Programmieren für Physiker: C++

SS 2010

Programmieren für Physiker: C++

Termine: SS 2010

- Vorlesung: Mo. 9:45-11:15 Uhr, Lehmann-HS
  (S. Gieseke)
- Hörsaalübungen: Fr. 8:00-9:30 Uhr, Lehmann-HS
  (A. Mildenberger)
- Computerübungen: (voraussichtlich)

  (Fragen stellen; Testate für Aufgaben)

  Di. 14:00-18:00 Uhr,
  Mi. 14:00-18:00 Uhr,
  Poolraum (Flachbau Physik)
  Beginn: Mo. 12. April 2010

`http://comp.physik.uni-karlsruhe.de/Lehre/Programmieren/`

Programmieren für Physiker: C++

Organisatorisches:

- Computerzugang:
  Sie benötigen einen Computeraccount im Poolraum Physik um einen Schein erlangen zu können.
  Bitte beantragen oder verlängern Sie Ihr Poolraumkonto bis spätestens Fr. 16.04.2010.
- Scheinkriterium:
  Auf den Übungsblättern sind einige Aufgaben als Pflichtaufgaben markiert.
  80% erfolgreich gelöster Pflichtaufgaben berechtigen zur Teilnahme an der Scheinklausur.
- Bachelor: Anmeldung zur Klausur im Studierendenportal.
- Klausurtermin: Di 13.07.2010, 17.30 Uhr.

Inhalt

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/`
- online: „SGI's STL Programmer's Guide“ `http://www.sgi.com/tech/stl/`

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

I. Einleitung

Rechner

Hardware: Monitor, Drucker, . . . , Speicher im Rechner

Software: Betriebsystem (Unix, Windows, . . .)

Anwenderprogramme (Firefox, Word, LaTeX, . . .)

Programmiersprachen (Fortran, Pascal, C/C++, . . .)

I. Einleitung

Rechner

Hardware: Monitor, Drucker, . . . , Speicher im Rechner

Software: Betriebsystem (Unix, Windows, . . .)

Anwenderprogramme (Firefox, Word, LaTeX, . . .)

Programmiersprachen (Fortran, Pascal, C/C++, . . .)

Programmiersprachen:

Maschinensprache (“0”, “1”)

Assembler (`addi #12, d0`)

Hochsprachen

I. Einleitung

Rechner

Hardware: Monitor, Drucker, . . . , Speicher im Rechner

Software: Betriebsystem (Unix, Windows, . . .)

Anwenderprogramme (Firefox, Word, LaTeX, . . .)

Programmiersprachen (Fortran, Pascal, C/C++, . . .)

Programmiersprachen:

Maschinensprache (“0”, “1”)

Assembler (`addi #12, d0`)

Hochsprachen

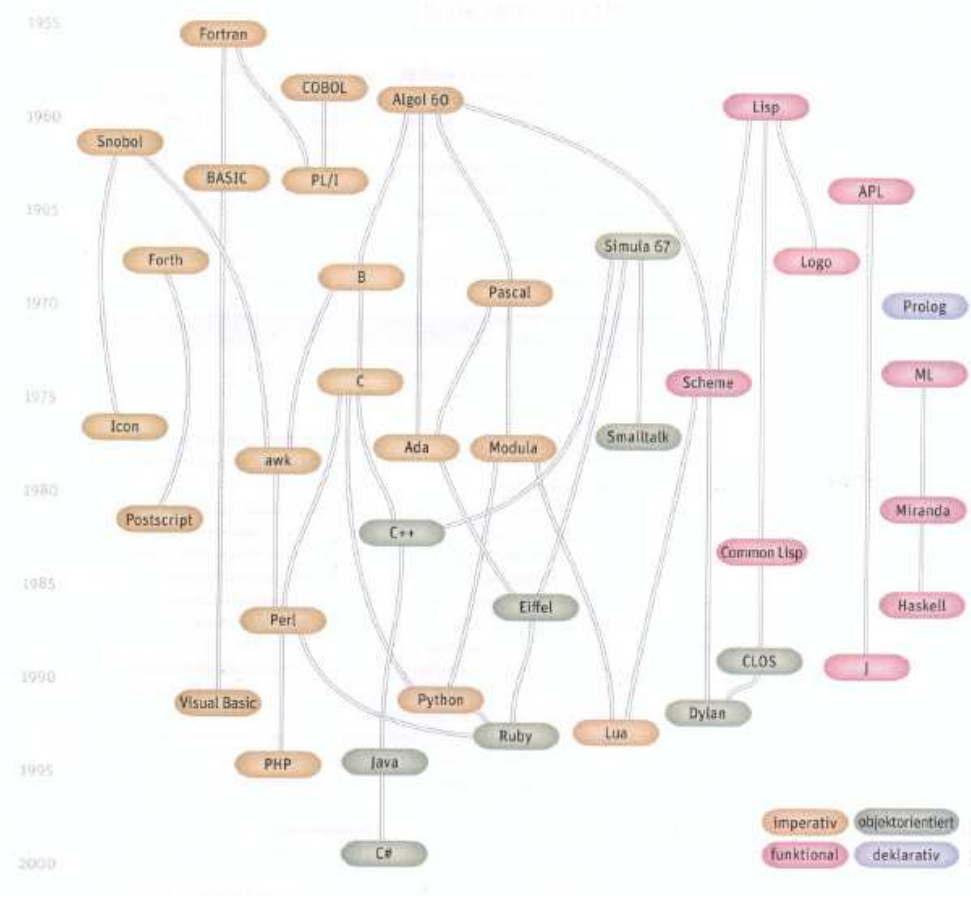
Hochsprachen:

— Befehle ähneln Umgangssprache (z.B.: “Tue . . . solange” $\leftrightarrow$ “do . . . while”)

— Kommandos werden mit Hilfe von Compilern in Maschinensprache übersetzt

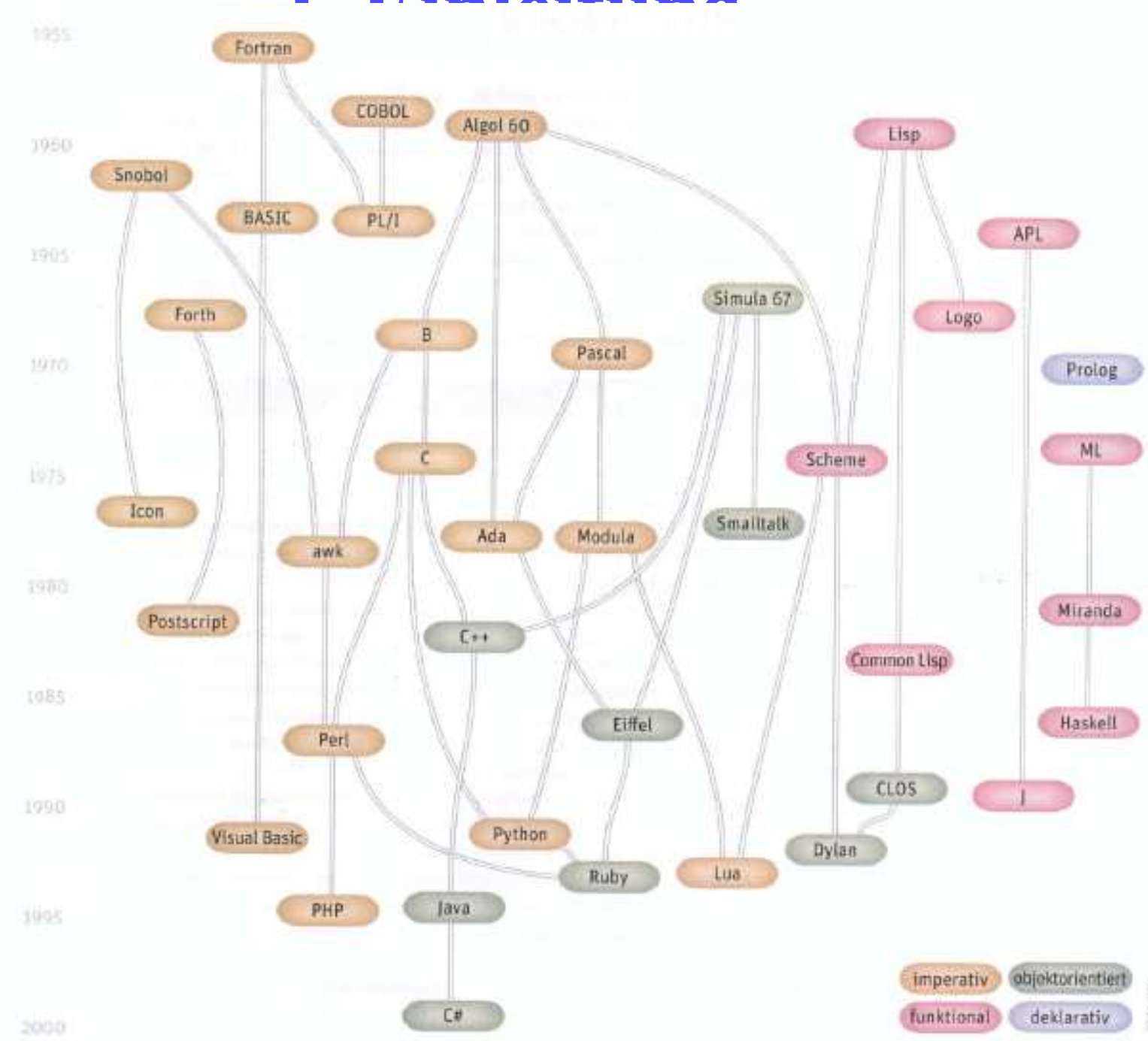
I. Einleitung

- Es gibt 6912 bekannte, auf der Erde gesprochene Sprachen
