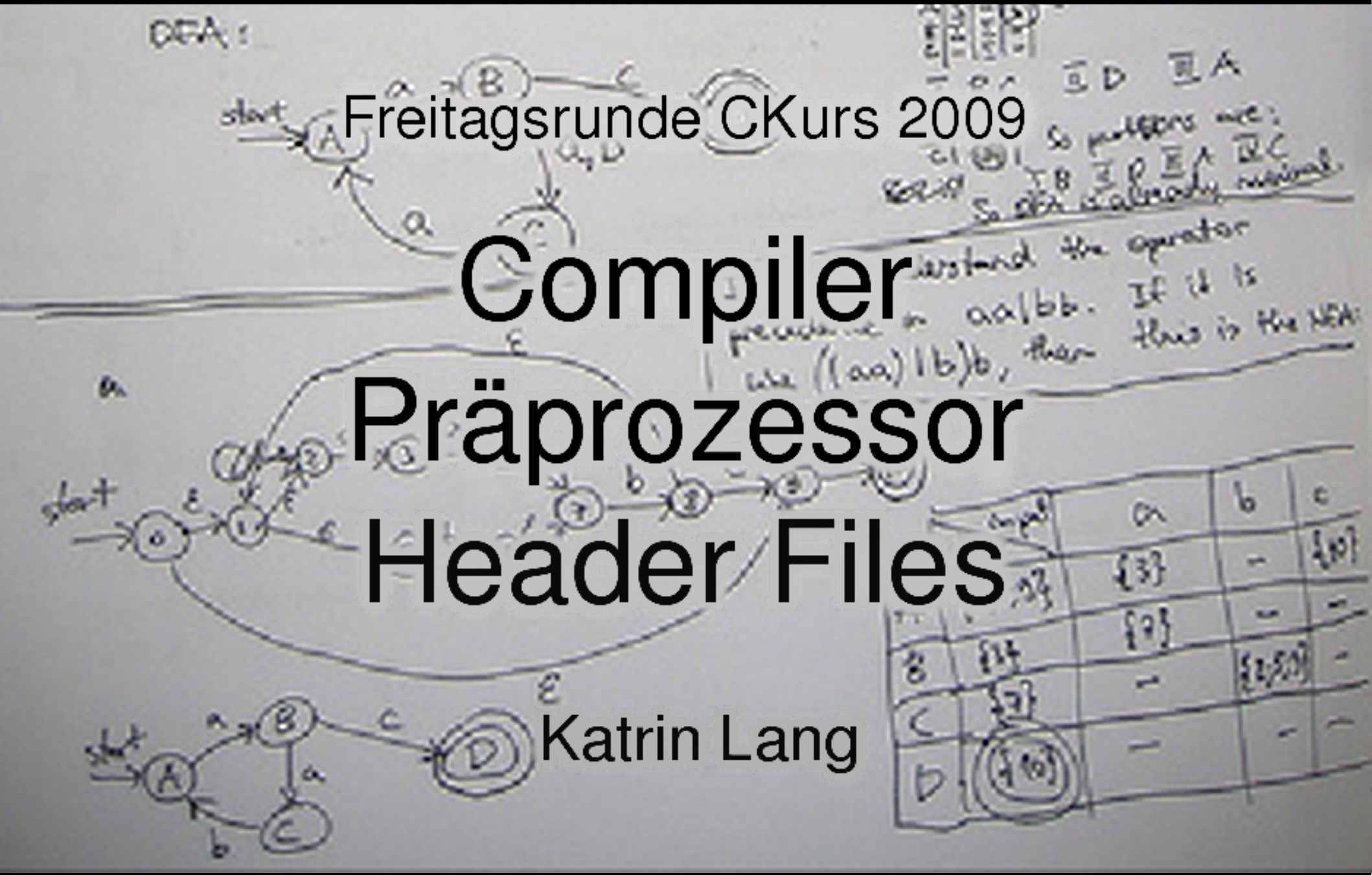


Freitagsrunde CKurs 2009

Compiler Präprozessor Header Files

Katrin Lang

Hello World Revisited

Datei hello.c

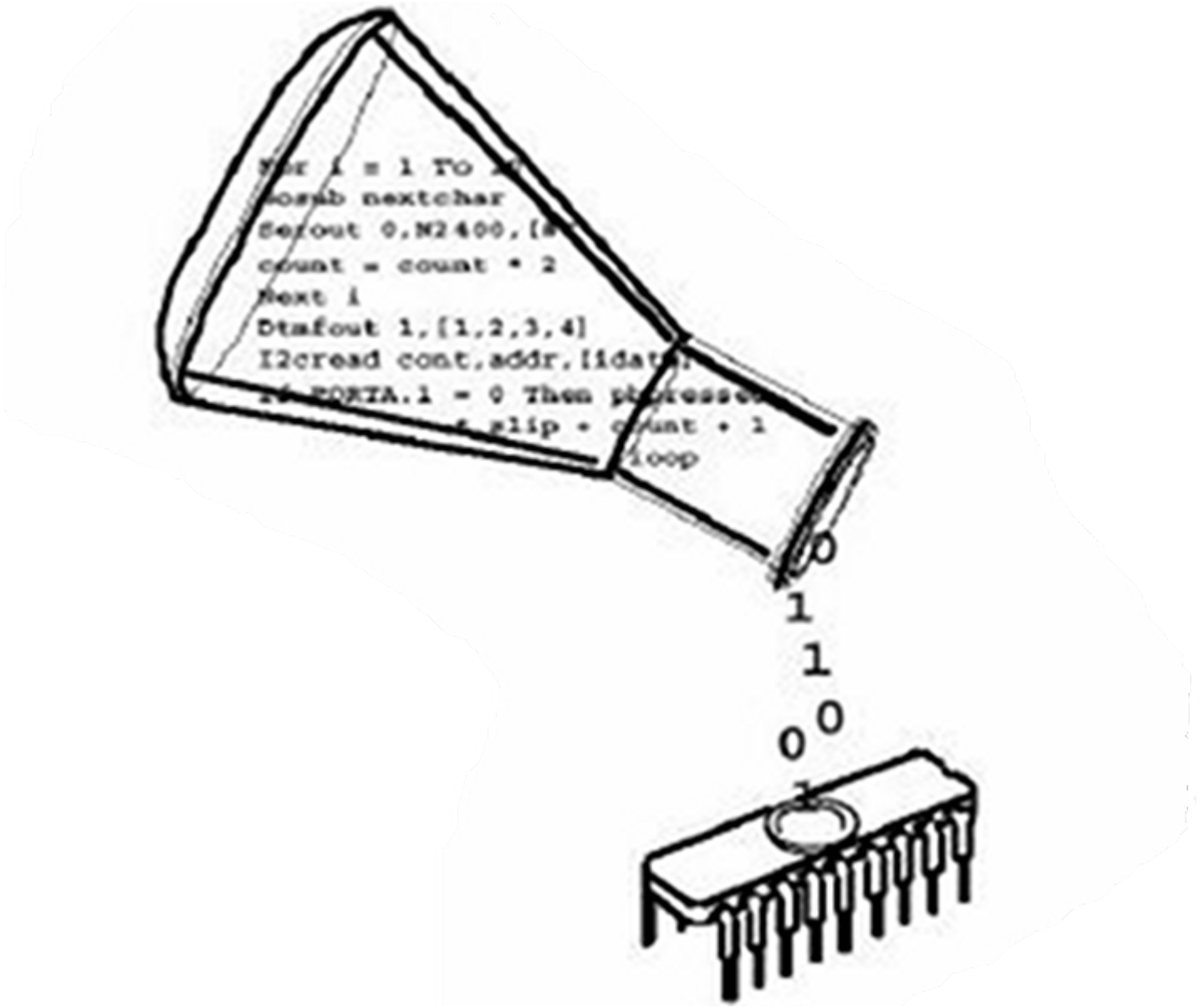
```
