Unterlagen

http://projects.eml.org/bcb/people/ralph/CPP-Uebungen-08/

http://www.katjawegner.de/lectures.html

Kommentare in C++

```
#include <iostream>

/* Dies ist ein langer Kommentar,
    der über zwei Zeilen geht. */
int main()
{
  // kurzer Kommentar: es folgt eine Ausgabe
  std::cout << “Hello World\n”;
  return 0;
}
```

Einfache Ausgabe

- Eingabe und Ausgabe in C++ werden via streams erledigt
- Die nötigen Deklarationen für die Bildschirmausgabe sind in der Datei **`iostream`**
- Die Deklarationen werden mit der Präprozessordirektive **`#include`** eingebunden
- Ausgabe eines Objektes:

```
std::cout << OBJECT1 [<< OBJECT2 ...];
```

main Routine

- gibt es in zwei Varianten:

```
int main()
{...}
```

oder

```
int main(int argc, char** argv)
{...}
```

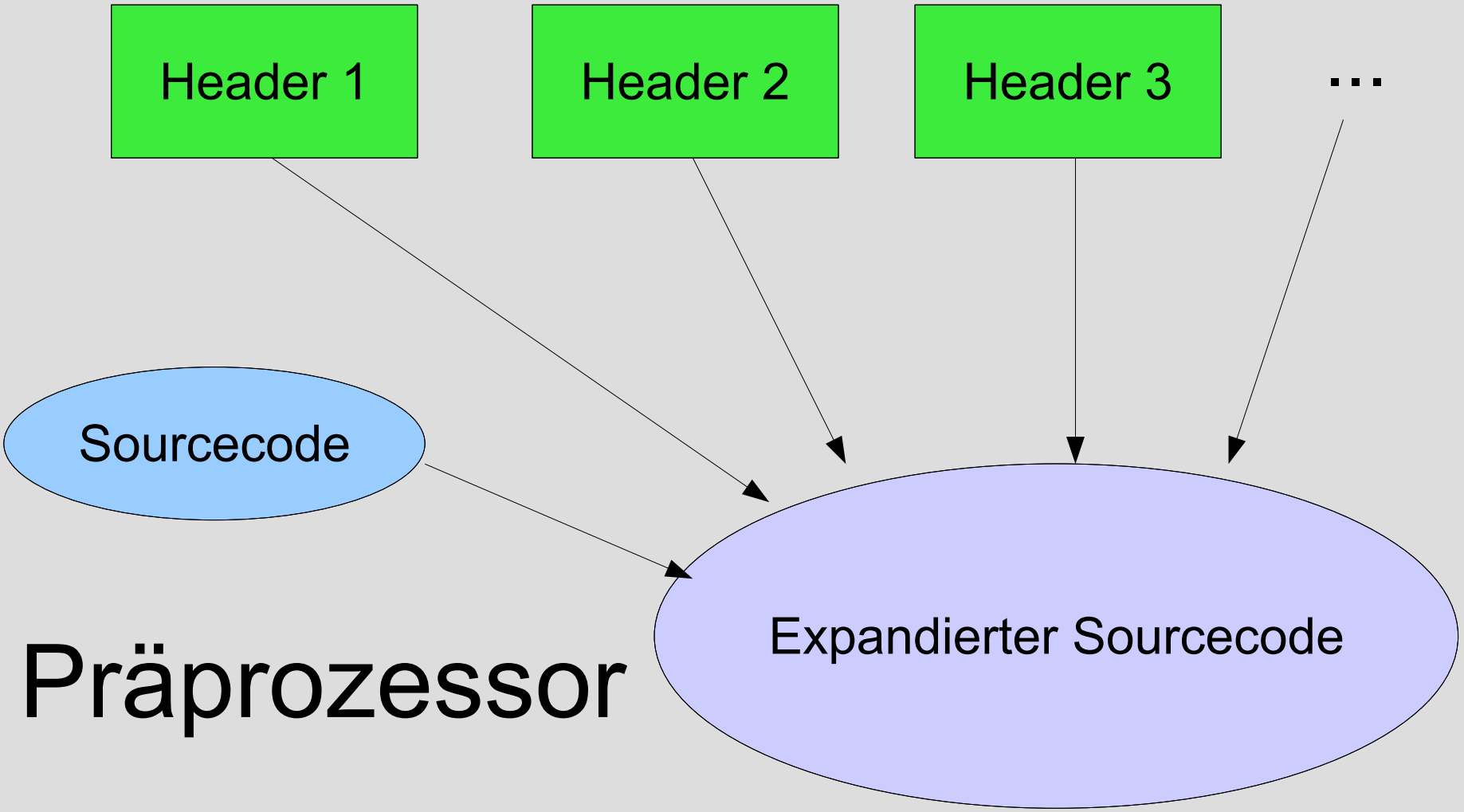
Erstellen einer ausführbaren Datei

g++ HelloWorld.cpp

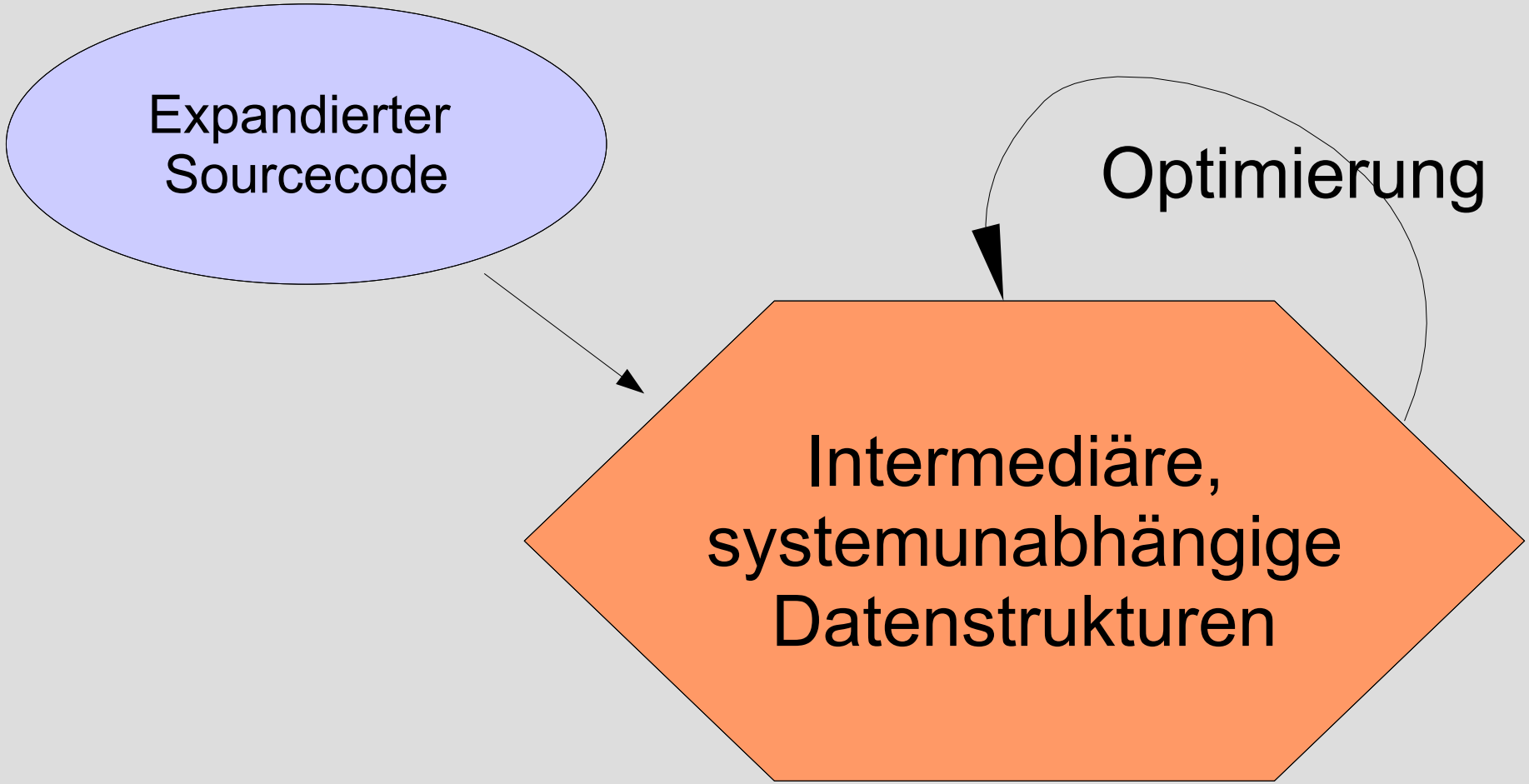
Generiert eine ausführbare Datei namens **a.out** welche mit **./a.out** gestartet wird.

Die Quelldatei muss die Endung .cc oder .cpp haben damit g++ sie als C++ Datei akzeptiert.

