BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

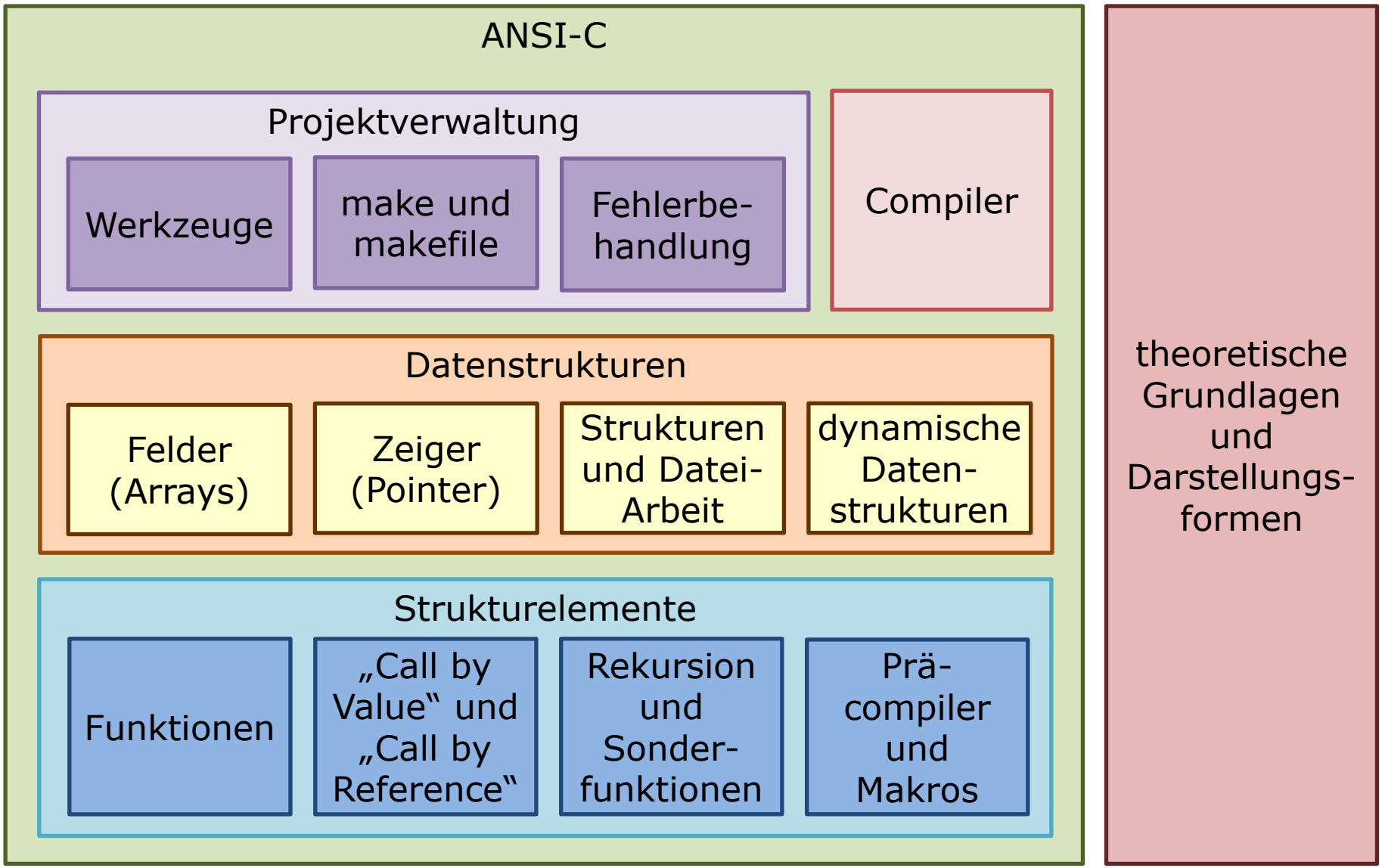
10

Imperative Programmierung

Programmprojekte, Fehlerbehandlung

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

Aufbau der Lehrveranstaltung



PROGRAMMPROJEKTE

größere Programme sollten in mehrere Module (Dateien) aufgeteilt werden

- Übersichtlichkeit der Programme
- kleinere Dateien sind besser editierbar
- Möglichkeit der getrennten Kompilierung
- Zusammenfassung von Funktionen für gleichartige Aufgaben
- Zusammenstellung von Bibliotheken
- Wiederverwendbarkeit für andere Programme
- Zusammenfassung der nicht-ANSI-Funktionen (Konsolen-E/A)
- bessere Pflegbarkeit, Wartbarkeit und Erweiterbarkeit der Programme