#include <stdio.h>

int main(){

    printf("Hello, World!\n");
    return 0;
}
```

Kommandozeile

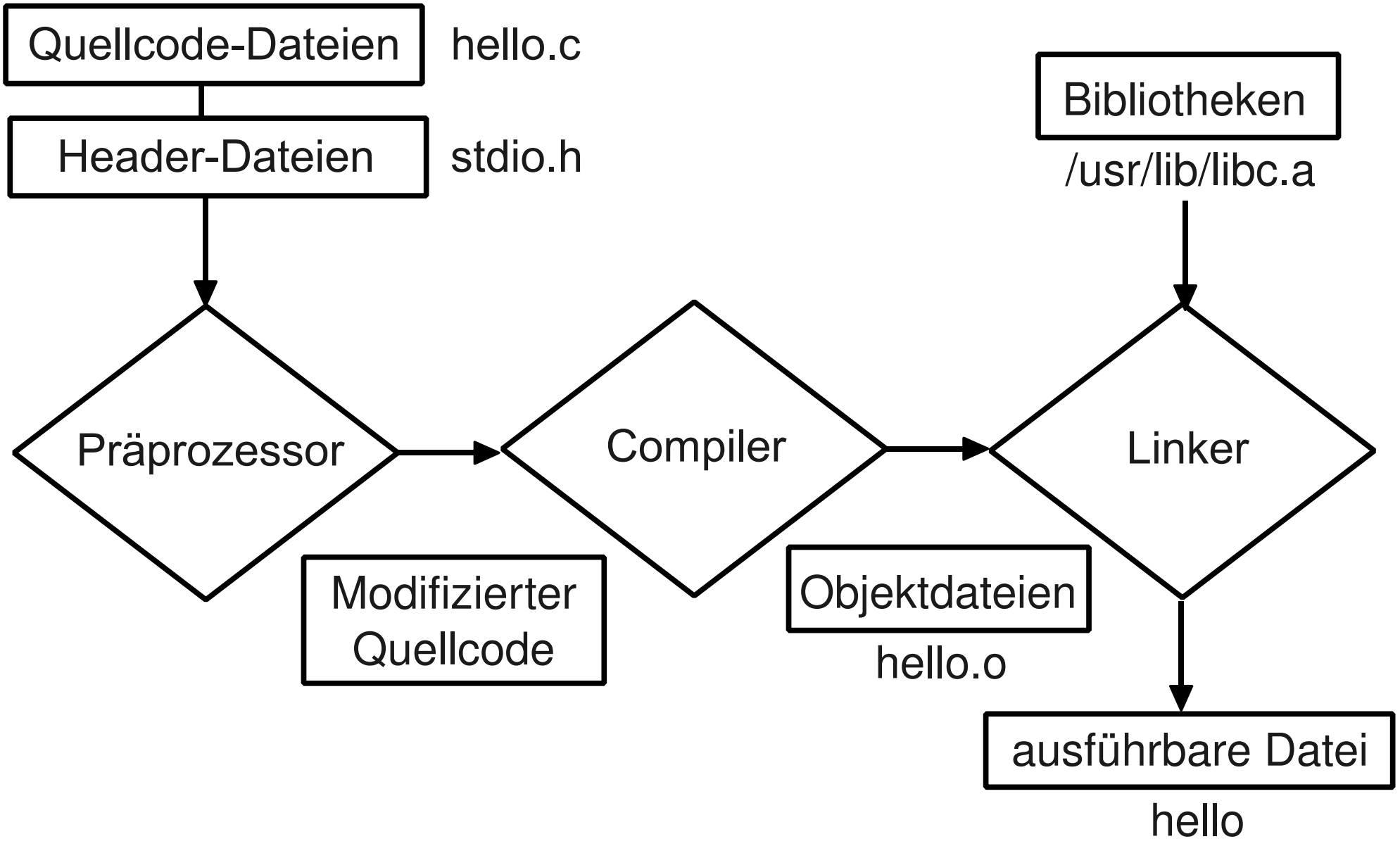
```
$ gcc -o hello hello.c
$ ./hello
Hello, World!
$
```