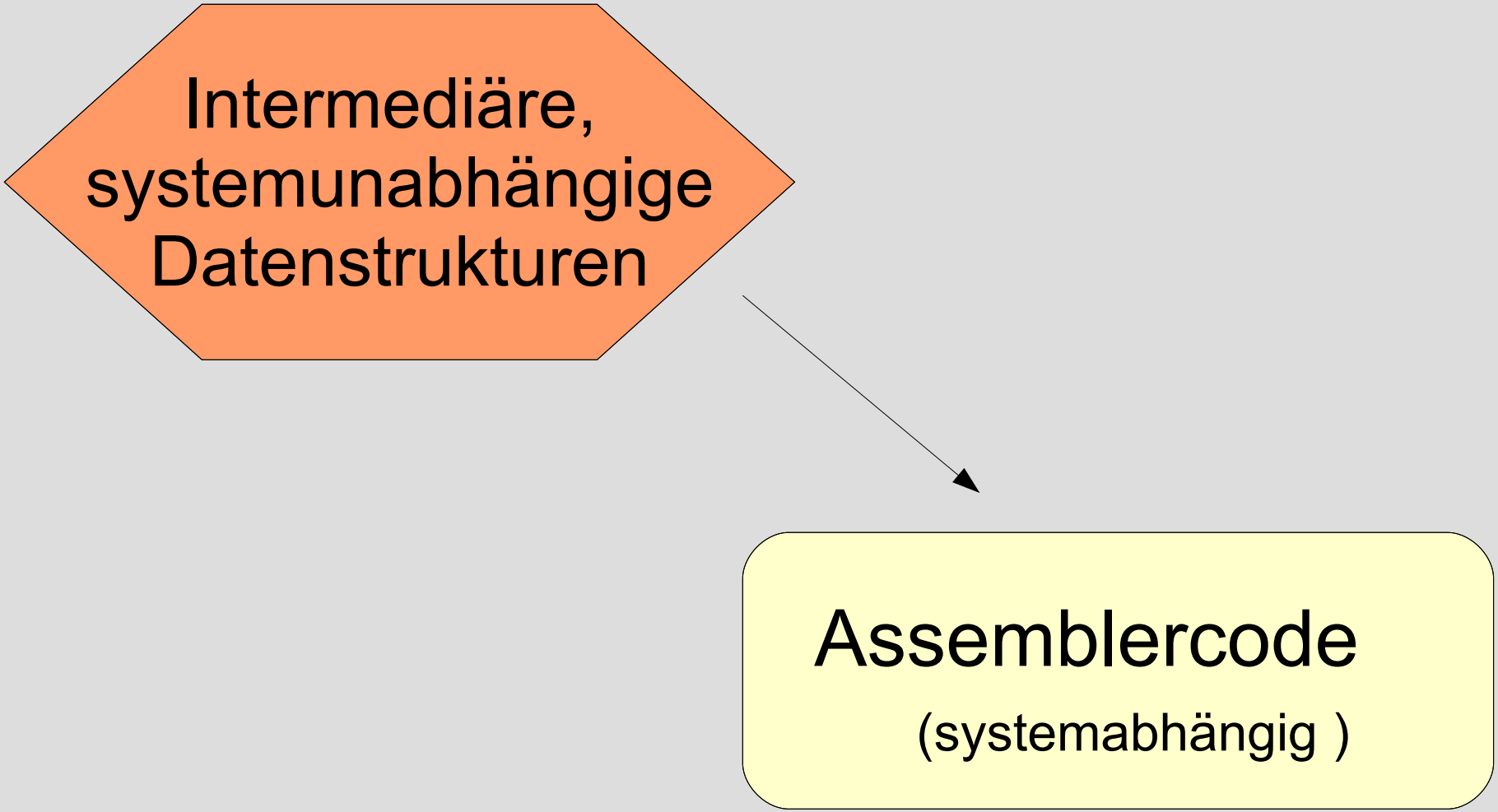
Was macht g++? (1/5)



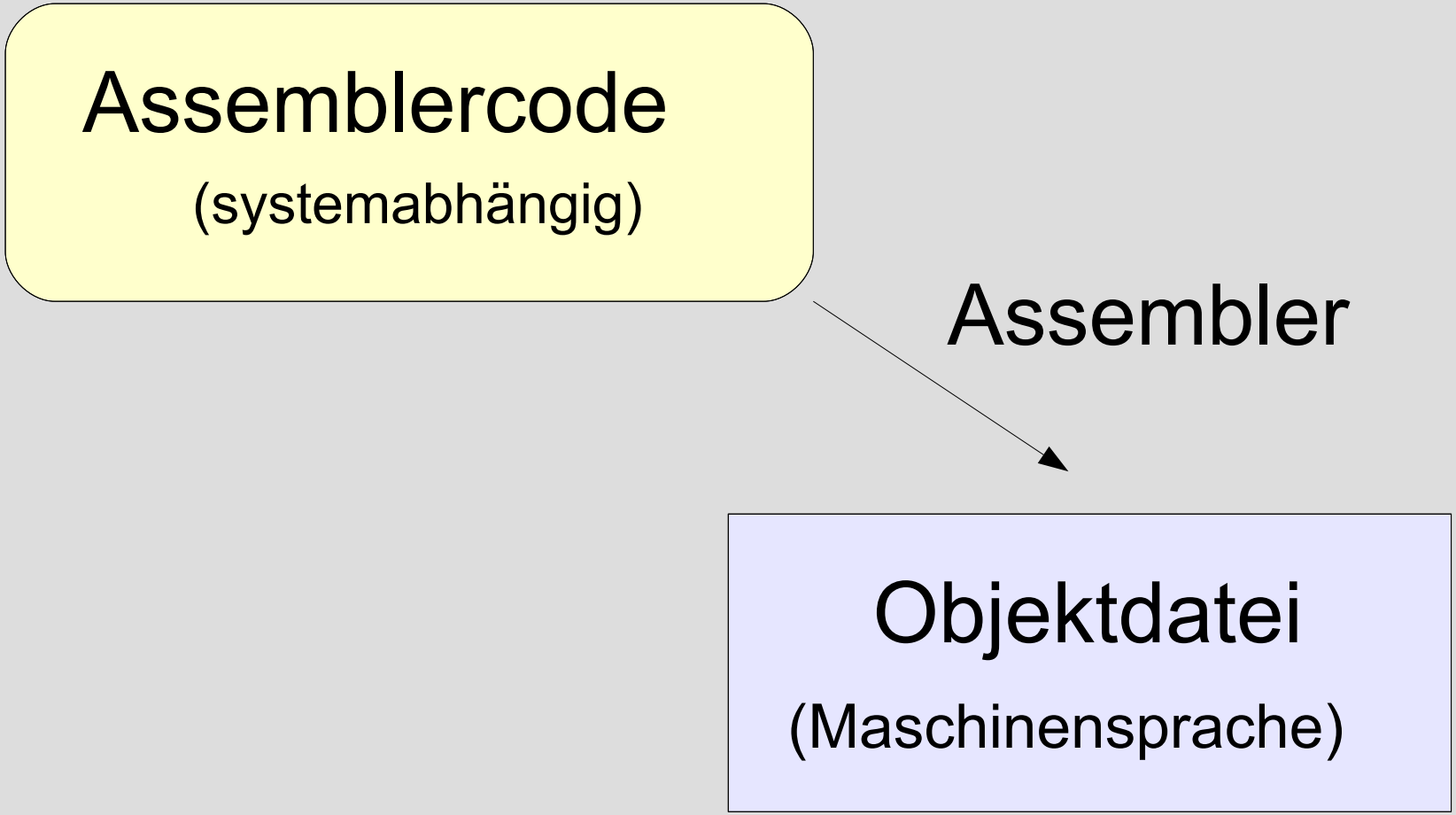
Was macht g++? (2/5)



