HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

# Kapitel 4

# Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

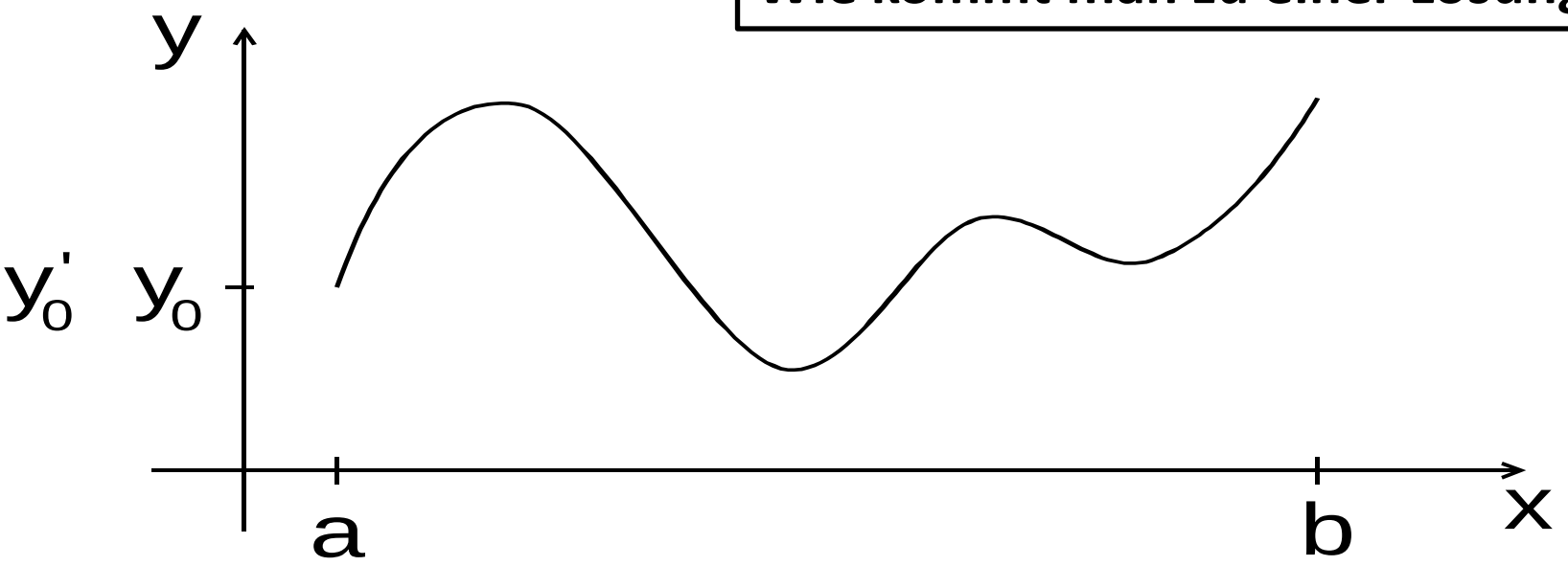
**Problem**:

- DGL und Anfangswerte gegeben

$$y'' + \omega^2 \cdot y = x^2 \cdot y + \lambda \cdot y^2$$

- Die Lösung der DGL soll grafisch dargestellt werden.

**Wie kommt man zu einer Lösung?**



HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

**Lösungsmethode: Strukturierte Programmierung**

- Schrittweise Verfeinerung des Problems
- Nur 3 einfache Kontrollstrukturen

Bei der **schrittweisen Verfeinerung** wird eine Aufgabe solange in kleinere/einfachere Teilaufgaben zerlegt, bis diese so einfach sind, dass eine Codierung erfolgen kann.

Bei der grafischen Darstellung dieser Methode durch Struktogramme wird jede Teilaufgabe durch einen sog. Strukturblock dargestellt. Ein Strukturblock besitzt folgende Eigenschaften:

- Er erfüllt eine klar definierte Aufgabe
- Jeder Strukturblock besitzt einen Eingang (von oben) und einen Ausgang (nach unten)

# 4.1. Strukturierte Programmierung



Die Reihenfolge, in der die einzelnen Teilaufgaben bearbeitet werden, wird durch Kontrollstrukturen festgelegt.

