Die Programmiersprache C++
Einführung / Datentypen

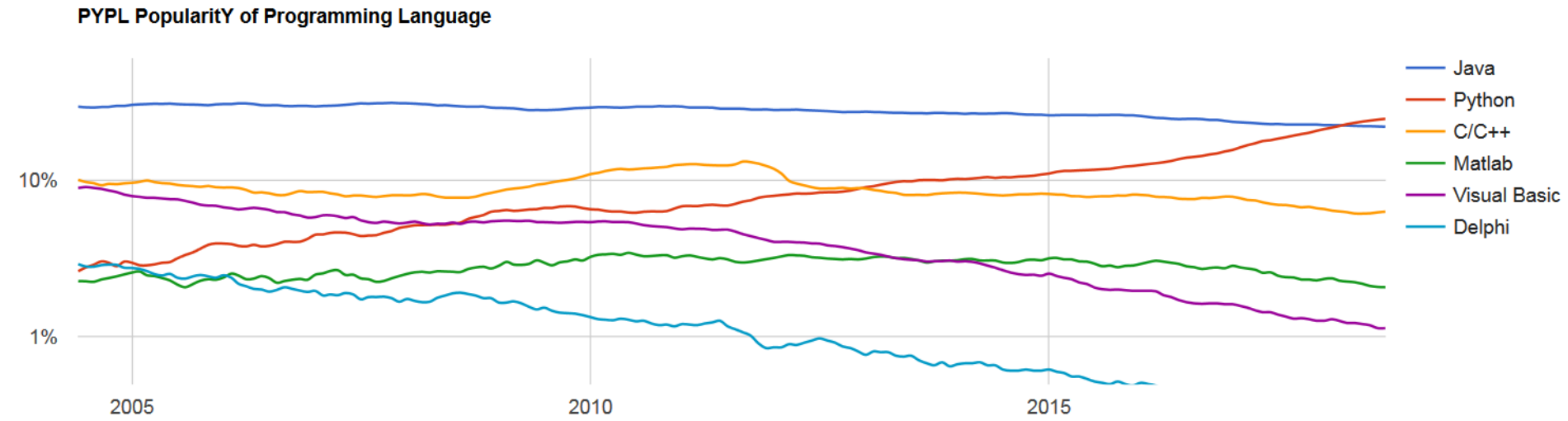
Karsten Rink

Department for Environmental Informatics, Helmholtz-Centre for Environmental Research, Leipzig

Warum C++?

- Universalsprache
- objektorientiert
- performant
- sehr weit verbreitet
- Seit über 30 Jahren etabliert

Rank	Change	Language	Share	Trend
1	↑	Python	24.72 %	+5.4 %
2	↓	Java	22.01 %	-0.7 %
3	↑	Javascript	8.4 %	+0.1 %
4	↑	C#	7.71 %	-0.4 %
5	↓↓	PHP	7.42 %	-1.6 %
6		C/C++	6.32 %	-0.5 %
7		R	4.11 %	-0.1 %
8		Objective-C	3.29 %	-0.9 %
9		Swift	2.69 %	-0.8 %
10		Matlab	2.08 %	-0.3 %



Die Ursprünge von C++

- Basierend auf der Programmiersprache C
  - Entwickelt 1972 von Dennis Ritchie
  - Prozedurale Sprache
  - Universalsprache: Anwendbar zur Bearbeitung aller Arten von Problemen, d.h. nicht nur für bestimme Gruppe von Anwendungen konzipiert
  - Compiler übersetzt Quelltext in Maschinencode, d.h. Programm ist auf allen Systemen lauffähig, für die es einen Compiler gibt

- C++
  - Entwickelt 1985 von Bjarne Stroustrup
  - Objekt-orientierte Sprache
  - 100% C-kompatibel (C++ ist ein Superset von C)
    d.h. man kann insbesondere auch C-Bibliotheken in C++ verwenden

- C/C++ werden noch immer aktiv entwickelt (letztes Update war C/C++ 17)

Objektorientierte Programmierung

- Fokus auf Daten statt auf Algorithmen
- Entspricht eher der menschlichen Sicht auf die Welt als prozedurale Programmierung

