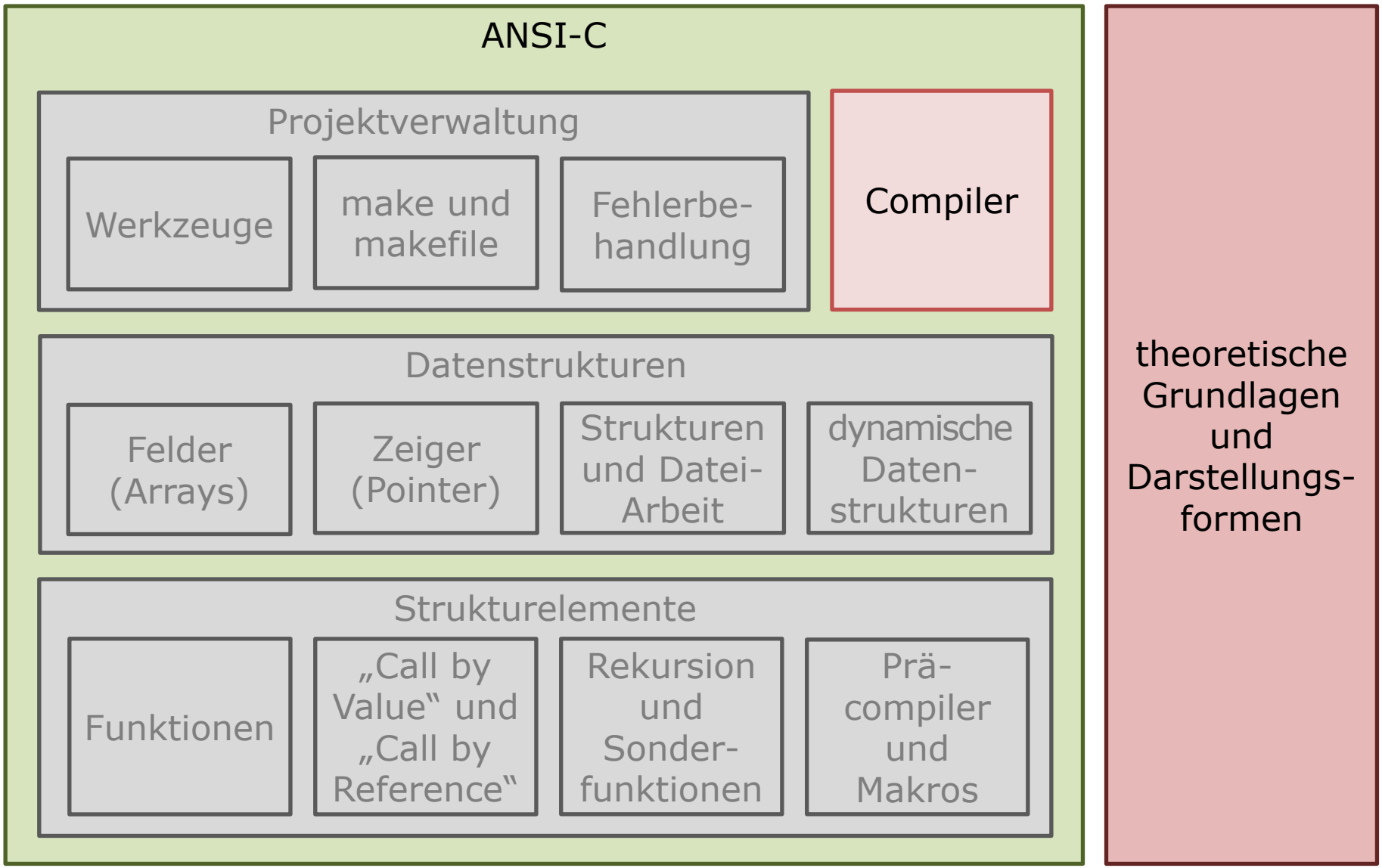
BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Imperative Programmierung

Programmier-hinweise

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung



Wiederholung Code Style

diverse Empfehlungen und Firmenrichtlinien zur optischen Kodegestaltung

- haben (wenn richtig definiert) keinen Einfluss auf Syntax des Programms
- verfolgen in der Regel gemeinsame Ziele
  - einheitliche Gestaltung bei mehreren Projektbeteiligten
  - leichte Lesbarkeit des Kodes
  - leichte Identifikation der Semantik
  - Automatisierbarkeit von Aufgaben (bspw. Dokumentation generieren)

