
MODULARISIERUNG

(Verknüpfung bzw. Aufteilung)
eines C\C++ Programms

Motivation und Ziel der Modularisierung

Die erste Motivation lautet:

Warum müssen wir selber Programmieren etwas,
das man schon (sogar besser als wir)
programmiert hat?

Motivation und Ziel der Modularisierung

Die schon vorprogrammierten Dateien kommen zu uns als **MODULE** – entweder als Bibliotheken (z.B. `iostream`, `cmath...`) oder als Dateien (z.B. `matrix2x2.cpp`, `statistik.cpp...`)

Unsere Arbeit besteht „lediglich“ darin, die Module mit unserem Hauptprogramm **Verknüpfen (LINKEN)**.

Motivation und Ziel der Modularisierung

Die zweite Motivation lautet:

Wie können wir am besten von uns schon programmierten Code (besonders Funktionen) in anderen Programmen weiterverwenden?

Motivation und Ziel der Modularisierung

Wir können die in unserem Hauptprogramm programmierten Funktionen aus dem Hauptprogramm entfernen und daher ein **eigenes MODUL** erstellen.

Unsere Arbeit besteht hier, das Hauptprogramm richtig **aufzuteilen**, um ein Modul zu erstellen.

Grundvoraussetzung

Damit wir überhaupt mit Modulen arbeiten können, müssen wir das **Erstellungsprozess** (**Build** auf Englisch) einer ausführbaren Datei verstehen.

Mit dieser Lehre befassen wir uns im ersten Teil der Vorlesung (siehe unten Inhaltsverzeichnis).

Inhaltsverzeichnis der Vorlesung

I Das Erstellungsprozess

- Diagramm
- Definitionen und Erläuterungen

II Verknüpfung von Quelldateien

- Beispiele (Aufgabe1 und Aufgabe2)

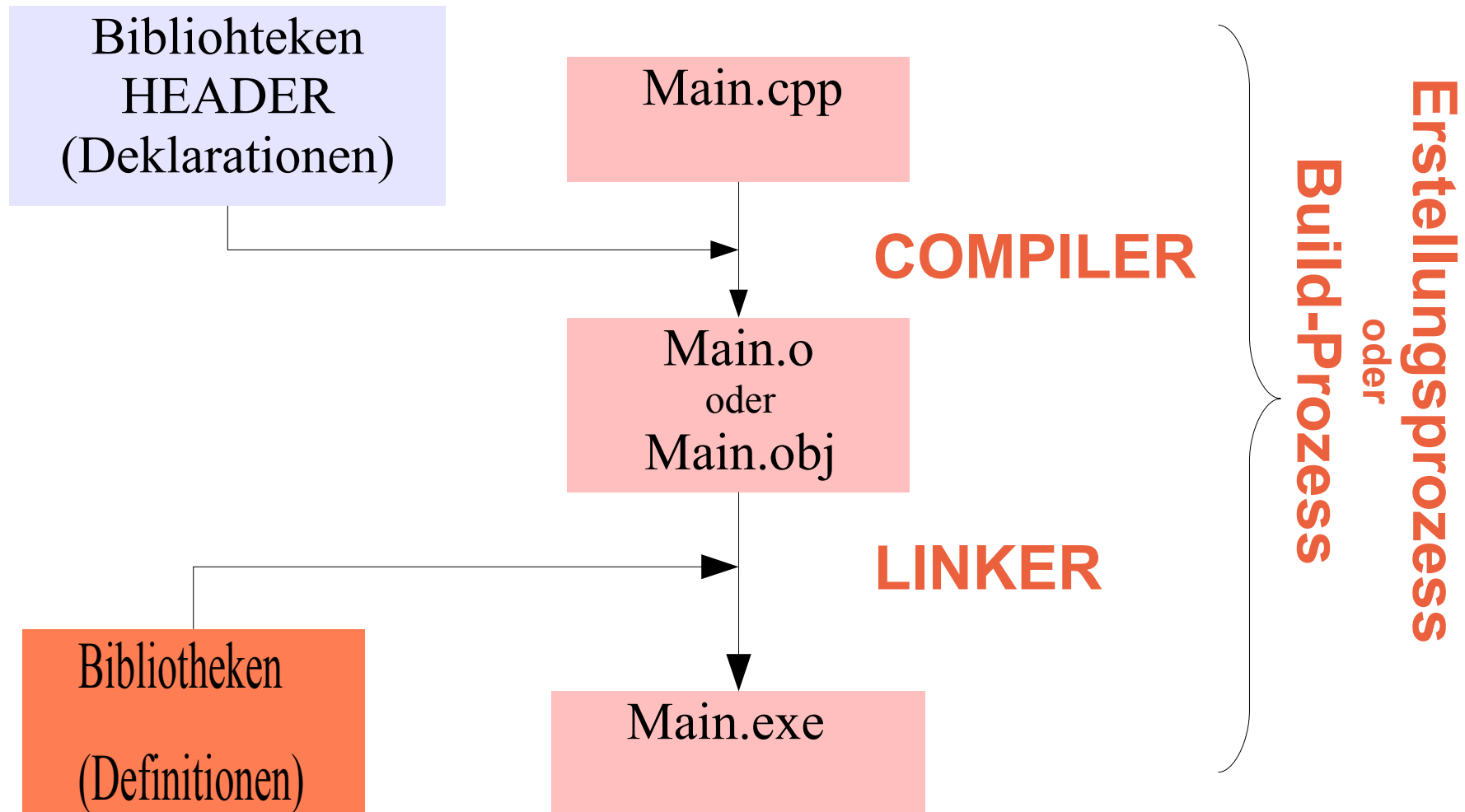
III Aufteilung eines Programms

- Erstellung eigener Module
- Beispiel (Aufgabe 3)