hello kann nach der Kompilierung wie jedes andere Unix-Kommando ausführt werden

Der C-Compiler



Kompilierung als mehrstufiger Prozess



Hello World bestehend aus mehreren Dateien

Datei main.c

```
extern hello(char* who);
extern bye(char* who);

int main(){

    hello("World");
    bye("World");
    return 0;
}
```

Dateien hello.c und bye.c

```
#include <stdio.h>
hello(char* who) {
    printf("Hello, %s!\n", who);
}
```

```
#include <stdio.h>
bye(char* who) {
    printf("Bye, %s!\n", who);
}
```

Separate Kompilierung

```
$ ls
bye.c       hello.c         main.c
$ gcc -c hello.c
$ gcc -c main.c
$ gcc -c bye.c
$ ls
bye.c       hello.c         main.c
bye.o       hello.o         main.o
$
```

Linken

```
$ gcc -o hello main.o hello.o bye.o
$ ./hello
Hello, World!
Bye, World!
$
```

Typen von Fehlern

- Präprozessor-Fehler, z.B
  - falsch geschriebene Präprozessoranweisung
  - undefinierte symbolische Konstante
- Compiler-Fehler, z.B
  - Syntaxfehler
  - Typfehler
- Linker-Fehler, z.B

  undefined reference to `foo'
  collect2: ld returned 1 exit status
- Laufzeitfehler, z.B.
  - divide by zero
  - Speicherzugriffsfehler: segmentation fault / bus error

Der Präprozessor

Präprozessoranweisungen

- am Zeichen # zu Beginn der Anweisung zu erkennen
- der Präprozessor bearbeitet nur Zeilen beginnend mit #

- Einfügen von Dateien:
  - #include
- Ersetzen von Text (Makros):
  - #define
- Bedingte Kompilierung:
  - #if, #ifdef, #ifndef, #else, #elif, #endif

Parameterlose Makros

- Syntax: **#define** name [replacement]

- Anwendung: Definition von symbolischen Konstanten

  #define PI 3.141529

- Präprozessor ersetzt vor der Kompilierung jedes Vorkommen von PI mit 3.141529

- Erhöht die Lesbarkeit und Wartbarkeit des Programms

Makros mit Parametern

- Syntax: #define name(dummy1[,dummy2][,...]) tokenstring
- Expression Makros
  - übersetzt in einen Ausdruck
  - ähnlich einer Funktion, die einen Wert zurückgibt

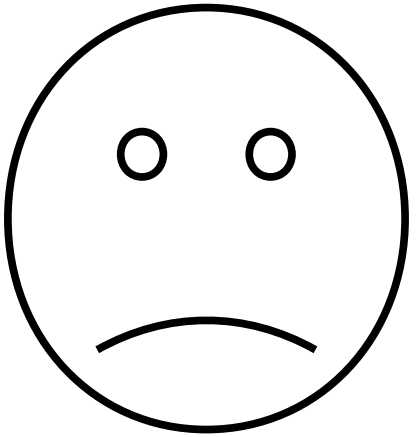
```
#define RADTODEG(x) ((x) * 57.29578)
```

- Statement Makros
  - übersetzt in ein oder mehrere volle C statements
  - ähnlich einer Funktion, die void zurückgibt

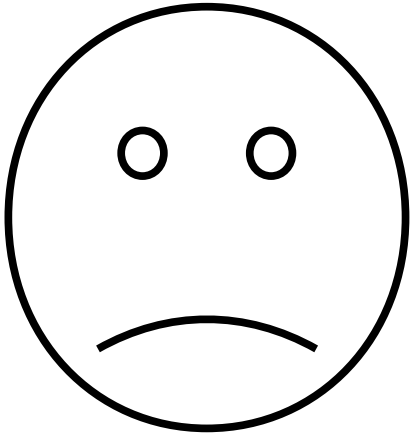
```
#define SWAP(x, y) \
  do{  tmp = x;   x = y;   y = tmp;  } while(0)
```