  Entwicklungszeit: mehrere 1000 Jahre
- Es gibt zwischen 2500 und 8500 Programmiersprachen (je nachdem wer zählt ...)
  Entwicklungszeit: ca. 50 Jahre



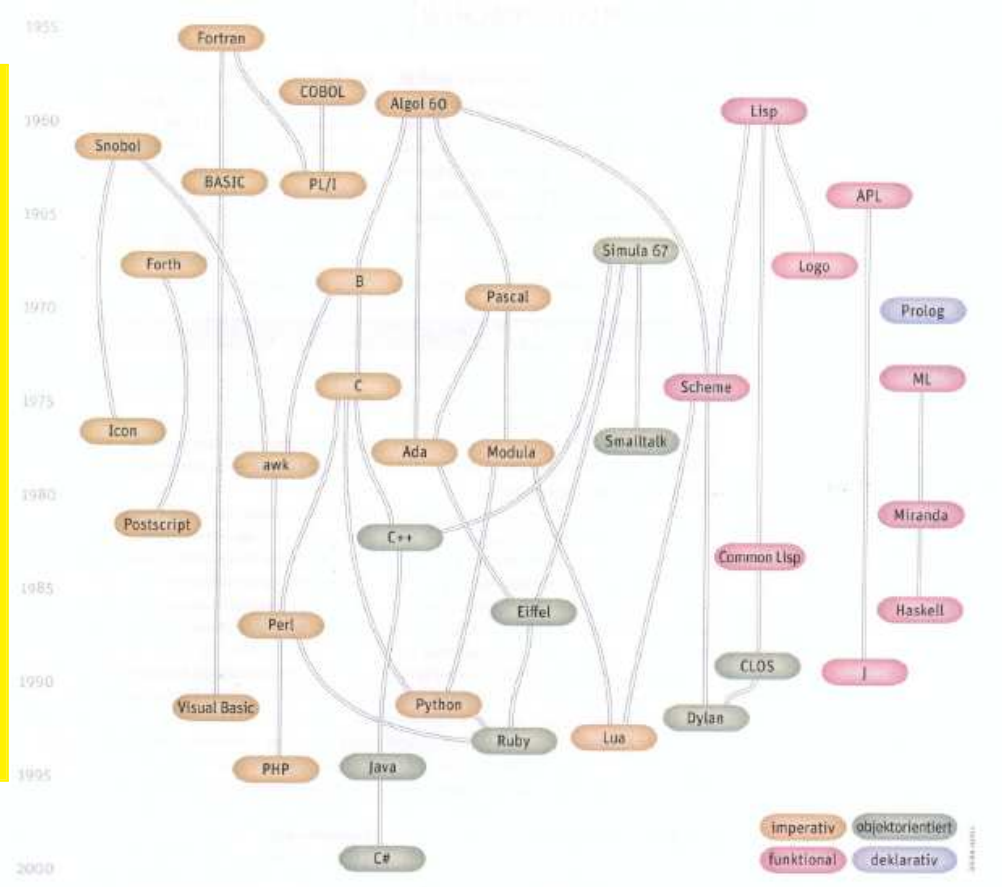
[Spektrum d. Wissenschaft Juni 07]