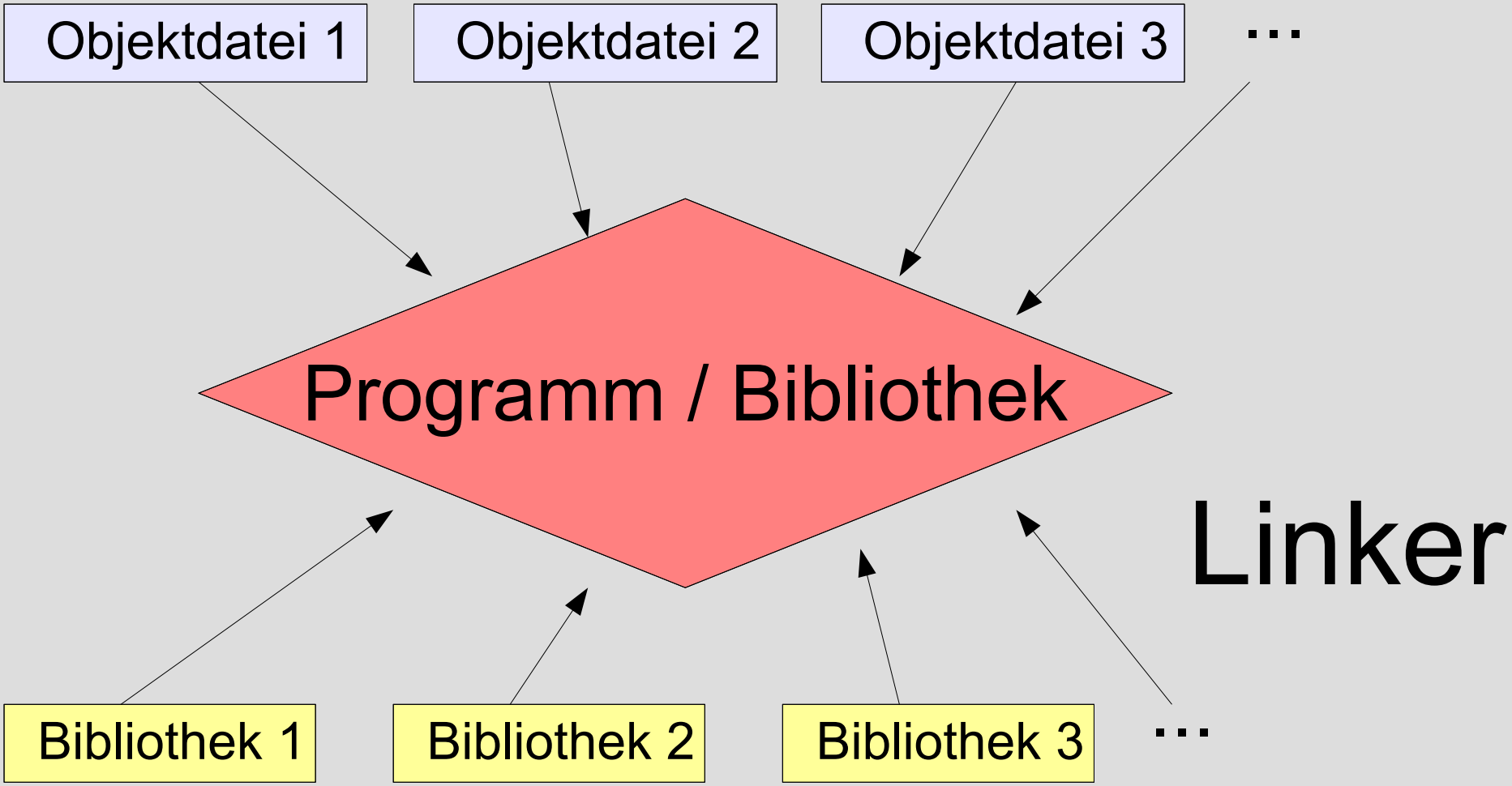
Was macht g++? (3/5)



