namespace std;
int main()
{
  int summe;
  int a;
  double b;

  a = 123;
  b = a;
  summe = 2 * a + 3;
 return 0;
}
```

Der Wert der Variablen **summe** wird durch einen Formelausdruck berechnet.

Rechenoperation

- Die Variable auf der linken Seite der Zuweisung kann auch auf der rechten Seite der Zuweisung vorkommen.
- Zuweisungen der Form a = a + 1; kommen daher ausgesprochene häufig vor. Zunächst wird die rechte Seite vollständig ausgewertet, dann wird das Ergebnis der Variablen links vom Gleichheitszeichen zugewiesen.
- Die Anweisung ist also kein mathematischer Widerspruch, sondern erhöht den Wert der Variablen a um 1.

Rechenoperation

- Das Ergebnis einer arithmetischen Operation, an der **nur** ganzzahlige Operanden beteiligt sind ist **immer** eine ganze Zahl.
- Im Falle der Division führt das zu überraschenden Ergebnissen. Denn die Operation „/“ im Falle von ganzzahligen Operanden eine **Division ohne Rest** bedeutet.
- Dies ist für die Programmierung genauso wichtig wie die „richtige“ Division.
- Beispiel:
  - int m; m = 19 / 3; ==> m = 6; (19:3 = 6 Rest 1)
  - int r; r = 19 % 3; ==> r = 1; (19:3 = 6 Rest 1) (modulo – Operator)
- Mit Gleitkommazahlen wird natürlich „richtig“ gerechnet, wobei hier Rundungsfehler auftreten können, wie sie auch vom Rechnen mit Taschenrechnern bekannt sind.

Vergleichsoperationen

- Zahlen und Variablen können untereinander verglichen werden.
- Zulässige Vergleichsoperatoren:

kleiner	<
kleiner oder gleich	<=
größer	>
größer oder gleich	>=
gleich	==
ungleich	!=

- Auf der linken bzw. rechten Seite eines Vergleichsausdrucks können dabei beliebige Ausdrücke mit Variablen oder Zahlen stehen:
  a > 7 a < = b (a + 1) = = (a * a)
- das Ergebnis ist dann ein logischer Wert („wahr“ oder „falsch“), der in C durch 1 (wahr) oder 0 (falsch) dargestellt wird. Mit diesem logischen Wert kann dann weitergearbeitet werden.
- Vergleiche stellt man nicht an, um mit dem Vergleichsergebnis weiter zu rechnen, sondern um anschließend im Programm zu verzweigen. Das wird aber später erklärt werden.

Variablendeklaration

- Variablen werden an 3 Orten deklariert:
  - innerhalb von Funktionen (**lokale Variablen**)
    (Beachte: auch main() gilt als eine Funktion!
  - in der Definition von Funktionsparametern (**formale Parameter**)
  - außerhalb aller Funktionen (**globale Variablen**)

- Variablendeklaration:

```
int i,j,l;
short int si;
unsigned int ui;
double balance, profit, loss;
```


Lokale Variablendeklaration

```
void func1(void)
{
  int x;
  x = 10;
}
void func2(void)
{
  int x;
  x = -199;
}
```

- Die Variable x ist jeweils lokal im Block bekannt. Beim Verlassen des Blockes wird sie gelöscht. Lokale Variable werden auf dem so genannten Stack gespeichert.

- In C müssen alle Deklarationen vor den ersten Zuweisungen oder Aufrufen stehen, in C++ können Sie beliebig platziert werden.

Globale Variablendeklaration

- Globale Variablen sind im ganzen Programm bekannt.

