§4 Programmstruktur - Vereinbarungen

Leitideen: *Bestimmte Konzepte in C++ lassen sich unter Implementierungsaspekten besser verstehen.*

Dazu gehört die Unterscheidung von Definitionen und Deklarationen und spezielle Deklarationsangaben (Speicherklassen), die die Bindungswirkung von Namen und die Lebensdauer von Variablen regeln.

Die Bindungswirkung (linkage) gibt an, ob ein Name in einem Programm aus mehreren Übersetzungseinheiten sich nur auf eine einzige oder mehrere gleichnamige Größen bezieht.

Die Lebensdauer von Variablen hängt wesentlich davon ab, in welchem Speichersegment sie angelegt werden.

Getrenntes Übersetzen und Linken wird meistens mit Hilfe von Werkzeugen (make) durchgeführt.

§4 Vereinbarungen - Themenübersicht

- Gültigkeitsbereich von Namen (scope) I,II
- Bindungswirkung von Namen (linkage)
- Deklarationen und Definitionen
- Speicherklassen
- Funktion strtok
- Linken - Beispiel, Reihenfolge
- Objektcodelayout
- Linken in C und C++
- Vorwärts- und Frienddeklarationen von Klassen I,II
- Inline-Funktionen

Gültigkeitsbereich von Namen (scope) I

- *Gültigkeitsbereich von Namen (z.B. von Variablen, Funktionen, Klassen)*: Teilbereich einer Übersetzungseinheit (Datei), in der ein Name bekannt ist
- Gültigkeitsbereiche: *lokal, global (Datei), Namensraum, Klasse, Funktion*
- *lokal:* Namen innerhalb eines Blocks (inkl. Funktionsblock) Gültigkeit: Ab Vereinbarungspunkt bis zum Ende des innersten umschließenden Blocks
- *global:* Namen außerhalb von Funktionen und Klassen Gültigkeit: Ab Vereinbarungspunkt bis zum Ende der Übersetzungseinheit (Datei)
- *Namensraum:* Ab Vereinbarungspunkt bis zum Ende des Namensraums inkl. der Namensräume, in die sie durch using-Direktiven eingeführt werden

Gültigkeitsbereich von Namen (scope) II

- *Klasse:* Komponentennamen innerhalb einer Klasse
  Gültigkeit: Ab Vereinbarungspunkt bis zum Ende der Klassenvereinbarung und in Komponentenfkt.rümpfen, Konstruktorinitialisierungen, Parametervoreinstellungen der Komponentenfkt. (auch rückwärts und außerhalb!)
- *Funktion:* Marken
  Gültigkeit: Funktionsblock (auch rückwärts!)

- Vorrangregel: Vereinbarungen in einem inneren Block haben Vorrang vor Vereinbarungen in einem äußeren Block oder außerhalb von Funktionen.
  Analog für Namensräume.
- Funktionsparameter: Gültigkeitsbereich ab Vereinbarungspunkt bis zum Funktionsende
  Gleichnamige Variablen im Funktionsblock überdecken die entsprechenden Funktionsparameter

Bindungswirkung von Namen (linkage)

- *Getrenntes Übersetzen:* Übersetzen eines Programms in Teilen („Übersetzungseinheiten“) zu unterschiedlichen Zeiten
  Bsp.: `sqrt` in Standardbibliothek `libm.a` bzw. `libm.so` bereits vorübersetzt enthalten
- Problem: Gleiche Namen in unterschiedlichen Übersetzungseinheiten
  - Soll sich der Name in allen Übersetzungeinheiten auf dasselbe Objekt beziehen? *“externe Bindung“*
  - Soll sich der Name sich in unterschiedlichen Übersetzungseinheiten auf unterschiedliche Objekte beziehen? *“interne Bindung“ bzw. “keine Bindung“*

Deklarationen und Definitionen

- Kein automatischer Import von Namen, stattdessen *Deklaration* erforderlich.
- Unterschied Definition/Deklaration: Deklaration macht Name bekannt, reserviert *keinen* Speicherplatz.
- Funktionsdeklaration: Funktionsblock wird durch Strichpunkt ersetzt, Parameternamen können weggelassen werden.
  Bsp.:
  ```
  double sqrt(double x){...}
  double sqrt(double);
  ```
- Klassendeklaration: Nur Klassenname, gefolgt von Strichpunkt
- Variablendeklaration: Kein syntaktischer Unterschied zur Variablendefinition
  Kompliziertere Regeln, wann eine Definition oder eine Deklaration vorliegt.
- static-Vereinb. in Klassen (Datenkomp./Komp.fkt) sind *immer* Deklarationen. (Definition außerhalb Klasse notw.)

