

DFA:



Freitagsrunde C-Kurs 2012

Compiler

Präprozessor

Header Files



So patterns are:
 $T \cup B \cup P \cup A \cup C$
 So DFA is already minimal.

understand the operator
 precedence in aa/bb . If it is
 like $((aa)/b)b$, then this is the NFA:



Tutorium 3

input	a	b	c
0	{3}	-	{0}
1	{2}	-	-
2	{1}	-	-
3	-	{2,5}	-
4	-	-	-
5	-	-	-
6	-	-	-
7	-	-	-
8	-	-	-

Compiler

Präprozessor

Header Files

Hello World Revisited

Datei hello.c

```
#include <stdio.h>

int main() {

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

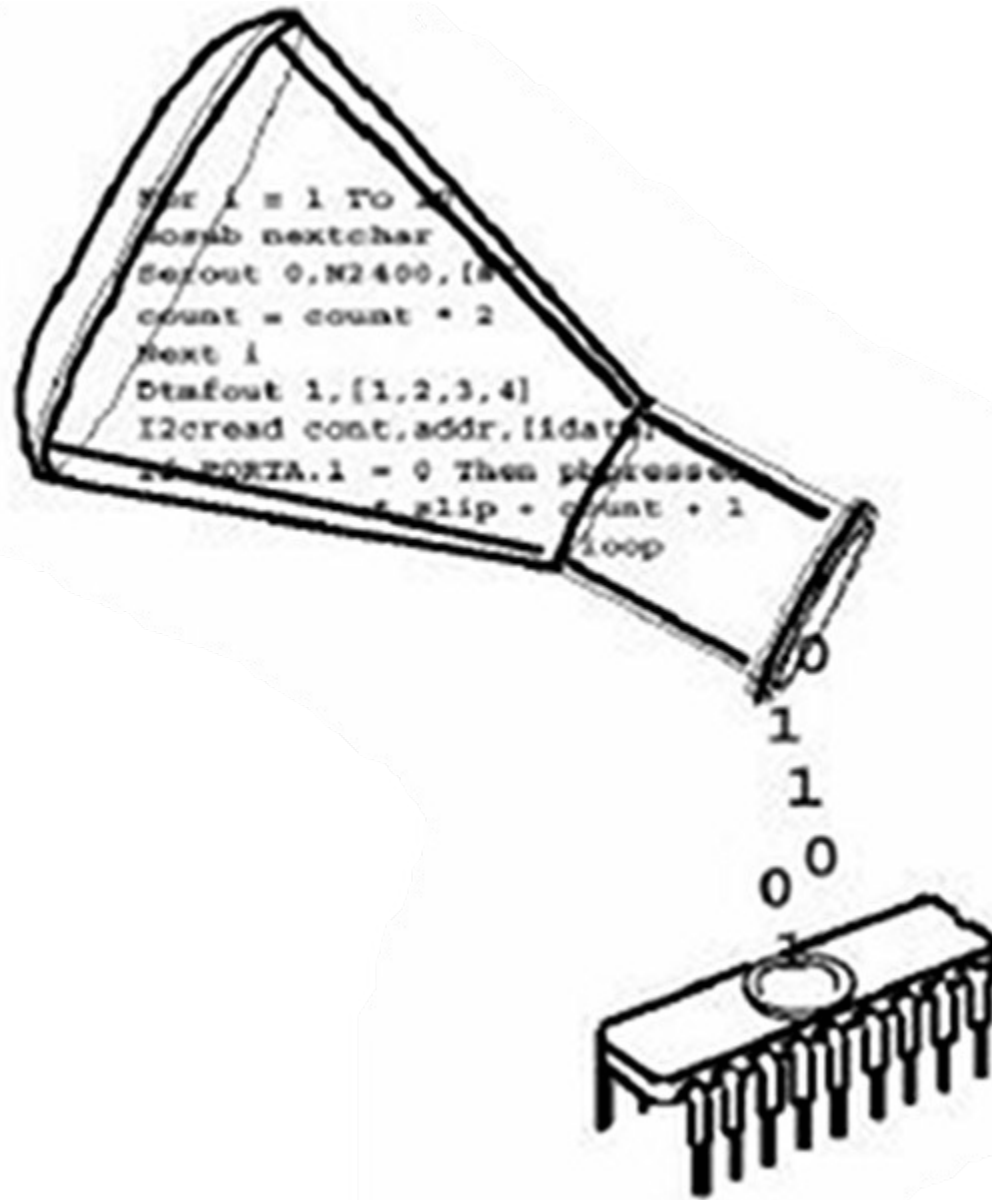
}
```

