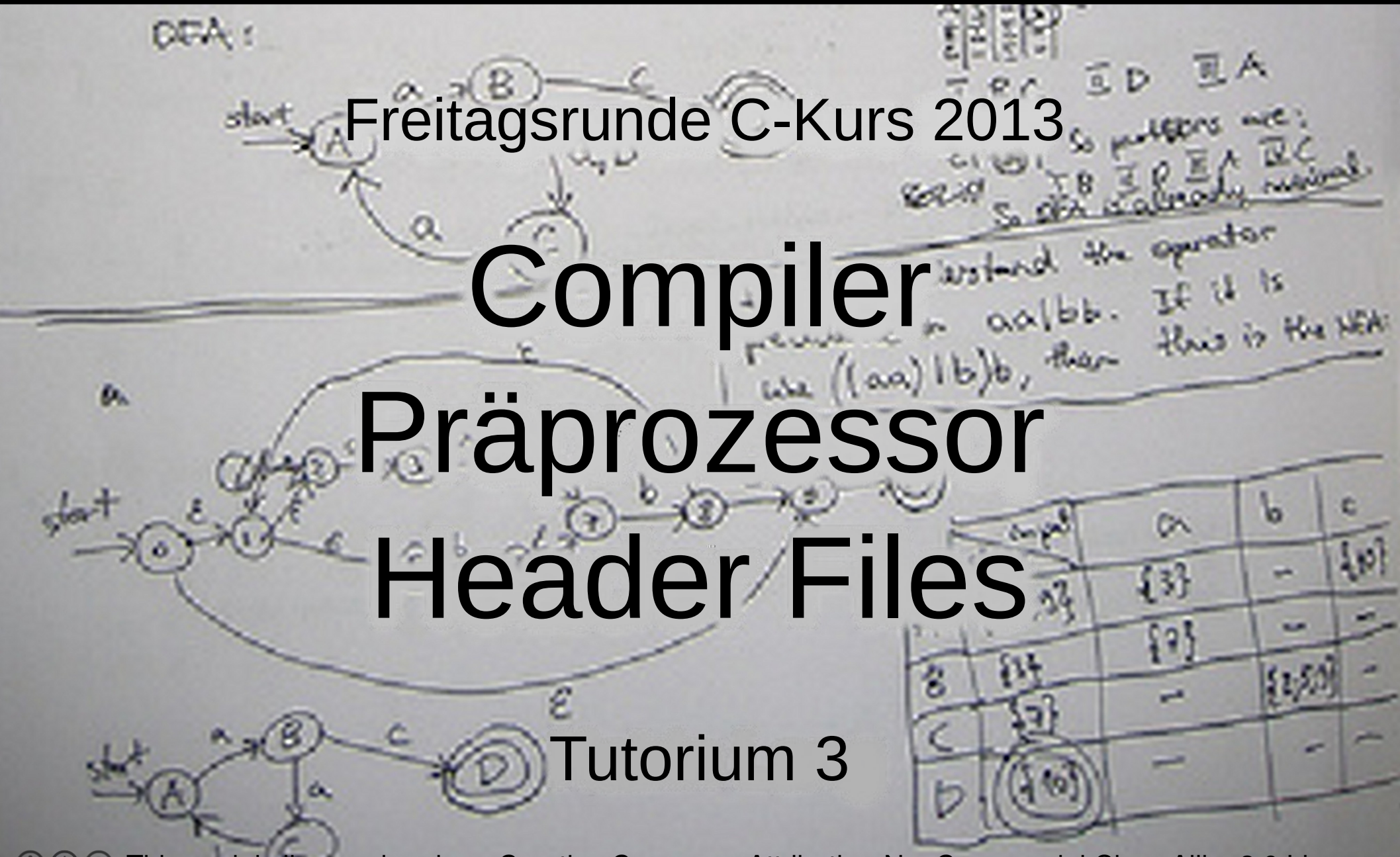


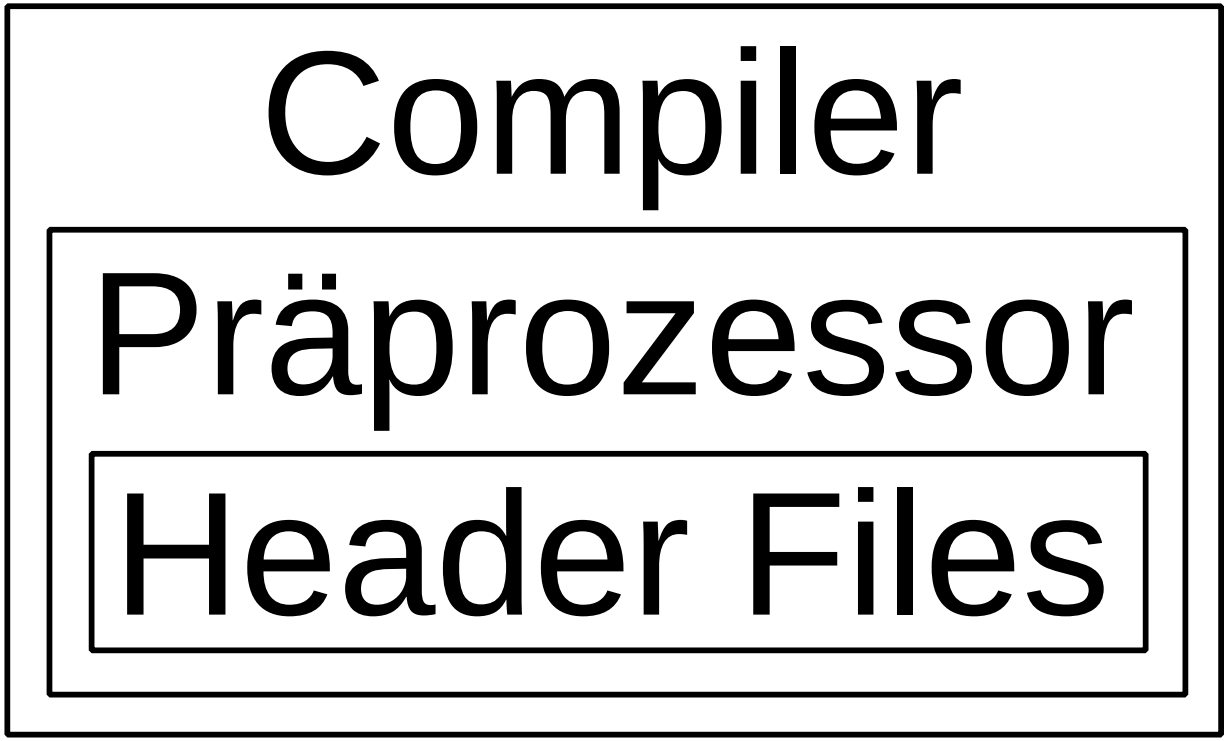
Freitagsrunde C-Kurs 2013

Compiler Präprozessor Header Files

Tutorium 3

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License

Gliederung



Hello World Revisited

Datei hello.c

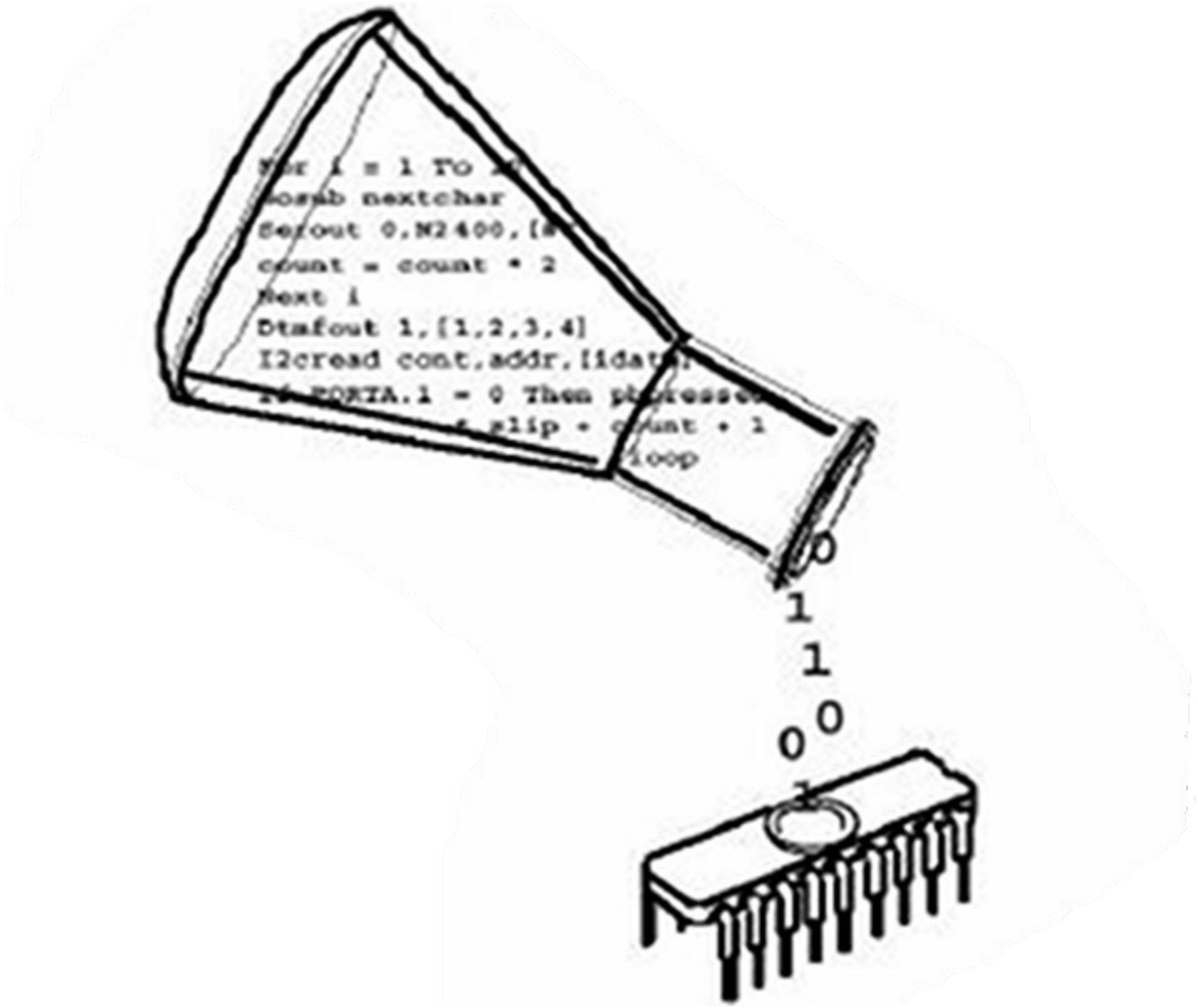
```
#include <stdio.h>

int main(){