Warum Klammern?

```
#define RADTODEG(x) (x * 57.29578)

RADTODEG(a + b)

(a + b * 57.29578)
```



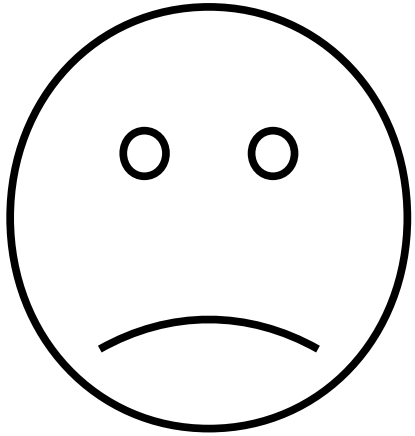
```
#define RADTODEG(x) (x) * 57.29578

1 / RADTODEG(a)

1 / (a) * 57.29578
```



Warum kein Semikolon am Ende eines Makros?

Semikolon beim Aufruf sieht natürlicher aus

-> Programmierer könnten Semikolons doppelt setzen
und damit den Control Flow eines Programms ändern



```
#define RADTODEG(x) ((x) * 57.29578);
#define DEGTORAD(x) ((x) * 0.017453);
```

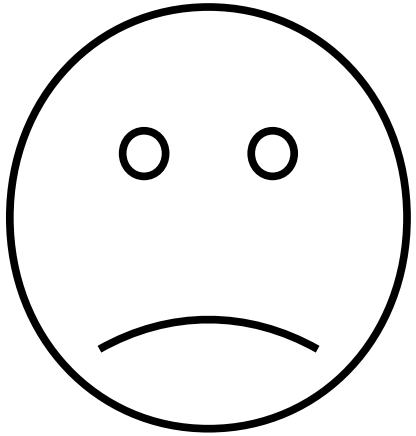
```
if(to_degree)
    y= RADTODEG(x);
else
    y= DEGTORAD(x);
```

```
if(to_degree)
    y= ((x) * 57.29578);;
else
    y= ((x) * 0.017453);;
```

Warum kein Semikolon am Ende eines Makros?

zusätzliches Problem bei Expression Makros

#define RADTODEG(x) ((x) * 57.29578);



if(RADTODEG(x)>180) ⟶ **if**(((x) * 57.29578);>180)

y= RADTODEG(x)+180; ⟶ y= ((x) * 57.29578);+180;

Mehrzeilige Statement Makros

```
/*
 * Swaps two values.
 * Requires tmp variable to be defined.
 */

#define SWAP(x, y) \
    do{ \
        tmp = x; \
        x = y; \
        y = tmp; \
    } \
    while(0)
```

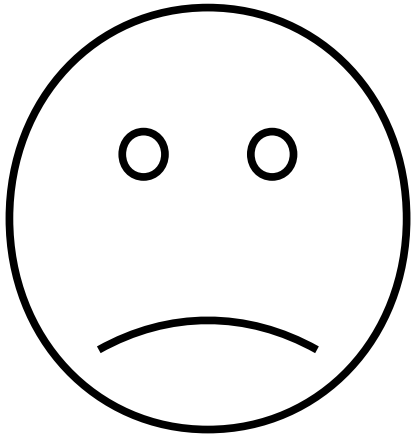
```
/*
 * Swaps two values.
 * Requires type passed as parameter
 */

#define SWAP(x, y, type) \
    do{ \
        type tmp = x; \
        x = y; \
        y = tmp; \
    } \
    while(0)
```

Elegante Lösung (nur gcc)

```
/*
 * Swaps two values.
 * Requires gcc typeof extension
 */

#define SWAP(x, y) \
    do{ \
        typeof(x) tmp = x; \
        x = y; \
        y = tmp; \
    } \
    while(0)
```

Warum do{...} while(0) ?

```
#define SWAP(x, y) \
    tmp = x; \
    x = y; \
    y = tmp
```

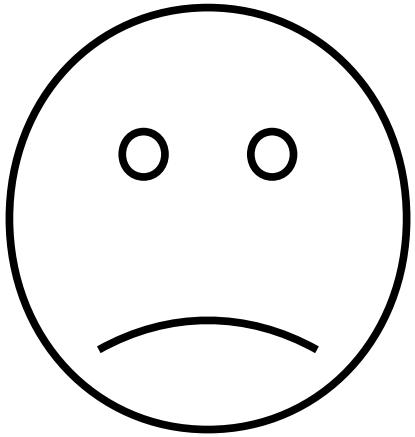


```
int x, y, tmp;
if (x > y)
    SWAP(x, y);
```

```
int x, y, tmp;
if(x > y)
    tmp = x;
x = y;
y = tmp;
```

Warum do{...} while(0) ?

```
#define SWAP(x, y){ \
    tmp = x; \
    x = y; \
    y = tmp; \
}
```



```
int x, y, tmp;
if (x > y)
    SWAP(x, y);
else
    ...
```

```
int x, y, tmp;
if (x > y) {
    tmp = x;
    x = y;
    y = tmp;
};
else
    ...
```