I. Das Erstellungsprozess (Build-Prozess) eines C++ Programms

C++ .exe Erstellungsdiagramm

Erstellungsprozess
aus einer .cpp Datei



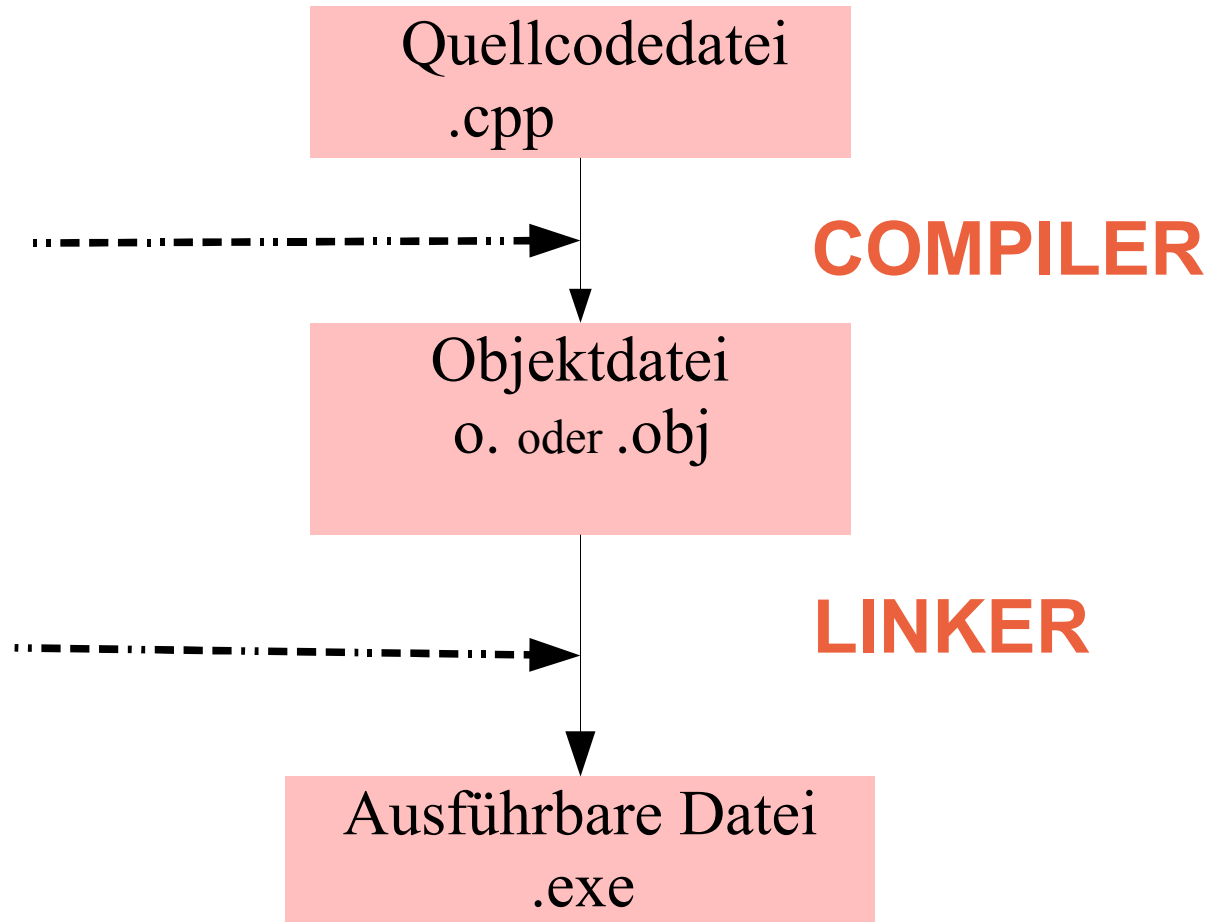
C++ .exe Erstellungsprozess

Erläuterung des Erstellungsdiagramm

- Das Erstellungsprozess beginnt, wenn die Programmierung des Quellcodes (.cpp-Dateien) endet.
- Das Ziel solches Prozesses ist die Erstellung einer ausführbaren Datei (.exe-Datei)
- Es besteht aus zwei Hauptschritte.
 - Compilieren: Die Objektdaten (.o oder .obj-Dateien) werden aus dem Quellcode vom Compiler erstellt
 - Verknüpfen (Linken): Die endgültige ausführbare Datei (.exe) wird vom Linker aus der Zusammenstellung von Objektdaten und Bibliotheken erstellt.

C++ .exe Erstellungsprozess

Erläuterung des Erstellungsdiagramm



C++ .exe Erstellungsprozess

Definitionen

- **Quellcodedatei** oder kürzer **Quelldatei** ist eine Textdatei, wo ausschließlich Anweisungen in einer bestimmten Programmiersprache sind.
(Versehen mit .cpp-Erweiterung für C++ und .c für C)
- **Objektdati** Kompilierte Quellcodedatei. Sie besteht aus einer Mischung von Maschinencode und Verweise auf Funktionen und andere Daten.
(Versehen mit .obj bzw. .o-Erweiterung)*
- **Ausführbare Datei** reines Maschinencodeprogramm. Deshalb ist diejenige, die vom Betriebssystem ausgeführt werden kann.
(Versehen mit .exe Erweiterung)*

* Diese Datei Erweiterungen gelten für Windows. Die Datei-Erweiterungen weichen zwischen Betriebssystemen ab

C++ .exe Erstellungsprozess

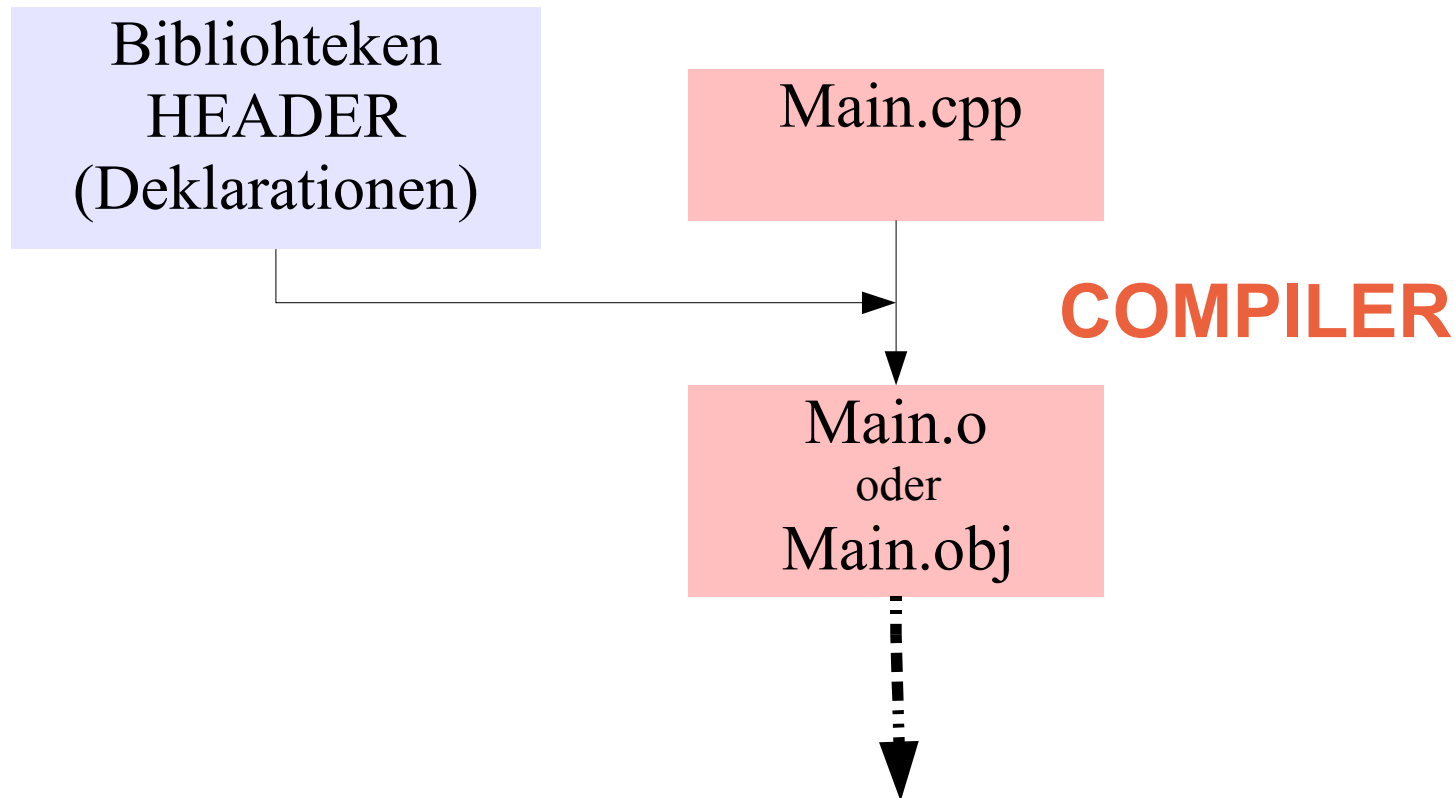
Definitionen

- **Kompilierung** Beim Kompilieren werden jeweilige Objektdateien aus den Quellcodedateien erstellt. Indem die Anweisungen in Maschinencode umgewandelt werden. Die Aufrufe der Funktionen wird aber nicht umgewandelt in Maschinencode; sie werden durch Verweise (Referenzen) ersetzt.