Kommandozeile

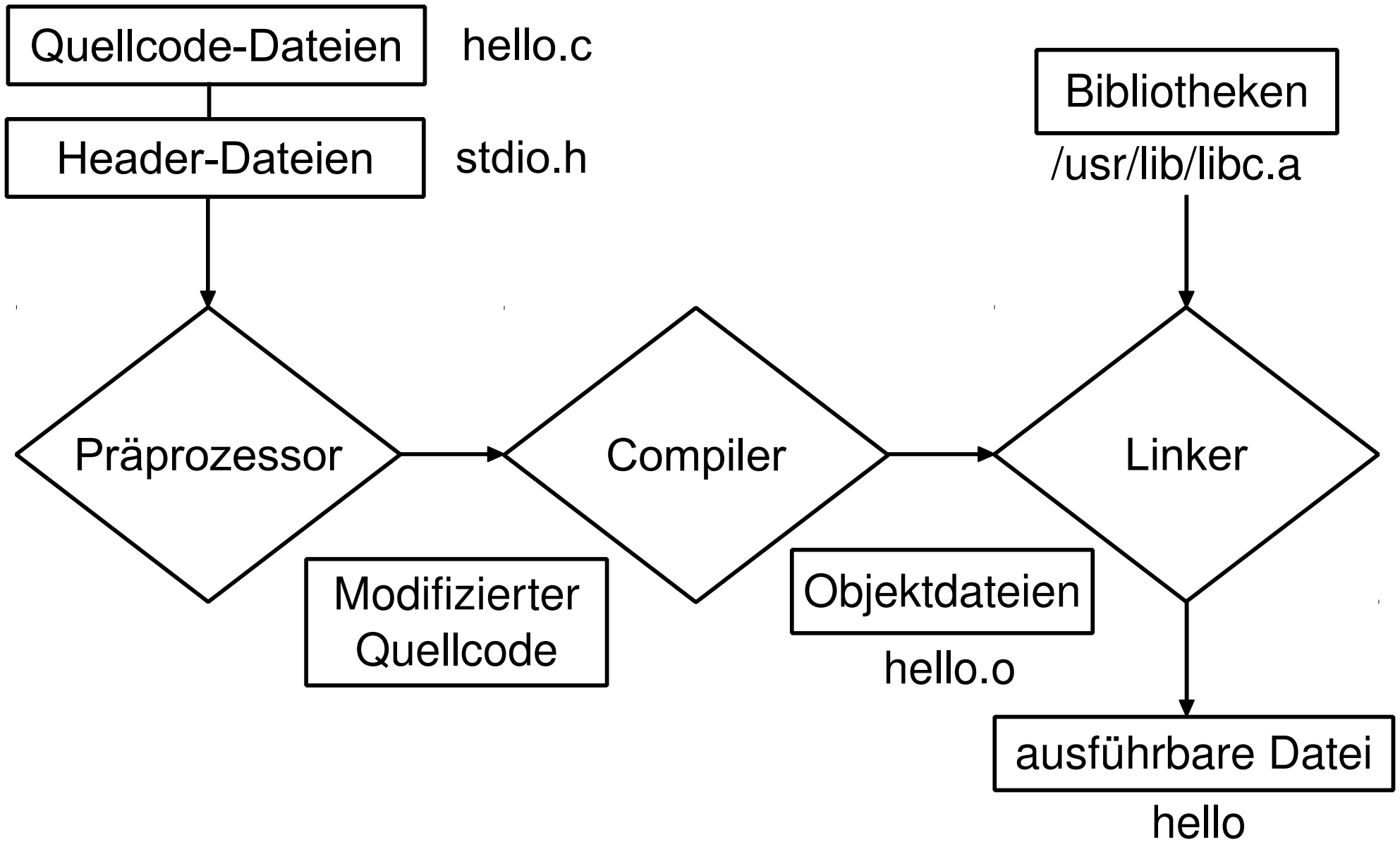
```
$ gcc -o hello hello.c
$ ./hello
Hello, World!
$
```