Beispiel: https://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html

→ vergleiche

```
for(a=1;a<10;a=a+1){printf("A: %d\n",a);}
```

mit

```
for ( a=1; a<10; a=a+1 ) {
   printf("A: %d\n", a);
}
```

Programmierhinweise

- Softwareentwicklung ist mehr als Implementieren von Algorithmen
- Prozess der Softwareentwicklung umfasst Problemanalyse, Planung, Implementation, Dokumentation, Tests, Einführung und Pflege/Wartung
- Softwareentwicklung ist i.d.R. Teamarbeit → Projektmanagement
- Ableitung von Regeln, Schreibweisen, Richtlinien, … für Programmierung
- Einhaltung von Richtlinien fördert Verständlichkeit und Übersichtlichkeit
- Richtlinien erleichtern die Fehlersuche bereits zur Programmierung

Kommentare

- Kommentare am Anfang von Funktionen dienen zur Identifizierung (Datum, Version, Name des Autors, usw.)
- Kommentieren hilft schon bei der Überlegung nach der richtigen Formulierung Denkfehler zu identifizieren
- Zeitaufwand für Kommentare gering ggü. Zeitaufwand für Fehlersuche (keine Entschuldigung, für das „Einsparen“ von Kommentaren!)
- selbst wenn für Programmteile bereits feststeht, dass sie später wieder gelöscht werden, ist das Kommentieren keine Zeitverschwendung (oft deuten „TODO“-Kommentare auf Vergessenes hin)
- Kommentare helfen dabei, dass Gedanken nachvollziehbar bleiben

Hinweise zur Schreibweise (Syntax)

- Softwarefirmen haben oft sehr genaue und restriktive Regeln für die Schreibweise von Programmen
- bereits bei ersten Programmierübungen sollten Mindest-Anforderungen erfüllt werden
  - bei eigenen Namen (Funktionsnamen, Variable, Makros, ...) sollte einheitlich entweder Deutsch oder Englisch gewählt werden
  - für die Schreibweise von Variablen gibt es allgemein bekannte Richtlinien, z.B. ungarische Notation (idFirst)
  - Namen aus Präcompiler-Direktiven und (echte) Konstanten werden meist in Großbuchstaben geschrieben (ANZ, MAX, MIN, ...)
  - möglichst selbsterklärende/selbstsprechende Namen verwenden
  - Einrückungen einheitlich, d.h. entweder exklusiv Leerzeichen oder exklusiv Tabulatoren
  - feste Anzahl von Leerzeichen bzw. Tabulatoren in allen Ebenen

Hinweise zur Schreibweise (Syntax)

```
if ( /* ... */ )
{
   // unter C-Entwicklern weit verbreitet
}

if ( /* ... */ ) {
   // unter Java-Entwicklern weit verbreitet
}

if ( /* ... */ )
   {
   // Block-Code
   }
```

viele weitere Varianten… → Wichtig: durchgängig eine Variante zu nutzen!

Hinweise zum Programmierstil

- in normalen Anwendungen ist Ausnutzung aller Möglichkeiten und Tricks (z.B. Seiteneffekte) der Programmiersprache nebensächlich
  - Programmiertricks *können* ein Programm effizient machen, **aber**
  - Programmiertricks machen Kode für andere schwer lesbar (erschwert Fehlersuche, Teamarbeit, Wartung und Erweiterungen)
  - Übertragbarkeit in andere Programmiersprachen kann dadurch extrem erschwert werden (z.B. von C nach Java)
- Beispiel

```
while (*zieltext++ = *quelltext++) ;
```

→ Ist für interne Systembibliothek u.U. gerade noch wegen Effizienz tragbar, aber für „normale" Anwendungen keineswegs zu empfehlen

Hinweise zum Programmierstil (Semantik)

- ressourcenverschwendende Programmierweise wird nicht geduldet (Ausnahme: Lastgeneratoren, Target-Size-Code)
- Beispiel

```
/* ... */
char quelltext[80+1], zieltext[80+1];
/* ... */
for (i = 0; i <= 80; i++)
    zieltext[i] = quelltext[i];
```

  - funktioniert zwar, ist aber unsinnig ineffizient

- Programmteile, die trivialerweise besser (effizienter und/oder eleganter) schreibbar sind, sollten auch so geschrieben werden
- es lohnt sich immer, über eventuell bessere Realisierung und Implementation nachzudenken → häufiges Refactoring einplanen!