```
#include <iostream>
using namespace std;

int count;

int  main( void )
{
  count = 100;
  cout << "count (main) ist " << count << endl;
  func1();
  cout << "count (main) ist " << count << endl;

  system("pause");
  return 0;
}
```


Globale Variablendeklaration

- Fortsetzung

```
void func1(void)
{
  count = 200;
  cout << "count (func1) ist " << count << endl;
  func2();
  count = 200;
  cout << "count (func1) ist " << count << endl;
}
void func2(void)
{
  int count;
  for ( count = 1; count < 10; count++)
    cout << "count (func2) ist " << count << endl;
}
```

Globale Variablendeklaration

- Fortsetzung - Bildschirmausgabe

```
count (main) ist 100
count (func1) ist 100
count (func2) ist 1
count (func2) ist 2
count (func2) ist 3
count (func2) ist 4
count (func2) ist 5
count (func2) ist 6
count (func2) ist 7
count (func2) ist 8
count (func2) ist 9
count (func1) ist 200
count (main) ist 200
```


Kommentare

- C Programme können durch Kommentare verständlicher gestalten werden
- Kommentare beginnen mit /* und enden mit */ bzw. bei einzeiligen Kommentaren genügt //
- Sie werden beim Übersetzen (Kompilieren) des Programms einfach ignoriert
- Kommentare sollten nur dort eingesetzt werden, wo sie wirklich etwas zum Programmverständnis beitragen

Kommentare

```
#include <iostream>
using namespace std;
int main()
{
  /*
          Variablendefinition
  */
  int summe;            // Dies ist ein Kommentar
  int a;                /**** Dies ebenfalls ****/
  double b;
  // Programmcode
  a = 123;
  b = a;
  summe = 2 * a + 3;
  return 0;
}
```


Konstanten

- **Variable** können zu verschiedenen Zeitpunkten **verschiedene Werte** enthalten.
- Alle Werte, die über die Programmlaufzeit hinweg von einer Variable eingenommen werden, gehören zum gleichen Typ.

- **Konstanten** sind **Speicherzellen**, die während der gesamten Programmlaufzeit **denselben Wert** enthalten.
- Im Programmcode fest eingetragene Werte werden ebenfalls als Konstanten bezeichnet.
- Konstanten sind beschrieben durch
  - Name (engl.: identifier),
  - Typ (engl.: type),
  - Wert (engl.: value)

Konstanten

- Variablen vom Typ const können vom Programm nicht geändert werden. Es kann allerdings ein Anfangswert zugewiesen werden. Dem Compiler steht es frei, Variablen diesen Typs im ROM (read only memory) anzulegen.

- **Const mit expliziter Initialisierung**

```
const double PI = 3.141592;
const int MAXIMUM = 10;
char line[MAXIMUM];
```

- Konstantenausdrücke wie oben werden zur Übersetzungszeit berechnet und vom Compiler als Konstante behandelt.

Elementare Ein- und Ausgabe

- Um erste einfache Programme schreiben zu können, müssen wir noch
  - Werte von der Tastatur in Variablen einlesen
  - Werte von Variablen auf dem Bildschirm

  können.
- **Bildschirmausgabe:**
  - Um einen Text auf dem Bildschirm auszugeben, verwenden wir **cout** und schreiben:

    ```
    cout << "Dieser Text wird ausgegeben" << endl;
    ```
  - Der auszugebende Text wird dabei in doppelte Hochkommata eingeschlossen.
  - Die Zeichenfolge endl erzeugt einen Zeilenvorschub.
  - Vergessen Sie nicht das Semikolon am Ende der Zeile.
  - In den auszugebenden Text können wir Werte einstreuen, indem wir den Variablennamen angeben.

Bildschirmausgabe

```
#include <iostream>
using namespace std;
int main()
{
  int wert;
  double preis;

  wert = 1;
  cout << "Dies ist die " << wert << ". Zeile." << endl;
  wert = 2;
  cout << "Hier kommt die " << wert << ". Zeile." << endl;
  preis = 10.99;
  cout << "Die Ware kostet " << preis << " EUR." << endl;
  return 0;
}
```


Bildschirmausgabe

Ausgabe:

```
Dies ist die 1. Zeile.
Hier kommt die 2. Zeile.
Die Ware kostet 10.99 EUR.
```

Tastatureingabe

- Tastatureingabe:
  - Eine oder mehrere Zahlen lesen wir mit **cin** von der Tastatur ein.
    - cin >> zahl;
  - Beim Einlesen müssen Variablen angegeben werden, in die Werte eingetragen werden sollen.
- Beispiel:
  - Programmauszug:

```
int zahl;
double preis;
cout << "Geben Sie bitte eine Zahl ein: ";
cin >> zahl;
cout << "Geben Sie bitte einen Preis ein: ";
cin >> preis;
cout << endl << "Sie haben die Zahl " << zahl << " und den Preis "
  << preis << " eingegeben." << endl;
```

  - Bildschirmausgabe:

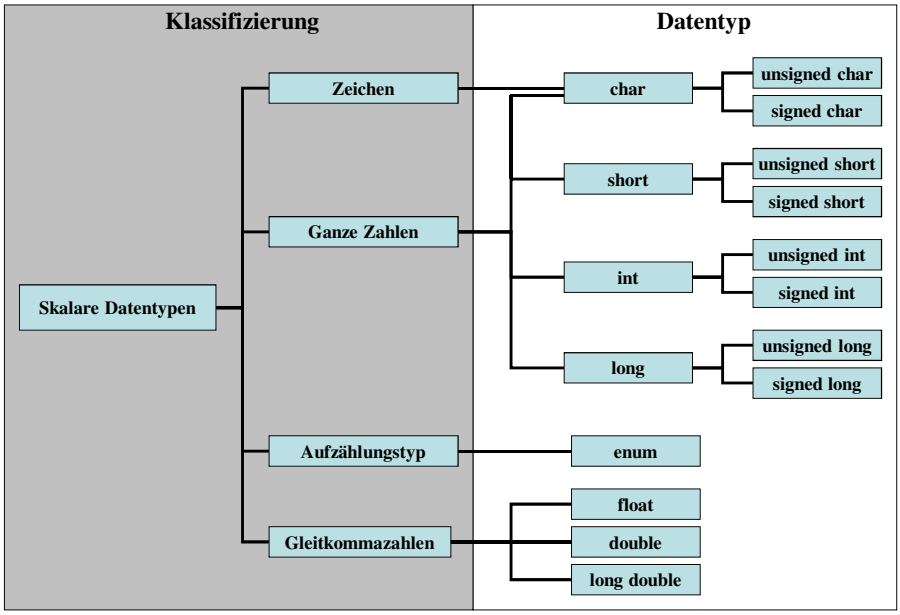
```
Geben Sie bitte eine Zahl ein: 123
Geben Sie bitte einen Preis ein: 10.99

Sie haben die Zahl 123 und den Preis 10.99 eingegeben.
```


Datentypen

- Jede Variable muss einen Typ haben. Bis jetzt int und float kennengelernt.
- In C gibt es nur wenige elementare so genannte skalare Datentypen: Datentyp zur Darstellung numerischer Werte
  - **char**: ein Byte, kann ein Zeichen aus dem Zeichensatz der Maschine aufnehmen
  - **int**: ganzzahliger Wert, in der für die Maschine „natürlichen“ Größe (16 bit = 2 Byte, Wertebereich: –(2^15):(2^15)-1 [-32768:32767]
  - **float**: einfach genauer Gleitkommawert
  - **double**: doppelt genauer Gleitkommawert

Skalare Datentypen



Ganze Zahlen

- Es gibt vier verschiedene Datentypen (char, short, int, long) für ganze Zahlen, die sich bzgl. der Größe der darstellbaren Zeichen unterscheiden. Da es jeweils eine Ausprägung für vorzeichenlose (unsigned) bzw. vorzeichenbehaftete (signed) Zahlen haben, ergeben sich insgesamt acht Kombinations-möglichkeiten.
- Der Zusatz signed ist optional und wird von C-Programmierern fast nie verwendet.
- Die angegebenen Zahlenwerte auf der nächsten Seite sind implementierungsabhängig, siehe auch in limits.h.

Ganze Zahlen

Variablendefinition	Bedeutung
char x signed char x	x ist eine „kleine“, vorzeichenbehaftete Zahl. Typischerweise eine 1-Byte-Zahl im Bereich von -128 bis +127.
unsigned char x	x ist eine „kleine“, vorzeichenlose Zahl. Typischerweise eine 1-Byte-Zahl im Bereich von 0 bis +255.
short x signed short x	x ist eine „mittelgroße“, vorzeichenbehaftete Zahl. Typischerweise eine 2-Byte-Zahl im Bereich von -32 768 bis +32 767.
unsigned short x	x ist eine „mittelgroße“, vorzeichenlose Zahl. Typischerweise eine 2-Byte-Zahl im Bereich von 0 bis +65 535.
int x signed int x	x ist eine „normale“, vorzeichenbehaftete Zahl. Typischerweise eine 2- oder 4- Byte-Zahl.
unsigned int x	x ist eine „normale“, vorzeichenlose Zahl. Typischerweise eine 2- oder 4- Byte-Zahl.
long x signed long x	x ist eine „große“, vorzeichenbehaftete Zahl. Typischerweise eine 4-Byte-Zahl im Bereich von -2 147 483 648 bis +2 147 483 647 (-2^{31} bis $2^{31}-1$).
unsigned long x	x ist eine „große“, vorzeichenlose Zahl. Typischerweise eine 4-Byte-Zahl im Bereich von 0 bis +4 294 967 295 (0 bis $2^{32}-1$).

Ganze Zahlen

- Wenn im Programm konkrete Integer-Zahlenwerte für Berechnungen oder Zuweisungen benötigt werden, gibt es drei verschiedene Möglichkeiten.
- Der Zahlenwert kann **dezimal**, **oktal** oder **hexadezimal** angegeben werden.
- Dezimaldarstellung: alltägliches Zahlenformat
- Oktaldarstellung: Den Ziffern wird eine „0“ vorangestellt.
- Hexadezimaldarstellung: Den Ziffern wird „0x“ vorangestellt.
- Beispiele:

```
long a = 12345;           /* dezimal 12345 */
unsigned int a = 0xaffe;  /* dezimal 45054 */
unsigned char a = 0377;   /* dezimal   255 */
```

Aufzählungstypen

- Beim Datentyp **enum** handelt es sich um einen benutzer-definierten **Aufzählungstyp**, bei dem der Programmierer selbst geeignete Namen für einzelne Werte vergeben kann.
- Beispiel Wochentag:

```
enum wochentag { Montag, Dienstag, Mittwoch, Donnerstag,
  Freitag, Samstag, Sonntag };
```

- Einmal deklariert ist `Wochentag` ein Datentyp wie `int`, der die Werte Montag – Sonntag annehmen kann. Verwendet wird ein Aufzählungstyp dann wie folgt:

```
enum wochentag tag;
int datum;
tag = Freitag;
datum = 13;
if (( Freitag == tag ) && ( 13 == datum ))
{
  cout << "Vorsicht, heute ist Freitag der 13ste!" << endl;
}
```


Aufzählungstypen

- Welche Zahlenwerte Montag – Sonntag zugeordnet werden, ist nicht festgelegt.
- In C wird nicht überprüft, ob Variablen von einem Aufzählungstyp wirklich nur die für diesen Aufzählungstyp definierten Werte enthalten. So kann man diesen Zahlen einen beliebigen int-Zahl zuweisen und wie mit int rechnen.
- In der C++ Entwicklungs wird dagegen überprüft, ob Variablen von einem Aufzählungstyp wirklich nur die für diesen Aufzählungstyp definierten Werte enthalten.
  - D.h. tag = freitag bzw. tag = 12 werden als Fehler angezeigt und verhindern, dass der Quellcode fehlerfrei kompiliert wird
- Aufzählungstypen haben also keine weitere Funktionalität, sondern dienen eher der Lesbarkeit eines Programms.

Gleitkommazahlen

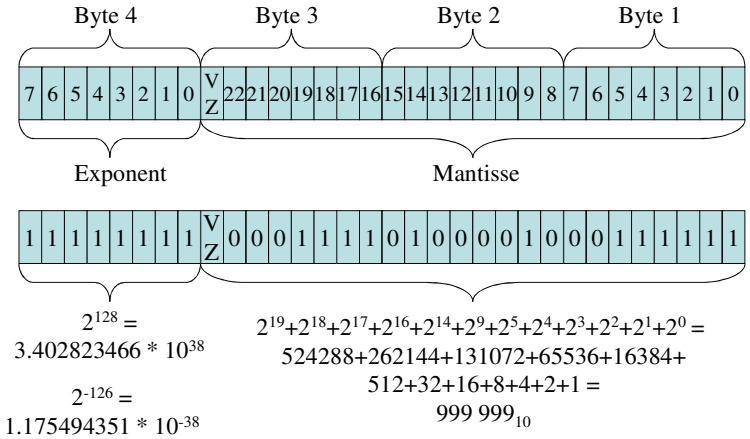