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
```

```
$
```

```
$ 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
→ gcc -o hello main.o hello.o bye.o

hello.o: hello.c
→ gcc -c hello.c

bye.o: bye.c
→ gcc -c bye.c

main.o: main.c
→ gcc -c main.c

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

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



- 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

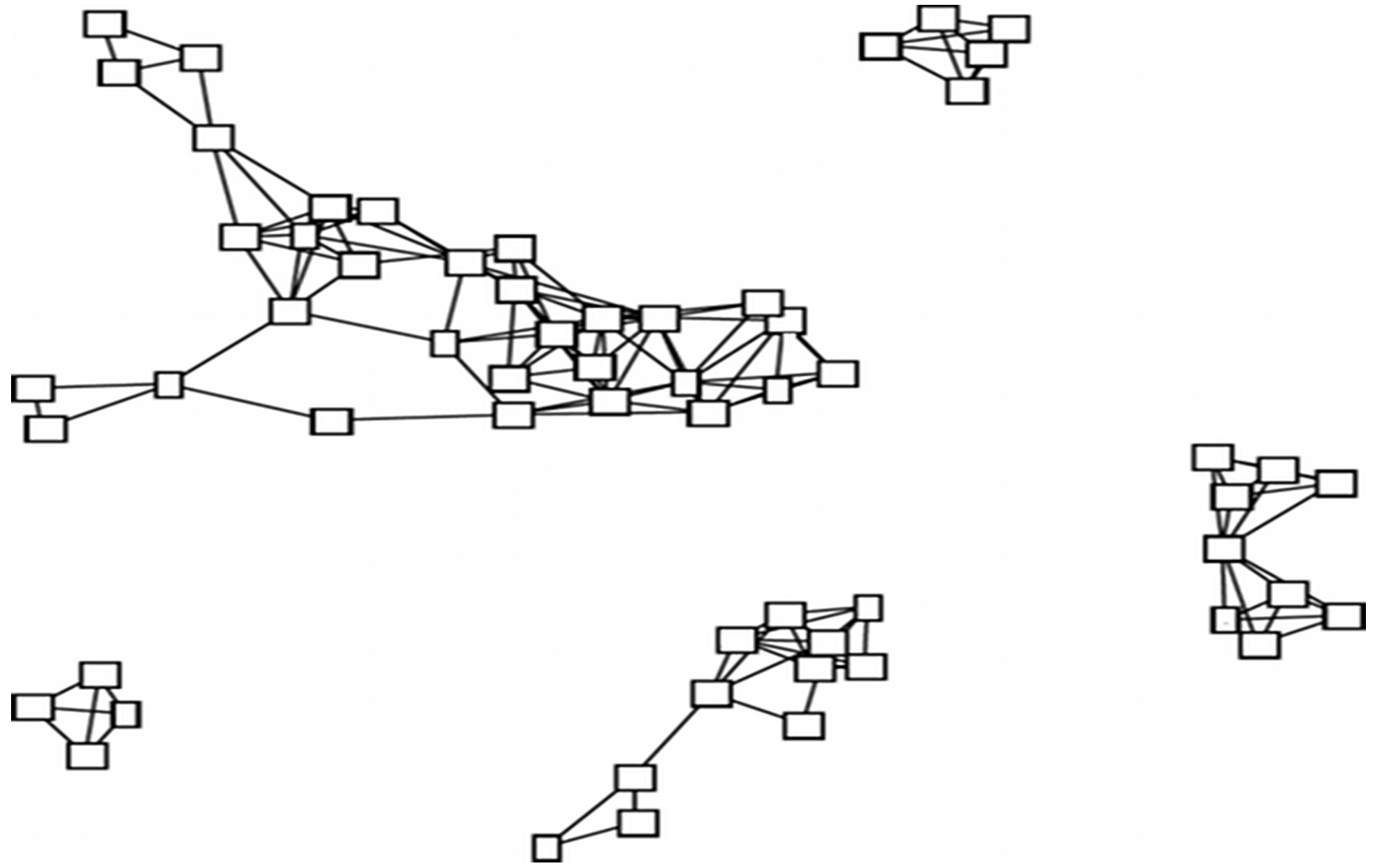
– Ersetzen von Text (Makros):

