Programmieren für Physiker: C++

SS 2017

Programmieren für Physiker: C++

Karlsruhe Institute of Technology

SS 2017

Termine:

- Vorlesung: Di. 8:00-9:30 Uhr, Lehmann-HS
  (Matthias Steinhauser)
- Hörsaalübungen: Fr. 8:00-9:30 Uhr, Lehmann-HS
  (Achim Mildenberger)
- Computerübungen (Fragen stellen; Testate für Aufgaben):
  Di. 14:00-18:00 Uhr,
  Mi. 14:00-18:00 Uhr,
  Poolraum FE-06 (Flachbau Physik)
  Beginn: Di./Mi. 2./3. Mai
- Beratungstutorium: Montag Nachmittag
  (keine Testate)

`http://comp.physik.kit.edu/Lehre/Programmieren/`

Organisatorisches

- Computerzugang:
  Sie benötigen einen Computeraccount im Poolraum Physik.
  Bitte beantragen oder verlängern Sie Ihr Poolraumkonto rechtzeitig vor Beginn des ersten Tutoriums.
- Prüfung: (Studienleistung, 6 ECTS-Punkte)
  Auf den Übungsblättern sind manche Aufgaben als Pflichtaufgaben markiert.
  80% erfolgreich gelöster Pflichtaufgaben berechtigen zur Teilnahme an der Klausur.
- Klausurtermin: Di. 25.07.2017, 17:30 Uhr, Dauer: 90 Minuten
  (Nachklausur zu Beginn des WS)
- Zusätzlich zum Poolraum-Account:
  elektronische Anmeldung in CAMPUS bzw. QISPOS erforderlich

`http://comp.physik.kit.edu/Lehre/Programmieren/`

Inhalt

I. Einleitung
II. Grundlagen
III. Kontrollstrukturen
IV. Datentypen
A: Lineares Gleichungssystem
B: Interpolation
V. Felder und Strukturen
VI. Funktionen
VII. Klassen und Objekte
VIII. Zeiger
IX. Klassen II
C: Numerische Integration
D: Differentialgleichungen

Literatur

- S. Oualline, Practical C++ Programming, O'Reilly
- P. Nootz, F. Morick, C/C++ Referenz, Franzis'
- B. Stroustrup, The C++ Programming Language, Add. Wesley
- online: `http://www.cplusplus.com/`
- online: "C++ Annotations"
  `http://www.icce.rug.nl/documents/cplusplus/`

- W. Press, B. Flannery, S. Teukolsky, W. Vetterlin, Numerical Recipes in C++, Cambridge
- Stoer/Bulirsch, Numerische Mathematik 1, Springer

I. Einleitung

Karlsruhe Institute of Technology

C++

- Hochsprache/höhere Programmiersprache
  $\Leftrightarrow$ Maschinensprache (“0”, “1”); Assembler (`addi #12, d0`)

I. Einleitung

Karlsruhe Institute of Technology

C++

- Hochsprache/höhere Programmiersprache
  $\Leftrightarrow$ Maschinensprache (“0”, “1”); Assembler (`addi #12, d0`)
- Befehle ähneln Umgangssprache (“do . . . while”)
  Kommandos werden mit Hilfe von Compilern in Maschinensprache übersetzt