- Neben den Integer-Datentypen gibt es in C drei Datentypen für Gleitkommazahlen (float, double, long double).

Variablendeklaration	Bedeutung
float x	x ist eine „normale“ Gleitkommazahl. Typischerweise eine 4-Byte-Zahl (3.4E+/-38 (7 Ziffern)).
double x	x ist eine Gleitkommazahl doppelter Genauigkeit. Typischerweise eine 8-Byte-Zahl (1.7E+/-308 (15 Ziffern)).
long double x	x ist eine Gleitkommazahl besonders hoher Genauigkeit. Typischerweise eine 10-Byte-Zahl (1.2E+/-4932 (19 Ziffern)).

- Zur Eingabe von Gleitkomma-Zahlenwerten verwendet man die technisch-wissenschaftliche Notation mit Vorzeichen, Mantisse und Exponent (z.B. -3.456e-78).
- Es wird kein Komma, sondern der Dezimalpunkt verwendet.
- Die angegebenen Zahlenwerte auf den nächsten Seiten sind implementierungsabhängig, siehe auch in float.h.

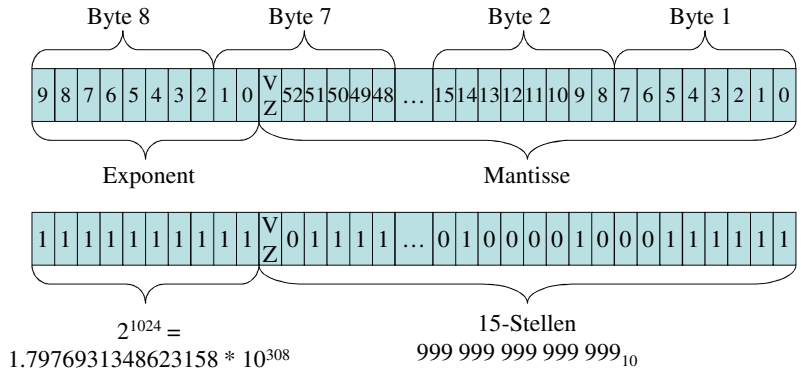
Gleitkommazahlen

- Darstellung des Typs **float** im Speicher



Gleitkommazahlen

- Darstellung des Typs **double** im Speicher



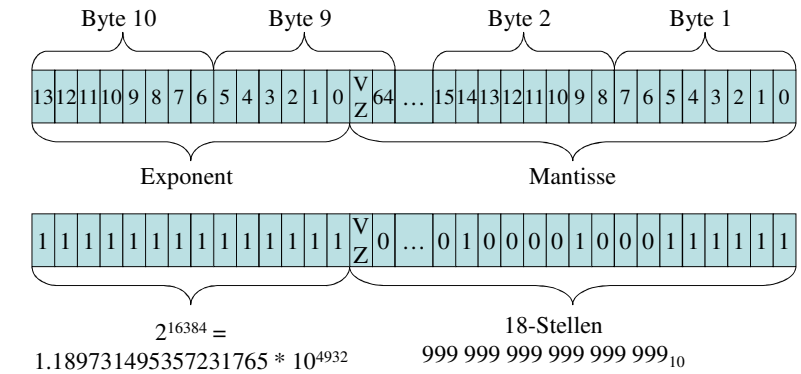
2^{1024} =
$1.7976931348623158 * 10^{308}$

15-Stellen
$999\ 999\ 999\ 999\ 999_{10}$

2^{-1022} =
$2.2250738585072014e * 10^{-308}$

Gleitkommazahlen

- Darstellung des Typs **long double** im Speicher
- Abgebildet auf double; long double nur theoretisch



2^{16384} =
$1.189731495357231765 * 10^{4932}$

18-Stellen
$999\ 999\ 999\ 999\ 999\ 999_{10}$

2^{-16382} =
$3.3621031431120935063e * 10^{-4932}$

Gleitkommazahlen

- Phänomen der Auslöschung bei der Subtraktion zweier fast gleicher Zahlen
- Beispiel: (a + b) + c <=> a + (b + c)