Zusammenfassung

- Wenn ein Dummy-Argument für einen Wert (oder einen Pointer zu einem Wert) steht, alle Vorkommen im Tokenstring klammern
- Den gesamten Tokenstring von Expression Makros klammern
- Keine Semikolons am Ende eines Makros
- Mehrzeilige Statement Makros mit \ trennen
- Tokenstring von mehrzeiligen Statement Makros mit do{..} while(0) umschließen

Weitere Probleme

- das Symbol erscheint beim Testen des Programms mittels Debugger nicht mehr
- Makros können sich ungewollt überschreiben
- Makros werden erst expandiert und daher vom Compiler überprüft, wenn sie tatsächlich aufgerufen werden
- Mehrfache Seiteneffekte

```
#define MIN(a, b) ((a)>(b)?(b):(a))
x = y = 1;
MIN(++x, ++y);
```

Makro Styleguide

- Namenskonvention Großbuchstaben
- Wenn möglich, Funktionen statt Makros verwenden
- Alternativen:
  - Parameterlose Makros – const (C90)
  - Makros mit Parametern – inline (C99)
- Vorsicht bei Makros zur Optimierung
- Makros kurz halten
- keine Hacks!

Bedingte Kompilierung

```
#ifdef _WIN32
    /* do Windows specific stuff here */
#endif
#ifdef __APPLE__
    /* do Mac specific stuff here */
#endif
#ifdef __linux__
    /* do Linux specific stuff here */
#endif
```

Beispiel: Debugging

```
#define PI 3.141529
#define RADTODEG(x) ((x) * 57.29578)

int debug= 1;

int main(){
    ...
    if(debug)
        printf("PI %f", RADTODEG(PI));
    ...
    return 0;
}
```

Ein einfaches Debugging Makro

Datei debug.h

```
#include <stdio.h>
#define DEBUG

#ifdef DEBUG
#define LOG printf
#else
#define LOG if(0) printf
#endif
```

Unser neues Hello World

Datei hello.c

```
#include “debug.h“

int main(){

    LOG("Hello World!\n");

    return 0;
}
```

Output des Präprozessors (gcc)

```
$gcc -E hello.c
... 748 weitere Zeilen
# 11 "hello.c"
int main(){

    printf("Hello, World!\n");
    return 0;
}
$
```

define per Kommandozeile (gcc)

gcc [-**D**macro[=defn]...] infile

%gcc -DDEBUG hello.c

% gcc -DDEBUG -DVERBOSE=1 hello.c

Hinzufügen von Debug Levels

Datei debug.h

```
#include <stdio.h>
#define VERBOSE 0

#ifdef DEBUG
#define LOG printf
#else
#define LOG if (0) printf
#endif
```

Unser neues Hello World mit Debug Levels

Datei hello.c

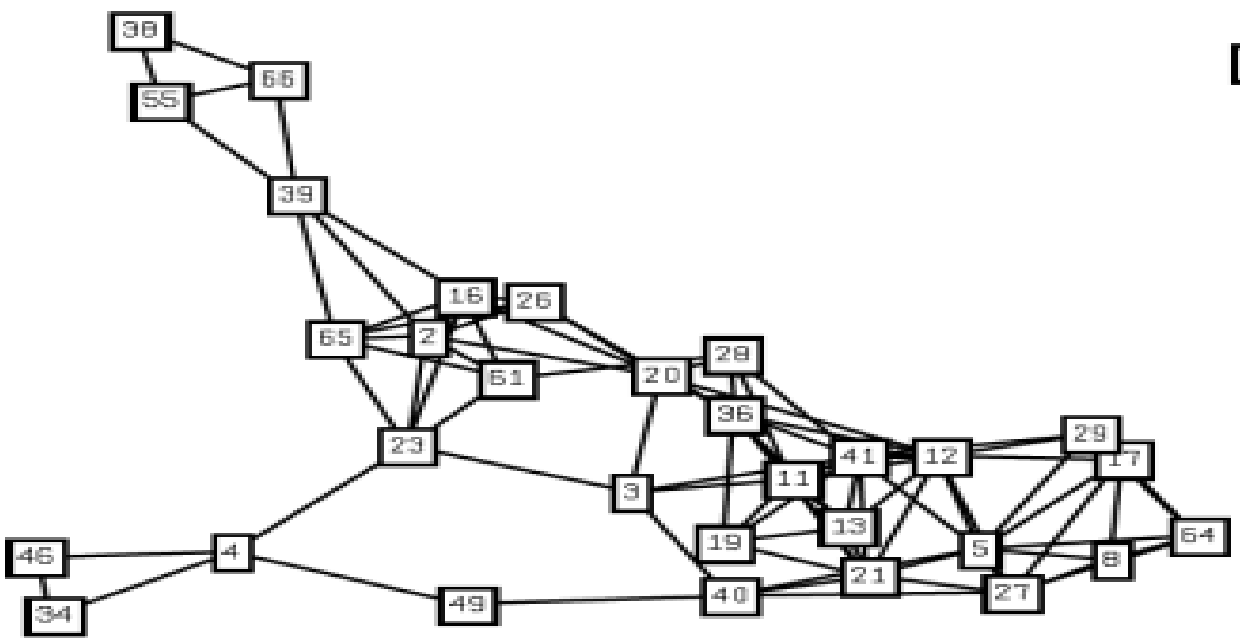
```
#include “debug.h“

int main(){

    if(VERBOSE== 1) LOG("Hello World!\n");

    return 0;
}
```