→ #define

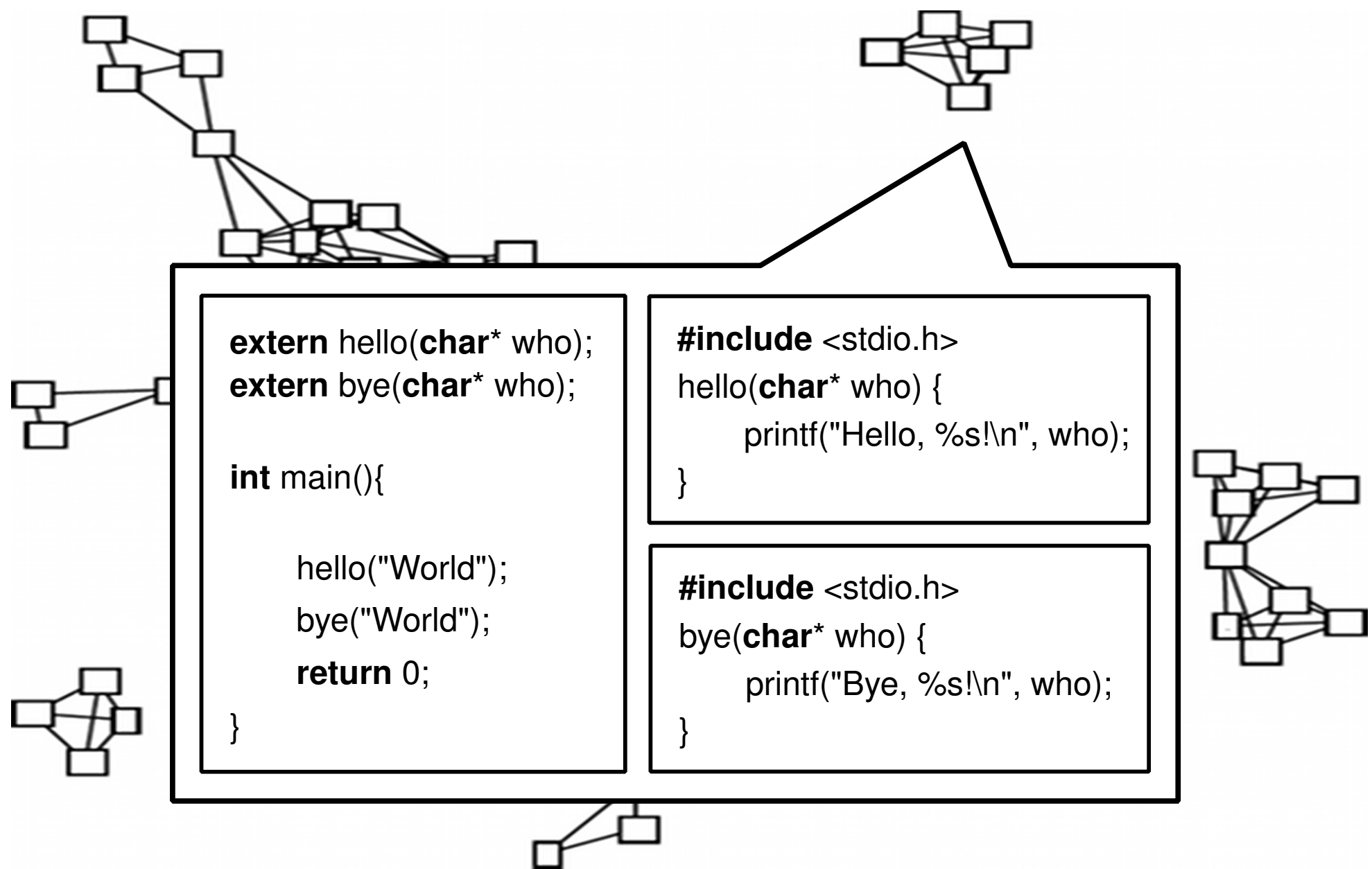
– Bedingte Kompilierung:

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

Header-Dateien



Header-Dateien



- 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 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, ...);
```