- Kapselung von Daten und darauf definierten Methoden in Klassen, d.h.
  - Daten und Funktionalität, die zusammengehören, sind auch im gleichen Objekt definiert/implementiert
  - Nach außen ist die Sicht auf die Daten abstrakt
  - Die Daten in Klassen können selbst wiederum Instanzen von Klassen sein
- Klassen können Eigenschaften und Funktionalität „vererben“, d.h. an spezialisierte Objekte weitergeben

$\Rightarrow$ Erleichtert das Erstellen von komplexen Programmen

Objektorientierte Programmierung: Beispiel

- Sie haben einige Datenlogger in einem Versuchsgelände installiert. Die angeschlossenen Sensoren messen Bodenfeuchte, relative Luftfeuchtigkeit und Niederschlag in festen Abständen. Sie möchten die Daten mit einem C++ Programm verwalten.

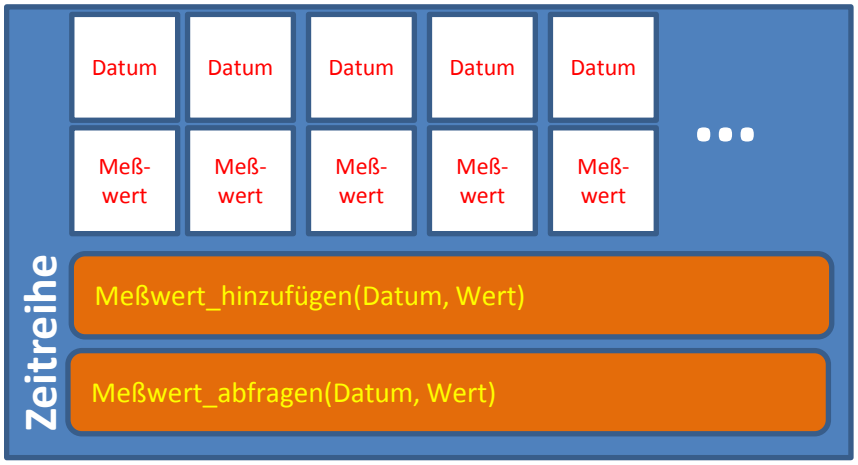
Objektorientierte Programmierung: Beispiel (2)

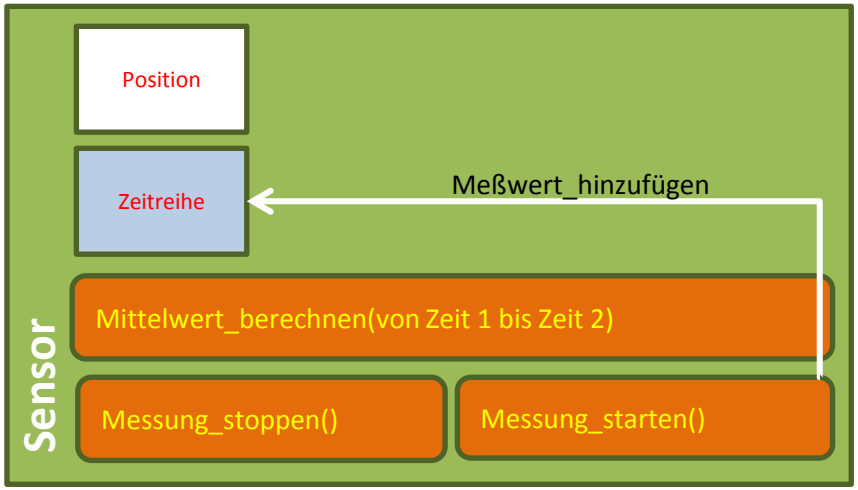
- Definition einer Klasse „Datenlogger“
  - An jeden Datenlogger sind ein oder mehrere Sensoren angeschlossen.
  - Jedem Datenlogger ist ein verantwortlicher Techniker zugeordnet
  - Wenn es zu stark regnet, soll keine Luftfeuchtigkeit gemessen werden, weil die Messwerte dann verfälscht sind