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

Kommandozeile

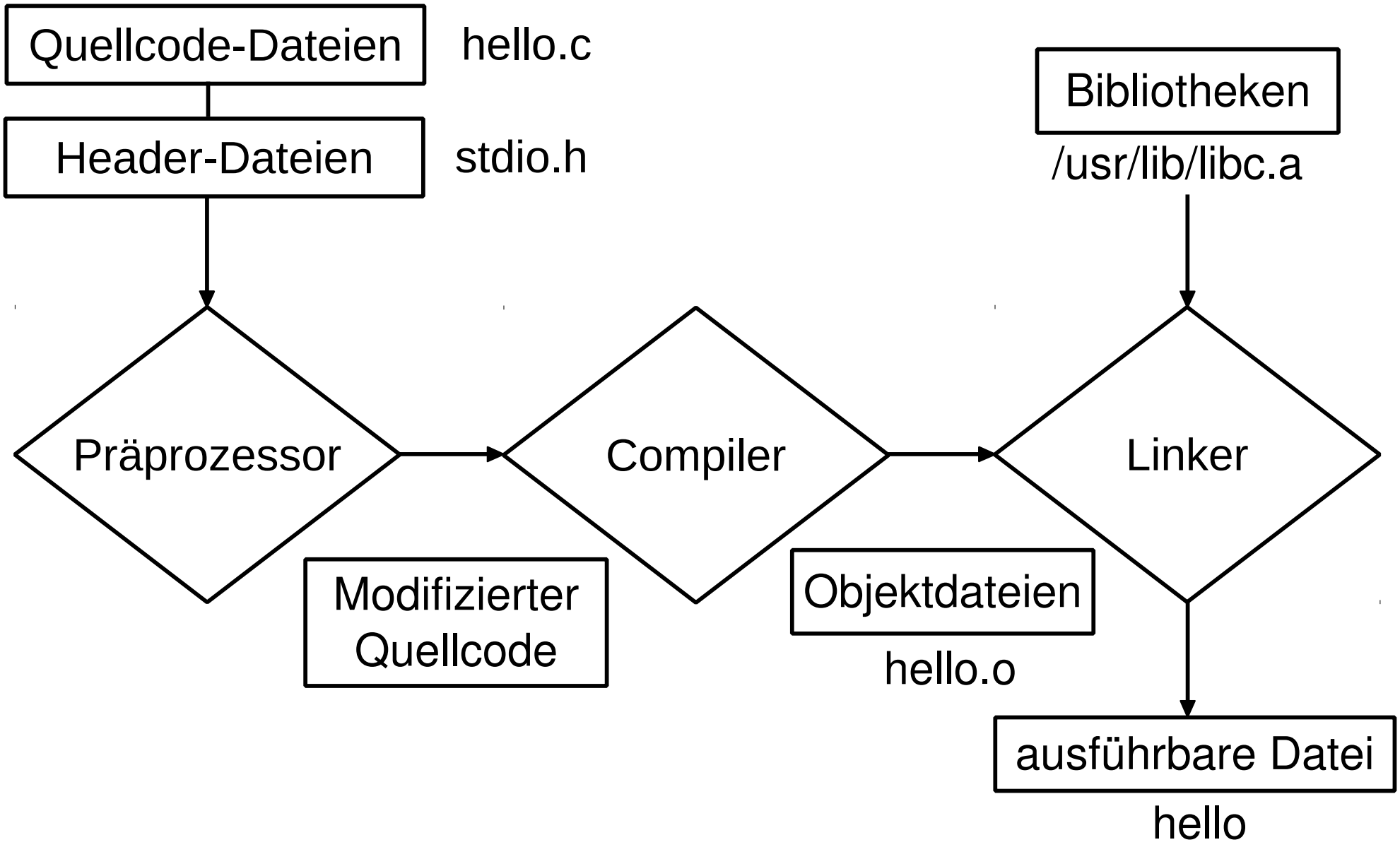
```
$ gcc -o hello hello.c
$ ./hello
Hello, World!
$
```

hello kann nach der Kompilierung wie jedes andere Unix-Kommando ausgeführt 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 main.c
$ gcc -c hello.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!
$
```

