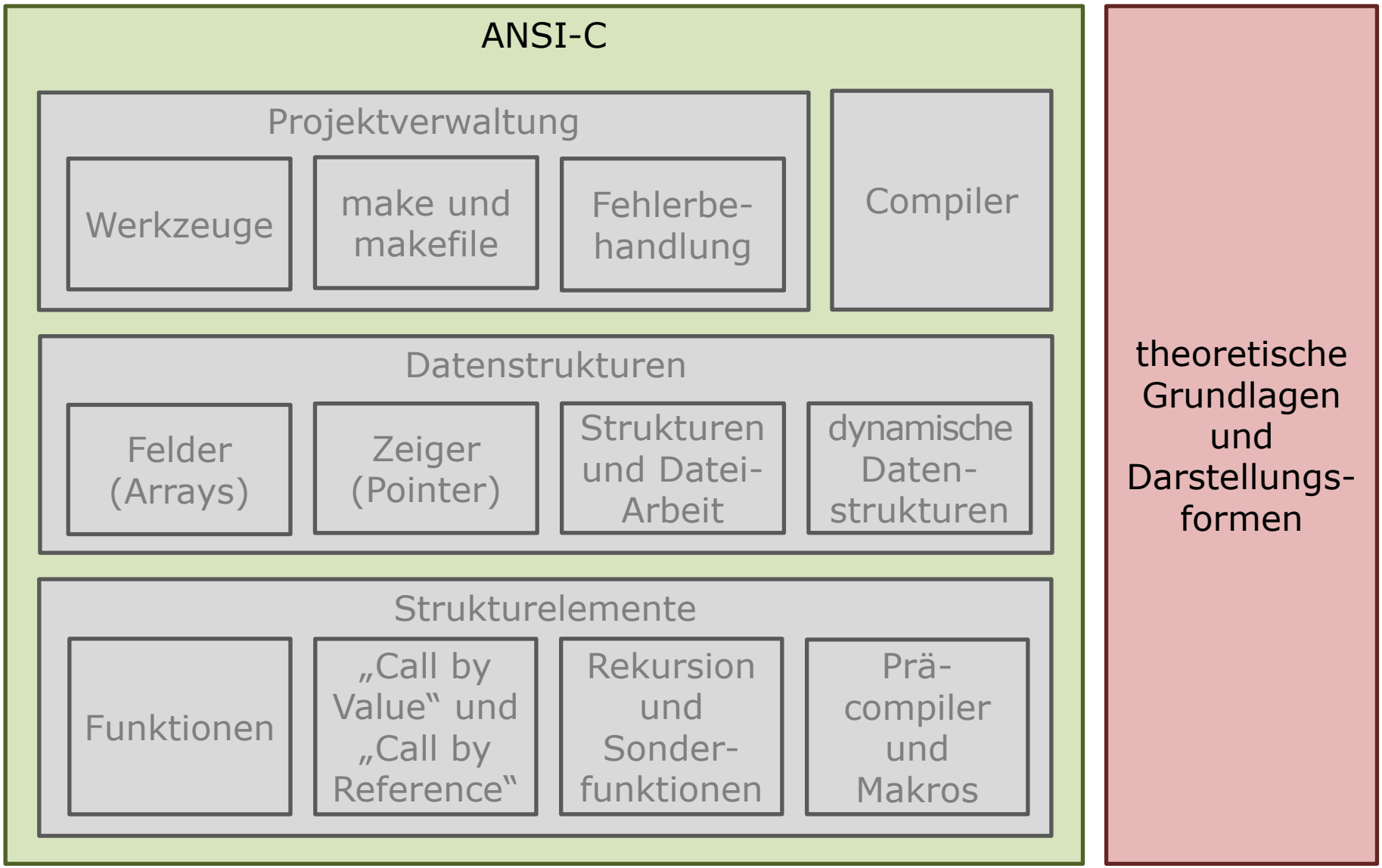
BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Imperative Programmierung