I. Einleitung

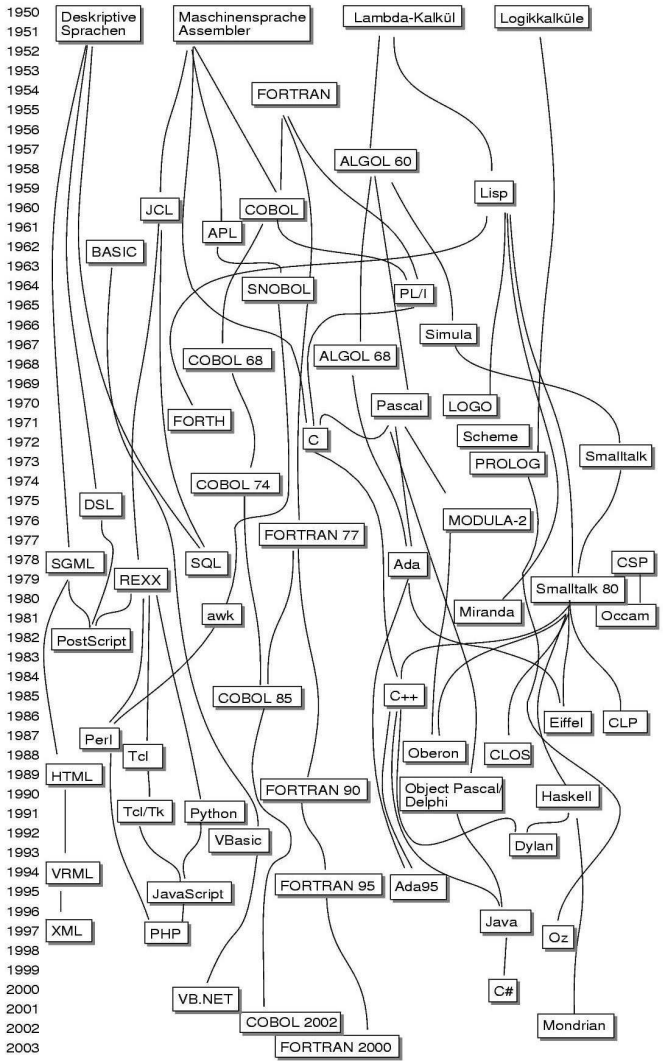
Karlsruhe Institute of Technology

C++

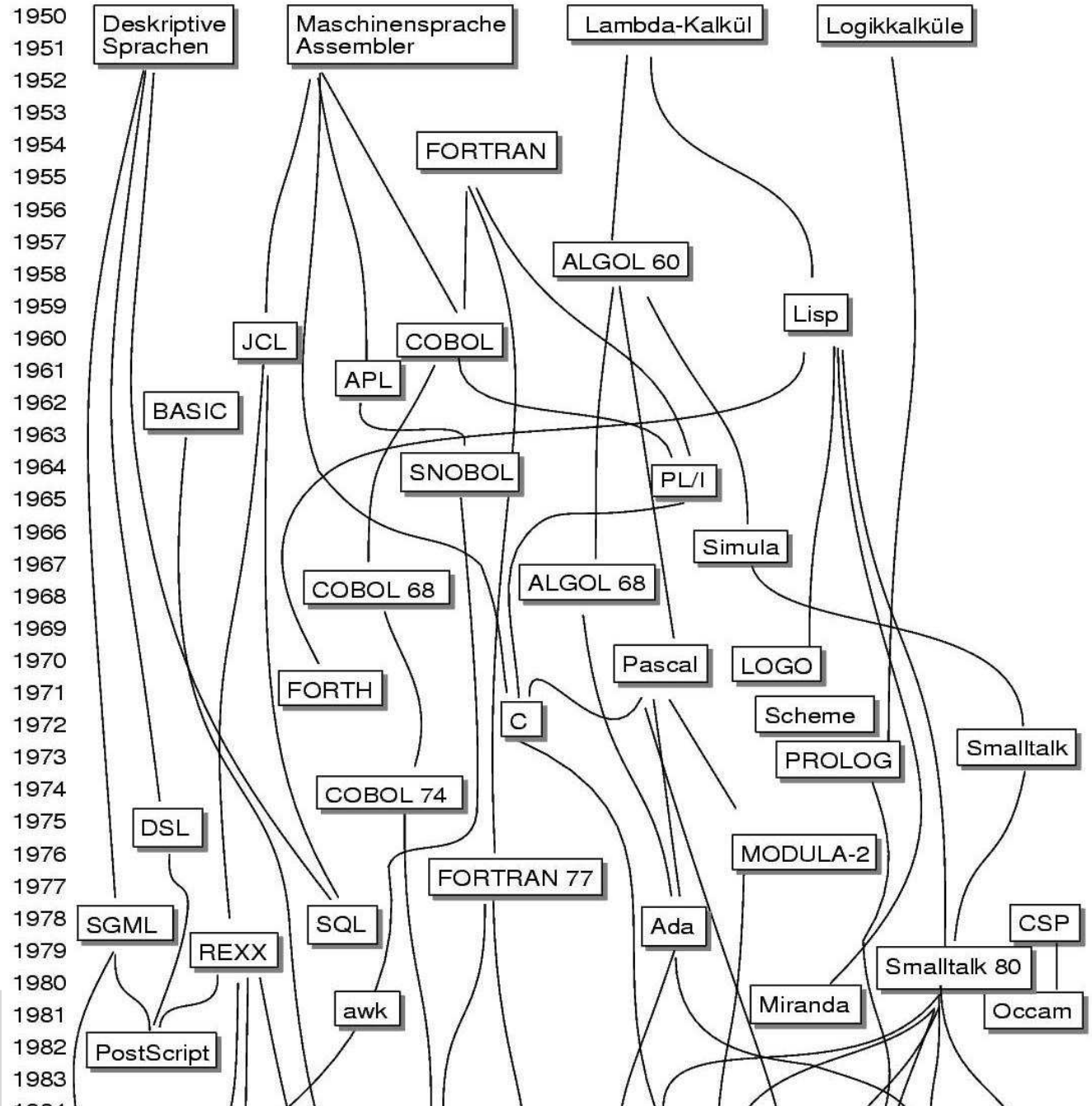
- Hochsprache/höhere Programmiersprache
  $\Leftrightarrow$ Maschinensprache (“0”, “1”); Assembler (`addi #12, d0`)
- Befehle ähneln Umgangssprache (“do . . . while”)
  Kommandos werden mit Hilfe von Compilern in Maschinensprache übersetzt
- Leistungsmerkmale von C/C++
  - schnelle Laufzeit
  - gute Portabilität
  - geringer Sprachumfang
  - ermöglicht objektorientierte Programmierung
  - überladen von Funktionen/Klassen
  - Templates
  - reichhaltige Bibliothek (STL)
  - . . .
  - weit verbreitet