Datei Makefile

```
all: main

main: main.o hello.o bye.o
→ $(CC) -o hello main.o hello.o bye.o

hello.o: hello.c
→ $(CC) -c hello.c

bye.o: bye.c
→ $(CC) -c bye.c

main.o: main.c
→ $(CC) -c main.c

clean:
→ rm -f main main.o hello.o bye.o
```

Übungsaufgabe

Kopiert euch die Beispieldateien und das Makefile in einen Ordner.

➜ Kompiliert die Dateien zunächst per Hand separat und linkt sie dann zusammen! Führt jedesmal **ls** aus!

➜ Löscht jetzt alle Objekt- und ausführbaren Dateien!

➜ Führt **make** aus und seht euch die Ausgabe an!

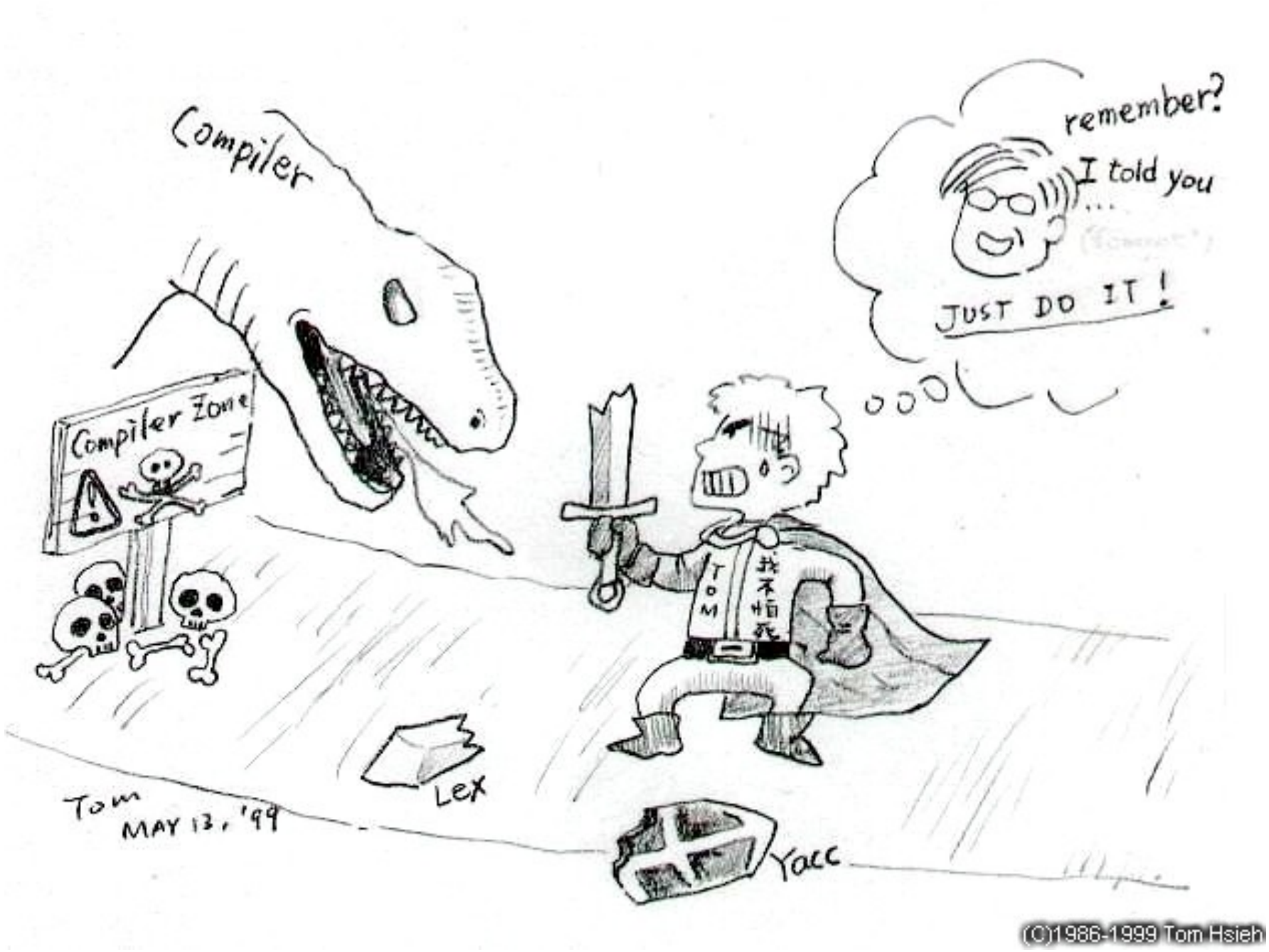
➜ Löscht die Datei **bye.o**!