```
01 #include <stdio.h>
02 #include <stdio.h>
03 #include <stdio.h>
04 #include <stdio.h>
05 #include <stdio.h>
06 #include <stdio.h>
07 #include <stdio.h>
08 #include <stdio.h>
09 #include <stdio.h>
10 #include <stdio.h>
11 #include <stdio.h>
12 #include <stdio.h>
13 #include <stdio.h>
14 #include <stdio.h>
15 #include <stdio.h>
16 #include <stdio.h>
17 #include <stdio.h>
18 #include <stdio.h>
19 #include <stdio.h>
20 #include <stdio.h>
21 #include <stdio.h>
22 #include <stdio.h>
23 #include <stdio.h>
24 #include <stdio.h>
25 #include <stdio.h>
26 #include <stdio.h>
27 #include <stdio.h>
28 #include <stdio.h>
29 #include <stdio.h>
30 #include <stdio.h>
31 #include <stdio.h>
32 #include <stdio.h>
33 #include <stdio.h>
34 #include <stdio.h>
35 #include <stdio.h>
36 #include <stdio.h>
37 #include <stdio.h>
38 #include <stdio.h>
39 #include <stdio.h>
40 #include <stdio.h>
41 #include <stdio.h>
42 #include <stdio.h>
43 #include <stdio.h>
44 #include <stdio.h>
45 #include <stdio.h>
46 #include <stdio.h>
47 #include <stdio.h>
48 #include <stdio.h>
49 #include <stdio.h>
50 #include <stdio.h>
51 #include <stdio.h>
52 #include <stdio.h>
53 #include <stdio.h>
54 #include <stdio.h>
55 #include <stdio.h>
56 #include <stdio.h>
57 #include <stdio.h>
58 #include <stdio.h>
59 #include <stdio.h>
60 #include <stdio.h>
61 #include <stdio.h>
62 #include <stdio.h>
63 #include <stdio.h>
64 #include <stdio.h>
65 #include <stdio.h>
66 #include <stdio.h>
67 #include <stdio.h>
68 #include <stdio.h>
69 #include <stdio.h>
70 #include <stdio.h>
71 #include <stdio.h>
72 #include <stdio.h>
73 #include <stdio.h>
74 #include <stdio.h>
75 #include <stdio.h>
76 #include <stdio.h>
77 #include <stdio.h>
78 #include <stdio.h>
79 #include <stdio.h>
80 #include <stdio.h>
81 #include <stdio.h>
82 #include <stdio.h>
83 #include <stdio.h>
84 #include <stdio.h>
85 #include <stdio.h>
86 #include <stdio.h>
87 #include <stdio.h>
88 #include <stdio.h>
89 #include <stdio.h>
90 #include <stdio.h>
91 #include <stdio.h>
92 #include <stdio.h>
93 #include <stdio.h>
94 #include <stdio.h>
95 #include <stdio.h>
96 #include <stdio.h>
97 #include <stdio.h>
98 #include <stdio.h>
99 #include <stdio.h>
100 #include <stdio.h>
```

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

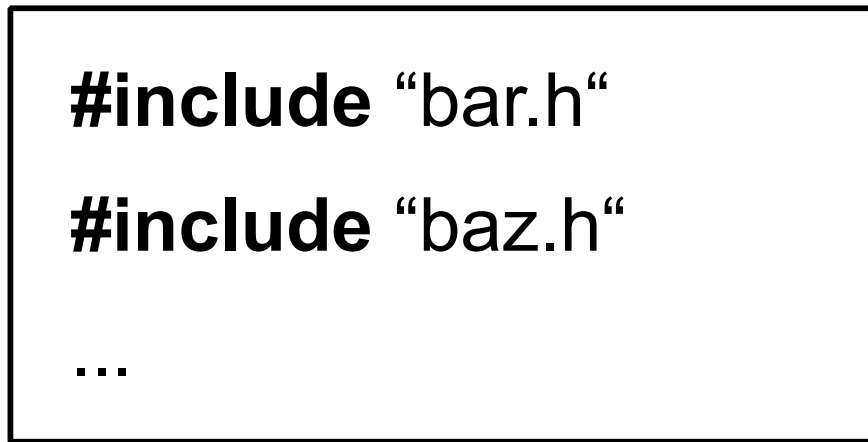
```
#include "baz.h"  
...
```

Datei baz.h