Stammbaum Programmiersprachen

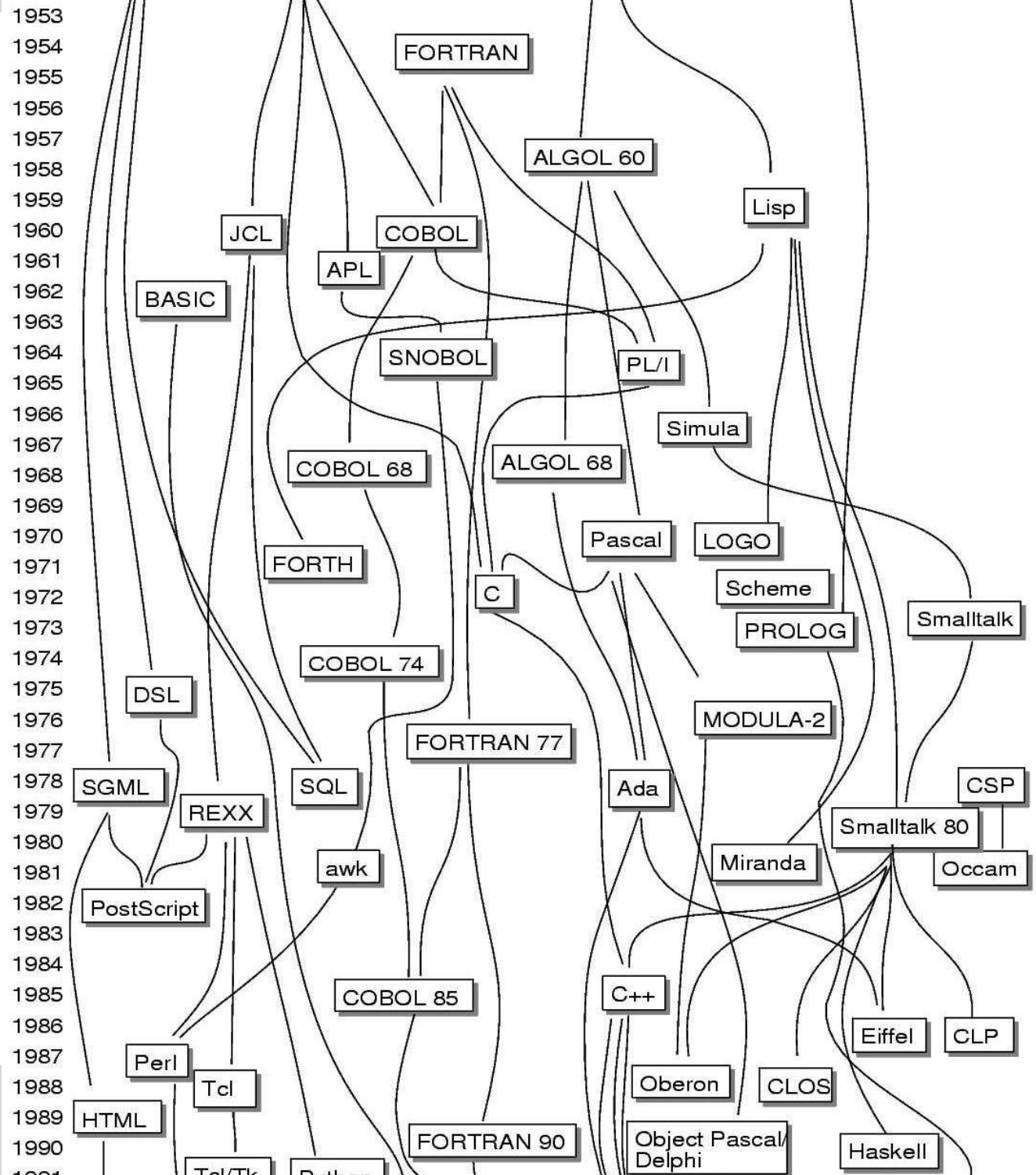
http://goessner.net/learn/inp/2006/lec01/slideous.html