➜ Führt erneut **make** aus und seht euch die Ausgabe an!

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 `hello'
    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 erkennt nur Zeilen beginnend mit #

Präprozessoranweisungen

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

- Einfügen von Dateien:

  ➜ #include
- Ersetzen von Text (Makros):

  ➜ #define
- Bedingte Kompilierung:

  ➜ #if, #ifdef, #ifndef, #else, #elif, #endif

Präprozessoranweisungen

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

- Einfügen von Dateien:

  ➜ #include

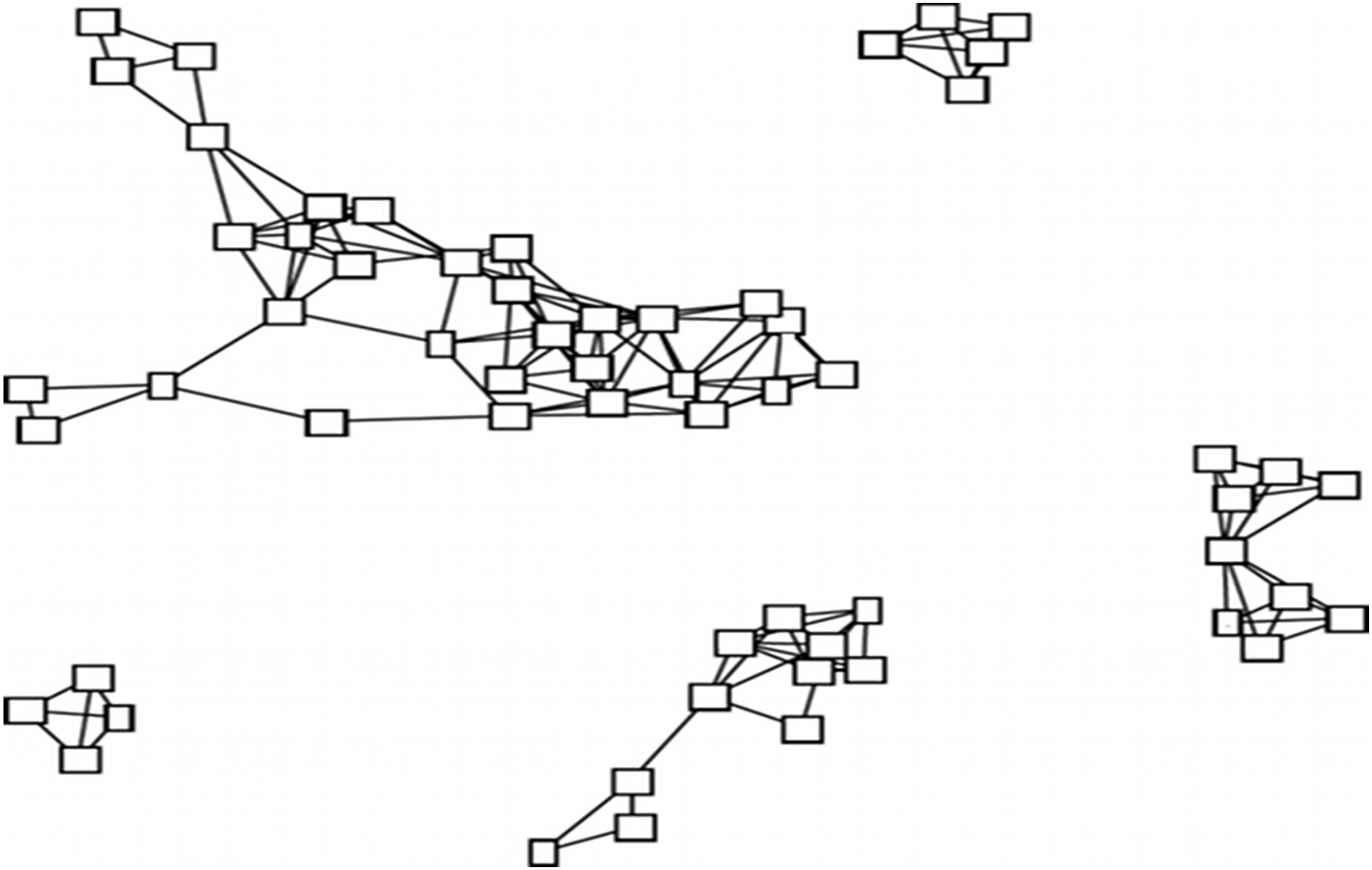
- Ersetzen von Text (Makros):