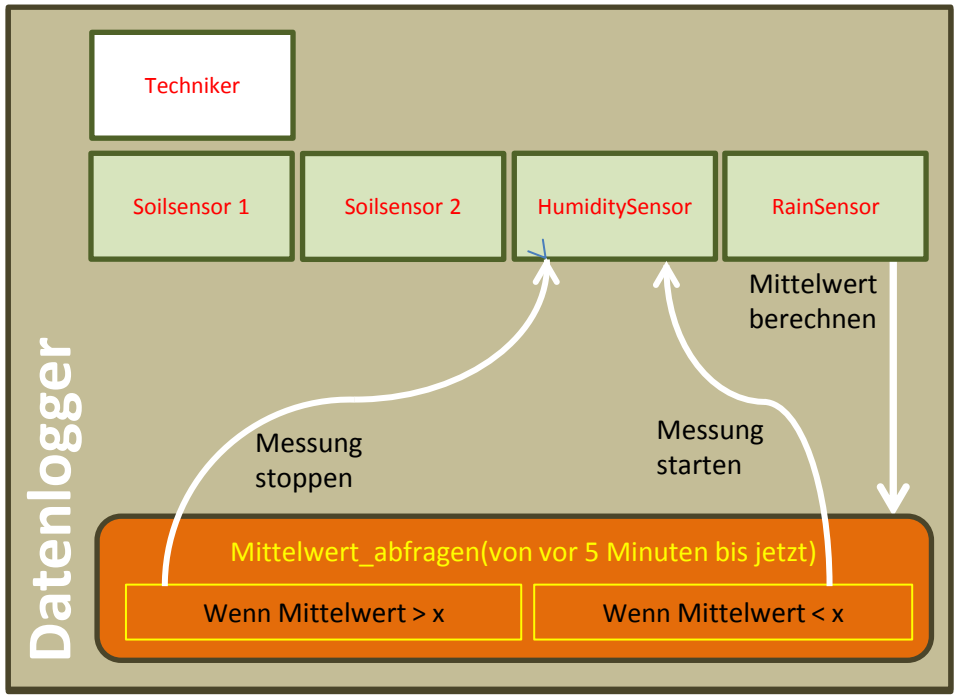
- Definition von Klasse „Sensor“ mit den davon abgeleiteten, spezialisierten Klassen „SoilSensor“, „HumiditySensor“ und „RainSensor“
  - Jeder Sensor kennt seine genaue Position
  - Jeder Sensor speichert eine Zeitreihe
  - Es soll möglich sein, den Mittelwert der Messwerte zwischen zwei Zeitpunkten zu berechnen

- Definition einer Klasse „Zeitreihe“
  - Jede Zeitreihe speichert eine Reihe von Datumsangaben und Messwerten
  - Man kann neue Messwerte hinzufügen oder alte Messwerte abfragen

Objektorientierte Programmierung: Beispiel (3)







Kapselung: Jede Klasse enthält genau die Daten und Funktionalität, die sie benötigt

Abstraktion: Der genaue Aufbau einer Klasse ist nach aussen verborgen. Der Datenlogger weiss z.B. nicht, wie eine Zeitreihe aufgebaut ist.

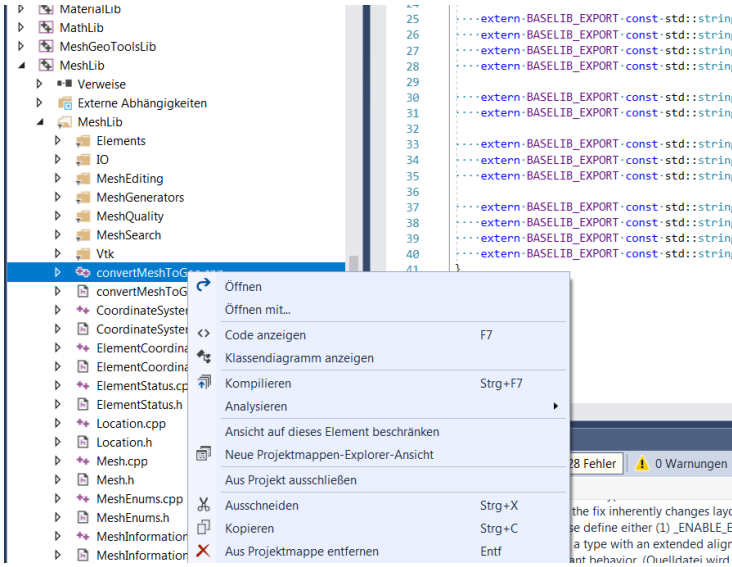
Wie schreibt man Quellcodedateien?