ANSI-C

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

Aufbau der Lehrveranstaltung



The Spirit of C (ANSI-Komitee)

- Keep it small and simple (KISS)
  (Halte es klein und einfach)
- Trust the programmer, don’t do any runtime checks
  (Vertraue dem Programmierer, mache keine Runtime-Checks)
- Make it fast, even if it is not guaranteed to be portable
  (Mache es schnell, auch wenn es möglicherweise nicht portabel ist)
- Provide only one way to do an operation (orthogonal)
  (Stelle nur einen Ausführungsweg für Operation zur Verfügung)
- Don't prevent the programmer from doing what needs to be done
  (Hindere den Programmierer nicht daran, Notwendiges zu tun)

Interpretation des „Geistes von C“

- Programmierer wissen was sie tun und tragen dafür die Verantwortung
- größtmögliche Freiheiten (Formatfreiheit, wenige Einschränkungen)
- Sprache klein und einfach halten
  - 32 Schlüsselwörter und 44 Operatoren bei C89/C90
  - seit C99 zusätzlich `_Bool`, `_Imaginary`, `restrict`, `Complex`, `inline`
- mit wenigen Zeichen viel ausdrücken (Redundanzfreiheit)
- effektive und effiziente Programmierung
  (Arbeit mit Adressen, Bitoperatoren, ...)

Eigenschaften von C (1/3)

…als höhere Programmiersprache (Middle-Level Language)

- Datenobjekte: Zeichen (Character), ganze Zahlen (Integer) und rationale Zahlen (Float)
- zusammengesetzte Datenobjekte: Zeiger, Felder (Array, Vector, String), Verbund (Structure), Variante (Union), Funktionen
- Arithmetik, Vergleichsoperatoren
- strukturierte Sprache mit Kontrollstrukturen
- globale und lokale Variablen
- unabhängig von Betriebssystem und Rechner(architektur)

Eigenschaften von C (2/3)

…als Systemprogrammiersprache (Assembler-Ebene)