Kompilierungsdiagramm:

Alle **Quellcode**dateien können in eine
Objektdaten-datei kompiliert werden

Siehe Folie 2



C++ .exe Erstellungsprozess

Definitionen

- ***Bibliothek*** wo gewisse Funktionen definiert sind. Eine Bibliothek besteht aus einer Sammlung von Objektdateien.
Es gibt 2 Art Bibliotheken
 - Statisch: mit nicht gelinkten Objektdateien (Versehen mit .lib Erweiterung)
 - Dynamisch: Mit gelinkten Objektdateien (Versehen mit .dll Erweiterung)

C++ .exe Erstellungsprozess

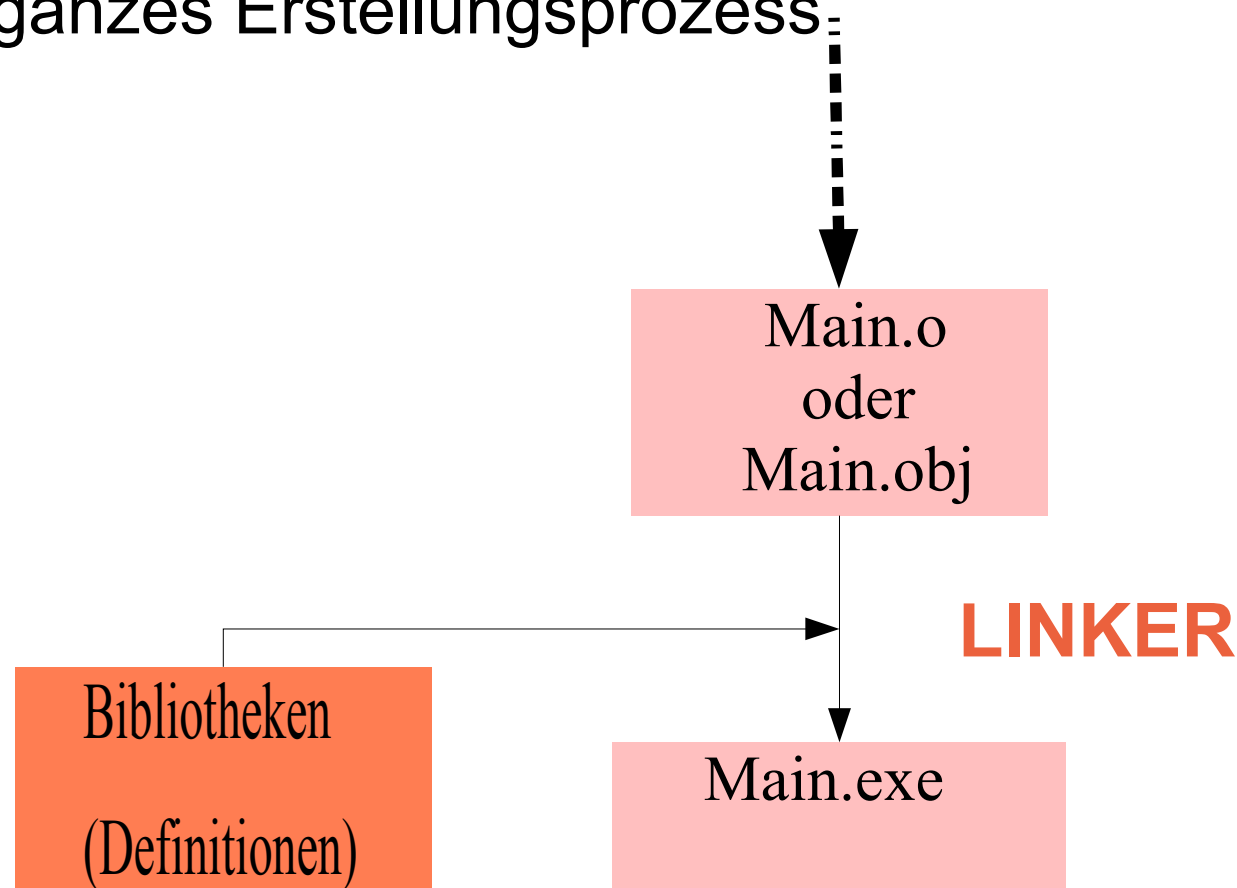
Definitionen

- ***Linken (Verknüpfen)*** Alle aus der Kompilierung entstandenen Verweise werden durch den jeweiligen Maschinencode ersetzt.
Die Definition der Funktionen-Verweise wird aus **Bibliotheken** oder **anderen Objektdaten** geholt. Das Ergebnis ist eine ausführbare Datei.

Verknüpfung (Linken) von Bibliotheken:

Funktionen **aus anderen Bibliotheken** werden beim Linken mit der **Hauptdatei** verknüpft (gelinkt).

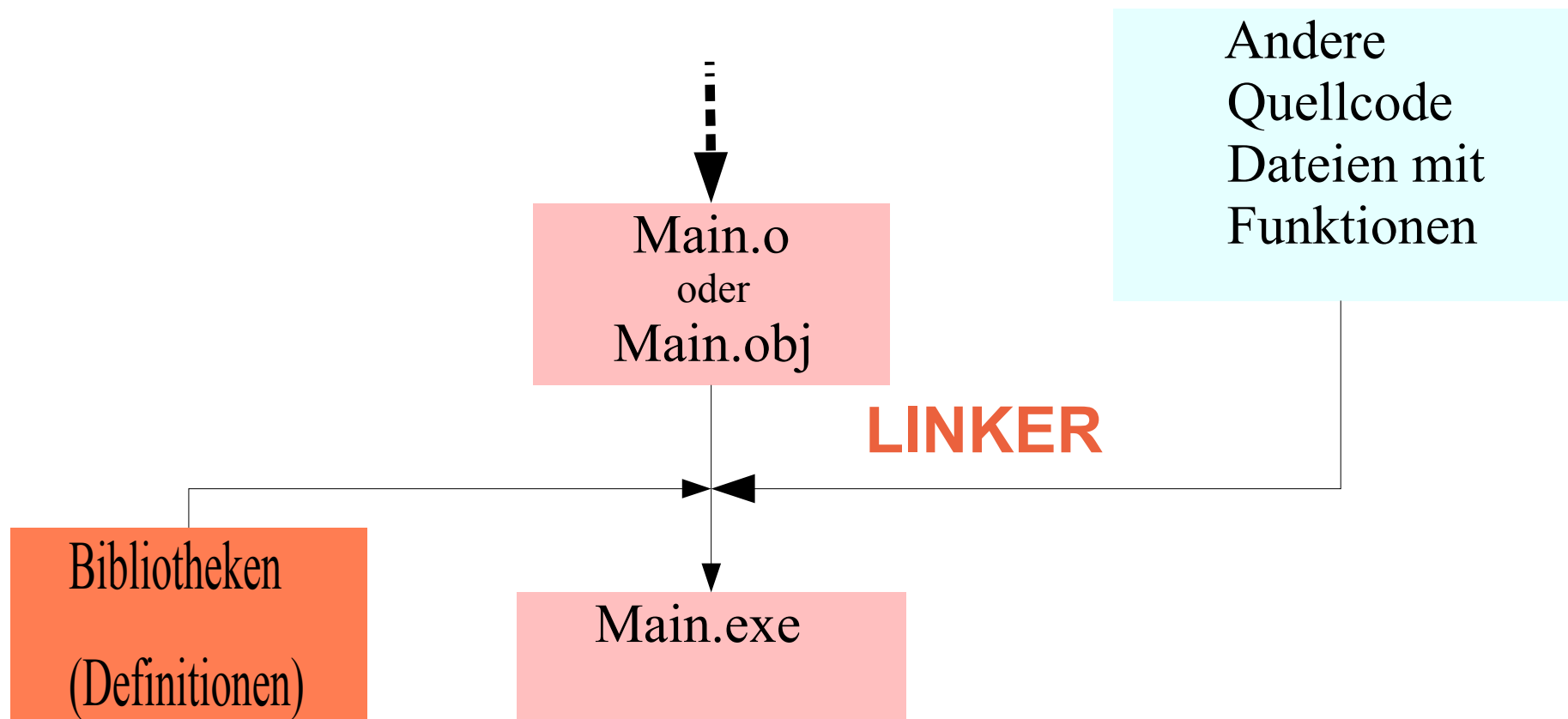
Siehe auch ganzes Erstellungsprozess:



II. Verknüpfung von Quelldateien

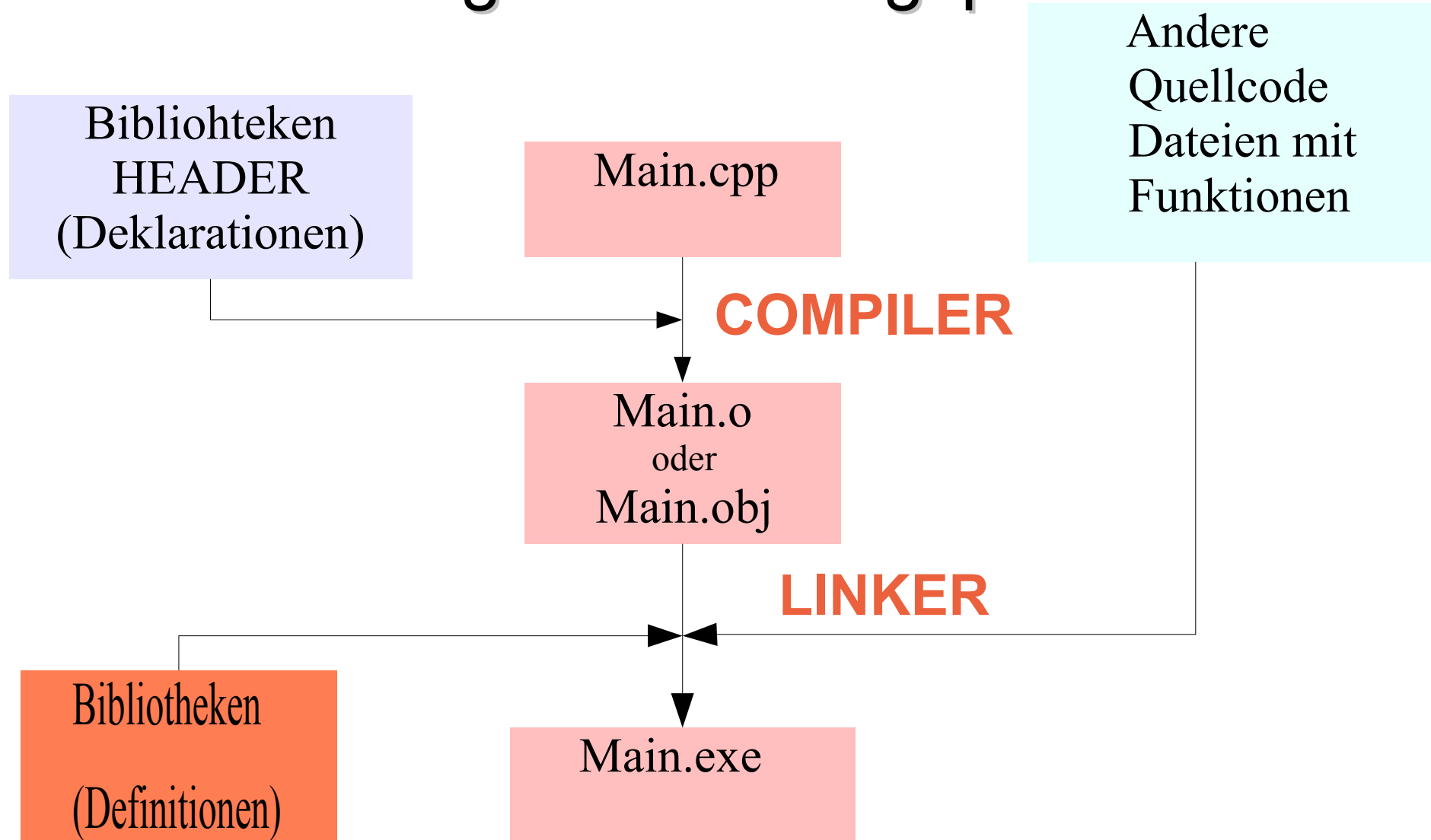
Verknüpfung von Quelldateien

Funktionen aus anderen Dateien können auch mit dem Hauptprogramm verknüpft (gelinkt).