Bei der strukturierten Programmierung werden nur drei einfache Kontrollstrukturen zugelassen:

- **Folge** (Sequenz)
- **Alternative** (Verzweigung, Auswahl oder Selektion)
- **Wiederholung** (Schleife, Iteration)

Hat man ein Problem schrittweise verfeinert und sich auf die drei genannten Kontrollstrukturen beschränkt, dann ist die Codierung einfach und die entstehenden Programme sind erfahrungsgemäß gut lesbar, wartungsfreundlich, leicht zu testen und wenig fehleranfällig.

Zur grafischen Darstellung der Methode der strukturierten Programmierung werden sogenannte **Nassi-Shneidermann-Diagramme** oder **Struktogramme** (DIN 66261) verwendet.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

**Problem:**

Schreibe ein Programm, das zwei ganze Zahlen einliest. Das Programm soll die Summe der eingegebenen Zahlen berechnen. Die Zahlen sollen am Bildschirm ausgegeben werden, ebenso der Wert der Summe.

## Verwendete Variablen:

z1 : erste eingelesene Zahl
z2 : zweite eingelesene Zahl
sum : Summe der eingelesenen Zahlen

# 4.2. Folge - Sequenz

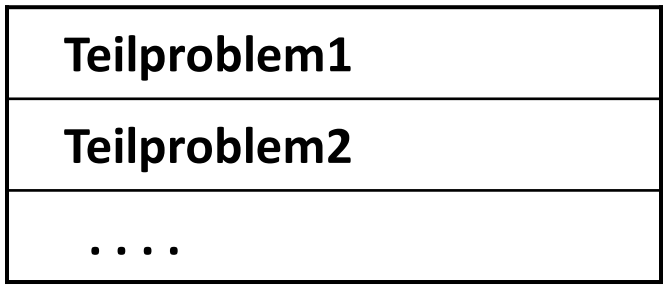
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>
int main(void)
{
  int z1, z2, sum;
  printf("Erste Zahl eingeben:\n");
  scanf("%d", &z1);
  printf("Zweite Zahl eingeben:\n");
  scanf("%d", &z2);
  /* berechne die Summe */
  sum = z1 + z2;
  printf("Zahl 1: %d Zahl2: %d\n", z1, z2);
  printf("Summe: %d \n", sum);
  return 0;
}
```

# 4.2. Folge - Sequenz

Die Lösung eines Teilproblems (z. B. eine einzelne Anweisung) wird im Struktogramm durch ein Rechteck grafisch dargestellt:


**Folge, Sequenz:**




**Bedeutung:**

- Die einzelnen **Strukturblöcke** (Teilprobleme) werden nacheinander abgearbeitet
- Im C-Programm: Anweisungen werden nacheinander geschrieben

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

**Ein Struktogramm kann in beliebige Sprachen übersetzt werden!**

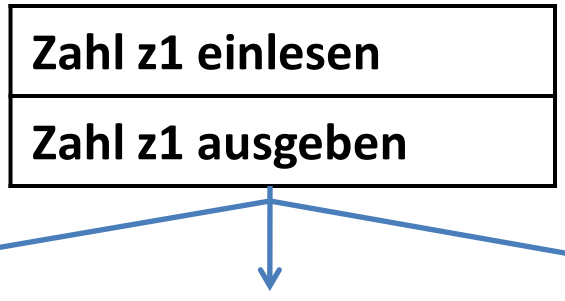
**Problem:**

Es soll eine Zahl eingelesen und der Wert zur Kontrolle wieder ausgegeben werden.

**Verwendete Variable:**

z1: eingelesene Zahl

**Struktogramm:**




| UNIX-SHELL | C | PASCAL |
|---|---|---|
| read z1 | scanf("%d", &z1); | readln(z1); |
| echo "Zahl = " $z1 | printf("Zahl=%d",z1); | writeln('Zahl=', z1); |

## Bei komplexeren Problemen:

- Ein Strukturblock entspricht vielen Anweisungen
- Strukturblock wiederum in Teilprobleme zerlegen
- Hierarchie von Struktogrammen

**Beispiel:**

10 Messwerte einlesen und sortiert wieder ausgeben

**Übersichts-Struktogramm (Grobstruktur):**

| |
|---|
| 10 Zahlen einlesen – TP1 |
| Zahlen der Größe nach sortieren – TP2 |
| 10 Zahlen ausgeben – TP3 |

**Verfeinerung (Feinstruktur):**

Struktogramme für einzelne Teilprobleme

**Struktogramm für TP1:**

Einlesealgorithmus


**Struktogramm für TP2:**

Sortieralgorithmus

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

**Problem:**

Schreibe ein Programm, das zwei Gleitkommazahlen einliest. Das Programm soll die Summe der eingegebenen Zahlen berechnen. Die Zahlen sollen am Bildschirm ausgegeben werden, ebenso der Wert der Summe aber nur, wenn dieser positiv oder null ist. Es soll eine Meldung ausgegeben werden, ob das Ergebnis positiv oder negativ ist. Am Ende soll eine Meldung erscheinen, dass das Programm beendet wurde.

**Variablen:**

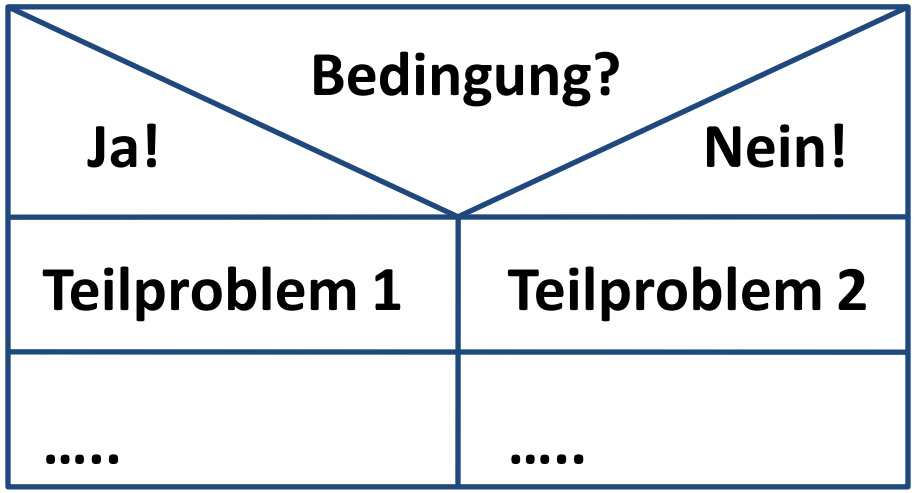
z1, z2 und sum - Typ float

# 4.3. Verzweigung - Alternative