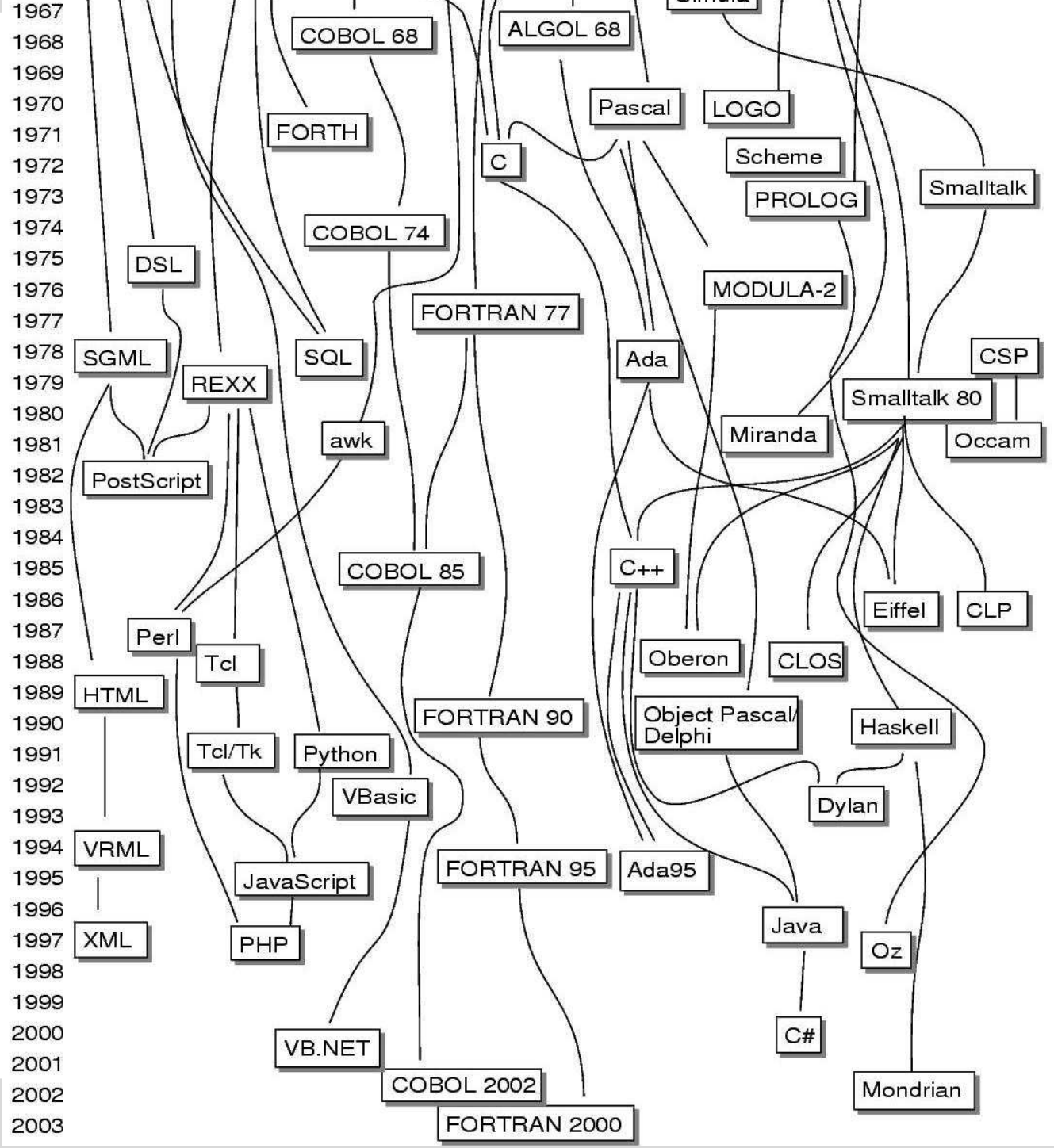
Stammbaum Programmiersprachen



Stamm


1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
FORTRAN
ALGOL 60
Lisp
JCL
COBOL
APL
BASIC
SNOBOL
PL/I
Simula
COBOL 68
ALGOL 68
Pascal
LOGO
FORTH
C
Scheme
PROLOG
Smalltalk
COBOL 74
DSL
MODULA-2
FORTRAN 77
SGML
SQL
Ada
CSP
REXX
Smalltalk 80
Miranda
awk
Occam
PostScript
COBOL 85
C++
Eiffel
CLP
Perl
Tcl
Oberon
CLOS
HTML
FORTRAN 90
Object Pascal/
Delphi
Haskell

Stamm


1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
COBOL 68
ALGOL 68
FORTH
Pascal
LOGO
C
Scheme
PROLOG
Smalltalk
COBOL 74
DSL
MODULA-2
FORTRAN 77
SGML
SQL
Ada
CSP
REXX
Smalltalk 80
awk
Miranda
Occam
PostScript
COBOL 85
C++
Eiffel
CLP
Perl
Tcl
Oberon
CLOS
HTML
FORTRAN 90
Object Pascal/
Delphi
Haskell
Tcl/Tk
Python
VBasic
Dylan
VRML
FORTRAN 95
Ada95
JavaScript
Java
Oz
XML
PHP
C#
VB.NET
COBOL 2002
Mondrian
FORTRAN 2000

II. Grundlagen

Karlsruhe Institute of Technology

1. Das erste Programm

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hallo Karlsruhe!" << endl;
  return(0);
}
```

1. Eintippen, unter 'hallo.cc' abspeichern
2. Compilieren: `g++ -o hallo hallo.cc`
3. Ausführen: `hallo`

II. Grundlagen

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bemerkungen:

(a) `#include`: Anweisung für Präprozessor (kein C++ Befehl); Einlesen von Datei

`iostream`: Ein/Ausgabe

`cmath`: mathematische Funktionen

...

II. Grundlagen

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bemerkungen:

(a) `#include`: Anweisung für Präprozessor (kein C++ Befehl); Einlesen von Datei

`iostream`: Ein/Ausgabe

`cmath`: mathematische Funktionen

...

(b) `using namespace std;`

In C++ gehören alle Variablen, Funktionen, ... zu einem sog. Namensraum ("namespace"). Standardobjekte (z.B. diejenigen, die in `iostream` definiert sind) sind im Namensraum `std` definiert. `using` macht diese im Programm verfügbar.

II. Grundlagen

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bemerkungen:

(a) `#include`: Anweisung für Präprozessor (kein C++ Befehl); Einlesen von Datei

`iostream`: Ein/Ausgabe

`cmath`: mathematische Funktionen

...

(b) `using namespace std;`

In C++ gehören alle Variablen, Funktionen, ... zu einem sog. Namensraum ("namespace").
Standardobjekte (z.B. diejenigen, die in `iostream` definiert sind) sind im Namensraum `std` definiert.
`using` macht diese im Programm verfügbar.

(c) `int main(){ ...}`

Funktionen mit dem Namen `main` muss in jedem Programm genau $1\times$ vorhanden sein.
Beim Programmstart wird diese Funktion ausgeführt.

II. Grundlagen

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)
`<<`: Operator, der auszugebenden Text in `cout` “schiebt”
`endl`: “end line” $\rightarrow$ Zeilenumbruch

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)
`<<`: Operator, der auszugebenden Text in `cout` “schiebt”
`endl`: “end line” $\rightarrow$ Zeilenumbruch
(e) `return(0);`: Verlasse Hauptprogramm; kann bei manchen Compilern weggelassen werden

II. Grundlagen

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)
`<<`: Operator, der auszugebenden Text in `cout` "schiebt"
`endl`: "end line" $\rightarrow$ Zeilenumbruch
(e) `return(0);`: Verlasse Hauptprogramm; kann bei manchen Compilern weggelassen werden
(f) Unterscheidung von Groß- und Kleinschreibung

II. Grundlagen