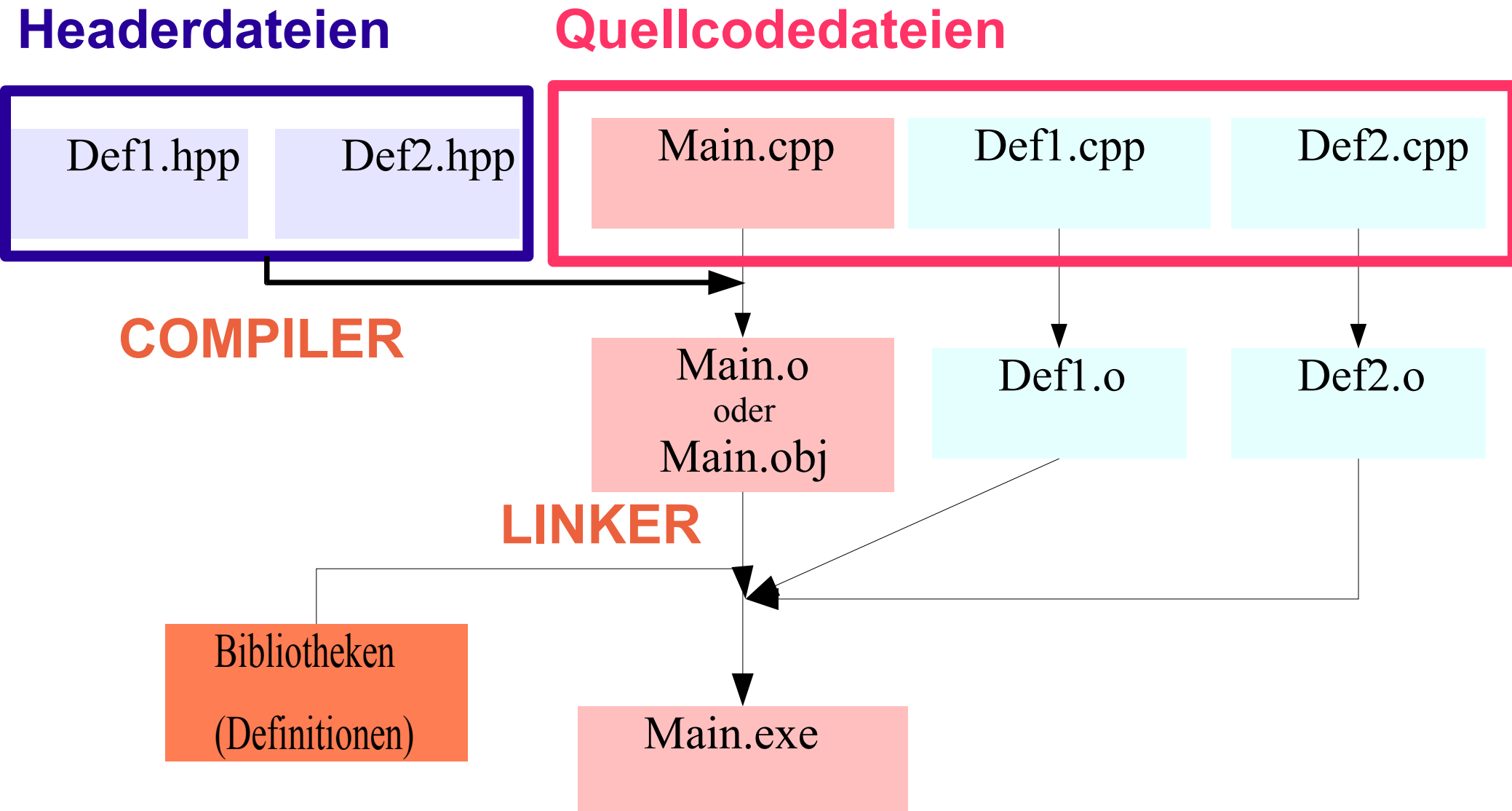
Verknüpfung von Quelldateien

Vollständiges Erstellungsprozess



Verknüpfung von Quelldateien

Detaillierte Ansicht



Verknüpfung von Quelldateien:

Definitionen

- **Headerdatei** Datei mit den Deklarationen der Funktionen und sogar lokalen Variablen jedoch ohne Befehle. (Versehen mit .hpp-Erweiterung für C++ und .h für C)
- **Modul** ist jede Datei die zu einem Hauptprogramm verknüpft (gelinkt) wird. Jedes Modul verschafft dem Hauptprogramm zusätzliche Funktionen.

Wir kennen zwei Arte Module

- Ein vollständiges Paar von Headerdatei und die entsprechende Quellcodedatei
Z.B. Def1.hpp und Def1.cpp sind ein Modul
matrix2x2.hpp und matrix2x2.cpp sind ein weiteres Modul
- Eine Bibliothek-Datei

Verknüpfung von Quelldateien:

Damit das Verknüpfen (Linken) erfolgreich ist muss man:

- 1) Die Headerdateien im Hauptdatei durch
`#include "HeadersDateiname"` verknüpfen.
- 2) Die zu verknüpfende cpp-Dateien als **Projekt** in unserer IDE (z.B. Bloodshed) kompilieren

Verknüpfung von Quelldateien:

Definitionen

- **IDE** (Integrated Development Environment =Integrierte Entwicklungsumgebung : Bloodshed C++, Code-Blocks, Microsoft Visual Studio)
ist die graphische Oberfläche die man nutzt für eine bequemere Programmierung, Verwaltung und Erstellung von ausführbaren Dateien.

Namensbestimmungen

Die Dateien wo man programmiert sind Text-Dateien.
(Deshalb können mit einem Text-Editor wohl
verarbeitet werden)

Sie werden aber durch die Dateierweiterung
unterschieden.

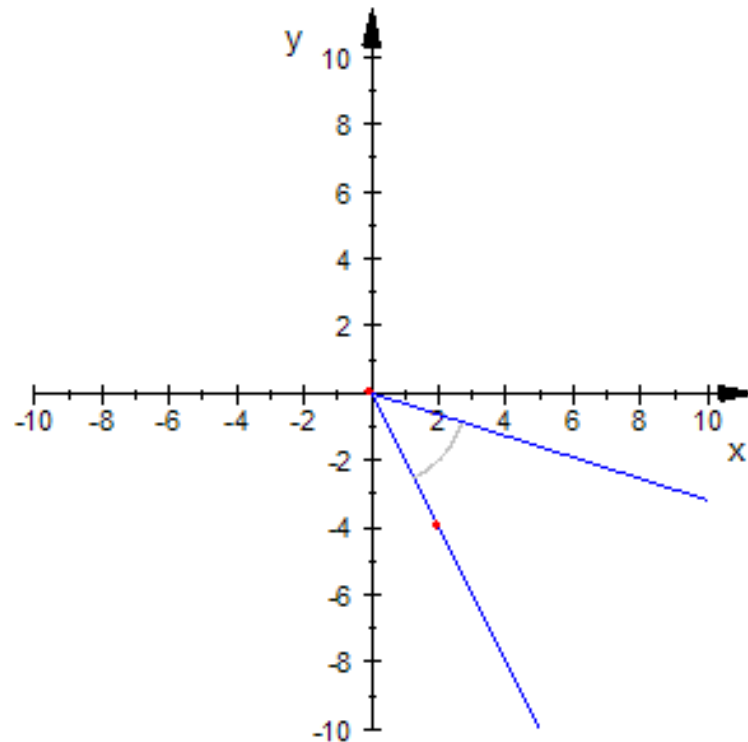
- 1) Die Header-Dateien werden mit *.h* bzw. *.hpp*
Erweiterung bezeichnet jeweils für C oder C++
- 2) Die Quellcode-Dateien werden mit *.c* bzw. *.cpp*
Erweiterung bezeichnet jeweils für C oder C++
- 3) Die Quellcode-Dateie und Header-Dateie, die auf die
gleichen Funktionen sich beziehen, **sollten** gleich
heißen. (z.B. *matrix2x2.cpp*, *matrix2x2.hpp*)

II. Verknüpfung von Quelldateien - Beispiele

Von Problem(en) zu Programm(en)

Aufgabe 1: Wir möchten Ein Punkt auf der Ebene um einen Winkel drehen.

Punkt $(2, -4)$ um 45° drehen



Aufgabe 1: Lösungsweg

Die Drehung eines Punktes wird anhand einer Rotationsmatrix ermittelt.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Können wir das schnell Programmieren?