1. **Texteditor**
   - Gute Texteditoren haben Syntaxhighlighting und helfen mit Klammern, Einrückungen, etc.
   - z.B.: Sublime Text, Kate, Visual Studio Code

2. **IDE** (Integrated Development Environment)
   - Editor und Managementfunktionalität
   - Grafische Benutzerumgebung für alle Schritte des Entwicklungsprozesses (z.B. Kompilieren, Linken, Projektverwaltung, etc.)
   - z.B. Microsoft Visual Studio, Eclipse, Netbeans, etc.

Quelltextdateien werden im nächsten Schritt mittels eines Compilers zunächst in Maschienencode und dann in ein ausführbares Programm übersetzt.

```
27  #include "Polyline.h"
28  #include "PointVec.h"
29
30  #include "MathLib/GeometricBasics.h"
31
32  extern double orient2d(double *, double *, double *);
33
34  namespace ExactPredicates
35  {
36  double getOrientation2d(MathLib::Point3d const& a,
37      MathLib::Point3d const& b, MathLib::Point3d const& c)
38  {
39      return orient2d(const_cast<double*>(a.getCoords()),
40          const_cast<double*>(b.getCoords()),
41          const_cast<double*>(c.getCoords()));
42  }
43  }
44
45  namespace GeoLib
```

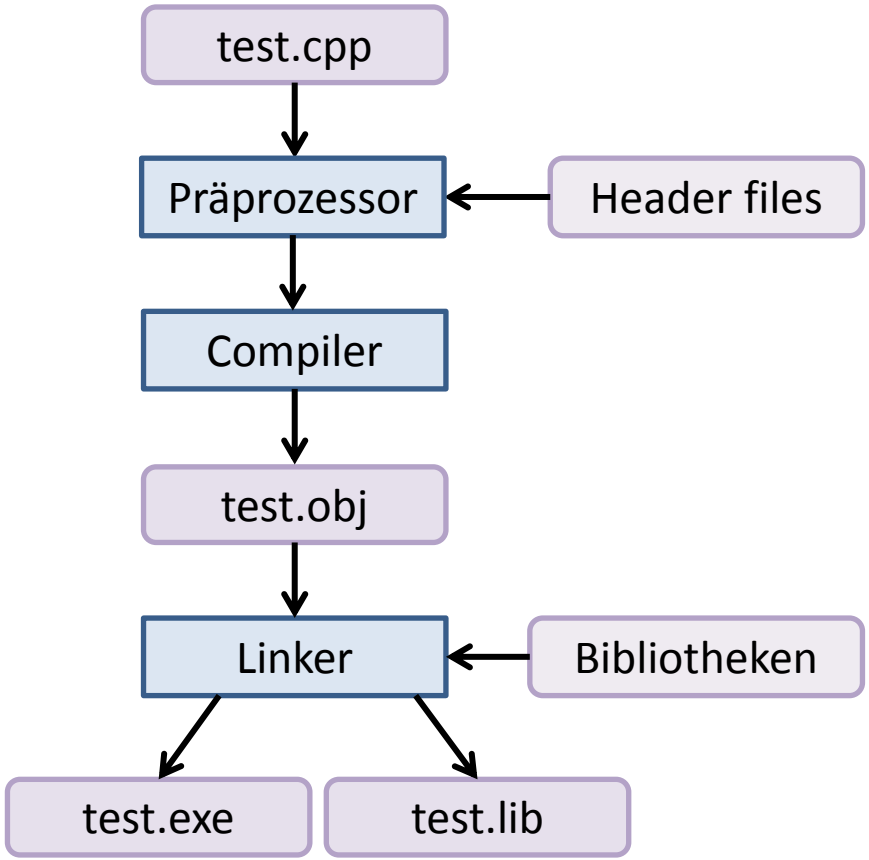


C++ Compiler

- Compiler übersetzen Quelltext in Maschinencode
  (d.h. Programme oder Teile von Programmen)

