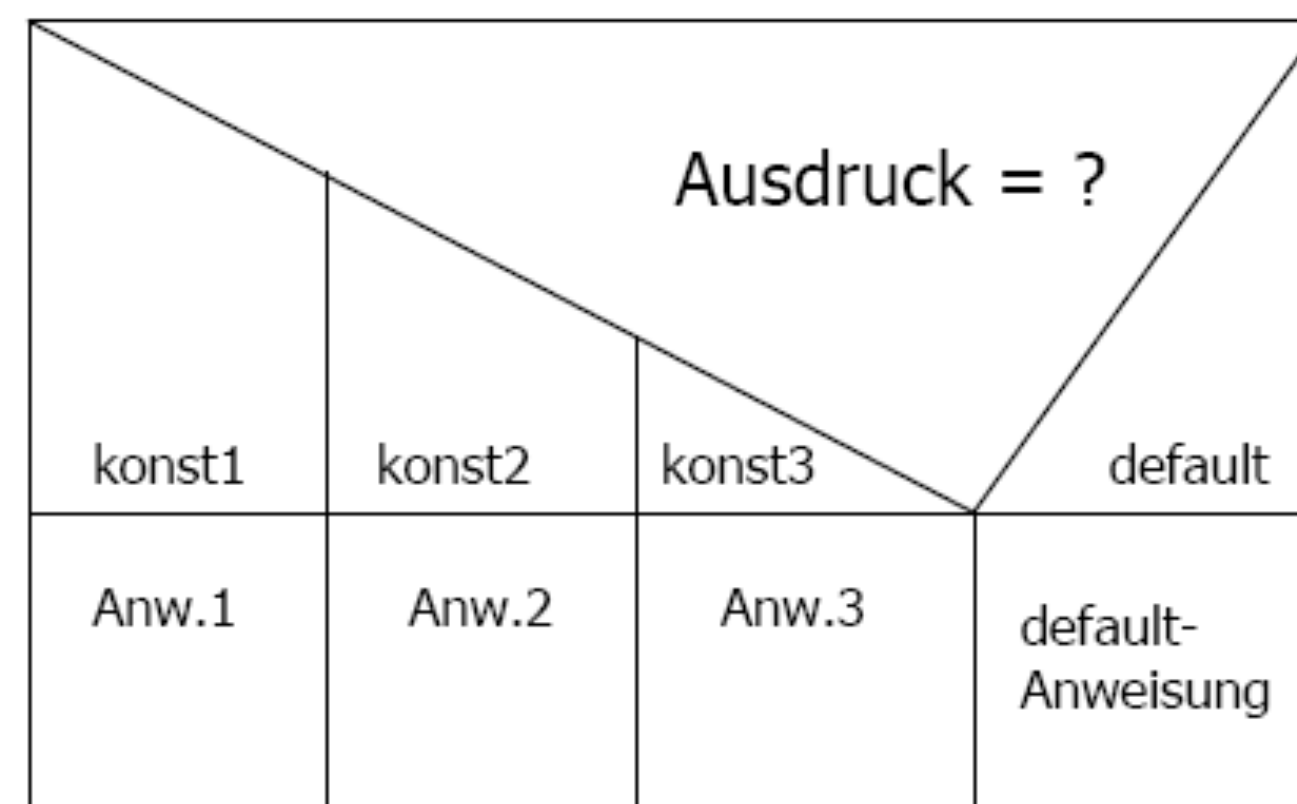
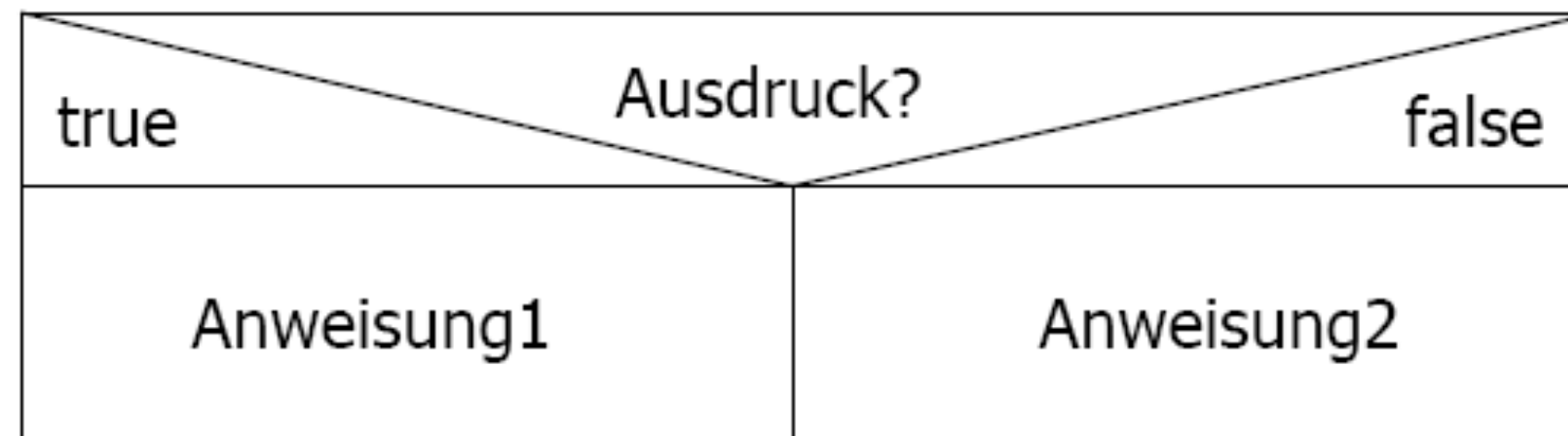