- schneller portierbar auf andere Plattformen
- mehrere Programmierer können gleichzeitig am selben Programm arbeiten (an verschiedenen Modulen)

Programmverwaltung mit Tools

Gründe für den Einsatz von Werkzeugen zur Programmverwaltung

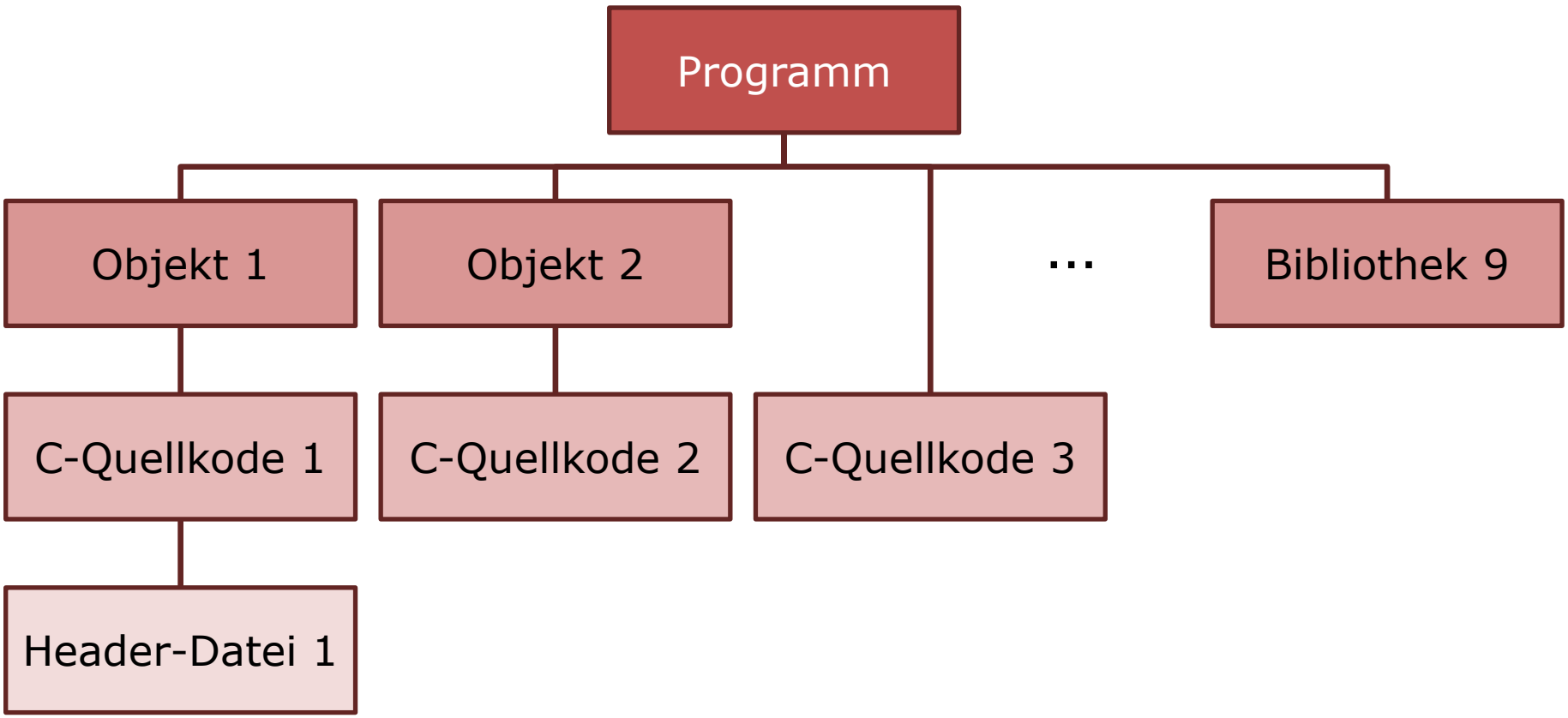
- bei sehr komplexen Programmen, die aus sehr vielen Teilen bestehen, muss das Bilden der Programme automatisiert werden (Übersichtlichkeit)
- es sollen nur die Teile neu gebildet werden, die sich zur vorhergehenden Version geändert haben
- es können verschiedene Varianten des Programms gebildet werden (Debug-Variante, Demo-Version, Vollversion, …)
- Verwaltung von verschiedenen Versionen (Bearbeitungsständen)