I. Einleitung



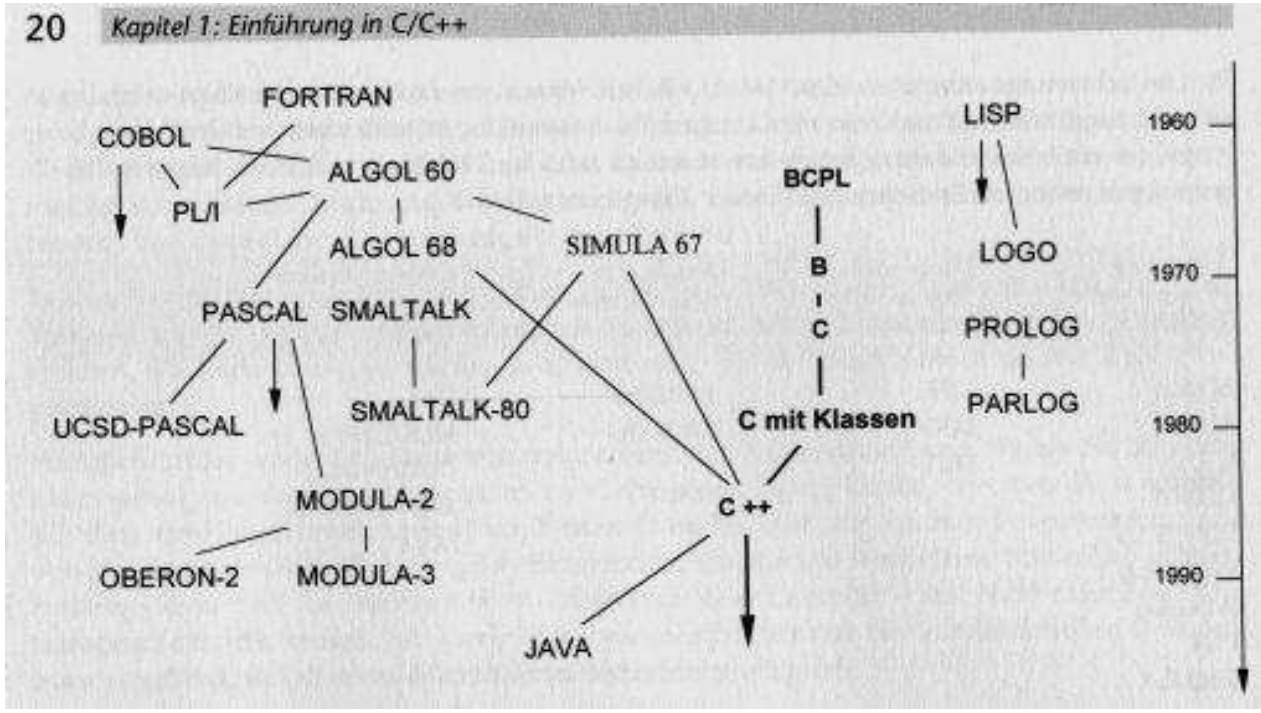
I. Einleitung

imperativ: basiert auf Befehlen
funktional: Funktionen mit Input und Output
objektorientiert: Befehle und Daten werden zu geschlossener Struktur verbunden
deklarativ: Programmierer schreibt Fakten und Relationen vor; Computer zieht daraus Schlüsse, um Fragen zu beantworten.



[Spektrum d. Wissenschaft Juni 07]

I. Einleitung



[Nootz/Morick]

I. Einleitung (2)

Entwicklungsgeschichte von C/C++

1967	BCPL	(Basic Combined Programming Language)
		Martin Richards
1970	B	frühe Implementierung von Unix
		Ken Thompson
1972	C	neue Version von B; Impl. von Unix
		Dennis Ritchie, Brian Kernighan
1983-1988	ANSI-C	(American National Standards Institute)
		eindeutige, maschinenunabh. Definition von C
seit 1980	C++	“C mit Klassen” $\rightarrow$ C++
		Bjarne Stroustrup
		Wesentlicher Unterschied: Erweiterung zur OO-Programmierung
1998		Standardisierung von C++

I. Einleitung (3)

Unterschiede zwischen Programmiersprachen

- Grundprinzip: imperativ, funktional, objektorientiert, deklarativ
- “high-level” vs. “low-level”
- Äußere Erscheinung: “wortkarg” vs. “ausführlich”; geschwungene Klammern; Groß- und Kleinschreibung; . . .
- Zielgruppe: Bsp.:
  Fortran = FORmula TRANslator
  Cobol = COmmon Business Oriented Language