Kontrollfluss

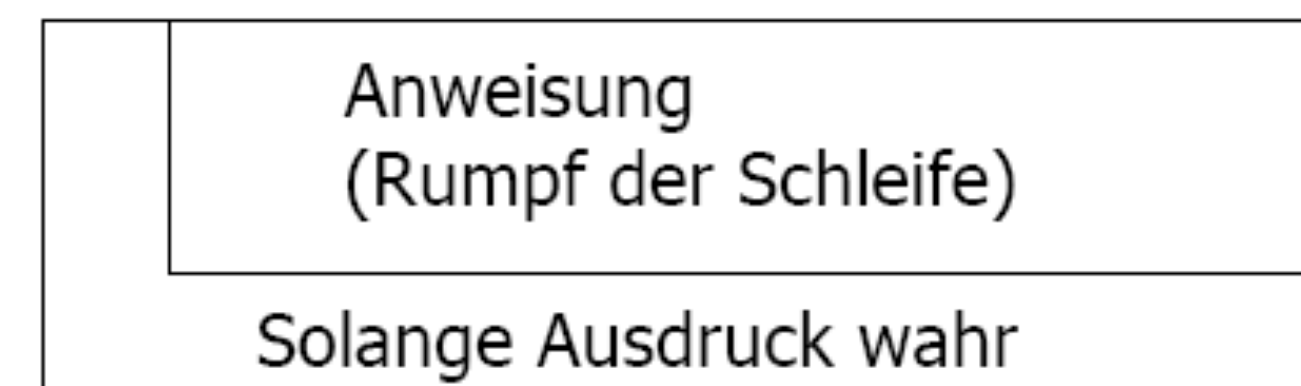
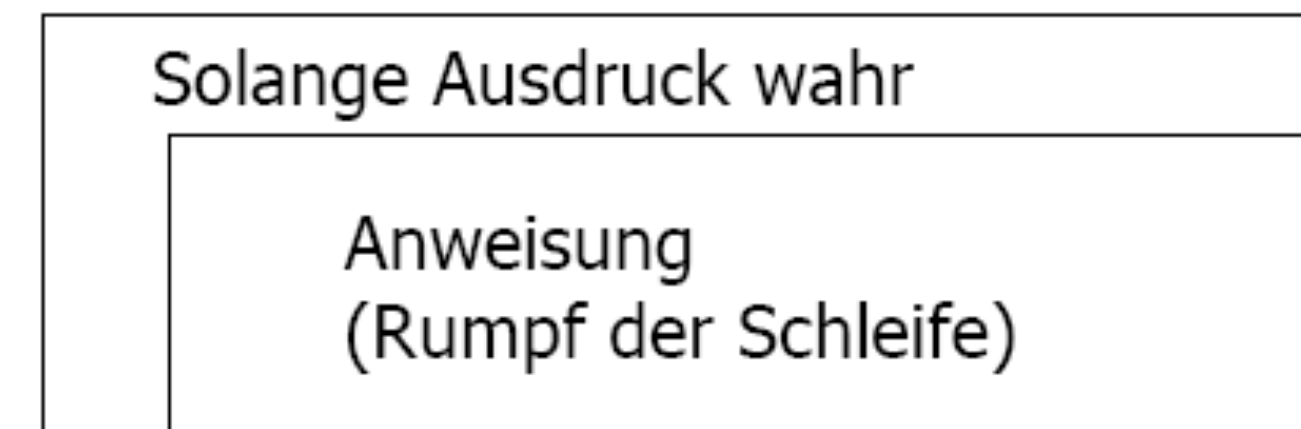
Kontrollfluss