Zur Programmverwaltung stehen verschiedene Werkzeuge (Tools) zur Verfügung. → Ein Tool ist das aus der UNIX-Welt stammende `make`.

MAKEFILE, MAKE

- `make` ist ein seit langem (besonders auf UNIX) eingesetztes Tool
- `make` ist das Tool, das die Informationen einer `Makefile` (Datei kann auch anders heißen) auswertet und das Programm bildet
- `Makefile` (Beschreibungsdatei) enthält alle Abhängigkeiten, Teile und Kommandos, die zur Bildung des Programms benötigt werden
- Syntax einer `Makefile` ist standardisiert, d.h. auf verschiedenen Plattformen kann dasselbe `Makefile` verwendet werden.

Abhängigkeiten (dependencies)



Fehlerbehandlung

Fehlerbehandlung (1/3)

- sorgfältige und umfassende Fehlerbehandlung ist sehr wichtig
- oft als lästig empfunden und daher sehr stiefmütterlich behandelt
- ist aber eine Voraussetzung für qualitativ gute Programme!
- Fehlerbehandlung kann in mehreren Stufen erfolgen
  1. Beschränkung und Überprüfung der Datenobjekte sowie Kodeabdeckung prüfen
  2. Abfangen von fehlerhaften und unsinnigen Eingaben, Test auf ungültige Zahlenbereiche, Vermeidung ungültiger Operationen, usw.
  3. aussagekräftige und ausreichende Fehlernachrichten
  4. sinnvolle Reaktionen auf Fehler (Wiederholung und/oder Abbruch von Aktionen, Programmabbruch)

Fehlerbehandlung (2/3)

- die verschiedenen Stufen der Fehlerbehandlung erfordern verschiedene Herangehensweisen bei der Programmierung
- Beschränkung und Überprüfung der Datenobjekte ist das, was häufig als lästig empfunden wird
  - Passen die Datentypen alle zueinander?
  - Wird nur auf instantiierte Variablen zugegriffen?
  - …
- Kodeabdeckungsprüfung durch Haltepunkte und Plausibilitätsprüfung
  - Wird Programmzeile X erreicht?
  - Wird Programmzeile Y garantiert ***nicht*** erreicht?
  - Müssen sich Nutzer einloggen bevor sie persönliche Daten ändern?
  - …

Fehlerbehandlung (3/3)

- Vermeidung von fehlerhaften und unsinnigen Nutzereingaben erfolgt meist in den Funktionen, wo die Eingabe erfolgt
- Ausgabe von Fehlernachrichten erfolgt **nicht** dort, wo der Fehler auftritt
  - Redundanz (viele gleichartige Fehler in verschiedenen Funktionen)
  - Funktionen ohne Ein- und Ausgabe erhalten auch Fehlerausgaben
  - Programme werden übersichtlicher und leichter wartbar, da sie nicht mit Fehlerbehandlungskode „zugekleistert" sind
  - Fehlermeldungen systematisieren, gruppieren und auflisten

Fehlerarten / Einteilung

Während der Laufzeit eines Programmes können diverse Fehler auftreten. Diese Fehler lassen sich in die folgenden Kategorien einteilen:

1. falsche bzw. unkorrekte Benutzereingaben
2. fehlerhafte Verwendung von Systemressourcen
   (`malloc`, `free`, `fopen`, `fread`, `fwrite`, `fclose`, …)
3. fehlerhafte interne Programmabläufe
   (Division durch `0`, Endlosschleifen, fehlerhafte Speicherzugriffe, …)
4. fehlerhafte Kompilierung, Fehler im Betriebssystem und Hardwarefehler

Fehlernachrichten

- Fehlernachrichten werden oft zentral in einer dafür vorgesehenen Funktion aufgelistet und bei Bedarf ausgegeben
- Reaktion auf Fehler hängt stark vom jeweiligen Fehler ab
  - direkte Behandlung in der Funktion, in der der Fehler auftritt (evtl. nur Nachricht ohne weitere Reaktion)
  - Weiterreichen des Fehlers mittels `return`-Wert (meist als negative ganze Zahl) an die aufrufende Funktion
  - zentrale Behandlung von gleichartigen Fehlern in einer dafür vorgesehenen Funktion

Es ist guter Style, Fehlernachrichten nicht im Programmkode zu definieren, sondern Platzhalter zu verwenden. Die eigentlichen Nachrichtentexte werden in separaten Dateien in unterschiedlichen Sprachen (Localization) gespeichert.

Ausgabe von Fehlernachrichten (1/4)

```
/* Locale-Datei für de_DE (errmsgs_49.h)
 * (49 - Ländervorwahl für Deutschland) */
#define ERRCNT 7 // kann auch im Hauptprogramm stehen

/* … */
const char errMsg[ERRCNT][60+1] = {
   " -1 Doppelt verkettete Liste ist leer",
   " -2 Speicherplatz konnte nicht bereit gestellt werden",
   " -3 Datei konnte nicht zum Lesen geöffnet werden",
   " -4 Datensatz konnte nicht gelesen werden",
   " -5 Datei konnte nicht zum Schreiben geöffnet werden",
   " -6 Datensatz konnte nicht geschrieben werden",
   " -7 Datei konnte nicht geschlossen werden" };
/* … */
```

Ausgabe von Fehlernachrichten (2/4)

```
/* Locale file for en_US (errmsgs_1.h)
 * (1 - Country code for USA) */
#define ERRCNT 7 // alternatively, can be placed in main file

/* … */
const char errMsg[ERRCNT][60+1] = {
   " -1 Doubly linked list is empty",
   " -2 Memory could not be allocated",
   " -3 Could not open file for reading",
   " -4 Could not read dataset",
   " -5 Could not open file for writing",
   " -6 Could not write dataset",
   " -7 Unable to close file" };
/* … */
```

Ausgabe von Fehlernachrichten (3/4)

```
/* Header im Hauptprogramm */
#include <locale.h>
#ifndef LOCALE
#define LOCALE 49
#endif
#define __gcc_header(x) #x
#define _gcc_header(x) __gcc_header(errmsgs_##x.h)
#define gcc_header(x) _gcc_header(x)
#include gcc_header(LOCALE)

/* … */
```

Beim Kompilieren wird dem Compiler die Locale mitgegeben:

```
g++ -DLOCALE=49 -o programm.exe programm.c
```

Ausgabe von Fehlernachrichten (4/4)

```c
/* Fehlerausgabe im Hauptprogramm */

/* … */

void throwError(char *function, int errNo) {
   int errIndex = (-errNo) - 1;
   if (errIndex >= 0 && errIndex < ERRCNT){
      printf("\nFunktion: %s\n", errMsg[errIndex]);
      perror(function); // Standardfunktion für Fehlerausgabe
   }
}

/* … */
```

Beispiel für Abfangen fehlerhafter Nutzereingaben

```c
/* … */

int fehler;

do {
   fehler= 0;

   /* entgegen Empfehlung, hier fest auf Deutsch… */
   printf("\nPreis von 0,00 bis 1000,00 in der Form xxx.xx: ");
   fflush(stdin);

   if (!scanf("%f", &preis))
      fehler = 1;

   if (preis < 0.0 || preis > 1000.0)
      fehler = 1;
} while(fehler);
```