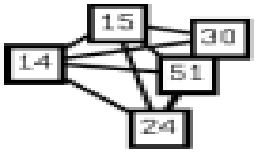
Debug Levels mit bedingter Kompilierung

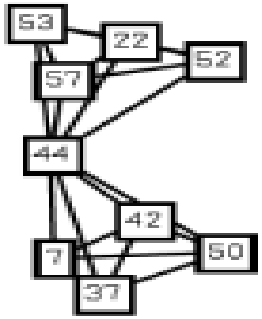
Datei hello.c

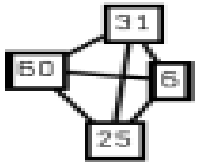
```
#include “debug.h“

int main(){

    #if VERBOSE==1
        LOG("Hello World!\n");
    #endif
    return 0;
}
```

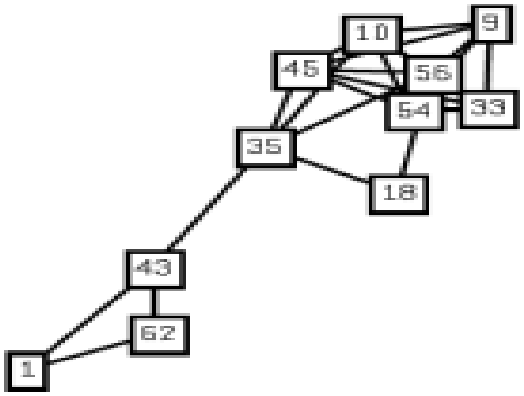
Header-Dateien



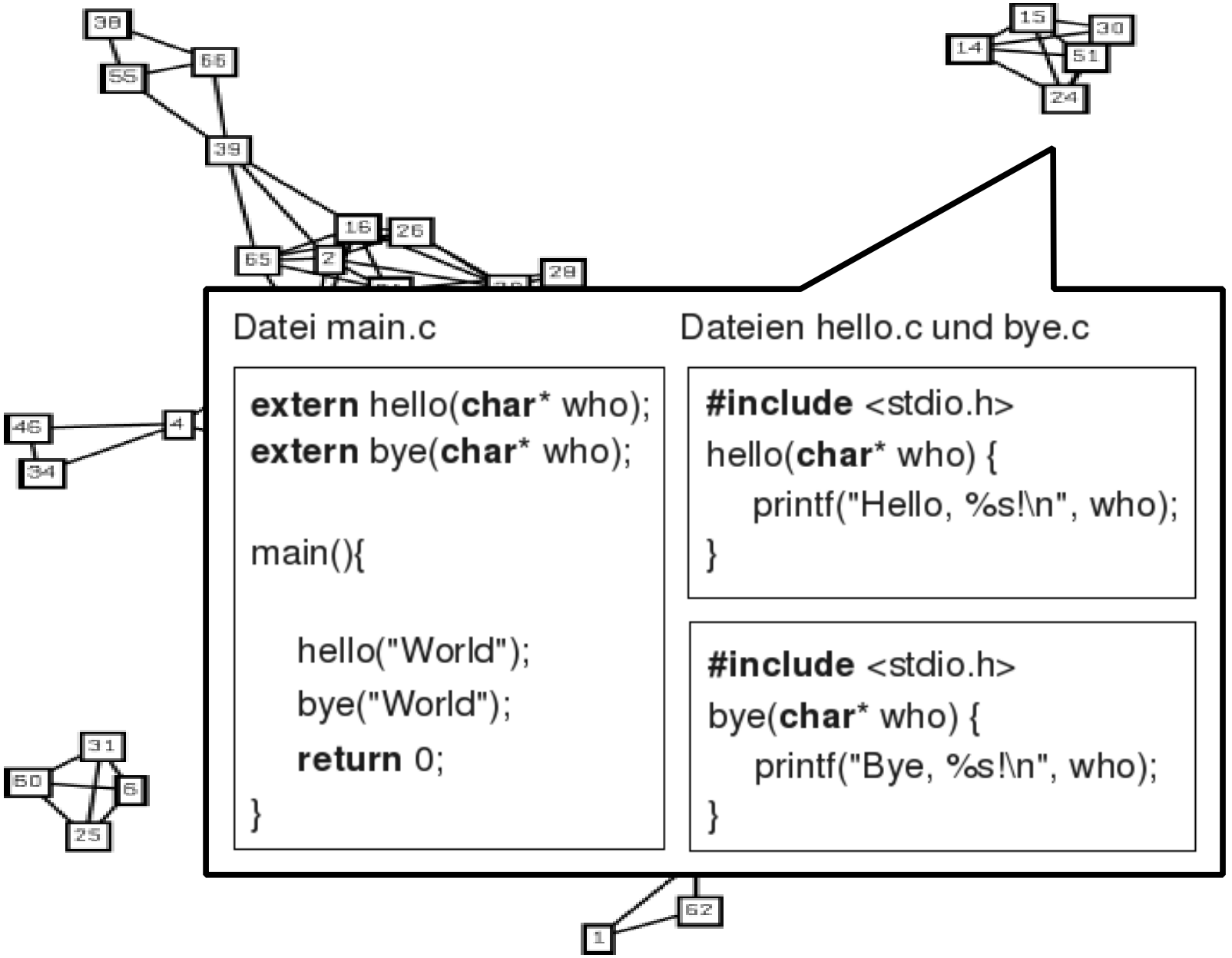








Header-Dateien



Header-Dateien

- Header Dateien erkennt man an der Endung ".h“
- Sie sind Teil von Schnittstellen zwischen Systemen
- Sie enthalten:
  - Funktions-Deklarationen
  - globale Variablen
  - symbolische Konstanten
  - Makros
  - Datentypen (z.B. Strukturen)

Inkludieren von Header-Dateien

#include <name>

- sucht zuerst im Verzeichnis der Systemdateien
- erst dann im Verzeichnis der Quelldatei