Was macht g++? (4/5)



Was macht g++? (5/5)



Gängige g++ Optionen

- `-o` ***DESTINATION***
- `-c`
- `-Wall`
- `-O`***level***
- `-g`
- `-I` ***INCLUDEPATH***
- `-L` ***LIBRARYPATH***
- `-l` ***LIBRARY***
- `-D` ***SYMBOLNAME[=VALUE]***

g++ Option (-o)

- -o ***DESTINATION*** weist g++ an das Ergebnis nach ***DESTINATION*** zu speichern
- z.B.

  “g++ -o HelloWorld HelloWorld.cpp”

  würde die Datei **HelloWorld** aus HelloWorld.cpp erstellen

g++ Option (-Wall)

- -Wall weist g++ an möglichst viele Warnungen zu erzeugen.
- g++ versucht so vor möglichen Fehlern zu warnen, selbst wenn es sich um gültigen C++ Code handelt.

  z.B. wenn eine Variable zwar deklariert aber nicht initialisiert wurde

Elementare Datentypen in C++

- bool
- signed/unsigned char
- signed/unsigned short int
- signed/unsigned int
- signed/unsigned long int
- (signed/unsigned long long int)
- float
- double
- long double
- void
- enum

boolean

- bool kann nur die Werte ***`true`*** oder ***`false`*** annehmen
- Zuweisung: z.B. ***`bool success = false;`***

(unsigned) int

- int = signed int
- wenn nicht anders angegeben ist ein **`int`** immer vorzeichenbehaftet
- z.B. ***`unsigned int x = 45671;`***

Variablen (1/2)

- Speicherplätze um Informationen eines bestimmten Typs zu speichern
- Variablen haben einen Typ und einen Namen
- Groß-/Kleinschreibung ist wichtig
- Namen beginnen immer mit einem Buchstaben oder einem Unterstrich (_), der Rest können Buchstaben, Zahlen oder Understrich sein
- Variablenname darf nicht identisch sein zu reservierten Wörtern in C++

Variablen (2/2)

- Variablen können einfache Datentypen (int, float,...), Zeiger, Referenzen oder Objekte enthalten
- Eine Variable muss deklariert werden bevor man sie benutzen kann
- Variablen sollten auf jeden Fall initialisiert werden
- Variablen die als `const` deklariert wurden können nicht geändert werden

Beispiele für Variablennamen

RICHTIG

```
hello    HELLO    Hello  _test
my_variable
DOUBLE     v1       ______
this_is_a_very_long_variable_name
```

FALSCH

```
012      my.variable    $test    double
2bad     class
```

Variablendeklaration

```
bool result;

int x;
signed int y;
unsigned int i;

int a,b,c,d;
```

Variableninitialisierung

```
unsigned int i;
i = 0;

bool result;
result = true;
```

Variablendeklaration & initialisierung

```
unsigned int i = 0;

unsigned int j = i;

bool result = false;

int a=1,b=2,c=3,d=4;
```

const Variablen

```
const unsigned int MAX  =  10;

MAX = 20; // ERROR
```

Operatoren

- Operatoren in C++ sind in vier Klassen eingeteilt:
  - primäre Operatoren
  - unäre Operatoren
  - binäre Operatoren
  - Zuweisungsoperatoren

Binäre Operatoren 1/2

- heissen so weil sie zwei Operanden besitzen
- die beiden Operanden stehen links und rechts des Operators und können beliebige Ausdrücke sein
- mathematische Operationen:
  - Multiplikation: z.B. 3 * 4
  - Division: z.B. 12 / 3
  - Addition: z.B. 3 + 4
  - Subraktion: z.B. 5 – 1
- Falsch: 4 / true

Binäre Operatoren 2/2

- Vergleichsoperatoren:
  - Gleich: z.B. x == 4 // Doppeltes == !!!
  - Ungleich: z.B. x != 4
  - Kleiner als: z.B. x < 4
  - Grösser als: z.B. x > 4
  - Kleiner oder Gleich: z.B. x <= 4
  - Gösser oder Gleich: z.B. x >= 4
- Das Ergebnis eines Vergleichs ist immer ein boolscher Wert (true oder false)

Zuweisungsoperatoren

- weisen einer Variablen einen Wert zu:

```
x = 4;
x += 4;  // äquivalent zu:  x = x + 4;
x -= 4;  //                 x = x - 4;
x *= 4;  //                 x = x * 4;
x /= 4;  //                 x = x / 4;
x %= 4; //                  x = x % 4;
```

Primäre Operatoren

- der einzige primäre Operator den wir an dieser Stelle einführen ist die runde Klammer
- dienen dazu die Ausführungsreihenfolge von Ausdrücken zu ändern
- z.B. 2 * 3 + 5 -> 11
  aber
  2 * (3 + 5) -> 16
- primäre Operatoren haben die höchste Priorität!!!