```
a = 1.23456e-4  =  0.000123456
b =             =  1.00000
                   ---------------
s1 = a + b      =  1.00012
c =             = -1.00000
                   ---------------
s1 = s1 + c     =  0.00012
s1 =            =  1.2e-4   (Exponent - Mantisse)

a = 1.23456e-4  =  0.000123456
d = b + c       =  0.00000
                   ---------------
s2 = a + d      =  0.000123456
s2 =            =  1.23456e-4   (Exponent - Mantisse)
```


Auslöschung

- **In C-Notation:**

```
#include <iostream>
#include <iomanip>
using namespace std;
void main ()
{
  double a, b, c, s1, s2;
  a = 1.23456789012345e-4;
  b = 1.0;
  c = -b;
  s1 = a + b;
  cout << setprecision(15);
  cout << "a  = " << a << endl;
  cout << "s1 = " << s1 << endl;
  s1 = s1 + c;
  cout << "s1 = " << s1 << endl;
  s2 = a;
  cout << "s2 = " << s2 << endl;
  s2 = s2 + (b + c);
  cout << "s2 = " << s2 << endl;
}
```

Auslöschung

- **Ausgabe:**

```
a  = 0.000123456789012345
s1 = 1.000123456789012
s1 = 0.000123456789012266
s2 = 0.000123456789012345
s2 = 0.000123456789012345
```


Speicherbedarf und Wertebereich

			signed		unsigned	
	Typ	**Bit**	**Minimum**	**Maximum**	**Minimum**	**Maximum**
Z	char	8	-128	127	0	255
	short int	16	-32.768	32.767	0	65.535
	int	32	-2.147.483.648	2.147.483.647	0	4.294.967.295
	long int	32	-2.147.483.648	2.147.483.647	0	4.294.967.295
			| Minimum |	**| Maximum |**	**Genauigkeit**	
R	float	32	$\pm 3.4 * 10^{-38}$	$\pm 3.4 * 10^{38}$	$1.2 * 10^{-7}$	6-Dezimalstellen
	double	64	$\pm 1.7 * 10^{-308}$	$\pm 1.7 * 10^{308}$	$2.2 * 10^{-16}$	15-Dezimalstellen
	long double	80	$\pm 1.2 * 10^{-4932}$	$\pm 1.2 * 10^{4932}$	$1.1 * 10^{-19}$	18-Dezimalstellen

Buchstaben

- Ein Computer soll nicht nur Zahlen, sondern auch Buchstaben und Text verarbeiten können. Da der Computer nur Dualzahlen kennt, muss es eine Zuordnung von Buchstaben zu Dualzahlen geben.
- Eine solche Zuordnung ist für die in der Datenverarbeitung am meisten verwendeten Zeichen im ASCII[1]-Zeichensatz festgelegt.
- Die meisten Rechner benutzen diesen Zeichensatz, haben aber oft individuelle Erweiterungen (nationale Zeichensätze, grafische Symbole).
- Grundsätzlich unterscheidet man druckbare und nicht druckbare Zeichen.
- Druckbar sind z.B. A, B, C, …
- Nicht druckbare Zeichen haben meist eine Steuerfunktion bzw. andere Sonderfunktion. Dafür sind im ASCII-Zeichensatz Kürzel festgelegt, die aber nicht alle erklärt werden.
  - EOT = End of Tape = Bandende, ACK = Acknowledge = akzeptiert

[1]: ASCII = American Standard Code for Information Interchange

Buchstaben

- Wichtig sind die Bildschirm-Steuerzeichen, in deren Bezeichnung (z.B. Wagenrücklauf) deutlich der Ursprung aus dem Fernschreiber-Bereich anzumerken ist:
  - BS = Backspace = Rückwärtsschritt
  - HT = Horizontal Tab = Horizontaler Tabulator
  - LF = Linefeed = Zeilenvorschub
  - VT = Vertical Tab = Vertikaler Tabulator
  - FF = Formfeed = Seitenvorschub
  - CR = Carriage Return = Wagenrücklauf
  - SP = Space = Leerzeichen
- Der ASCII-Zeichensatz nutzt nur die Hälfte der in einer 8-Bit-Darstellung möglichen Zeichencodes. Rechnersysteme nutzen üblicherweise auch den freien Bereich für Sonderzeichen (Umlaute, Grafikzeichen). Diese sind aber nicht normiert, daher werden sie hier nicht weiter aufgelistet.

Der ASCII - Zeichensatz

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HAT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

ASCII - Zeichensatz

- Das Zeichen 'A' hat den Zeichencode $(41)_{16} = 65$.
- Das Zeichen 'z' hat den Zeichencode $(7a)_{16} = 122$.
- Der Rechner speichert nicht das Zeichen, sondern den Zahlencode - also eine Zahl. Für den Rechner besteht kein Unterschied zwischen Zeichen und dem zugehörigen Zeichencode.
- Konsequenterweise verwendet C deshalb auch den Integer-Datentyp zur Ablage von Buchstaben.
- Da ein Code des ASCII - Zeichensatzes in den kleinsten verfügbaren Integer-Datentyp hineinpasst, verwendet man den Begriff **char** (engl. character = Buchstabe). Der Datentyp char beinhaltet also Zahlen und Buchstaben.
- Daher kann man mit Buchstaben auch rechnen.
- Beispiel: Addiert man zu 'A' eine 1 hinzu, erhält man das auf 'A' folgende Zeichen also ein 'B'.
- Daher kann man Buchstaben auch der Größe nach vergleichen; wichtig für Sortieraufgaben.

Buchstaben

- Schreibweise in der C-Programmierumgebung in Hochkommata, um es von Variablen unterscheiden zu können.