  ➜ #define

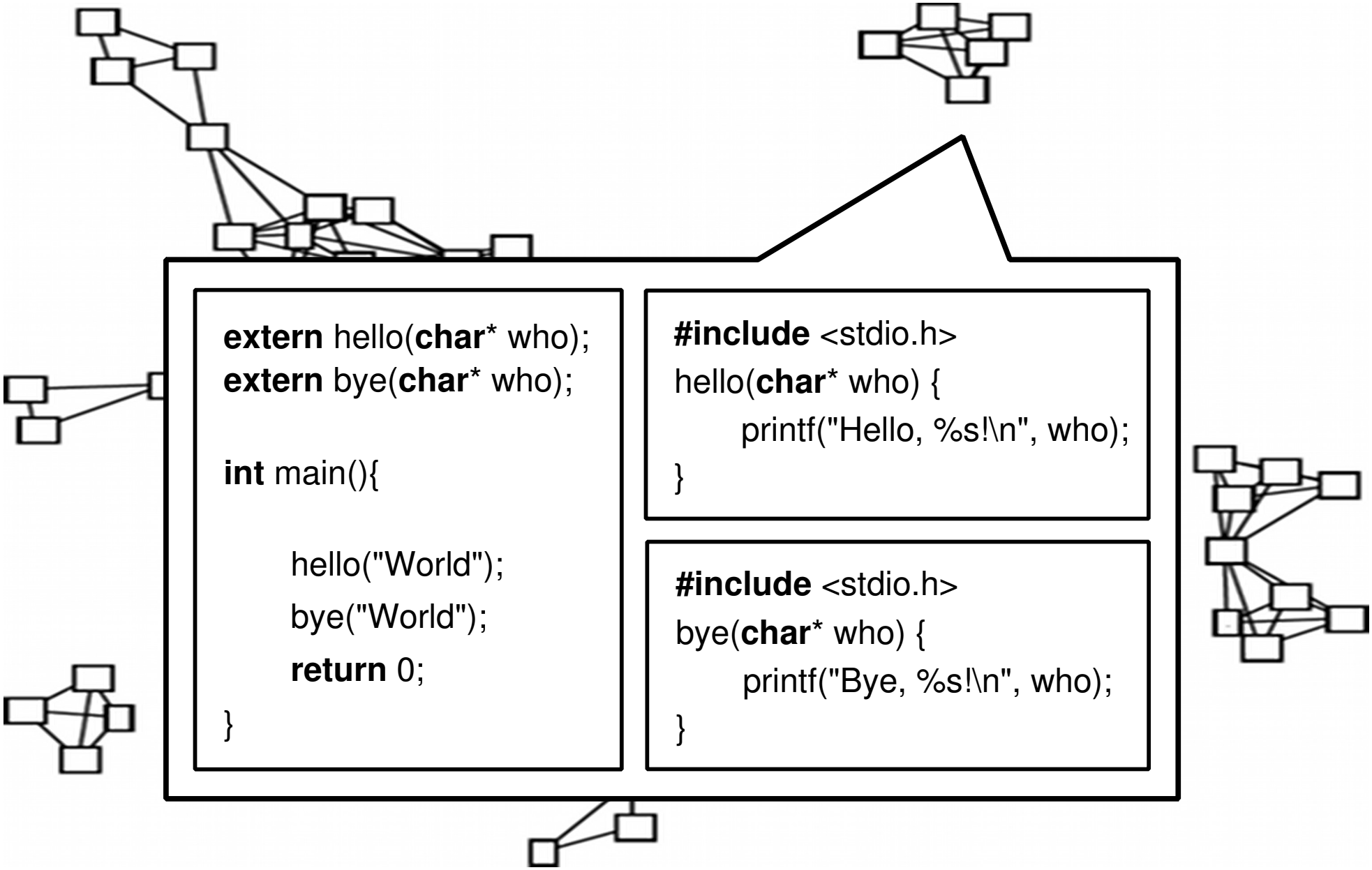
- Bedingte Kompilierung:

  ➜ #if, #ifdef, #ifndef, #else, #elif, #endif

Header-Dateien



Header-Dateien



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

int main(){

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

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

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

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 im Verzeichnis der Systemdateien
- wird 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 selbstgeschriebene 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 hello.c

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

-E-Compileroption (gcc)

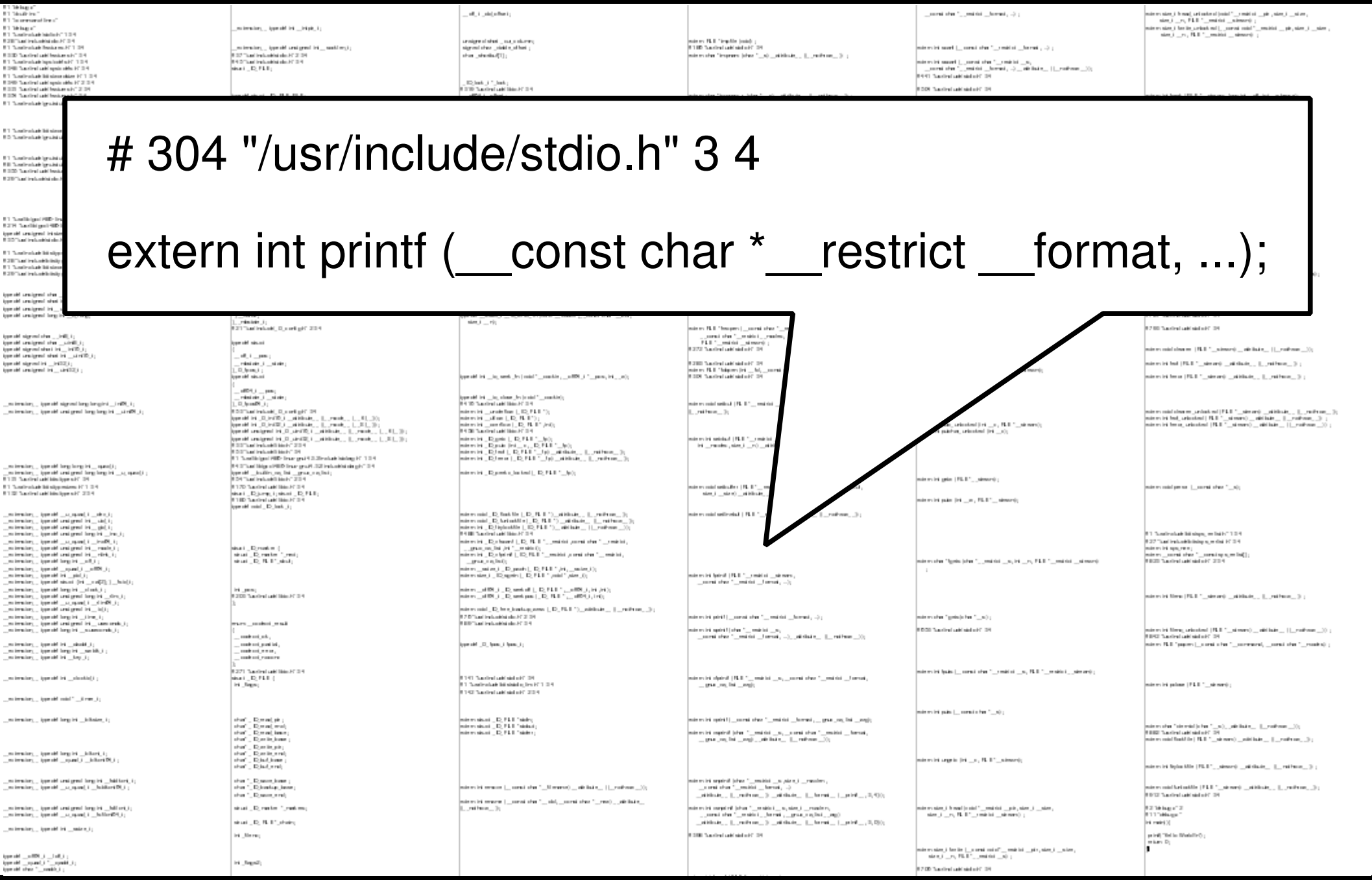
Was geschieht nun aber eigentlich beim Inkludieren eines Files?