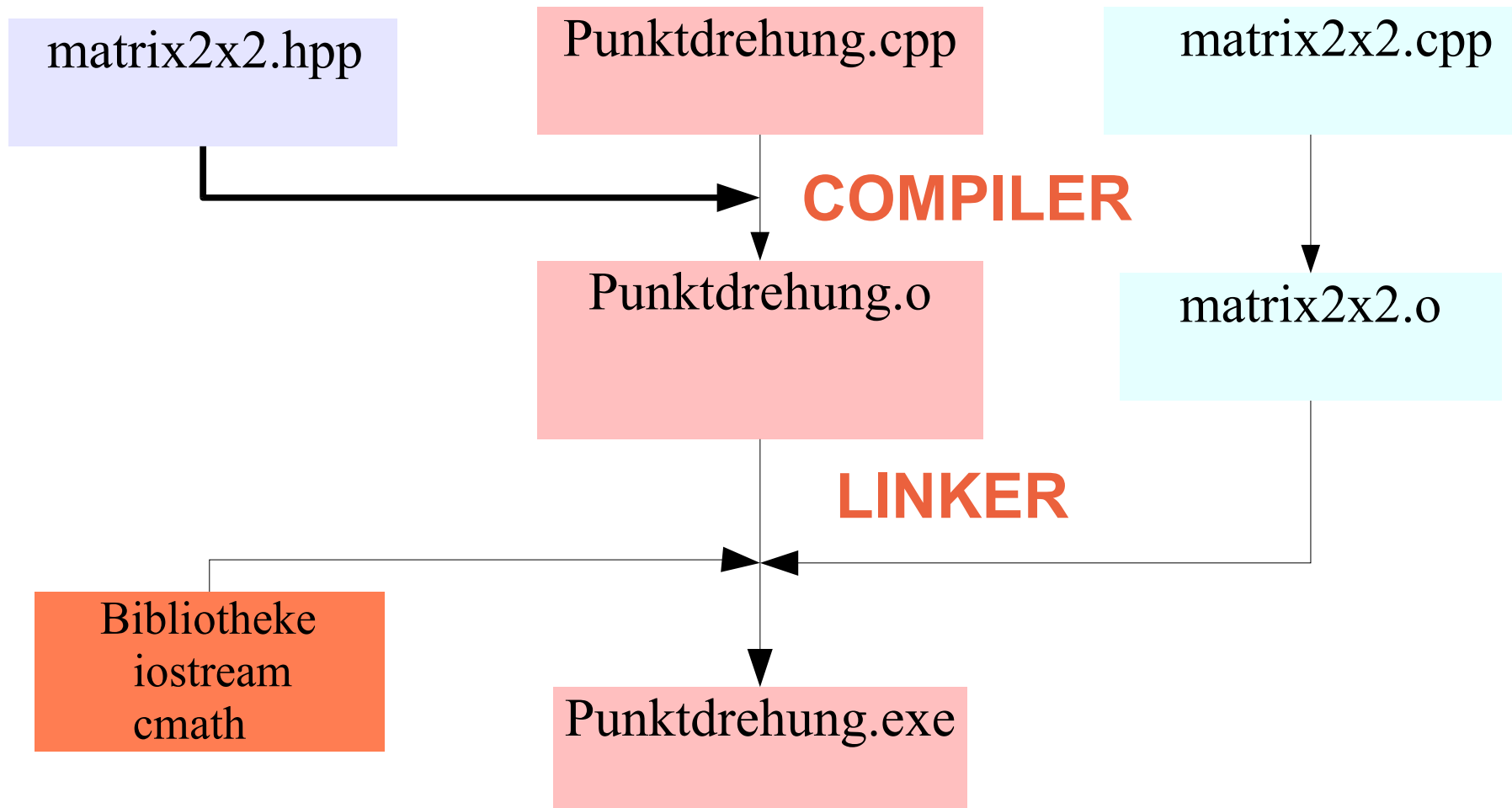
Ja!

Wir verfügen über ein Modul mit Matrizen und Vektoren Funktionen!

matrix2x2.hpp

matrix2x2.cpp

Aufgabe 1: Erstellungsprozess



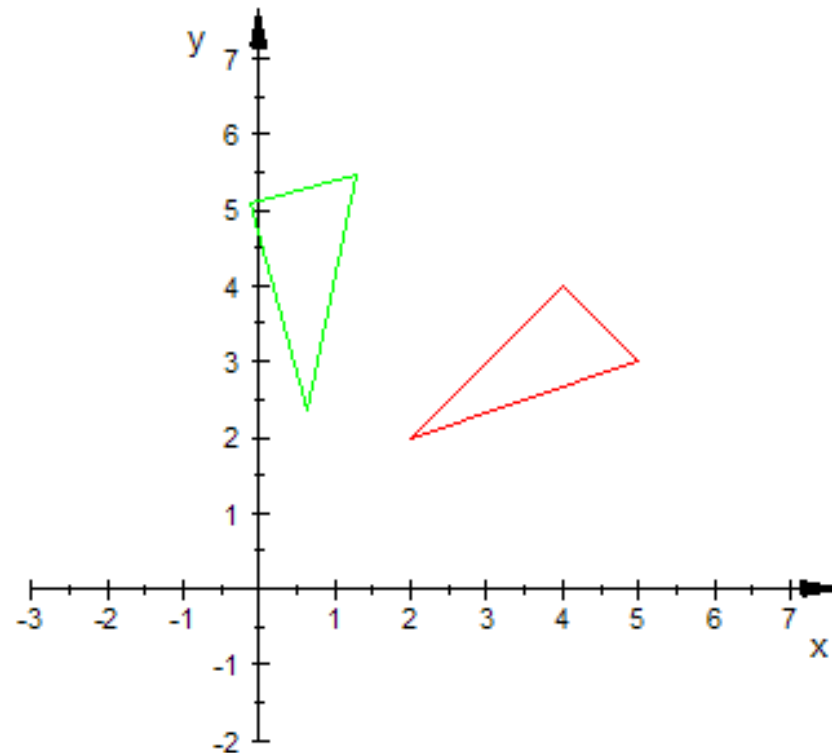
Von Problem(en) zu Programm(en)

Aufgabe 2: Wir möchten jetzt ein ganzes Dreieck drehen.

Punkte des roten
Dreiecks:

$(2,2)$, $(4,4)$, $(5,3)$

Wird um 30° gedreht



Aufgabe 2: Lösungsweg

Ein Dreieck drehen bedeutet dessen 3 Punkte drehen.

Also ist diese Aufgabe so ähnlich wie Aufgabe 1 wiederholt mit 3 Punkten

Können wir das schnell Programmieren?