- Es gibt 2 Arten von Kontrollflusssteuerung:

Fallunterscheidungen



Wiederholungen



Struktogramme bzw. Nassi-Shneiderman-Diagramme nach DIN 66261

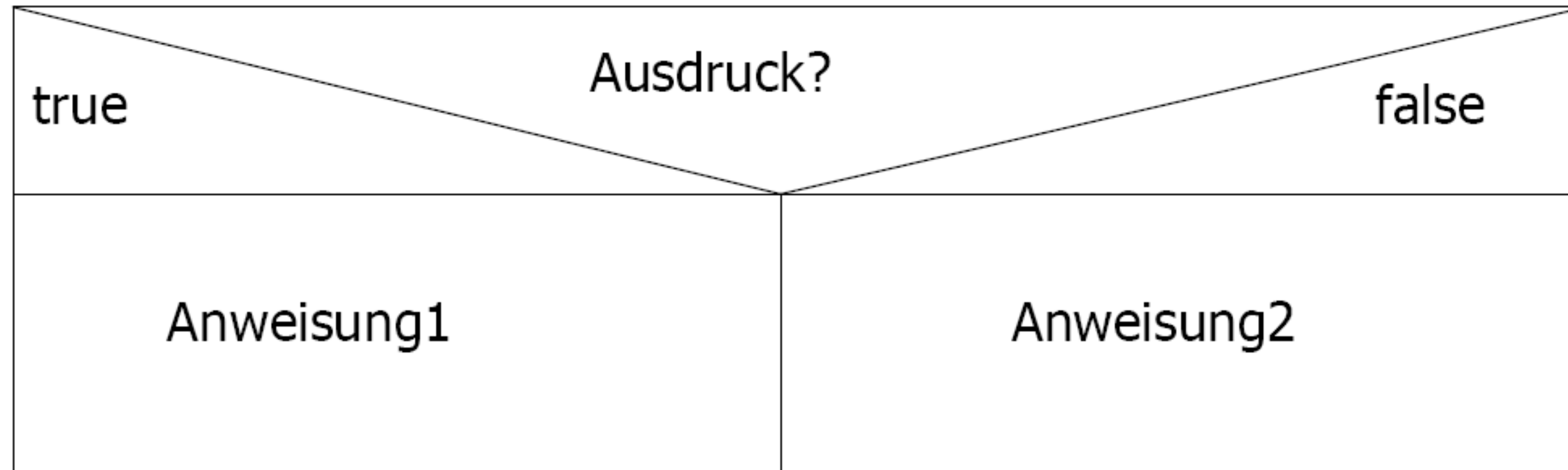
Fallunterscheidungen

- Fallunterscheidung mit zwei Alternativen: Genau eine Anweisung wird ausgeführt!
 - falls Haltbarkeit abgelaufen:
 - wegwerfen
 - sonst
 - verbrauchen
- Fallunterscheidungen mit mehreren Alternativen: Auswahl einer Anweisung
- Tätigkeit:
 - falls Informatiker: Gehalt = x
 - falls Kaminkehrer: Gehalt = y
 - falls Industriekaufmann: Gehalt = z
 - falls Konzernchef: Gehalt = u
 - falls Bäcker: Gehalt = v