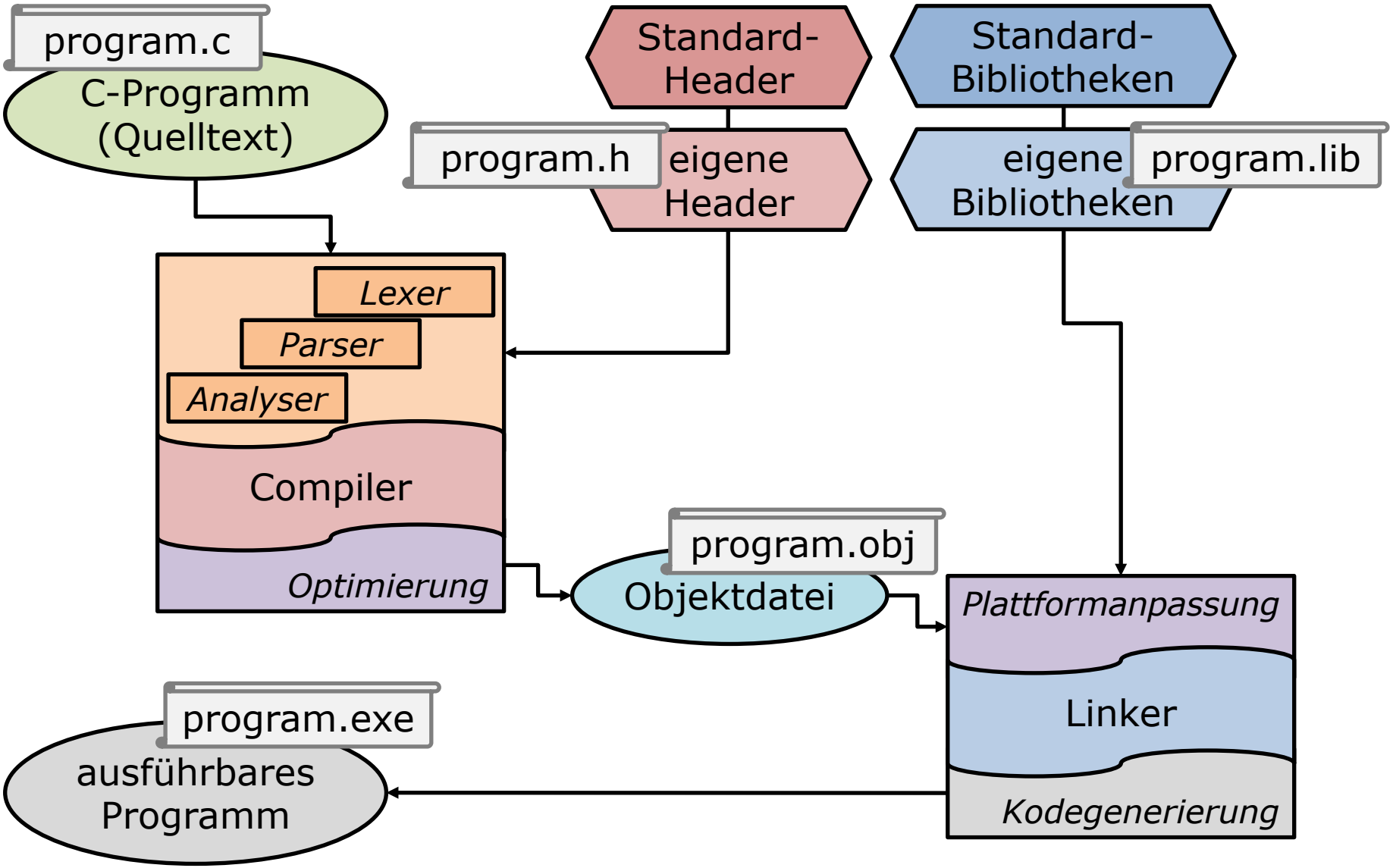
- wenig Fehlerüberprüfung zur Laufzeit
- nicht streng typgebunden (automatisches Casting)
- Datenobjekte entsprechend der meisten Maschinen (Zeichen, Zahlenwerte, Adressen)
- keine Elemente für E/A (I/O), Dateizugriffe, Speicherverwaltung und Felder
- Adressenarithmetik (Zeigerarithmetik)
- hardwarenahe Befehle (Bit-Operatoren, Register)

Eigenschaften von C (3/3)

…im weiteren Sinne

- C ist portable Sprache (leicht auf andere Betriebssysteme und Rechner(architekturen) übertragbar)
- funktionenorientierte Sprache (auch E/A-Operationen, Speicherverwaltung und Bearbeitung von Feldern als Funktionen)
- C ist vielseitig einsetzbare, universelle Programmiersprache (General Purpose Programming Language)
- entwickelt von Systemprogrammierern für Systemprogrammierer

Phasen der Programmentwicklung (1/2)



Phasen der Programmentwicklung (2/2)

1. Lexikalische Analyse (Lexer):
   Zerteilen des Quelltextes in Token; deren Quelltextposition assoziieren
2. Syntaktische Analyse (Parser):
   Umwandeln in Syntaxbaum; Prüfen auf korrekte Struktur
   („Entspricht Programm kontextfreier Grammatik der Quellsprache?“)
3. Semantische Analyse (Analyser):
   Attribuieren des Syntaxbaums; Prüfen auf Deklaration und Typisierung
4. Zwischencodeerzeugung:
   maschinennahen Zwischenkode aus Syntaxbaum erzeugen;
   Ausführungsplan erstellen und optimieren; Objektkode erzeugen
5. Programmoptimierung:
   Anpassung und Optimierung des Objektkodes an die Zielplattform
6. Kodegenerierung:
   Zusammenführung aller Module mit Objektkode;
   Erzeugen des ausführbaren Binärkodes