I. Einleitung (4)

Leistungsmerkmale von C/C++

- schnelle Laufzeit
- gute Portabilität
- geringer Sprachumfang
- ermöglicht objektorientierte Programmierung
- überladen von Funktionen/Klassen
- Templates
- reichhaltige Bibliothek (STL)
- . . .
- weit verbreitet

II. Grundlagen

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 (2)

```
#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 (2)

```
#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 (2)

```
#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 (3)

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

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

II. Grundlagen (3)

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

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

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

II. Grundlagen (3)

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

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

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

(f) Unterscheidung von Groß- und Kleinschreibung

II. Grundlagen (3)

```
/* 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)

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

(f) Unterscheidung von Groß- und Kleinschreibung

(g) Kommentare:

//: ganze Zeile

/* . . . */: auch mehrere Zeilen

II. Grundlagen (3)

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

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

(e) `return(0);` 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 (3)

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

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

(e) `return(0);` 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 “;”!)

II. Grundlagen (3)

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

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

(e) `return(0);` 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 “;”!)

(j) Bezeichner (Namen für Variablen, Funktionen, . . .) werden aufgebaut aus Buchstaben, Ziffern und “_” (Unterstrich)

II. Grundlagen (3)

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

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

(e) `return(0);` 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 “;”!)

(j) Bezeichner (Namen für Variablen, Funktionen, . . .) werden aufgebaut aus Buchstaben, Ziffern und “_” (Unterstrich)

(k) Trennzeichen: Leerzeichen, Zeilenende, Tabulator

II. Grundlagen (4)

2. Von der Hochsprache zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

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

II. Grundlagen (4)

2. Von der Hochsprache zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

↓

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

II. Grundlagen (4)

2. Von der Hochsprache zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

↓

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

↓ ← Editor

❸ Quelltext

II. Grundlagen (4)

2. Von der Hochsprache zum ausführbaren Programm

❶ Idee, Modell, Algorithmus

↓

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

↓ ← Editor

❸ Quelltext

↓ ← Compiler, Linker

❹ lauffähiges Programm

II. Grundlagen (4)

2. Von der Hochsprache zum ausführbaren Pr

❶ Idee, Modell, Algorithmus

↓

❷ Programmentwurf (Flussdiagramm, Datens
Funktionen, . . .)