- wird normalerweise verwendet, um Headerdateien, die vom System geliefert werden, einzubinden (z.B. #include <stdio.h>)

#include "name"

- sucht zuerst im Verzeichnis der Quelldatei
- erst dann im Verzeichnis der Systemdateien
- wird normalerweise verwendet, um selbst geschriebene Header-Dateien einzubinden (z.B. #include "debug.h")

-I-Compileroption (gcc)

- Erweitert beim Übersetzen eines Programmes die Liste der Verzeichnisse in denen nach einer Datei gesucht wird.

  gcc -Iinclude helloworld.c

- sucht nach stdio.h zuerst als include/stdio.h, und erst dann als /usr/include/stdio.h.

Präprozessor-Output von Hello World

```
# 304 "/usr/include/stdio.h" 3 4

extern int printf (__const char *__restrict __format, ...);
```

Problem: Mehrfachinklusion

Datei foo.h

```
#include “bar.h“
#include “baz.h“
...
```

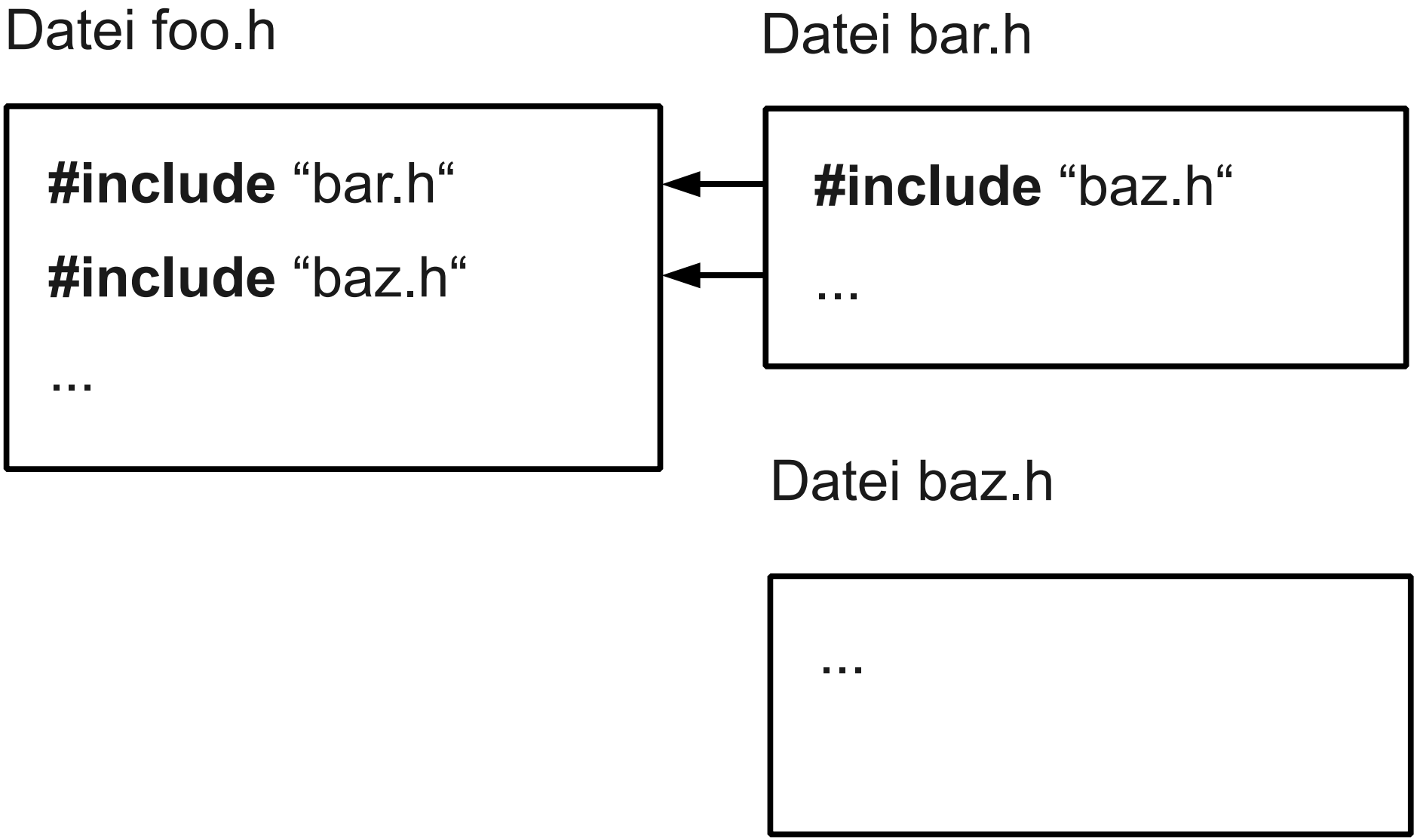
Datei bar.h

```
#include “baz.h“
...
```