Speicherklassen

- Speicherklasse: Deklarationsangabe

  Bsp.: `extern double sqrt(double);`

  `static int f(....) { ... }`
- Speicherklasse regelt zwei verschiedene Eigenschaften:
  - Dauer der Speicherreservierung
  - Bindungswirkung
- Speicherklassen für lokale Variablen: Hauptsächlich zur Festlegung der Dauer der Speicherreservierung

  `auto` Variable wird neu angelegt (Voreinstellung)

  `static` Variable behält Wert auch nach Verlassen des Blocks
- Speicherklassen für Variablen und Funktionen (global oder in Namensräumen): V. a. Festlegung der Bindungswirkung

  `extern` externe Bindungswirkung (Voreinstellung)

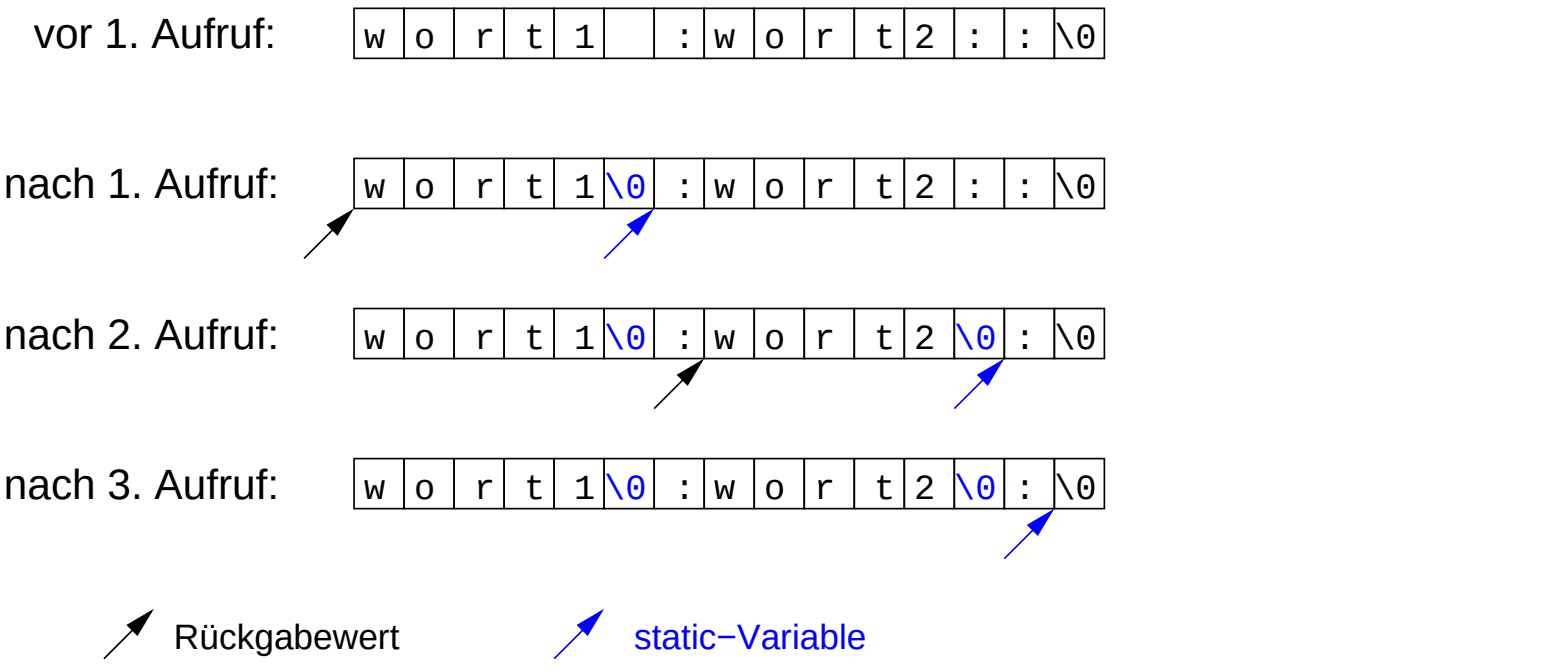
  `static` interne Bindungswirkung
- Lebensdauer globaler Variablen : Initialisierung beim Programmstart (ggf. per Default oder Konstruktoraufruf), Freigabe am Programmende (ggf. Destruktoraufrufe)