Unäre Operatoren

- heissen so, weil sie nur einen Operanden besitzen:
  - unäres Minus: -1
  - logisches Not: !true -> false
  - Inkrement: x++ oder ++x
    x++ ist das Gleiche wie x=x+1
  - Dekrement: x-- oder --x
    x-- ist das Gleiche wie x=x-1

Operatorpriorität

- primäre Operatoren > unäre Operatoren > binäre Operatoren > Zuweisungsoperatoren
- die Prioritäten unterscheiden sich manchnal auch innerhalb einer Gruppe
- bei den mathematischen Operationen kommt Punktrechnung (*,/,%) vor Strichrechnung (+,-)
- Um die Auswertungsreihenfolge zu ändern benutzt man den Klammeroperator () (Operator mit höchster Priorität)

Beispiele

- 5 + 3 * 4 -> 17
- (5 + 3) * 4 -> 32
- int x = 4 * 2; -> x = 8;

Postfix/Präfix In- & Decrement

```
#include <iostream>
int main()
{
    int x = 4 * 2;
    int y = ++x;
    y = x++;
    return 0;
}
```

Was sind Kontrollstrukturen?

Normalerweise hat jedes Programm einen Anfang und ein Ende und all Anweisungen dazwischen werden nacheinander abgearbeitet.

Kontrollstrukturen gestatten es dem Programmierer in Abhängigkeit einer Bedingung Verzweigungen in den Programmablauf einzubauen.

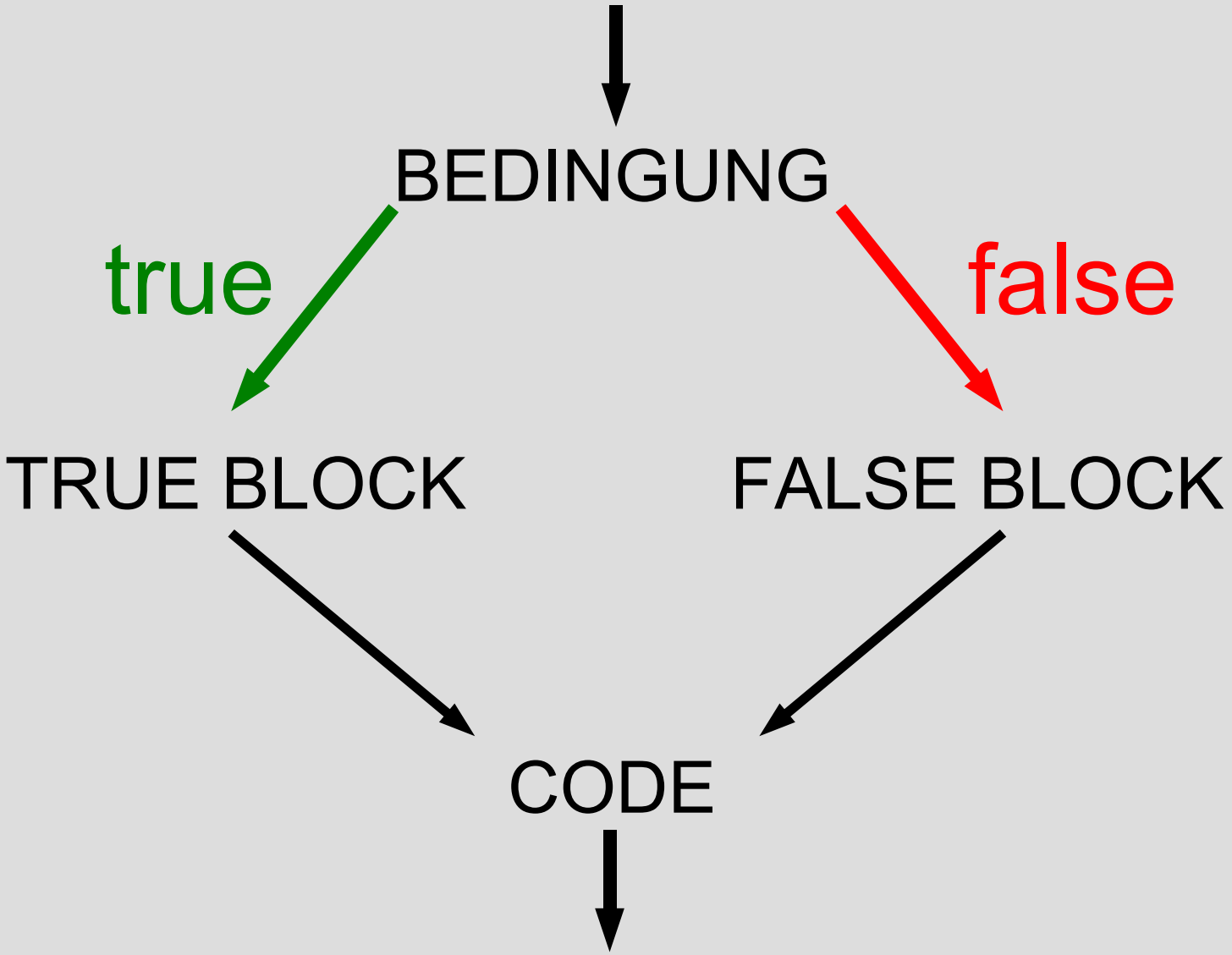
C++ Kontrollstrukturen

- if ... else
- for ...
- while ...
- do ... while
- switch ... case ...