```
/* hallo.cc
   Ausgabe von Text auf Bildschirm */
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)

`<<`: Operator, der auszugebenden Text in `cout` "schiebt"

`endl`: "end line" $\rightarrow$ Zeilenumbruch

(e) `return(0);`: Verlasse Hauptprogramm; kann bei manchen Compilern weggelassen werden

(f) Unterscheidung von Groß- und Kleinschreibung

(g) Kommentare:

//: ganze Zeile

/* ... */: auch mehrere Zeilen

II. Grundlagen

```
/* hallo.cc
   Ausgabe von Text auf Bildschirm */
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)
`<<`: Operator, der auszugebenden Text in `cout` "schiebt"
`endl`: "end line" $\rightarrow$ Zeilenumbruch
(e) `return(0);`: Verlasse Hauptprogramm; kann bei manchen Compilern weggelassen werden
(f) Unterscheidung von Groß- und Kleinschreibung
(g) Kommentare:
//: ganze Zeile
/* ... */: auch mehrere Zeilen
(h) ";" am Ende von Befehlen

II. Grundlagen

```
/* hallo.cc
   Ausgabe von Text auf Bildschirm */
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)

`<<`: Operator, der auszugebenden Text in `cout` "schiebt"

`endl`: "end line" $\rightarrow$ Zeilenumbruch

(e) `return(0);`: Verlasse Hauptprogramm; kann bei manchen Compilern weggelassen werden

(f) Unterscheidung von Groß- und Kleinschreibung

(g) Kommentare:

//: ganze Zeile

/* ... */: auch mehrere Zeilen

(h) ";" am Ende von Befehlen

(i) { ... } definiert Block (= Zusammenfassung von Anweisungen); kein ";" nach "}"!

II. Grundlagen

```
/* hallo.cc
   Ausgabe von Text auf Bildschirm */
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)
`<<`: Operator, der auszugebenden Text in `cout` "schiebt"
`endl`: "end line" $\rightarrow$ Zeilenumbruch
(e) `return(0);`: Verlasse Hauptprogramm; kann bei manchen Compilern weggelassen werden
(f) Unterscheidung von Groß- und Kleinschreibung
(g) Kommentare:
//: ganze Zeile
/* ... */: auch mehrere Zeilen
(h) ";" am Ende von Befehlen
(i) { ... } definiert Block (= Zusammenfassung von Anweisungen); kein ";" nach "}"!
(j) Bezeichner (Namen für Variablen, Funktionen, ...) werden aufgebaut aus Buchstaben, Ziffern und "_" (Unterstrich)

II. Grundlagen