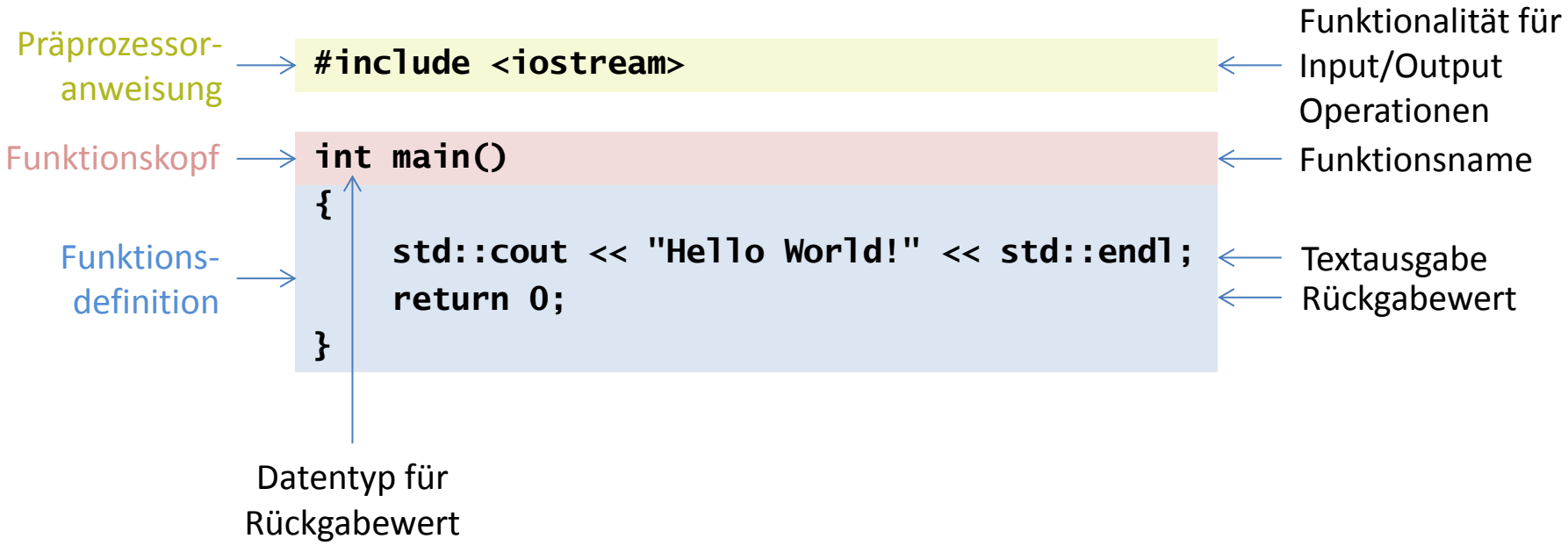
- Typische C++ Compiler:
  - **gcc/g++** : GNU Compiler Collection
    - Nativ für *nix, Windows-port: **MinGW**
    - Ursprünglich zur Anwendungsentwicklung unter Unix
  - **clang** : LLVM Compiler
    - Ursprünglich als Forschungsprojekt der Uni Chicago gestartet
    - Heute integraler Bestandteil von macOS, iOS, PS4 SDK
  - **cl** : Microsoft Visual C++ Compiler
    - Anwendungsentwicklung unter Windows (Microsoft Visual Studio)
  - **icc** : Intel C++ Compiler
    - Zusätzliche Optimierungen bzgl. Intel-Prozessoren, z.B. Nutzung für High-Performance Computing
  - Und viele andere mehr…

Compilieren und Linken



- **Präprozessor:** In allen C++-Files werden nacheinander `#include`-Anweisungen durch den Inhalt der Dateien ersetzt, Makros ersetzt (`#define`) und Teile des Quelltexts anhand definierter Bedingungen ausgewählt (`#if`/`#ifdef`)
- **Compiler:** Übersetzt den C++-Quellcode in maschienenspezifischen Binärcode und schreibt diesen in Objekt-Dateien
- **Linker:** Verknüpft aller Objektdateien, ersetzt Referenzen werden durch korrekte Speicher-adressen und schreibt das Ergebnis entweder als eine shared library (z.b. unter Windows eine *.dll-Datei oder *.dylib unter macOS) oder als eine ausführbare Datei.

Ein erstes Beispielprogramm



Der tatsächliche Rückgabewert muss den Datentyp haben, der im Funktionskopf definiert ist.
Der Datentyp `void` definiert, dass es keinen Rückgabetyp gibt.