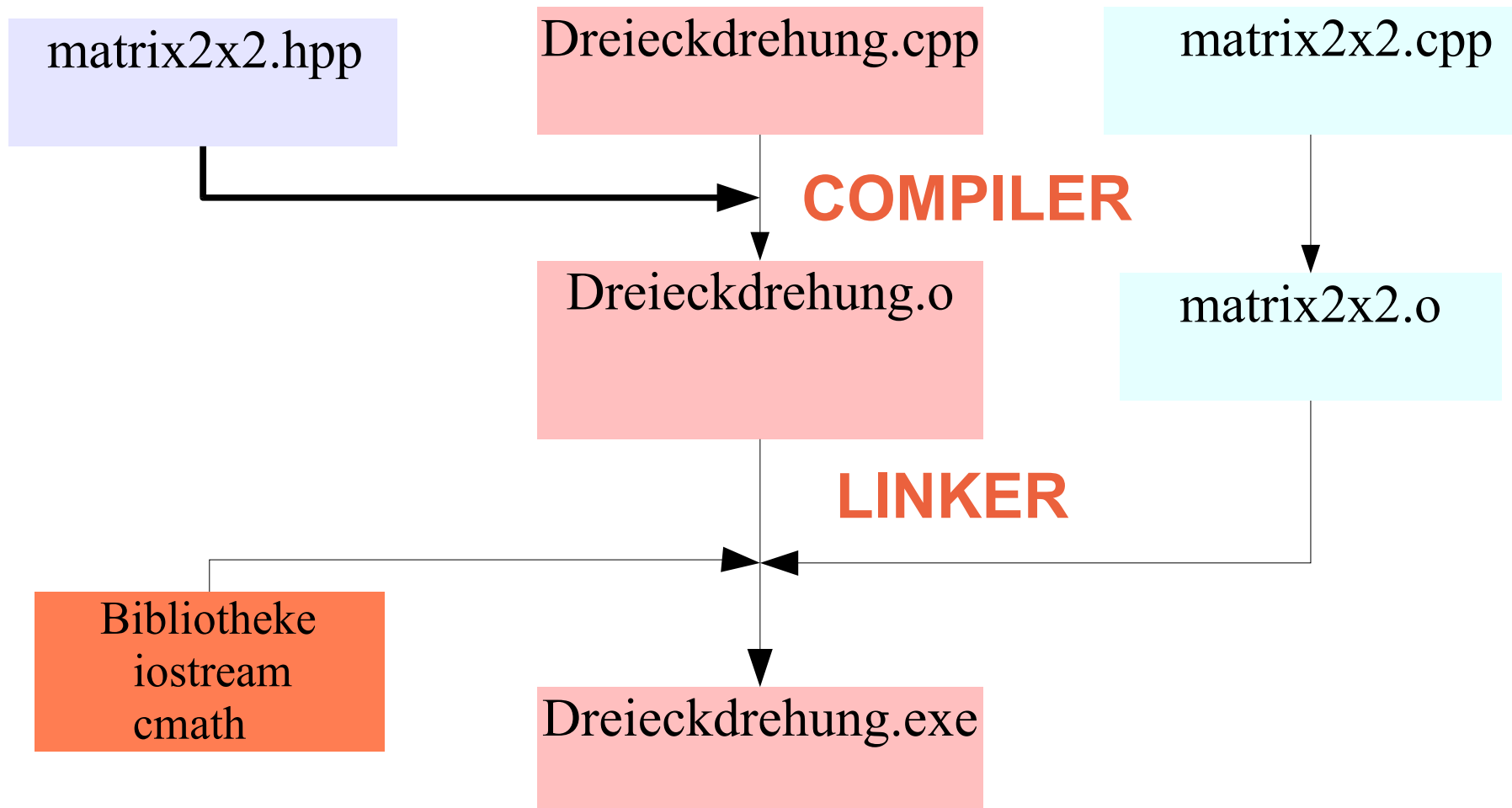
Ja!

Wir verfügen über ein Modul mit Matrizen und Vektoren Funktionen!

matrix2x2.hpp

matrix2x2.cpp

Aufgabe 2: Erstellungsprozess



Aufteilung von eines C++ Programms

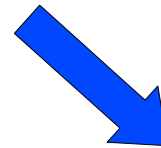
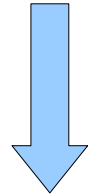
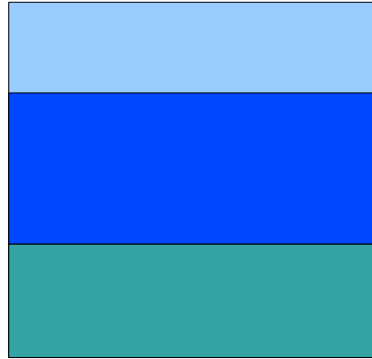
Aufteilung von C++ Programmen:

Eine **einzigste Datei** mit einem Cpp-Programm kann in ***verschiedenen Dateien*** aufgeteilt werden.

Man sagt, die Datei wird **modularisiert**

Aufteilung von cpp-Dateien:

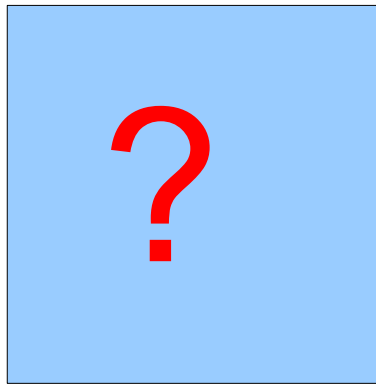
PROGRAMM_MIT_FUNKTIONEN.CPP



funktionen.cpp



funktionen.hpp



main.cpp



Ziel der Aufteilung einer .cpp Datei

- Die main.cpp Datei wird **übersichtlicher**:

Der ganze Code der Funktionen ist weg, es bleiben nur die Aufrufe der Funktionen

- Die Funktionalitäten sind getrennt und können einfacher weiterverwendet können. (**eigenes Modul**)

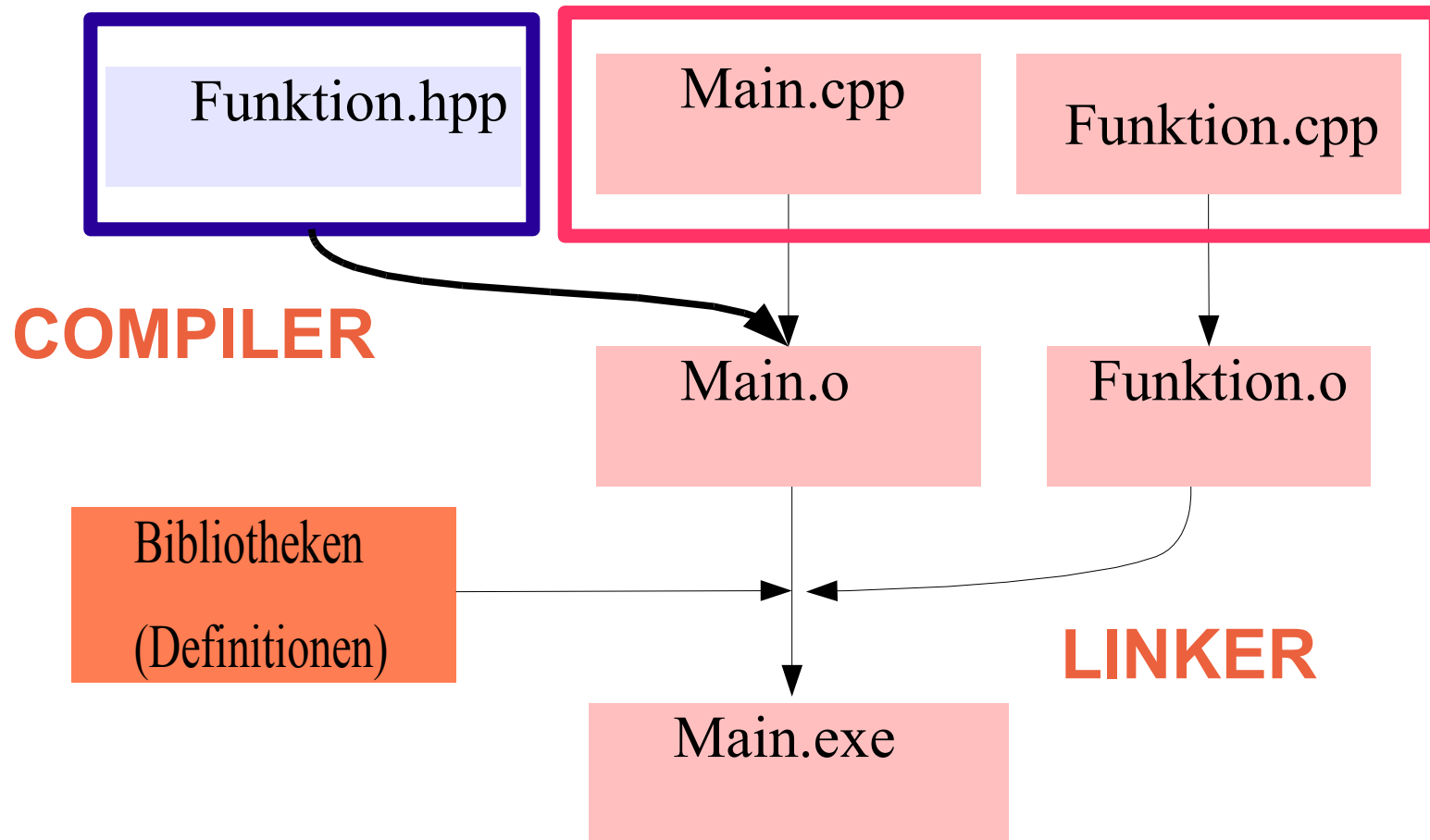
Falls diese Funktionen in einem anderen Programm verwendet werden, müssen sie einfach verknüpft (gelinkt werden).