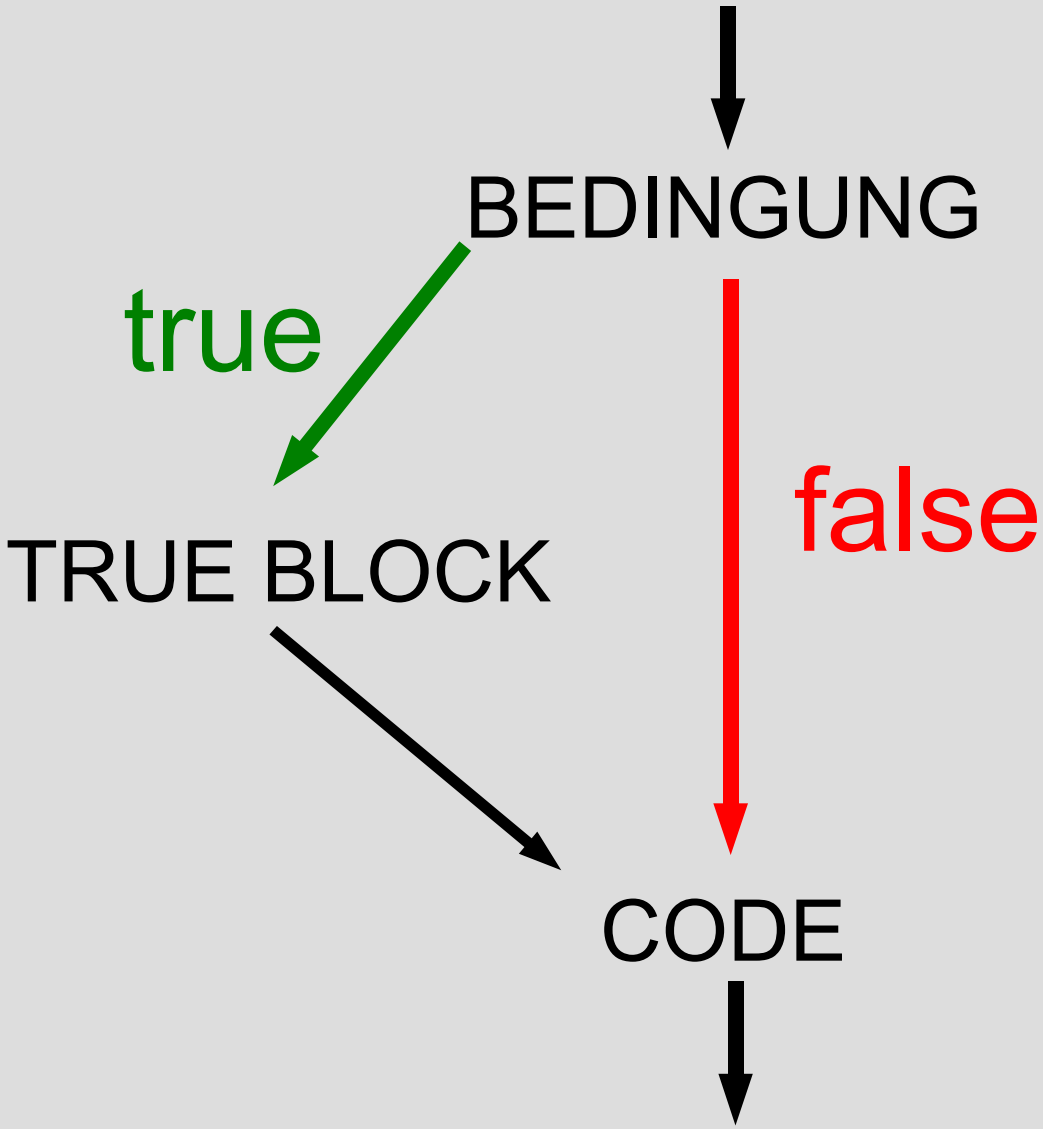
if ... else

- wertet eine Bedingung aus und
- wenn sie zutrifft wird ein Programmblock ausgeführt (`if`), und wenn nicht ein anderer (`else`)
- der Block falls die Bedingung (`else`) nicht erfüllt ist ist optional

if-Flussdiagramm 1



if-Flussdiagramm 2



if Beispiel

```
if( x < 5 )
{
  std::cout << "x is smaller than 5\n";
}
```

if Beispiel

```
if( x < 5 )
{
  std::cout << "x is smaller than 5\n";
}
```

```
if( x < 5 ) std::cout << "x is smaller than 5\n";
```

if ... else Beispiel

```
if( x < 5 )
{
  std::cout << "x is smaller than 5\n";
}
else
{
  std::cout << "x is not smaller than 5\n";
}
```