Funktion strtok

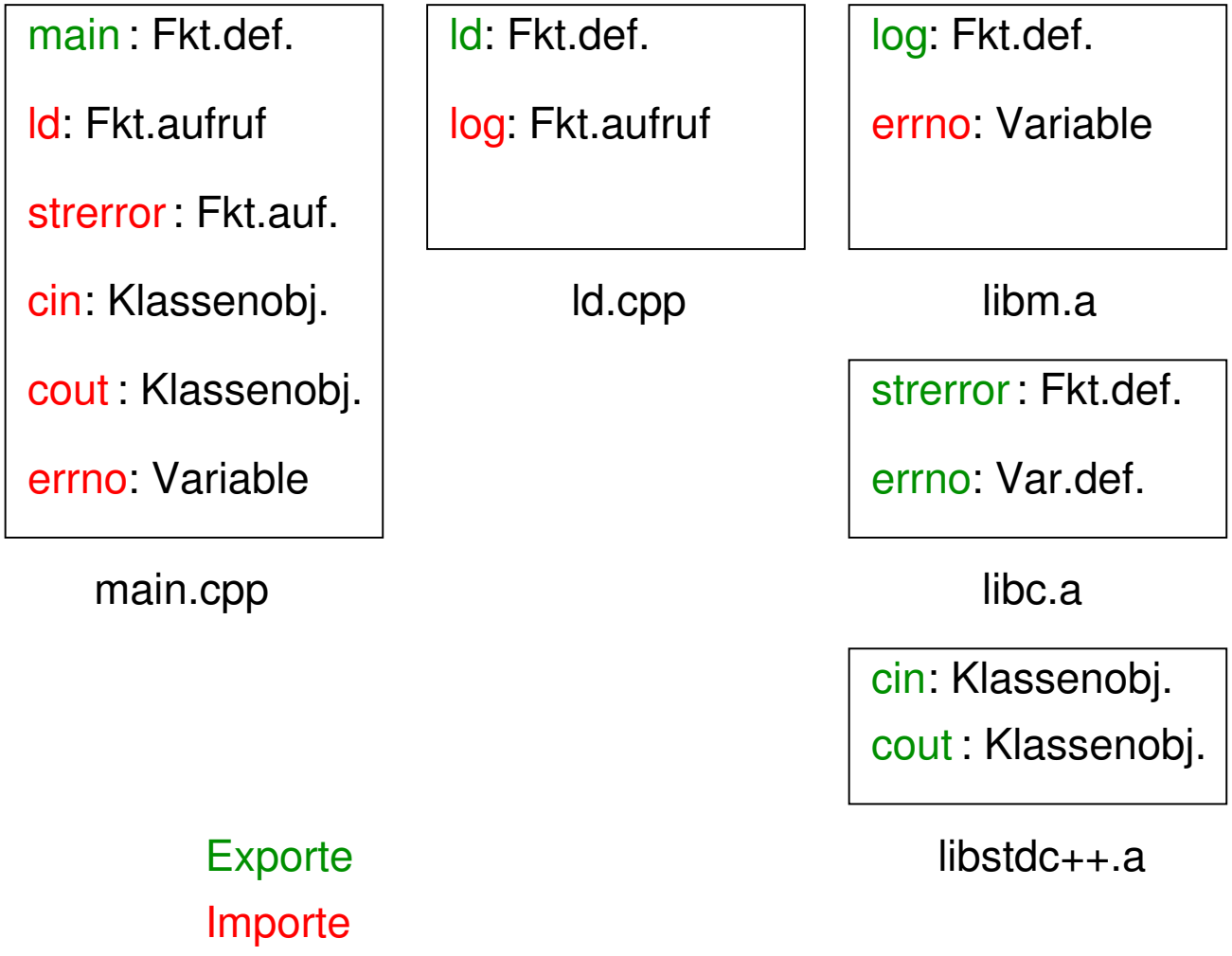
- Lokale static-Variable ermöglicht Funktion mit Gedächtnis
- Zerlegt Zeichenkette anhand von Trennzeichen in “Worte“
- *Bsp.:* `char zk[]="wort1 :wort2::";`