Namespaces

- Namespaces bezeichnen eine Umgebung, in der eine Menge von Symbolen definiert sind, z.b. bezeichnet `std` den Standard-Namespace
- Man kann Namespaces explizit dem Programm bekannt machen:

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

- Vorteil: Man muss nicht immer `std::` vor Symbolen aus dem Namespace schreiben
- Nachteil: Bei Funktionen mit identischem Namen aus verschiedenen Bibliotheken kann der Compiler nicht entscheiden, welche die korrekte ist

Dateinamen

- Die Datei, in der `main()` definiert ist, ist der Programmeinstiegspunkt `[Programmname].cpp`

- Bei größeren Projekten oder der Verwendung von Bibliotheken, werden weitere Dateien mittels `#include` eingebunden
  - Systembibliotheken: `#include <iostream>`
  - Benutzerdefiniert: `#include "hydro.h"`

- Alle Programmteile ausser dem Programmeinstiegspunkt bestehen aus
  - Headerdatei: `"hydro.h"` (Deklaration von Methoden)
  - Implementationsdatei: `"hydro.cpp"` (Implementation von Methoden)

- Durch das Einbinden der Headerdateien, werden existierende Funktionen dem Programm bekannt gemacht, ohne die Implementation zu kennen

Datentypen

- Alle Daten in einem Programm werden in Variablen gespeichert, z.B.:

```
string  Vorlesungsname = "C++Datentypen";
   int  Dauer_in_Minuten = 90;
 float  Aufmerksame_Studenten = 27.3;
  bool  Ist_Vertretungsveranstaltung = true;
```

- Dafür muss das Programm folgende Dinge wissen:
  1. Wo ist die Information gespeichert? ⇒ Speichermanagement
  2. Welche Art von Information ist gespeichert? ⇒ Datentyp
  3. Welche Information ist gespeichert? ⇒ Wert

- Der einfachste Datentyp: **bool**
  - Speichert Wahrheitswerte, d.h. `true` (1) oder `false` (0)
  - Benötigt 1 *byte* Speicherplatz

```
bool x = 17 > 3;
std::cout << "Ist 17 grösser als 3? " << x << std::endl;

>Ist 17 grösser als 3? true
```

Ganzzahlige Datentypen in C++

Vorbemerkung:

- Daten werden in *bytes* gespeichert
- Ein *byte* ist eine digitale Informationseinheit und besteht aus 8 *bit*

1 byte = | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | = $2^6+2^3+2^1+2^0 = 75$

- Jedes *bit* speichert eine binäre Zahl, d.h. entweder **0** oder **1**
- Das erste *bit* einer Zahl codiert das Vorzeichen (0 = positiv, 1 = negativ)

Datentypen für ganze Zahlen:

Datentypname	Speicherbedarf	Wertebereich
short	2 byte	[-32767, 32768], d.h. $-2^{15}-1$ bis 2^{15}
int	2-4 byte	Min. so groß wie *short*, höchstens so groß wie *long*
long	min. 4 byte	[-2147483647, 2147483648], d.h. $-2^{31}-1$ bis 2^{31}

Anzahl der bytes ist systemabhängig

Gleitkommazahlen (Reelle Zahlen)

- Reelle Zahlen werden als Kombination von **Mantisse** und **Exponent** gespeichert,

z.B.

1741.67	1.75167E+3	$1.75167 * 10^{3}$
0.0833	8.33E-2	$8.33 * 10^{-2}$
-25.876	-2.5876E+1	$-2.5866 * 10^{1}$

- Durch die Speicherung von Zahlen in Binärcode und die begrenzte Anzahl von *bits* können Zahlen ggf. nicht exakt gespeichert werden und deshalb gerundet, z.b.

```
double x = 0.1;
std::cout << "x = " << x << std::endl;
```

```
>x = 0.10000000000000001
```

Datentypen für Gleitkommazahlen:

Datentypname	Speicherbedarf	Wertebereich
float	4 byte	±3.4E+38, 6 Stellen Genauigkeit
double	8 byte	±1.7E+308, 15 Stellen Genauigkeit

Zeichen/Text

char

- Zahlen von 0 bis 255 *oder*
  entsprechendes Zeichen in der ASCII-Tabelle
- Größe: 1 byte

```
char x = 'G';
std::cout << "Eingegebenes Zeichen: " << x << std::endl;
int y = x;
std::cout << "Der ASCII-Code für " << x << " ist " << y << std::endl;
```

```
> Eingegebenes Zeichen: G
> Der ASCII-Code für G ist 71
```

string

- Zeichenketten
- Größe: abhängig von der Textlänge, 1 + ein *byte* pro Zeichen

```
std::string text = "Learn C++ at http://www.learncpp.com";
```