Mit **-E** könnt ihr euch die Ausgabe des Präprozessors ansehen. Probiert's aus:

Kommandozeile:

```
$ gcc -E hello.c
```

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

```
…
```

Datei baz.h

```
…
```

Problem: Mehrfachinklusion

Datei foo.h

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

Datei bar.h

```
#include “baz.h“
...
```

Datei baz.h

```
...
```

Problem: Mehrfachinklusion

Datei foo.h

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

Datei bar.h

```
#include “baz.h“
...
```

Datei baz.h

```
...
```

Problem: Mehrfachinklusion

Datei foo.h

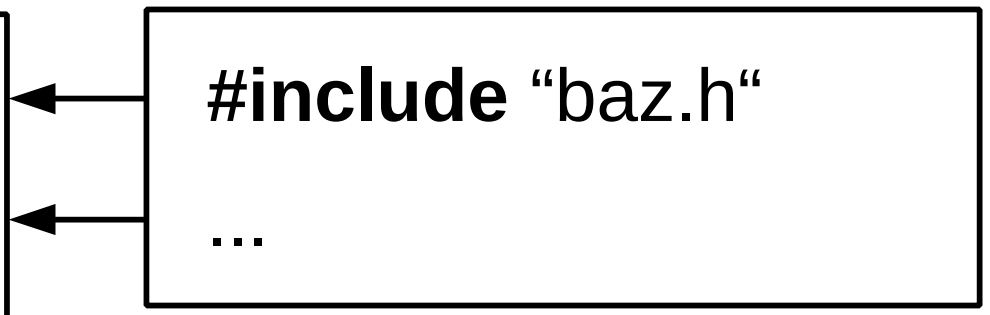
```
#include “bar.h“
#include “baz.h“
...
```