↓ ← Editor

❸ Quelltext

↓ ← Compiler, Linker

❹ lauffähiges Programm

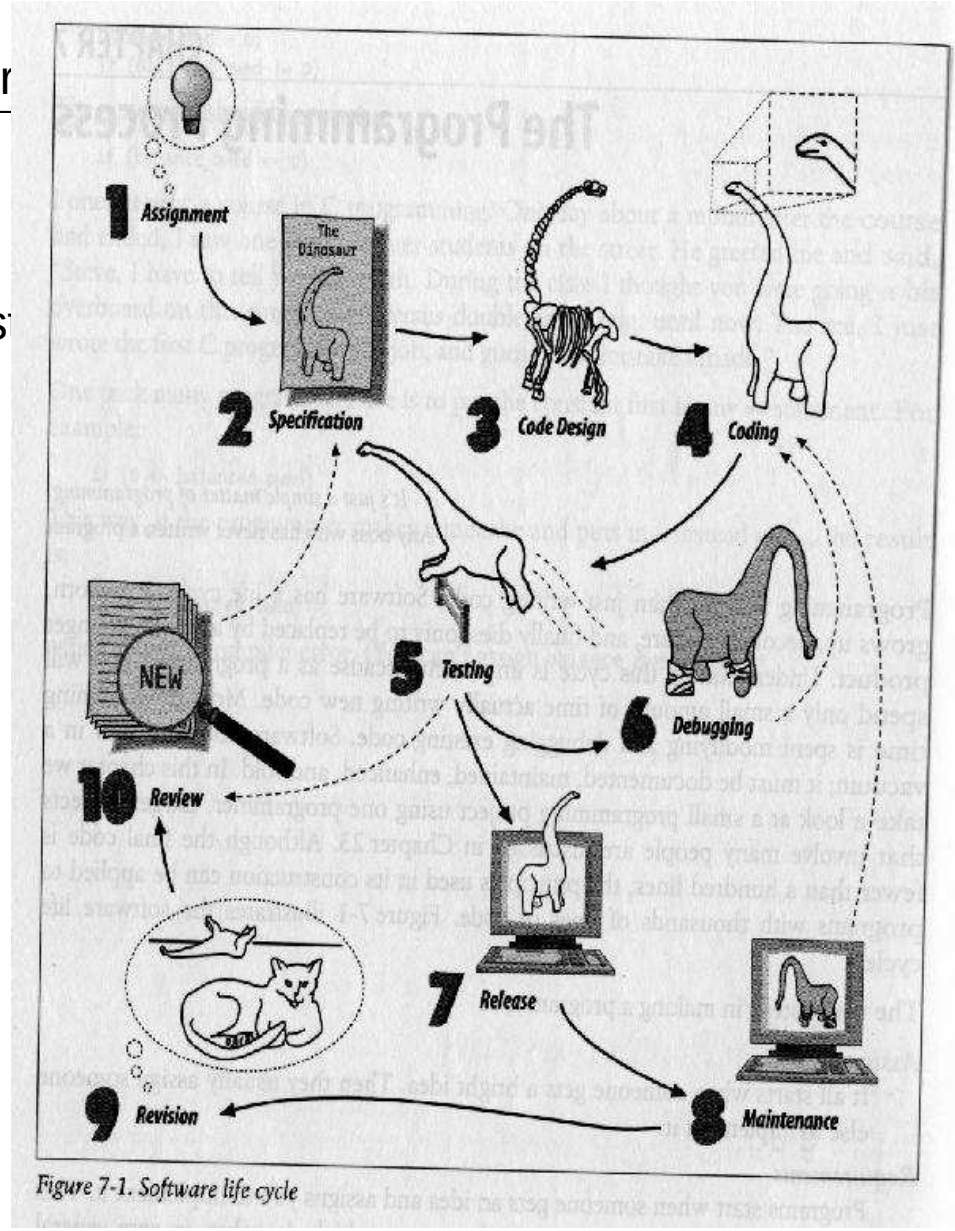


Figure 7-1. Software life cycle

[Qualline

II. Grundlagen (5)

3. Beispiel: Umrechnung Fahrenheit in Celsius

❶ Idee, Analyse (Modell, Algorithmus)

↓

❷ Programmentwurf (Flussdiagramm, Datenstruktur, . . .)

↓

❸ Quelltext

↓

❹ lauffähiges Programm

II. Grundlagen (5)

3. Beispiel: 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

II. Grundlagen (5)

3. Beispiel: 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

II. Grundlagen (5)

3. Beispiel: 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 (5)

3. Beispiel: 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
```

II. Grundlagen (6)

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
- “Standard”: i.d.R. Tastatur und Bildschirm
- Formatierung der Ausgabe; Bsp.: `cout_form.cc` (mehr in den Übungen)

II. Grundlagen (6)

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
- “Standard”: i.d.R. Tastatur und Bildschirm
- Formatierung der Ausgabe; Bsp.: `cout_form.cc` (mehr in den Übungen)

4.2 Ein/Ausgabe von/in Datei

Prinzip:
(i) öffne Datei
(ii) lese/schreibe
(iii) schließe “Datenstrom”
Bsp.: `datei.cc`

II. Grundlagen (7)

4.3. Wichtige Datentypen

(a)	`int`	ganze Zahlen	$-2^{31} \ldots 2^{31} - 1$	32 Bit
			$\left[2^{31} - 1 + (2^{31} + 1) = 2^{32}\right]$	
	`float`	Gleitkommadatentyp	$\pm 1.17 \cdot 10^{-38} \ldots \pm 3.40 \cdot 10^{38}$	32 Bit
	`char`	Zeichen		8 Bit

Syntax: `Typ Variablenname;`

Bsp.:

```
char ch;
ch = 'A';
```

oder:

```
char ch = 'A';
```

II. Grundlagen (7)

4.3. Wichtige Datentypen

(a)	`int`	ganze Zahlen	$-2^{31} \ldots 2^{31}-1$ $\left[2^{31}-1+(2^{31}+1)=2^{32}\right]$	32 Bit
	`float`	Gleitkommadatentyp	$\pm 1.17 \cdot 10^{-38} \ldots \pm 3.40 \cdot 10^{38}$	32 Bit
	`char`	Zeichen		8 Bit

Syntax: `Typ Variablenname;`

Bsp.:

```
char ch;
ch = 'A';
```

oder:

```
char ch = 'A';
```

(b) Felder

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

Bsp.:

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

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