for Schleifen

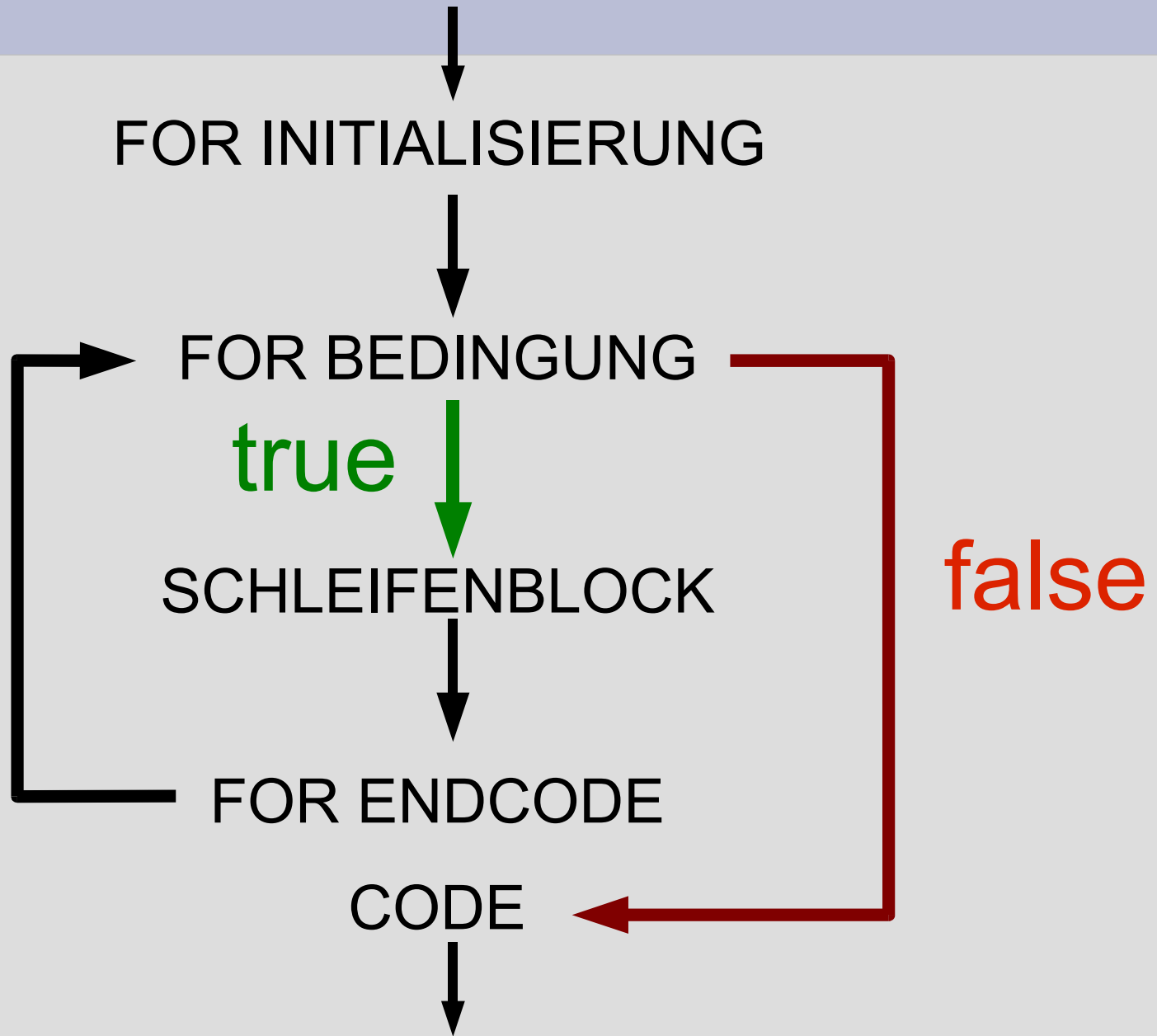
- eine `for` Schleife iteriert über einen Programmblock solange eine bestimmte Bedingung erfüllt ist
- meist ist die Bedingung eine feste Zahl, welche mit einem Zähler verglichen wird, der in jedem Schleifendurchgang erhöht wird

for Beispiel

Initialisierung Bedingung Endcode


```
for(int x=1;    x<=10;      ++x)
{
   std::cout << "x: " << x << "\n";
}
```

for-Flussdiagramm



for Example 2

```
int x=1;
for(  ;   x<=10   ;  ++x)
{
     ...
}
```

for Example 3

```
for(int x=1  ;   x<=10   ;  )
{
     ...
     ++x;
}
```

for Example 4

```
for(int x=1 , y=10  ;   x<=10   ;  ++x , ++y)
{
    ...
}
```

Workaround for MS VC++6

```
int x;
for(x=1  ;   x<=10   ;  ++x)
{
     ...
}
```

Verschachtelte for Schleife

```
for(int x=1; x<=5; ++x)
{
    int result = 1;
    for(int y=1; y<=x; ++y)
    {
        result *= x;
    }
    std::cout << result << "\n";
}
```