```
char buchstabe = 'a';
. . .
if ( buchstabe <= 'z' )
    . . .
```

- Gilt nur für druckbare Zeichen.
- Ersatzdarstellung für die Nicht-druckbaren Zeichen. Dazu "opfert" man ein druckbares Zeichen den Backslash "\" als Indikator für ein Sonderzeichen.
- Der Backslash heißt daher auch Escape - Zeichen.
- Zusammen mit den nachfolgenden Zeichen spricht man von einer Escape - Sequenz.

Escape - Sequenz

Escape-Sequenz	Bedeutung
\n	neue Zeile
\t	horizontaler Tabulator
\v	vertikaler Tabulator
\b	backspace (Rückschritt, Löschen)
\r	carriage return (Wagenrücklauf)
\f	form feed (Seitenvorschub)
\'	einfaches Hochkomma
\"	doppeltes Hochkomma
\a	bell (Klingelzeichen)
\?	Fragezeichen
\\	backslash
\zzz	Oktaldarstellung des Zeichencodes (z = 0, 1, … , 7)
\xzz	Hexdarstellung des Zeichencodes (z = 0, 1, … , 9, a, … , f)

Escape - Sequenz

- Beispiele:

```
'\n'     /* ein Zeilenvorschub    Hex-Code: 0a */
'\t'     /* ein Tabulatorzeichen  Hex-Code: 09 */
'\\'     /* ein Backslash         Hex-Code: 5c */
'\''     /* ein einf. Hochkomma   Hex-Code: 27 */
'\101'   /* ein A (Oktalcode)     Hex-Code: 41 */
'\x41'   /* ein A (Hex-Code)      Hex-Code: 41 */
```

- Auch wenn bis zur Darstellung bis zu 4 Zeichen verwendet werden, handelt es sich immer nur um 1 Zeichen (1-Byte-Zeichencode).

Formatierte Ausgabe

- Manipulatoren werden in den Ausgabestrom eingefügt. Sie bewirken keine Ausgabe, sondern bewirken eine Zustands-änderung des Ausgabestroms
- Folgende Ausgabemanipulatoren existieren:

Manipulator	Bedeutung	beeinflusst
dec	dezimale Ausgabe (default)	Z
hex	hexadezimale Ausgabe	Z
oct	oktale Ausgabe	Z
flush	Leeren des Puffers auf das Ausgabemedium	
endl	Ausgabe von '\n' und Aufruf von flush	
setw (int n)	n ist minimale Ausgabebreite (nur für unmittelbar folgende Ausgabe, wird danach auf 0 gesetzt)	
setprecision (int n)	n ist die Ausgabegenauigkeit	R
setfill (char c)	c ist Füllzeichen	