Ausdruck?	
true	false
Anweisung1	Anweisung2

Ausdruck = ?			
konst1	konst2	konst3	default
Anw.1	Anw.2	Anw.3	default-Anweisung

Fallunterscheidung mit **if**



if (Ausdruck)

Anweisung1 /* Entweder Anweisung1 (falls Ausdruck wahr) */

else

Anweisung2 /* oder Anweisung2, aber niemals beide */
/* Hier wird die Ausführung nach if fortgesetzt */

Ausdruck: vom Typ **int** und sonst nichts.

Semantik von **if**

- Die Verzweigung mit **if** dient zum Programmieren von Fallunterscheidungen.
- Man kann eine Entweder-oder-Entscheidung treffen oder eine Anweisung nur unter einer bestimmten Bedingung ausführen lassen.
- Auswahl einer Anweisung:
 - entweder Fall 1
 - oder Fall 2, wenn ein **else**-Zweig gegeben ist (optional)
- Oder: Bedingte Ausführung einer Anweisung
 - **if** (Ausdruck)
 - Anweisung /* Nur falls Ausdruck wahr: ausführen */
 - Hier wird die Ausführung **nach if** fortgesetzt
- Bedingung gilt immer dann, wenn der Ausdruck **!= 0** ist!

Vergleiche und Ausdrücke

- Vergleiche liefern ganzzahlige Werte. Sie können daher als Ausdruck in einer **if**-Anweisung verwendet werden

Bedeutung

Falls a kleiner ist als b

Falls a kleiner ist als b oder gleich b

Falls a gleich b ist

Falls a ungleich b ist

Falls a größer ist als b

Falls a größer ist als b oder gleich b

C

if (a < b)

if (a <= b)

if (a == b)

if (a != b)

if (a > b)

if (a >= b)

Logische Verknüpfungen

&&	logisches und
	logisches oder
!	logische Negation

Bedeutung	C
b liegt zwischen a und c	<code>if (a < b && b < c)</code>
b ist kleiner als a oder größer als c	<code>if (b < a b > c)</code>
i ist nicht kleiner als 5	<code>if (!(i < 5))</code>

Logische Verknüpfung

- Beachte die Vorrangregeln!
- Beispiel: **!** Bindet stärker als **<**
 - **!3 < 5** liefert **true**
- Warum?
 - Das Ergebnis eines logischen Ausdrucks ist ein **int** mit Wert **0** im Falle von **false** und **1** im Falle von **true**
 - Logische Operatoren und Fallunterscheidungen erwarten daher ebenfalls **int** Werte als Operanden
- Interpretation:
 - 0 -> **false**, Beispiel: **!0 = 1**
 - Alle Werte $\neq 0$ -> **true**, Beispiele: **!1 = 0**, **!3 = 0**, **(-5 && 3) = 1**
- **!** besser immer klammern
 - **!(3 < 5)** liefert **false** (0)

Fallunterscheidungen von **if**

- Mehrere Fallunterscheidungen von **if** mittels **else if** möglich:

```
1  #include <stdio.h>
2
3  int main() {
4      int a = 0;
5
6      if (a < 0)
7          printf("a < 0");
8      else if (a == 0)
9          printf("a = 0");
10     else
11         printf("a > 0");
12
13     return 0;
14 }
```

Formatierung

- Welche Wert hat a am Ende des Codes? Initialwert: 5

```
6      if (a < 0)
7          printf("a < 0");
8          a = 0;
```

Formatierung

- Welche Wert hat a am Ende des Codes? Initialwert: 5


```
6      if (a < 0)
7          printf("a < 0");
8          a = 0;
```

- a hat den Wert 0, da **if** nur eine Anweisung ausführt
- -> a = 0; // wird immer ausgeführt