ASCII	Char	ASCII	Char	ASCII	Char
0	(Null)	46	.	92	\
1	☺	47	/	93	]
2	☻	48	0	94	^
3	♥	49	1	95	_
4	♦	50	2	96	`
5	♣	51	3	97	a
6	♠	52	4	98	b
7		53	5	99	c
8	◘	54	6	100	d
9	○	55	7	101	e
10	◙	56	8	102	f
11	♂	57	9	103	g
12	♀	58	:	104	h
13	♪	59	;	105	i
14	♫	60	<	106	j
15	☼	61	=	107	k
16	►	62	>	108	l
17	◄	63	?	109	m
18	↕	64	@	110	n
19	‼	65	A	111	o
20	¶	66	B	112	p
21		67	C	113	q
22	▬	68	D	114	r
23	↨	69	E	115	s
24	↑	70	F	116	t
25	↓	71	G	117	u
26	→	72	H	118	v
27	←	73	I	119	w
28	∟	74	J	120	x
29	↔	75	K	121	y
30	▲	76	L	122	z
31	▼	77	M	123	{
32	(Leerstelle)	78	N	124	|
33	!	79	O	125	}
34	„	80	P	126	~
35	#	81	Q	127	⌂
36	$	82	R	128	Ç
37	%	83	S	129	ü
38	&	84	T	130	é
39		85	U	131	â
40	(	86	V	132	ä
41	)	87	W	133	à
42	*	88	X	134	å
43	+	89	Y	135	ç
44	,	90	Z	136	ê
45	„-„	91	[	137	ë

Escape Characters

- Manche Zeichen können nicht einfach über **cout** ausgegeben werden, weil sie besondere Bedeutung für die C++ Syntax haben.
- Durch Voranstellen eines „\“ wird C++ mitgeteilt, dass das Zeichen wie Text behandelt werden soll
- Beispiele: ", ', %, \, ?

```
std::cout << "Warum belegt der Order \'C:\\Qt\\\' 17\% des Speichers\?" << std::endl;
```

```
> Warum belegt der Ordner 'C:\Qt\' 17% des Speichers?
```

- Weitere Escape Characters dienen der Textformatierung,
  z.B.: \n (line feed), \r (carriage return), \t (horizontal tab), \v (vertical tab)

```
std::cout << "Qt\t= 17\%\nWindows\t= 52\%" << std::endl;
```

```
> Qt       = 17%
> Windows  = 52%
```

Übung 1

Der Befehl `sizeof()` gibt die Größe eines Datentyps zurück, also z.B.

```
int a = sizeof(bool);
```

schreibt die Anzahl der benötigten *bytes* für eine Variable des Datentyps bool in die Variable a. Nach der Ausführung sollte a = 1 sein.

Schreiben Sie ein Programm, das die Grössen aller in dieser Vorlesung vorgestellten Datentypen ausgibt, d.h. bool, short, int, long, float, double, char, string.

Da strings eine variable Größe haben, können sie den Speicherbedarf mit einigen Beispielen testen, z.B. `sizeof("Hello World!");`

Übung 2

1. Schreiben Sie eine Funktion, der man einen Wert x und eine (ganzzahlige) Prozentzahl y übergeben kann und dann y% von x als Zahlenwert zurückgibt.

2. Rufen Sie diese Funktion von `main()` mit beliebigen festen Werten auf.

3. Modifizieren Sie `main()` so, dass ein Benutzer die Zahlen über die Kommandozeile eingeben kann. Texteingaben können mittels

   ```
   std::cin >> [Variablenname];
   ```

   abgefragt werden.

4. Verschieben Sie die Funktion in eine seperate Datei und binden sie diese dann im Orginalprogramm ein.

Testfragen

1. Was ist der genaueste Datentyp in C++ ?
2. Wie groß ist der Speicherbedarf von einem string Datentyp ?
3. Mit welcher Anweisung können wir den Speicherbedarf von elementaren Datentypen ermitteln ?
4. Was sind Escape-Sequenzen ?
5. Was ist der Unterschied zwischen den Escape-Sequenzen \n und \r ?
6. Was ist die C++ Entsprechung der Klasse `cout` für einen Zeilenumbruch ?

- Online-Kurs:
  https://www.learncpp.com/

- Buch:
  S. Lippman, J. Lajoie, B.E. Moo: *C++ Primer* (5th Edition, Addison Wesley)

- Weitere Buchempfehlungen:
  https://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list