Das aufgeteilte C Programm

Von Quellcode zum Ausführbaren

Header

Quellcode



Aufteilung von Dateien:

Wir versuchen Anhand einer Aufgabe den Konzept zu erläutern.

Aufteilung von Dateien:

Aufgabe 3:

Es soll ein Programm geschrieben werden, das die Standardabweichung der eingegeben Zahlen berechnet.

Der Mittelwert von N Werten $x_0, x_1, x_2, \dots, x_{N-1}$ ist:

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x_i = \frac{x_0 + x_1 + x_2 + \dots + x_{N-1}}{N}$$

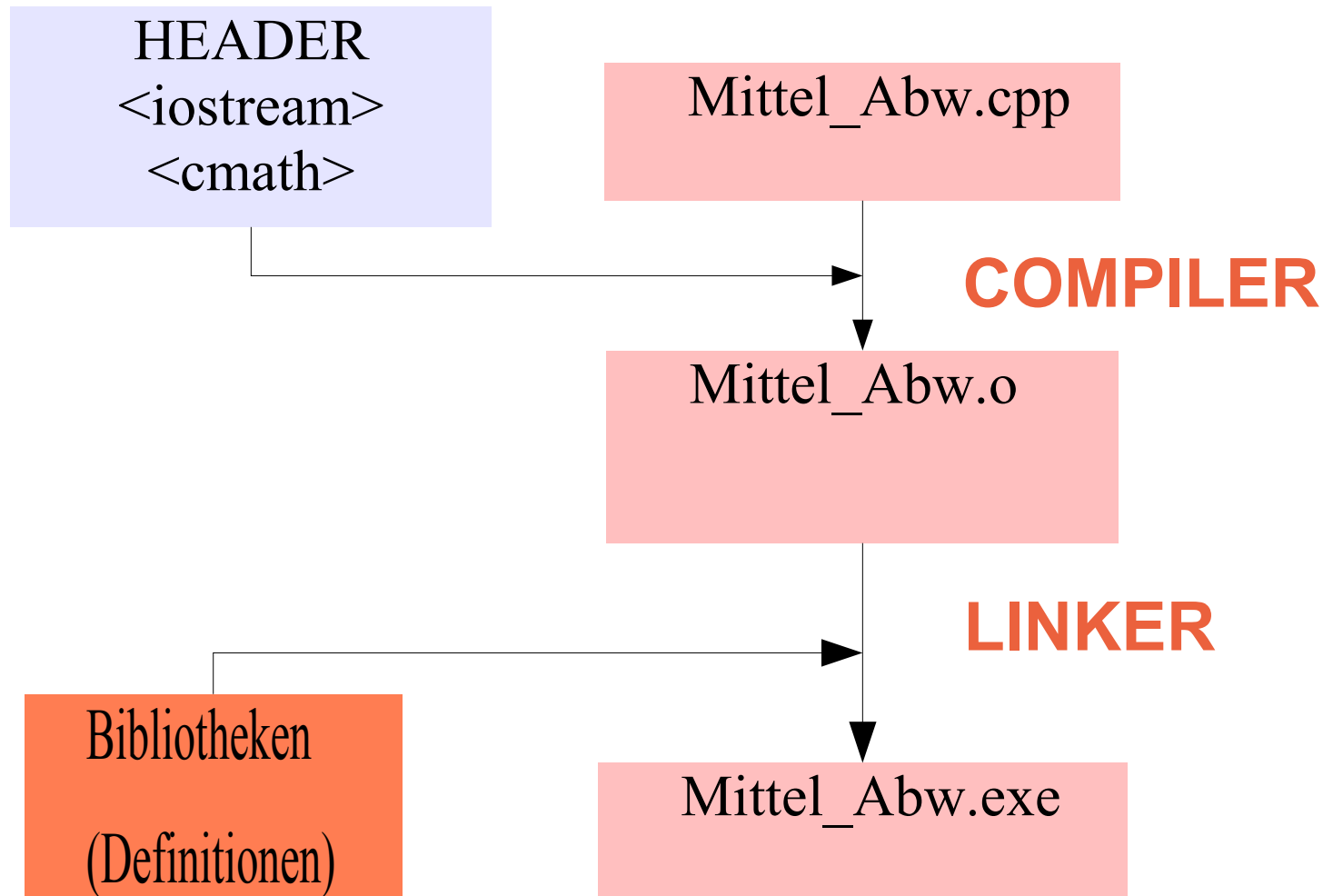
Die Standardabweichung als Maß für die mittlere Abweichung der Einzelwerte vom Mittelwert ist so definiert:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \bar{x})^2}$$

Aufgabe 3: Erstellungsprozess ohne Modul

Lösung mit einer einzigen Datei:

Die ganze Lösung ist in der Datei *Mittel_Abw.cpp*



Aufgabe 3: Modularisierung

Aufteilung des Lösungsprogramms:

1. Deklaration der Funktion `mittelwert` in eine Header-Datei `statistik.hpp`.

```
double mittelwert(double [ ],int);  
double streuung(double[ ],int );
```

**Man darf im Header-Dateien nur
deklarieren nicht ausführen**

Aufgabe 3: Modularisierung

2. Definition der Funktion `mittelwert` und `streuung` in einer Quellcodedatei `statistik.cpp`.

```
double mittelwert(double x[],int anz)
{
    int i;
    double m;
    m=0.0;
    for(i=0;i<anz;i++)
        m=m+x[i];
    m=m/(double)anz;

    return m;
}

double streuung(double x[],int anz)
{
    double m = mittelwert(x,anz);
    double s = 0.0;
    int i;
    for(i=0;i<anz;i++)
        s+=(x[i]-m)*(x[i]-m);
    return sqrt(s);
}
```

Aufgabe 3: Modularisierung

3. Definition des mains bleibt in der Datei
`main_ohne_funkt.cpp`.

Aufgabe 3: Erstellungsprozess mit Modul

Lösung mit Modul:

Die ganze Lösung ist in der Datei *Main_Ohne_Funkt.cpp*