Formatierung

- Achtung! Keine leere Anweisung schreiben:

```
6      if (a < 0);  
7      printf("a < 0");
```



- `printf` wird hier immer ausgeführt.
- Richtig ist:

```
6      if (a < 0)  
7      printf("a < 0");
```

Formatierung

- Mehrere Anweisungen können mit einem Block ausgeführt werden:

```
6      if (a < 0)
7      {
8          printf("a < 0");
9          a = 0;
10     }
```

- Auch bei einer Anweisung möglich:

```
6      if (a < 0)
7      {
8          printf("a < 0");
9      }
```

Formatierung

- Schreibweise mit Blöcken:

```
6      if (a < 0)
7      {
8          printf("a < 0");
9      }
10     else if (a == 0)
11     {
12         printf("a = 0");
13     }
14     else
15     {
16         printf("a > 0");
17     }
```

```
6      if (a < 0)
7      {
8          printf("a < 0");
9          a = 0;
10     }
11     else if (a == 0)
12     {
13         printf("a = 0");
14     }
15     else
16     {
17         printf("a > 0");
18     }
```

Formatierung

- Schöner und kürzer:

```
6      if (a < 0) {  
7          printf("a < 0");  
8          a = 0;  
9      }  
10     else if (a == 0)  
11         printf("a = 0");  
12     else  
13         printf("a > 0");
```

```
6      if (a < 0)  
7      {  
8          printf("a < 0");  
9          a = 0;  
10     }  
11     else if (a == 0)  
12         printf("a = 0");  
13     else  
14         printf("a > 0");
```

- Den Block nur schreiben, wenn er nötig ist

Formatierung

- Wird hier “a = 0” auch ausgegeben?

```
1  #include <stdio.h>
2
3  int main() {
4      int a = 0;
5
6      if (a <= 0)
7          printf("a < 0");
8      else if (a == 0)
9          printf("a = 0");
10     else
11         printf("a > 0");
12 }
```

Formatierung

- Wird hier “a = 0” auch ausgegeben?

```
1  #include <stdio.h>
2
3  int main() {
4      int a = 0;
5
6      if (a <= 0)
7          printf("a < 0");
8      else if (a == 0)
9          printf("a = 0");
10     else
11         printf("a > 0");
12 }
```

- **Nein**, `if` stoppt nach der ersten Bedingung die Wahr ist!

Auswahl mit **switch**

Ausdruck = ?			
konst1	konst2	konst3	default
Anw.1	Anw.2	Anw.3	default- Anweisung

switch (Ausdruck)

```
{
  case konst1 : Anw1; break;
  case konst2 : Anw1; break;
  ...
  default: default-Anweisung;
}
```

Die sind hier Pflicht!

break ist wichtig!
Es bewirkt Verlassen des **switch**-Blocks

default wird immer dann ausgewählt,
Wenn kein **case** ausgewählt wird

Auswahl mit **switch**

- Ausdruck muss ganzzahlig sein (**short**, **long**, **int**, **char**)
- Beispiele:
- **int** i;
- **switch** (i) { ...
- **switch** (i%42) { ...
- Andere Datentypen sind nicht erlaubt!

```
switch (Ausdruck)
{
    case konst1 : Anw1; break;
    case konst2 : Anw1; break;
    ...
    default: default-Anweisung;
}
```

Auswahl mit **switch**

- konst1 bis konstN müssen **ganzzahlige Konstanten** (keine Variablen!) sein, die kompatibel zum Typ von Ausdruck sind

- Beispiele:

- **int** i;

- **switch** (i) {

- **case** 42 : ...