Erstes C-Programm („hallo.c")

- Einstellen der Entwicklungsumgebung (Editor, ANSI-Quellcode und Verzeichnisse)
- Arbeitsfähigkeit herstellen

```c
/* erstes C-Programm */

#include <stdio.h>

int main(void)
{
   printf("Hallo Welt!\n");
   return 0;
}
```

Kommentar

Einbindung von Header-Dateien (hier: Ein-/Ausgabe)

Haupt-Funktion

Ausgabefunktion

Rückgabewert

KÜRZESTES C-PROGRAMM

- mit Warnungen

```
main(){}
```

- ANSI-konform

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

- Achtung: die Semantik der Kontrollstrukturen wird durch Programmablaufpläne (bspw. PAP nach DIN 66 001) beschrieben.

Grundsätzlicher Aufbau (1/3)

- Sammlung von Funktionen, auch über mehrere Dateien (Module)
  - mindestens eine Funktion, nämlich `main`
  - genau eine Funktion `main` in einem Programm
  - meist Aufrufe von Bibliotheksfunktionen im Funktionsblock
- Funktionen bestehen aus:
  - Typ des Rückkehrwertes (**`return`**-Wert)
  - Name der Funktion
  - Parameterliste (Argumente in runden Klammern)
  - Funktionsblock (oder einfach Block)

Grundsätzlicher Aufbau (2/3)

- außerhalb der Funktionsdefinitionen stehen
  - Anweisungen für Präcompiler (`#include`, `#define`, ...)
  - Funktionsdeklarationen (Prototypen)
  - globale und externe Variablen
- außerdem können dort stehen
  - Typ-Definitionen
  - Kommentare (diese können überall stehen)
- Kommentare
  - `/* … */` auch über mehrere Zeilen; muss mit `*/` terminiert werden
  - `// …` ab C99; endet mit nächstem Zeilenumbruch

Grundsätzlicher Aufbau (3/3)

- ein Block besteht aus
  - öffnender und schließender geschweifter Klammer
  - Vereinbarungen (Definition von Variablen und Feldern); müssen am Anfang des Blockes stehen
  - Anweisungen bzw. Kontrollstrukturen
  - weitere Blöcke (Blöcke können geschachtelt werden)
- ab C99 können Vereinbarungen und Anweisungen in `for`-Schleifen gemischt untereinander stehen
- EBNF-Syntax eines Blockes:

```
<Block> ::= "{" ⊙
                { <Vereinbarung> ⊙ }
                { ( <Anweisung> | <Block> ) ⊙ }
            "}"
```

Elementare Datentypen

ganzzahlige Datentypen

- `short` (short int) 2 Byte $-2^{15} \ldots +2^{15}-1$
- `unsigned short` 2 Byte $0 \ldots 2^{16}-1$
- `int` 2 Byte, 4 Byte oder 8 Byte (implementationsabhängig)
- `long` (long int) 4 Byte $-2^{31} \ldots +2^{31}-1$
- `unsigned long` 4 Byte $0 \ldots 2^{32}-1$
- `char` 1 Byte $-128 \ldots +127$
- `unsigned char` 1 Byte $0 \ldots 255$

Elementare Datentypen

Gleitkommazahlen

- `float` 4 Byte $\approx -3{,}4 \cdot 10^{38}$ … $\approx 3{,}4 \cdot 10^{38}$
  mindestens 6 Stellen Genauigkeit (dezimal)
- `double` 8 Byte $\approx -1{,}7 \cdot 10^{308}$ … $\approx 1{,}7 \cdot 10^{308}$
  mindestens 15 Stellen Genauigkeit (dezimal)
- `long double` 16 Byte $\approx -1{,}1 \cdot 10^{4.932}$ … $\approx 1{,}1 \cdot 10^{4.932}$
  mindestens 19 Stellen Genauigkeit (dezimal)

Zeichen und Symbole

- `char` 1 Byte −128 ... +127
- `unsigned char` 1 Byte 0 ... 255

→ siehe ASCII-Tabelle, Codepage 437 oder Codepage 850

Achtung! Es gibt keine elementaren Datentypen für Zeichenketten und Text (`String`) und auch nicht für binäre Rohdaten (`byte`).