```
/* hallo.cc
   Ausgabe von Text auf Bildschirm */
#include <iostream>
using namespace std;
int main() {
    cout << "Hallo Karlsruhe!" << endl;
    return(0);
}
```

Bem (Forts.): (d) `cout`: Ausgabe (auf Bildschirm; später mehr)
`<<`: Operator, der auszugebenden Text in `cout` "schiebt"
`endl`: "end line" $\rightarrow$ Zeilenumbruch
(e) `return(0);`: Verlasse Hauptprogramm; kann bei manchen Compilern weggelassen werden
(f) Unterscheidung von Groß- und Kleinschreibung
(g) Kommentare:
//: ganze Zeile
/* ... */: auch mehrere Zeilen
(h) ";" am Ende von Befehlen
(i) { ... } definiert Block (= Zusammenfassung von Anweisungen); kein ";" nach "}"!
(j) Bezeichner (Namen für Variablen, Funktionen, ...) werden aufgebaut aus Buchstaben, Ziffern und "_" (Unterstrich)
(k) Trennzeichen: Leerzeichen, Zeilenende, Tabulator

II. Grundlagen

2. Von der Problemstellung zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

Algorithmus: exakte Verfahrensvorschrift zur Lösung eines Problems; insbesondere zur Realisierung auf einem Rechner

2. Von der Problemstellung zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, welche Variablen, welche Funktionen, . . .)

Algorithmus: exakte Verfahrensvorschrift zur Lösung eines Problems; insbesondere zur Realisierung auf einem Rechner

2. Von der Problemstellung zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, welche Variablen, welche Funktionen, . . .)

↓ ← Editor

❸ Quelltext

Algorithmus: exakte Verfahrensvorschrift zur Lösung eines Problems; insbesondere zur Realisierung auf einem Rechner

II. Grundlagen

2. Von der Problemstellung zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, welche Variablen, welche Funktionen, . . .)

↓ ← Editor

❸ Quelltext

↓ ← Compiler, Linker

❹ lauffähiges Programm

Algorithmus: exakte Verfahrensvorschrift zur Lösung eines Problems; insbesondere zur Realisierung auf einem Rechner

II. Grundlagen

2. Von der Problemstellung zum au

❶ Idee, Modell, Algorithmus

↓

❷ Programmentwurf (Flussdiagran
Funktionen, . . .)

↓ ← Editor

❸ Quelltext

↓ ← Compiler, Linker

❹ lauffähiges Programm

Algorithmus: exakte Verfahrensvor
insbesondere zur Realisierung auf

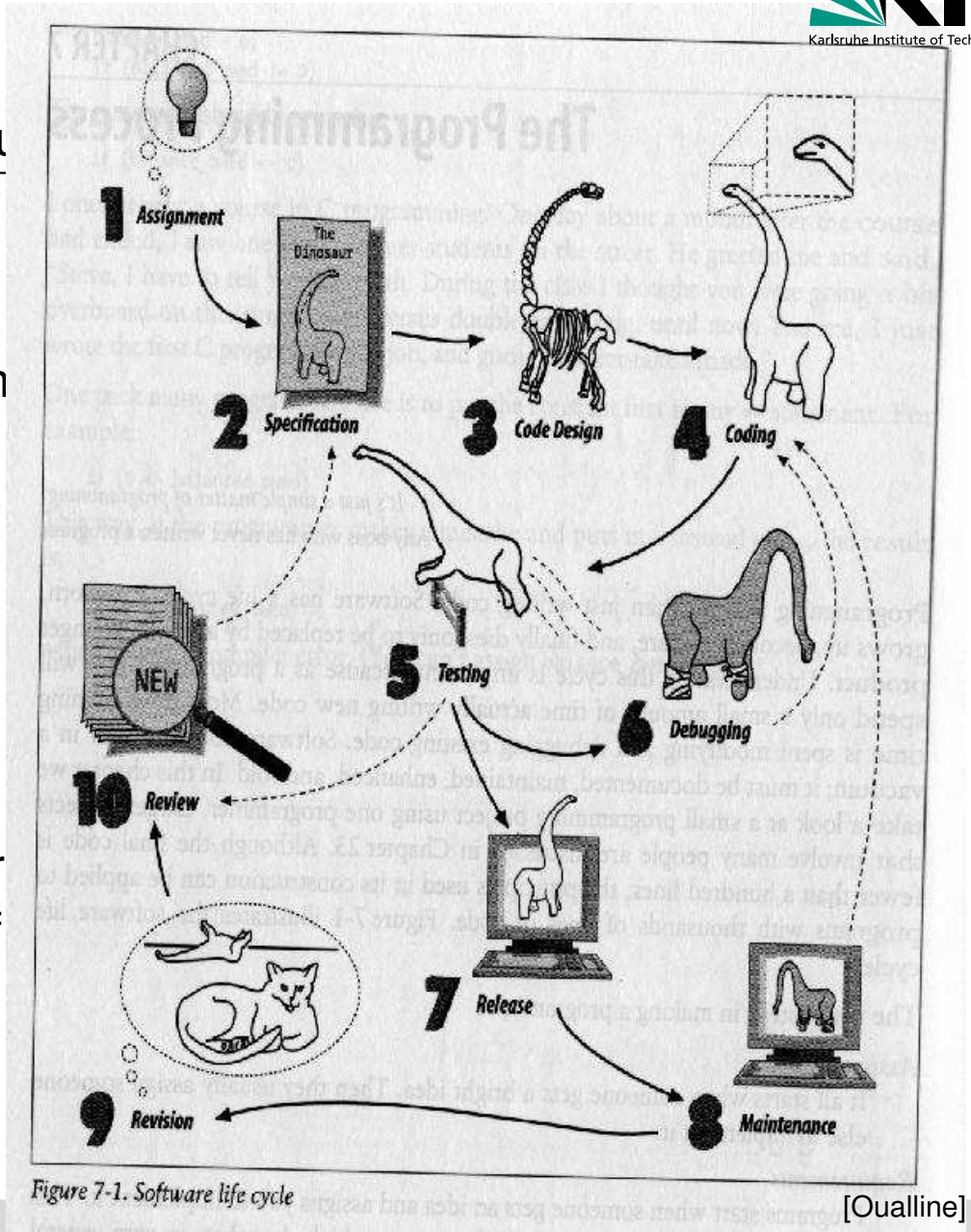


Figure 7-1. Software life cycle

[Oualline]

3. Beispiel: (a) Umrechnung Fahrenheit in Celsius

❶ Idee, Analyse (Modell, Algorithmus)

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, . . .)

↓

❸ Quelltext

↓

❹ lauffähiges Programm

Karlsruhe Institute of Technology

3. Beispiel: (a) Umrechnung Fahrenheit in Celsius

❶ Lese (pos. oder neg.) Zahl ein. Interpretiere sie als Temperatur in Grad-Fahrenheit. Rechne entsprechenden Wert in Celsius aus.
Gebe Ergebnis aus.
Benutze: $T_c = \frac{5}{9}(T_F - 32)$

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, ...)

↓

❸ Quelltext

↓

❹ lauffähiges Programm

Karlsruhe Institute of Technology

3. Beispiel: (a) Umrechnung Fahrenheit in Celsius

❶ Lese (pos. oder neg.) Zahl ein. Interpretiere sie als Temperatur in Grad-Fahrenheit. Rechne entsprechenden Wert in Celsius aus. Gebe Ergebnis aus.
Benutze: $T_c = \frac{5}{9}(T_F - 32)$

↓

❷

1) lies Wert von T_F in Variable `tf`
2) berechne T_c: `tc=5*(tf-32)/9`
3) gebe `tf` und `tc` aus

↓

❸ Quelltext

↓

❹ lauffähiges Programm

Karlsruhe Institute of Technology

3. Beispiel: (a) Umrechnung Fahrenheit in Celsius

❶ Lese (pos. oder neg.) Zahl ein. Interpretiere sie als Temperatur in Grad-Fahrenheit. Rechne entsprechenden Wert in Celsius aus. Gebe Ergebnis aus.
Benutze: $T_c = \frac{5}{9}(T_F - 32)$

↓

❷

1) lies Wert von T_F in Variable `tf`
2) berechne T_c: `tc=5*(tf-32)/9`
3) gebe `tf` und `tc` aus

↓

❸ Schreibe Quellcode in File `fah2cel.cc`:

↓

❹ lauffähiges Programm

II. Grundlagen

3. Beispiel: (a) Umrechnung Fahrenheit in Celsius

❶ Lese (pos. oder neg.) Zahl ein. Interpretiere sie als Temperatur in Grad-Fahrenheit. Rechne entsprechenden Wert in Celsius aus.
Gebe Ergebnis aus.
Benutze: $T_c = \frac{5}{9}(T_F - 32)$

↓

❷

1) lies Wert von T_F in Variable `tf`
2) berechne T_c: `tc=5*(tf-32)/9`
3) gebe `tf` und `tc` aus

↓

❸ Schreibe Quellcode in File `fah2cel.cc`:

↓

❹