- **case** 0xFF : ...

char c;

switch (c) {

case 'a' : ...

switch (Ausdruck)

{

case konst1 : Anw1; **break**;

case konst2 : Anw1; **break**;

...

default: default-Anweisung;

}

Auswahl mit **switch**

- **default** sollte immer als letzter Fall stehen. Wenn nicht, braucht auch **default** ein **break**!
- Konstanten müssen nicht sortiert sein

switch (Ausdruck)

```
{  
    case konst1 : Anw1; break;  
    case konst2 : Anw1; break;  
    ...  
    default: default-Anweisung;  
}
```

Beispiel mit `switch`

- Für `case` ist es nicht erforderlich einen Block `{ }` zu definieren.
- Das `break;` ist allerdings wichtig und darf nicht vergessen werden.
- Was gibt der rechte Code bei den jeweiligen Eingaben aus?

```
#include <stdio.h>

int main() {
    int i;
    printf("Eingabe: ");
    scanf("%d", &i);
    switch (i) {
        case 2:
            printf("2");
            break;
        case 5:
            printf("5");
        case 3:
            printf("3");
        default:
            printf("D");
        case 6:
            printf("6");
            break;
        case 0:
            printf("0");
    }
    printf("\n");
    return 0;
}
```


Beispiel mit `switch`

- Was gibt der rechte Code bei den jeweiligen Eingaben aus?

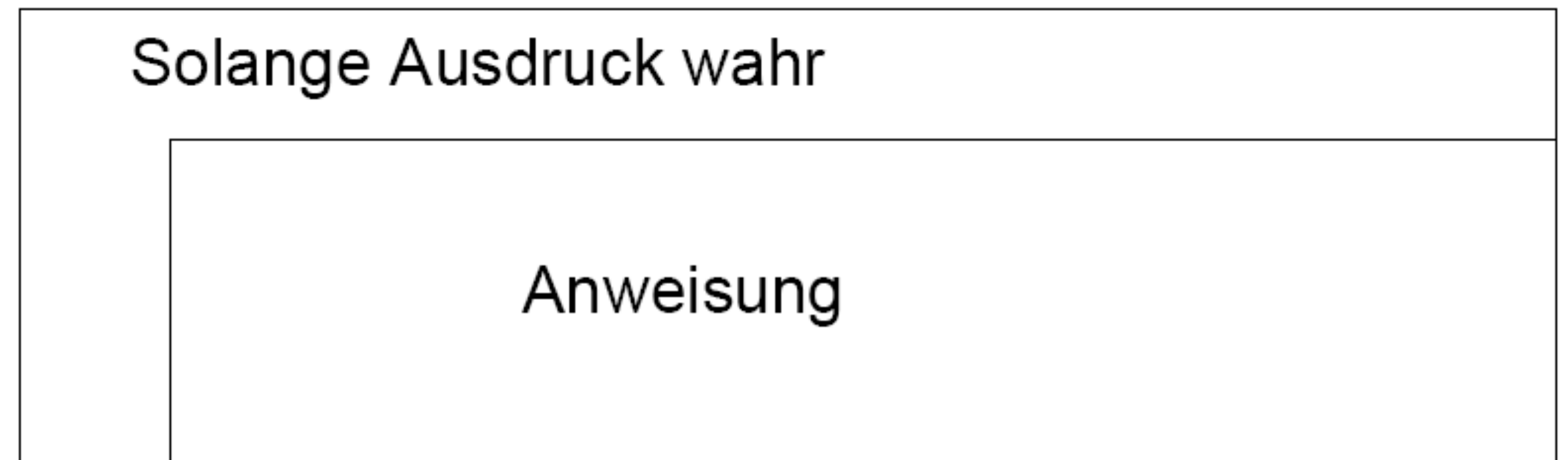
Eingabe (i)	Ausgabe
0	0
1	D6
2	2
3	3D6
4	D6
5	53D6
6	6
7, 8, ...	D6

```
#include <stdio.h>

int main() {
    int i;
    printf("Eingabe: ");
    scanf("%d", &i);
    switch (i) {
        case 2:
            printf("2");
            break;
        case 5:
            printf("5");
        case 3:
            printf("3");
        default:
            printf("D");
        case 6:
            printf("6");
            break;
        case 0:
            printf("0");
    }
    printf("\n");
    return 0;
}
```