  Erster Aufruf: `wort=strtok(zk," :")` Trenner: `: ␣`

  Weitere Aufrufe: `wort=strtok(0," :")`



Linken (Binden) - Demobeispiel



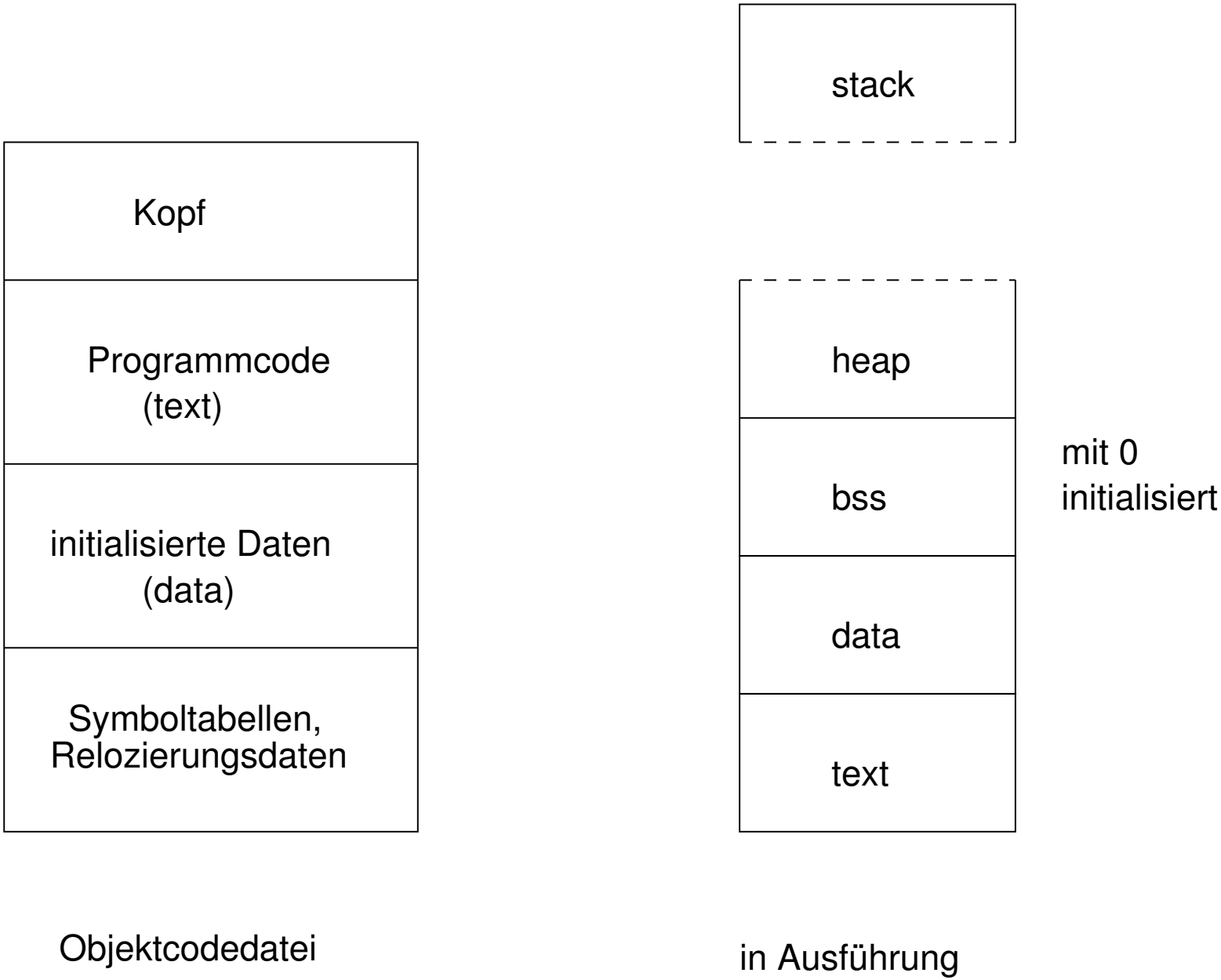
Linken - Reihenfolge

- `c++ main.cpp ld.cpp`
  - erzeugt `main.o` und `ld.o`
  - bindet diese mit `libm.a` (und weiteren C/C++-Systembibliotheken) zu `a.out`.
  - löscht `main.o` und `ld.o` nach dem Linken
- Linken kann auch explizit erfolgen:

```
c++ -c main.cpp
c++ -c ld.cpp
c++ main.o ld.o
```

- Die Reihenfolge war früher wichtig, denn:
  - schrittweise Abarbeitung von links nach rechts
  - jeweils Ersetzung derjenigen undefinierten Symbole, die in der hinzuzulinkenden Objektcodedatei (oder Bibliothek) enthalten sind
  - Falsch war früher: `c++ ld.cpp main.cpp`

Objektcodelayout (Modell)



Linken in C und C++

- Benutzung des (C-)Linkers durch C++ erfordert Modifikationen
  Grund: Gleichnamige Funktionen möglich (Überladen!)
- Abhilfe: Linkersymbol enthält Funktionsname
  und Parametertypen
- Kein allgemein akzeptiertes Schema zur Bildung von Linkersymbolen.
  Objektcodedateien und Bibliotheken müssen oft sogar mit derselben Compilerversion übersetzt sein.
- Kennzeichnung zu importierender C-Funktionen durch `extern "C"`-Anweisungen oder -Blöcke