Datei bar.h

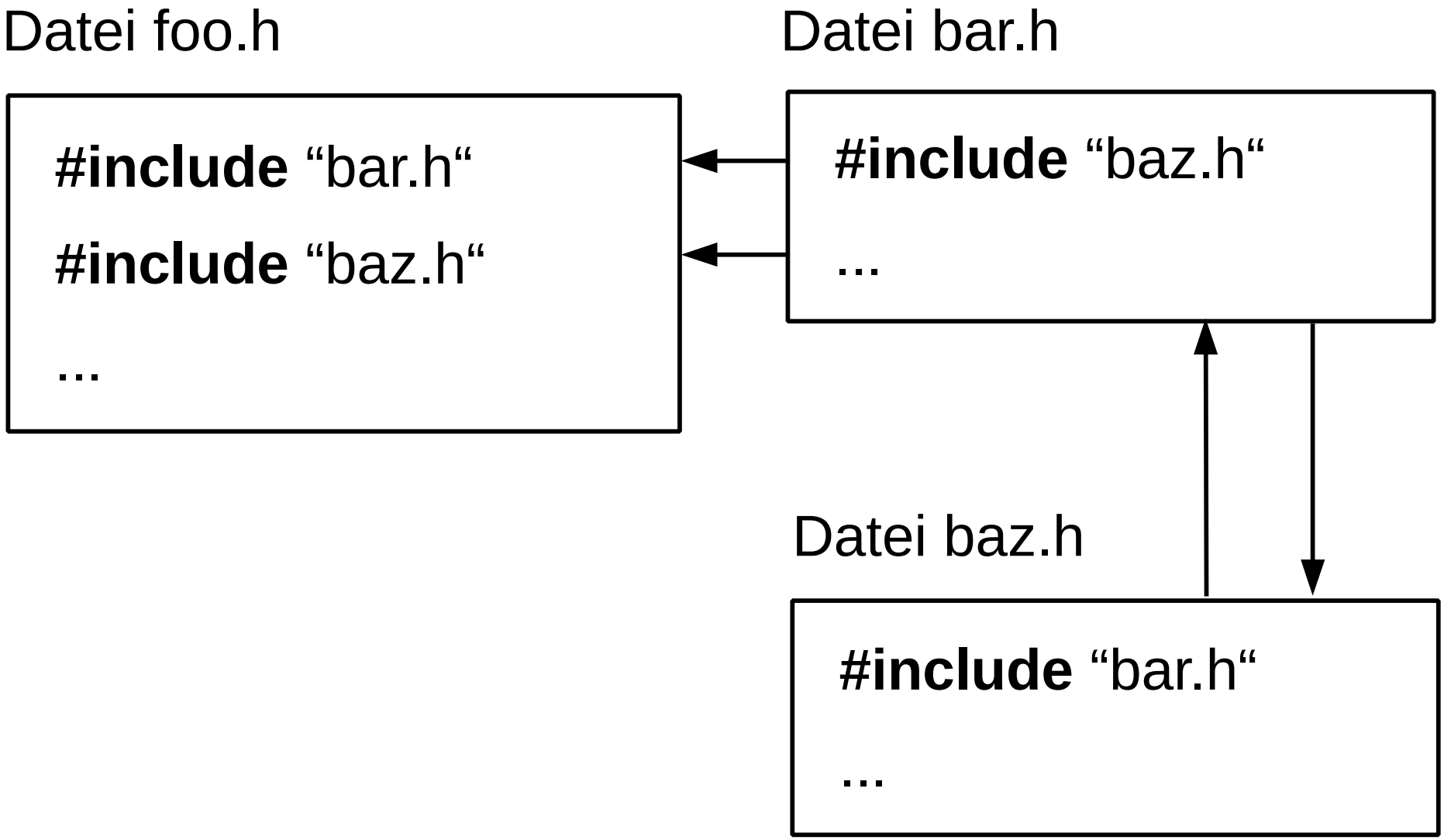
```
#include “baz.h“
...
```

Datei baz.h

```
#include “bar.h“
...
```



Problem: Mehrfachinklusion

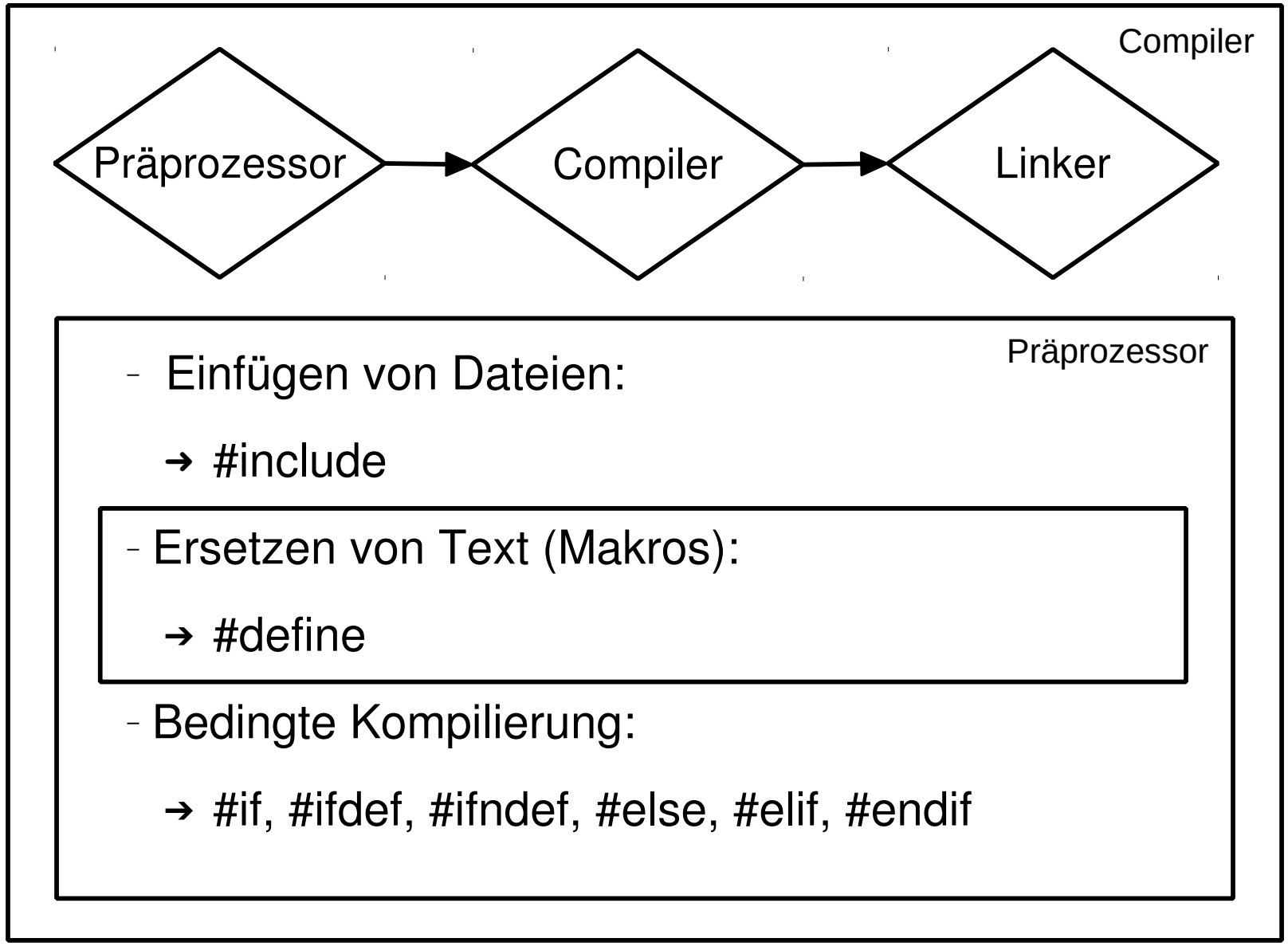


Vermeidung von Mehrfachinklusion

Datei foo.h

```
#ifndef FOO_H
   #define FOO_H


   extern int foo(int x, int y);


#endif
```

Die nötigen Sprachmittel, um diesen Code verstehen zu können, werden wir uns im Folgenden aneignen


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

Parameterlose Makros

- Syntax: **#define** NAME [replacement]
- Präprozessor ersetzt vor der Kompilierung jedes Vorkommen von **NAME** mit dem Ersetzungstext
- Fehlt das replacement, so ist der **NAME** dem System im Folgenden bekannt, anstatt undefiniert zu sein
- Um den semantischen Unterschied klarzumachen, sollten Makros immer GROSS geschrieben werden!
- Ein ausführliches Beispiel werden wir später betrachten

Syntax:

kein Leerzeichen Leerzeichen

#define NAME(dummy1 [[,dummy2], ...]) ... dummy1 ...