Datei baz.h

```
...
```

Problem: Mehrfachinklusion



Problem: Mehrfachinklusion

Datei foo.h

```
#include “bar.h“
#include “baz.h“
...
```

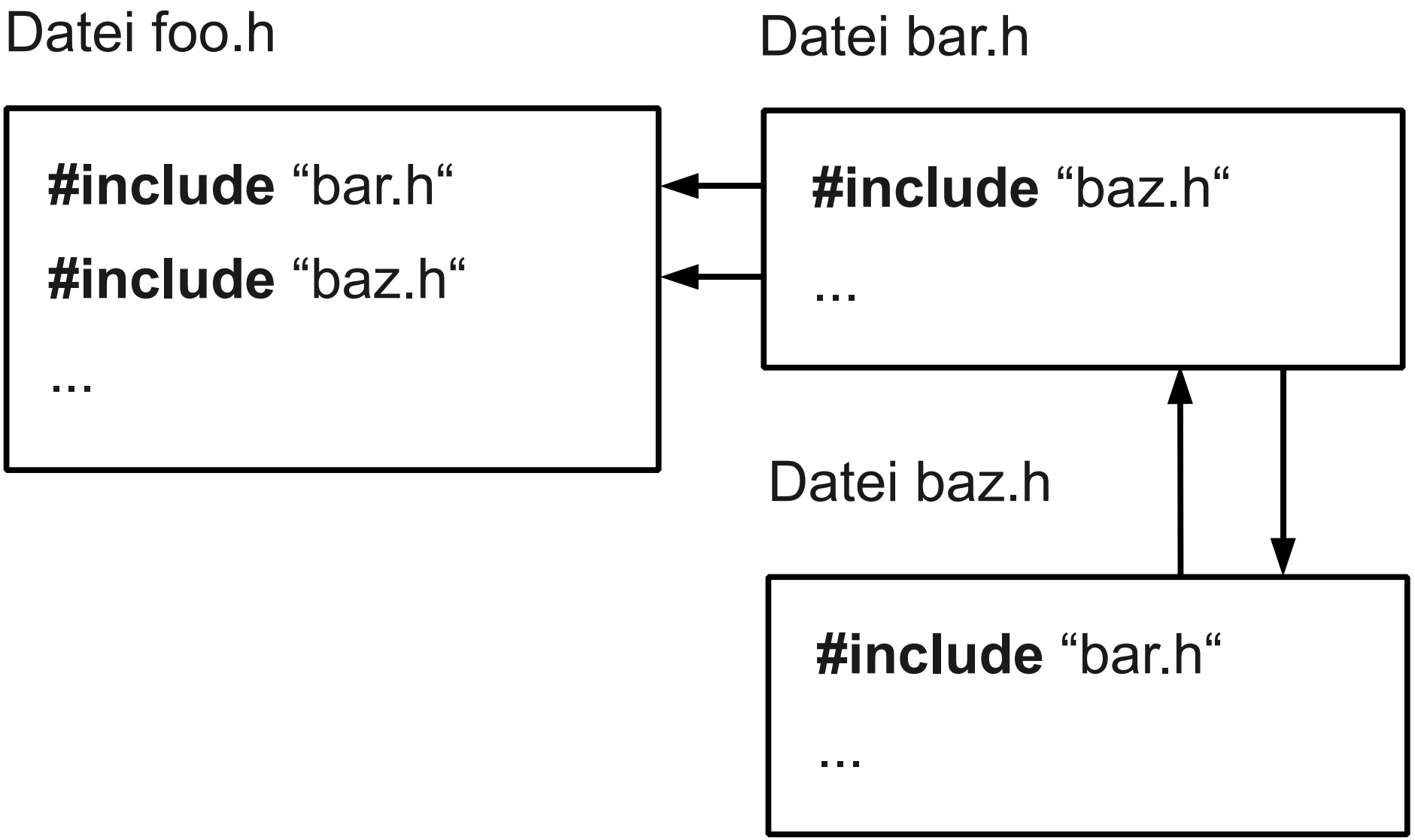
Datei bar.h

```
#include “baz.h“
...
```

Datei baz.h

```
#include “bar.h“
...
```

Problem: Mehrfachinklusion



Vermeidung von Mehrfachinklusion

Datei foo.h

```
#ifndef FOO_H
   #define FOO_H


   extern int foo(int x, int y);


#endif
```

Unknown compiler

Scrapbook
Melbourne, [1860–1916]
Louise Hanson-Dyer Music Library Rare Collections, University of Melbourne

Danke!