- Oft bereits in den Headern von C-Bibliotheken enthalten (Steuerung durch Präprozessormakros: `__STDC__` bzw. `__cplusplus`)

Vorwärts- und Frienddeklarationen von Klassen

Ziel: Wechselseitige Benutzung von Klassen

- Vorwärtsdeklaration: Klassenname ist in nachfolgenden Klassenvereinbarungen bekannt.
- Frienddeklaration: Befreundete Klasse darf Komponenten benutzen
- *Vorsicht:* Vorwärtsdeklaration ist unvollständige Typdef., die Klassenmethoden sind in der nachfolgenden Klasse noch nicht benutzbar.
  Deshalb: Ggf. zuerst Methoden nur deklarieren und erst danach definieren

Vorwärts- und Frienddeklarationen von Klassen II

Beispiel aus dem C++-Standard:

```
class Vector;

class Matrix {
  // ...
  friend Vector operator*(Matrix&, Vector&);
};

class Vector {
  // ...
  friend Vector operator*(Matrix&, Vector&);
};
```

- *Bem.:* Wird später ausführlich behandelt.

Inline-Funktionen

Begriff

Vermeidung des Funktionsaufruf (Stackaufbau, Sprung, Rücksprung, Stackabbau) aus Effizienzgründen
Ersetzung durch geeignete Ausdrucksanweisungen

Bsp.
```
double sqr(double x) { return x*x; }
z = sqr(x+y);
```
wird beispielsweise ersetzt durch
```
double tmp; z = ((tmp=x+y),(tmp*tmp));
```

Verwendung

- Deklarationsangabe `inline`: Hinweis an Compiler, möglichst entsprechenden Code zu erzeugen
- In Klassen *definierte* Funktionen: automatisch inline