Wiederholung von Anweisungen

- Prüfung des Ausdrucks am Anfang (Schleifenkopf)
- Wenn Ausdruck true ($\neq 0$), dann Rumpf ausführen
- Zurück zur Prüfung des Ausdrucks
- In C:
 - **while**-Schleifen
 - **for**-Schleifen



while-Schleife

- Die allgemeine Verwendung ist
 - `while` (Ausdruck)
 - Anweisung;
- Schleifen können auch geschachtelt werden:

```
4      int i = 0;
5
6      while (i < 5)
7      {
8          printf("%i\n", i);
9          i++;
10     }
```

```
4      int i = 0;
5      int x = 0;
6
7      while (i < 5)
8      {
9          x = 0;
10         while (x < 3){
11             printf("%i%i\n", i, x);
12             x = x + 1;
13         }
14         i++;
15     }
```

Die **for**-Schleife

- Die allgemeine Verwendung ist:

for (Initialisierer; Bedingung; Ausdruck)

Anweisung;

- Beispiel:

```
int i;
```

```
for ( i=0; i<5; i++)
```

```
    printf("%I" ,i);
```

- Ergibt die Ausgabe:

0 1 2 3 4

for(Initialisierer; Bedingung; Ausdruck)

Anweisung;

ist eine abkürzende Schreibweise für:

Initialisierer;

while(Bedingung)

{

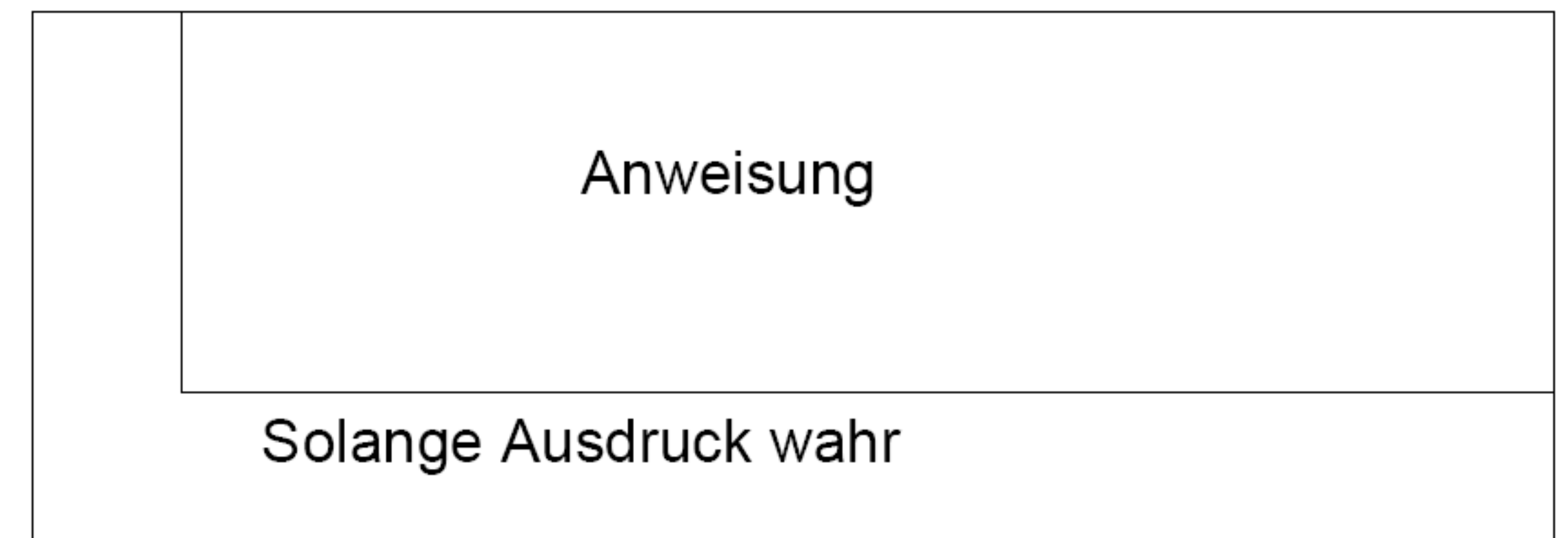
Anweisung;

Ausdruck;