```
#include "bar.h"  
...
```

Problem: Mehrfachinklusion

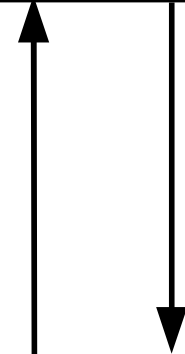
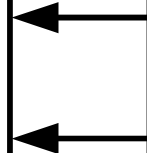
Datei foo.h

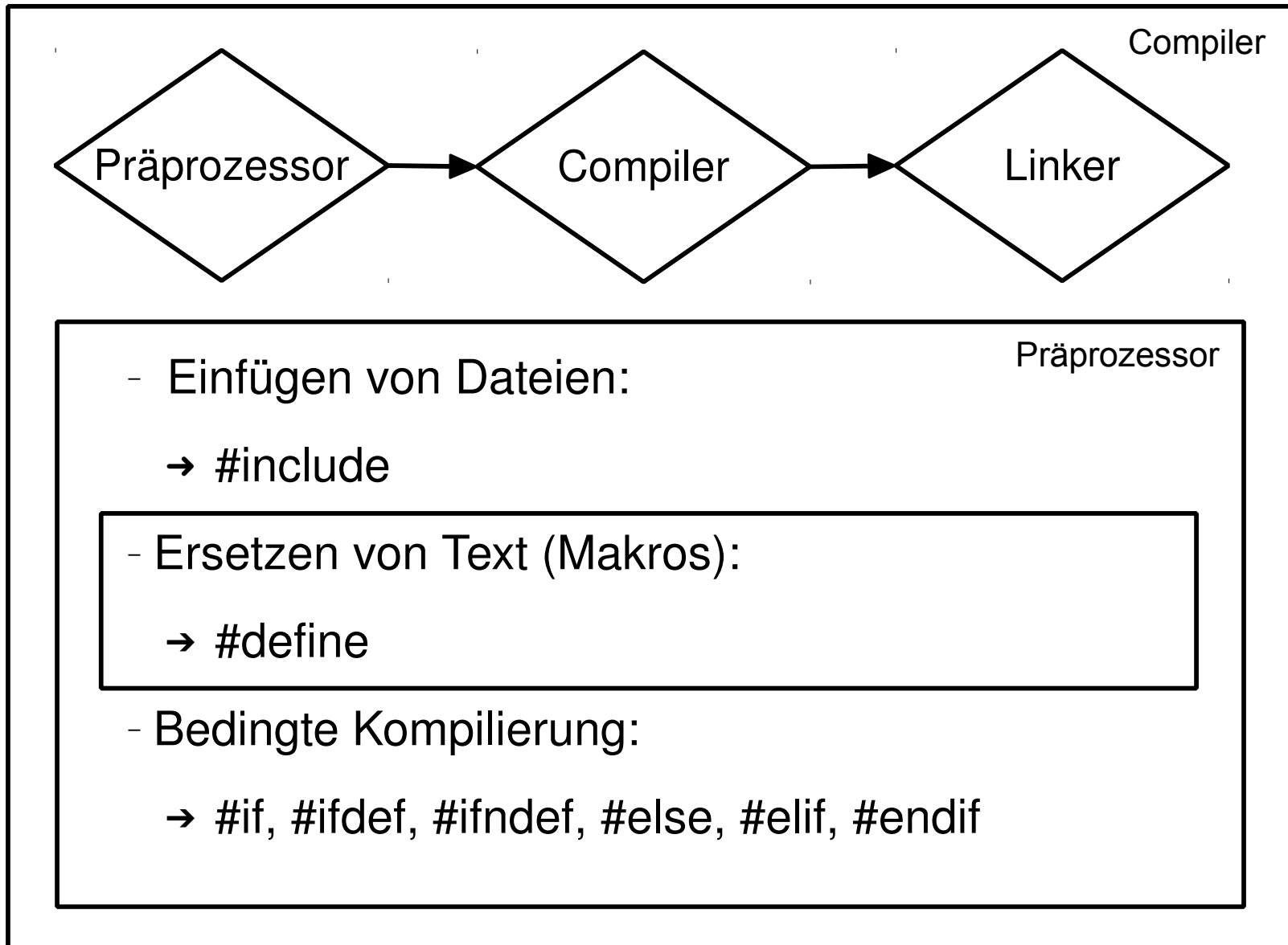


Datei bar.h



Datei baz.h





- 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

Makros mit Parametern

Syntax:

kein Leerzeichen

Leerzeichen

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