Parameterliste Tokenstring

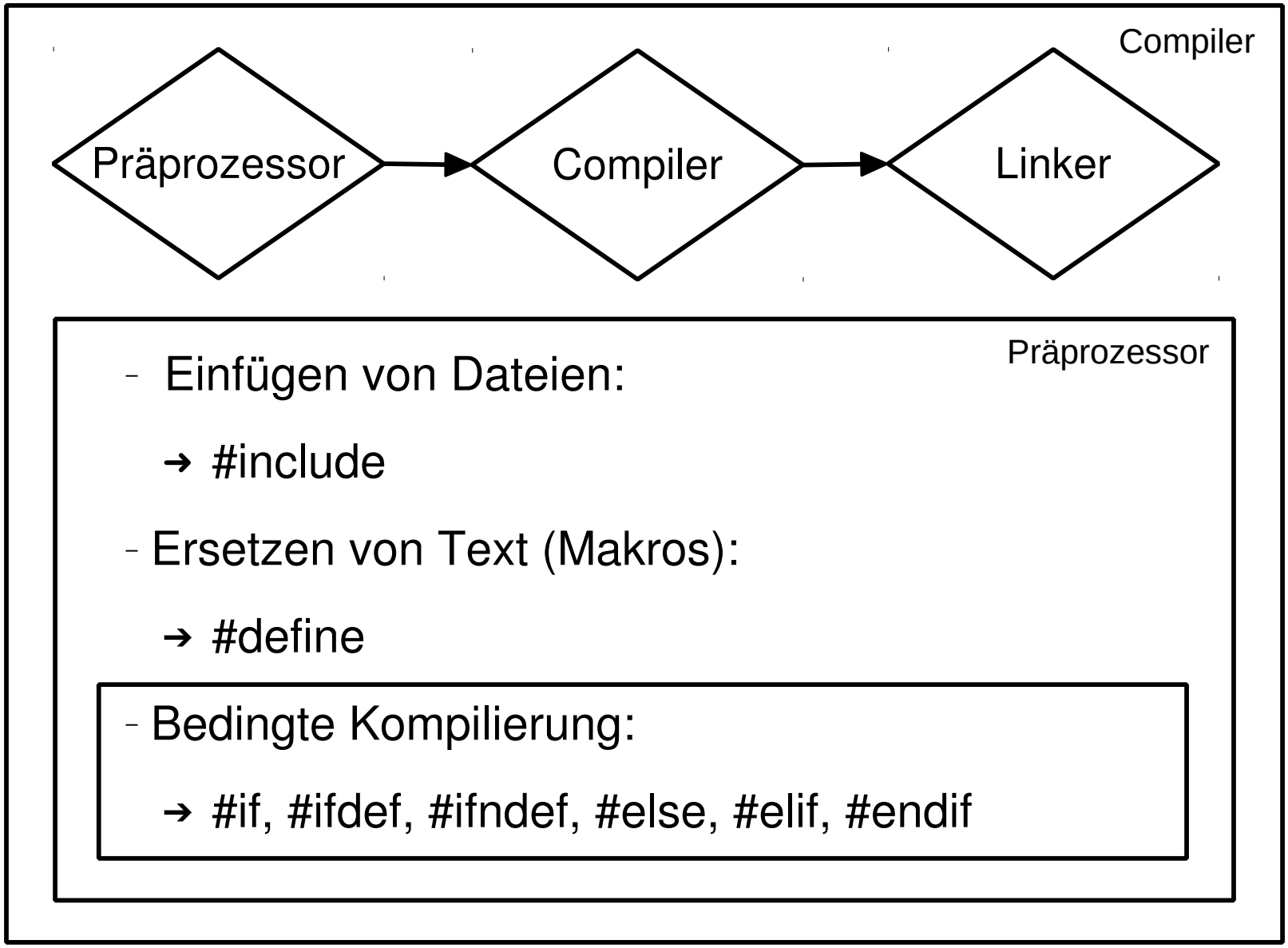
Beispiel:

#define SIZEOF(array) sizeof(array)/sizeof(array[0])

Ein Wort zur Warnung

Da man sich mit Makros sehr, sehr leicht ins Knie schiessen kann, empfehle ich euch für den Fall, dass ihr tiefer ins Thema einsteigen wollt, dringend, euch die Folien vom Vorjahr anzugucken, bevor ihr anfangt, eigene Makros zu schreiben!

Bedingte Kompilierung



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
```

Vermeidung von Mehrfachinklusion

Datei foo.h

```
#ifndef FOO_H
   #define FOO_H


   extern int foo(int x, int y);


#endif
```

Was bewirkt nun dieser Code?

Beispiel: Debugging

```
int debug= 1;

int main(){

    if(debug)
        printf("entering main...\n“);
    …
    if(debug)
        printf("exiting main...\n“);

    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=2 hello.c

Unser neues Debugging Makro

Datei debug.h

```
#include <stdio.h>

#ifndef VERBOSE
#define VERBOSE 0
#endif

#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;
}
```

Das Debugging-Makro kann man noch beliebig schöner und aussagekräftiger gestalten (Mit Angabe von Zeilennummer, Funktion, in der man sich gerade befindet, etc). Ihr könnt euch ja mal die Folien von vorigem Jahr angucken!

Übungsaufgabe

Schreibt ein Makefile für unser neues Hello-World-Beispiel!

➜ Definiert targets **release, debug0** und **debug1** für Debug-Builds mit verschiedenen verbosity levels!

➜ Beachtet, dass auch header-Dateien als Abhängigkeiten angegeben werden müssen!

➜ Kompiliert euren Code für verschiedene Debug-Levels und überzeugt euch anhand des Konsolen- sowie des Präprozessoroutputs, dass er funktioniert

Danke!