}

Wiederholung von Anweisungen

- Führe zunächst den Rumpf aus
- Dann Prüfung des Ausdrucks am Ende der Schleife
- Wenn Ausdruck true ($\neq 0$), dann Rumpf erneut ausführen
- Weiter zur erneuten Prüfung des Ausdrucks
- In C:
 - **do-while**-Schleifen



do-while-Schleife

- Schleifen mit der Überprüfung am Ende werden in C durch die do-while-Schleife geschrieben:

do

{

Anweisung;

} while (Ausdruck);


```
4      int i = 0;
5
6      do
7      {
8          printf("%i\n", i);
9          i++;
10     } while (i < 5);
11
12     return 0;
```

- ; am Ende!**

Verlassen von Schleifen: **break**

- Mit **break** können Schleifen vorzeitig verlassen werden
- Dies gilt für alle Schleifen (**for**, **while** und **do-while**)
- Es wird immer die Schleife verlassen, in der die **break**-Anweisung steht
- Mit **break** werden auch switch-Anweisungen verlassen
- Mögliche Anwendungen sind Suchen in großen Datenmengen

```
for (f=0; f<NrOfItems; f=f+1)
{
    if (f-tes_Item == Was_wir_suchen)
        break;
}
Anweisung;
```



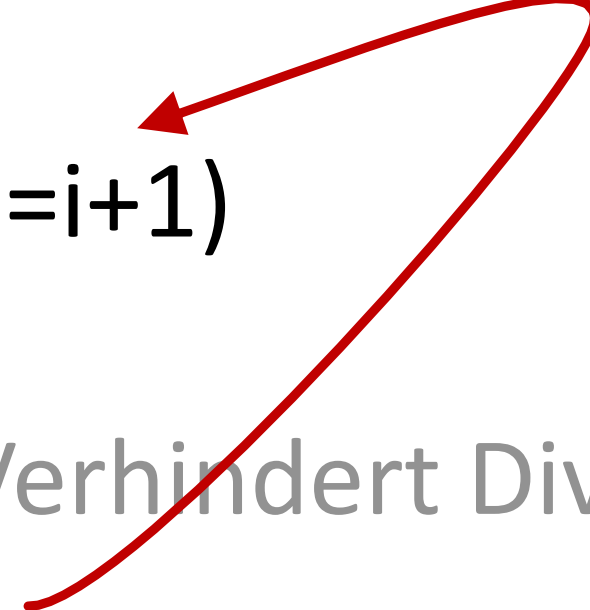
Überspringen von Anweisungsteilen

- Fortsetzen von Schleifen ohne Abarbeitung verbleibender Anweisungen des Rumpfes kann durch `continue` erreicht werden
- `continue` kann immer auch durch entsprechende Abfragesequenzen im Rumpf ersetzt werden, ist aber bequemer zu schreiben
- Bei `while`- und `do-while`-Schleifen springt `continue` grundsätzlich den Ausdruck an
- Bei `for`-Schleifen der Form `for(I;B;A)` sorgt `continue` dafür, dass zunächst A ausgeführt und dann B getestet wird

Beispiel – continue

- Wird oft benutzt, um bestimmte weitergehende Analyse von Daten im Rumpf zu verhindern (Effizienzgründe), Beispiel Kehrwertberechnung:

```
int i;  
for(i=-10;i<10;i=i+1)  
{  
    if (i==0) /* Verhindert Division durch 0 */  
        continue;  
    printf("Kehrwert von %i ist %f \n",i,1.0/i);  
}
```



Algorithmus Beispiel

```
1  #include <stdio.h>
2
3  int main() {
4      int ausgangsbbox = 0;
5      int eingangsbbox = 3;
6      int merkbbox = 0;
7
8      scanf("%d",&eingangsbbox);
9
10     merkbbox = eingangsbbox;
11
12     while(eingangsbbox > 0)
13     {
14         eingangsbbox--;
15         ausgangsbbox += merkbbox;
16     }
17
18     printf("Ausgangsbbox: %d\n", ausgangsbbox);
19
20     return 0;
21 }
```