KONSTANTEN

- Einzelne Zeichen (als ganze Zahlen)
  - durch Winkelminuteneinheitszeichen (') begrenzt
  - Beispiele: `'A'`, `'1'`, `'\n'` (Zeilenumbruch), `'\011'` (Tab), `'\x84'` („)
  - der Backslash markiert als Escape-Zeichen alle Sonderzeichen und macht aus sich selbst das \-Zeichen (angegeben durch `'\\'`)
  - Achtung! `'A'` ungleich `"A"` (beachte: Winkelsekundeneinheitszeichen)
- ganze Zahlen (Standard ist `int`)
  - dezimal: `123`, `-12`, `45L` (vom Typ `long`), `678U` (vom Typ `unsigned`)
  - oktal: `0173`, `-\14`, `055L`, `\1246U` (`0` *und* `\` markieren Oktale)
  - hexadezimal: `0x7B`, `-0xC`, `0x2DL`, `0x2A6U`
- Real-Zahlen (Standard ist `double`)
  - `123.45`, `.667`, `-3.5e-5`, `65.43F` (vom Typ `float`), `12345678912345.0L` (vom Typ `long double`)

NEUERE DATENTYPEN (ANSI)

- `const`: konstanter Datentyp – nach Zuweisung unveränderlich
- `void`: unbestimmt, nicht existent
- `volatile`: flüchtig
  - entsprechend vereinbarte Variablen können auch durch Ereignisse außerhalb des Programms verändert werden
  - `const` und `volatile` können kombiniert werden:
    Variable kann nicht durch Programm, jedoch extern verändert werden
- ab 1999 (C99)
  - `long long` 8 Byte -2^{63} ... $+2^{63}-1$
  - `unsigned long long` 8 Byte 0 ... $+2^{64}-1$
  - `complex`, `imaginary` (inkompatibel zu C++)

Vereinbarungen

Vereinbarungen elementarer Datentypen

- müssen am Anfang eines Blockes stehen (bis C99)
- reservieren Speicherplatz entsprechend der Größe der Datentypen (mit `sizeof` ermittelbar)
- Variablen gleichen Typs werde durch Komma getrennt deklariert
- jede Vereinbarung wird durch Semikolon abgeschlossen
- Variablen sind zufällig belegt!
  (können aber eine Anfangsbelegung (Konstante) zugewiesen bekommen)
  → Instantiierung vor/bei erster Nutzung nicht vergessen!
- Beispiele
  - `int i, j, k;`
  - `float op1=0.0f, op2=0.0f;`
  - `char zeichen='E';`

Programmierung und Nassi-Shneiderman-Diagramme

- Vorüberlegungen mit Zettel und Bleistift können sehr hilfreich sein
- Programme durch reines Probieren zum Laufen zu bringen ist nicht zielführend und kostet viel Zeit
- obwohl Kontrollstrukturen und Nassi-Shneiderman-Diagramme äquivalent sind, erfordert das Erstellen der Diagramme mehr Nachdenken über den zugrunde liegenden Algorithmus
- Programmierer werden gezwungen, sich gründlicher mit dem Algorithmus zu beschäftigen
- Programme können mit Nassi-Shneiderman-Diagrammen vereinfacht und übersichtlicher beschrieben werden
- beim späteren Debuggen können die Diagramme gemeinsam mit Variablenbelegungstabellen beim Korrigieren der Fehler helfen