Parameterliste

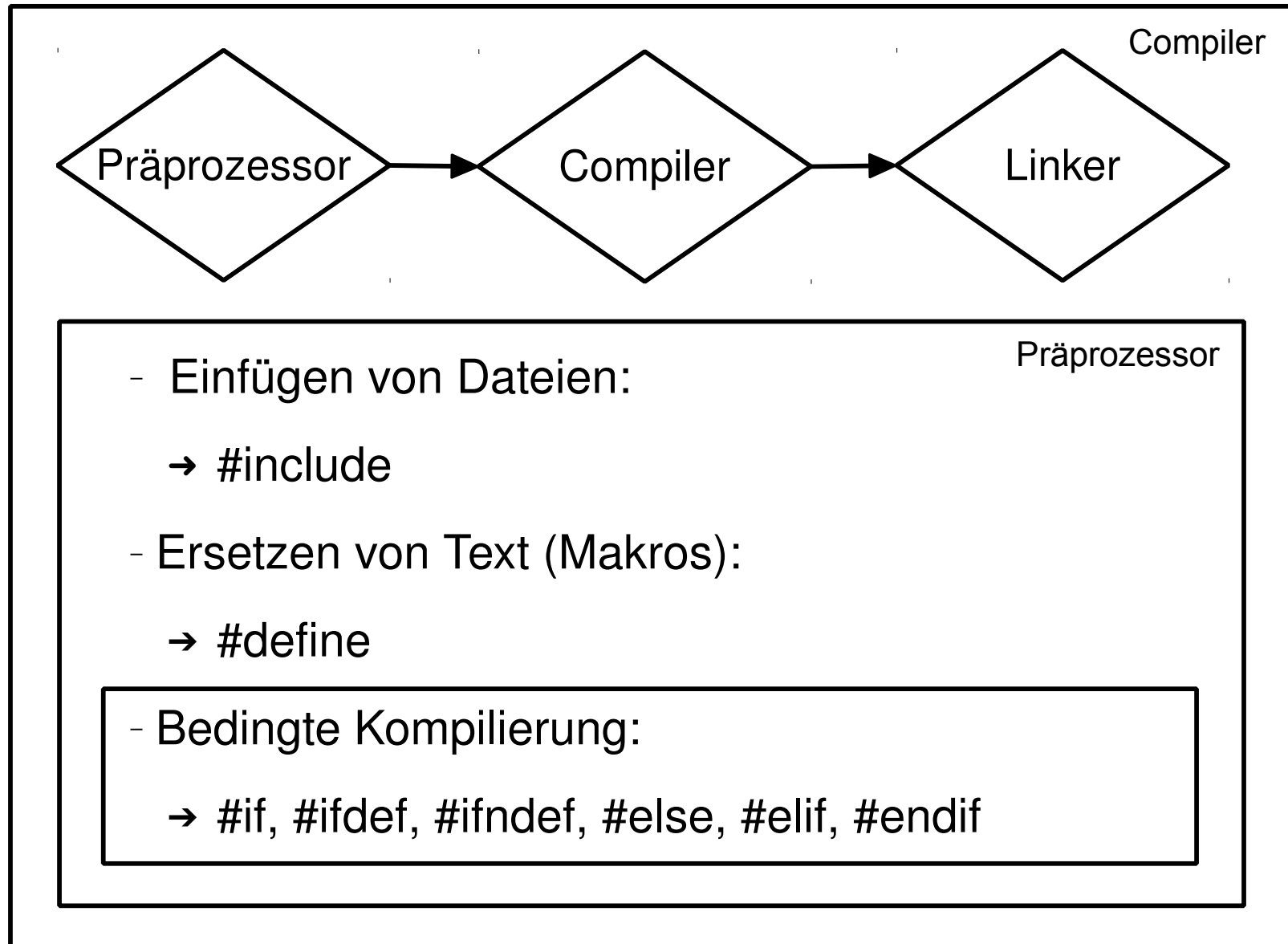
Tokenstring

Beispiel:

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



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
```

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 [-Dmacro[=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;  
}
```

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!