```
> g++ -o fah2cel fah2cel.cc
> ./fah2cel
```

3. Beispiel: (b) Berechne $\sqrt{c}$ $(c > 0)$

❶ Idee, Analyse (Modell, Algorithmus)

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, . . .)

↓

❸ Quelltext

↓

❹ lauffähiges Programm

II. Grundlagen

3. Beispiel: (b) Berechne $\sqrt{c}$ $(c > 0)$

❶ Lese positive Zahl c ein. Definiere (numerische) Genauigkeit ϵ. Berechne Nullstelle von $f(x) = x^2 - c$ (Halbierungsverfahren): Wähle Intervall $[a, b]$ mit $f(a) < 0$ und $f(b) > 0$. Definiere $m = (a + b)/2$. Wähle Teilintervall, in dem die Nullstelle liegt und passe a bzw. b an. Variiere a bzw. b solange bis $|a - b| < \epsilon$ erreicht ist. Gebe $a = \sqrt{c}$ aus.

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, ...)

↓

❸ Quelltext

↓

❹ lauffähiges Programm

II. Grundlagen

3. Beispiel: (b) Berechne $\sqrt{c}$ $(c > 0)$

❶ Lese positive Zahl c ein. Definiere (numerische) Genauigkeit ϵ.
Berechne Nullstelle von $f(x) = x^2 - c$ (Halbierungsverfahren): Wähle Intervall $[a, b]$ mit $f(a) < 0$ und $f(b) > 0$. Definiere $m = (a + b)/2$. Wähle Teilintervall, in dem die Nullstelle liegt und passe a bzw. b an. Variiere a bzw. b solange bis $|a - b| < \epsilon$ erreicht ist. Gebe $a = \sqrt{c}$ aus.

↓

❷

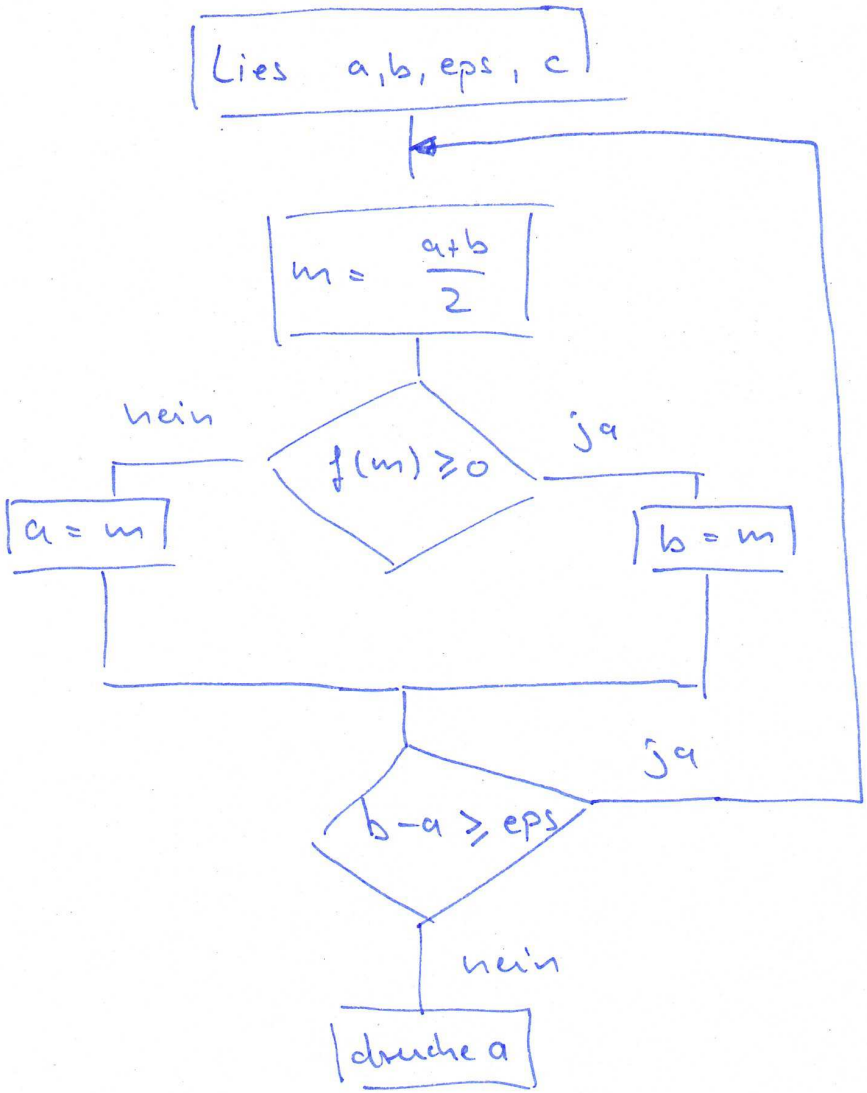
1) Variablen: `a,b,c,eps,m`
2) Lies Werte von a, b, c in Variablen `a,b,c`.
3) Schleife für Intervall-Halbierung; Redefinition von `a,b`.
Ende falls Abstand zwischen `a` und `b` kleiner als `eps`.

↓

❸ Quelltext

↓

❹ lauffähiges Programm



ıle Intervall
eilintervall, in
ange bis

4. Ein- und Ausgabe; wichtige Datentypen

4.1 Standardein- und ausgabe: `cin`, `cout`

- Bsp.:
```
cout << "Bitte Grad in Fahrenheit eingeben:";
cin >> tf;
```
- Ein/Ausgabe mittels Datenstrom (“stream”)
- “`<<`” bzw. “`>>`” gibt “Fluss”-Richtung an; “Schiebeoperatoren”
- `endl` oder “`\n`”: Zeilenumbruch
- Formatierung der Ausgabe; Bsp.: `cout.cc`
- “Standard”: i.d.R. Tastatur und Bildschirm

II. Grundlagen

4. Ein- und Ausgabe; wichtige Datentypen

4.1 Standardein- und ausgabe: `cin`, `cout`

- Bsp.:
```
cout << "Bitte Grad in Fahrenheit eingeben:";
cin >> tf;
```
- Ein/Ausgabe mittels Datenstrom (“stream”)
- “`<<`” bzw. “`>>`” gibt “Fluss”-Richtung an; “Schiebeoperatoren”
- `endl` oder “`\n`”: Zeilenumbruch
- Formatierung der Ausgabe; Bsp.: `cout.cc`
- “Standard”: i.d.R. Tastatur und Bildschirm

4.2 Ein/Ausgabe von/in Datei

Prinzip:

(i) öffne Datei
(ii) lese/schreibe
(iii) schließe “Datenstrom”

Bsp.: `cout_datei.cc`, `cout_datei_2.cc`

II. Grundlagen

4.3. Wichtige Datentypen:

Alle Variablen müssen vor der Verwendung deklariert werden.

Syntax: `<Typ> <Variablenname>;`

(a)

`int`	ganze Zahlen	$-2^{31} \ldots 2^{31} - 1$ $\left[2^{31} - 1 + 2^{31} + 1 = 2^{32}\right]$
`float`	Gleitkommadatentyp	$\pm 1.17 \cdot 10^{-38} \ldots \pm 3.40 \cdot 10^{38}$

siehe auch `double`

`char` Zeichen

Bsp.:

```
int i,j;
float f1,f2,f3;
char ch;
ch = 'A';
char ch2 = 'Z';
```

4.3. Wichtige Datentypen:

Alle Variablen müssen vor der Verwendung deklariert werden.

Syntax: `<Typ> <Variablenname>;`

(a)
Bsp.:

```
int i,j;
float f1,f2,f3;
char ch;
ch = 'A';
char ch2 = 'Z';
```

(b) Felder: Verarbeiten von Daten mit gemeinsamen Eigenschaften

Bsp.:
```
int vek[3]; // 3-dim.  Vektor
vek[0] = 1; vek[1] = -10; vek[2] = 100;
```

4.3. Wichtige Datentypen:

Alle Variablen müssen vor der Verwendung deklariert werden.

Syntax: `<Typ> <Variablenname>;`

(b) Felder: Verarbeiten von Daten mit gemeinsamen Eigenschaften

Bsp.:
```
int vek[3]; // 3-dim.  Vektor
vek[0] = 1; vek[1] = -10; vek[2] = 100;

int mat[4][3]; // Matrix mit 4 Zeilen und 3 Spalten
int mat2[4][3] = {1,2,3, 4,5,6, 7,8,9, 10,11,12};
```

array.cc

4.3. Wichtige Datentypen:

Alle Variablen müssen vor der Verwendung deklariert werden.

Syntax: `<Typ> <Variablenname>;`

(b) Felder: Verarbeiten von Daten mit gemeinsamen Eigenschaften

Bsp.:
```
int vek[3]; // 3-dim.  Vektor
vek[0] = 1; vek[1] = -10; vek[2] = 100;

int mat[4][3]; // Matrix mit 4 Zeilen und 3 Spalten
int mat2[4][3] = {1,2,3, 4,5,6, 7,8,9, 10,11,12};
```
array.cc

Achtung: Indexbereich von “0” bis “Dimension-1”

Syntax: `<Typ> <Arrayname>[<Größe`$_1$`>]...[<Größe`$_n$`>];`