```c
#include <stdio.h>
int main(void)
{
    float z1, z2, sum;
    printf("Erste Zahl eingeben:\n");
    scanf("%f", &z1);
    printf("Zweite Zahl eingeben:\n");
    scanf("%f", &z2);
    sum = z1 + z2;
    printf("Zahl1: %f Zahl2: %f\n", z1, z2);
    if(sum < 0.0)
    {
        printf("Ergebnis ist negativ!");
    }
    else
    {
        printf("Summe: %f\n", sum);
        printf("Ergebnis ist positiv oder null!");
    }
    printf("Ende des Programms");
    return 0;
}
```

# Kontrollstruktur: einfache Verzweigung




## Bedeutung:

- Ist die Bedingung erfüllt, wird Teilproblem1 bearbeitet, andernfalls Teilproblem2
- Genau eines der beiden Teilprobleme wird bearbeitet

## Umsetzung einer einfachen Verzweigung in C:

```
if ( Bedingung )
    Anweisungsblock_1
else
    Anweisungsblock_2
```

## Bedingung

- Die Bedingung muss so formuliert sein, dass sie entweder erfüllt („wahr“) oder nicht erfüllt („falsch“) ist
- Beispiele für Bedingungen sind: (x > 1), (2*x < 10), (x == y), (x != y)
- Achtung: (x = 5) ist keine Bedingung, sondern eine Zuweisung!

## Anweisungsblock:

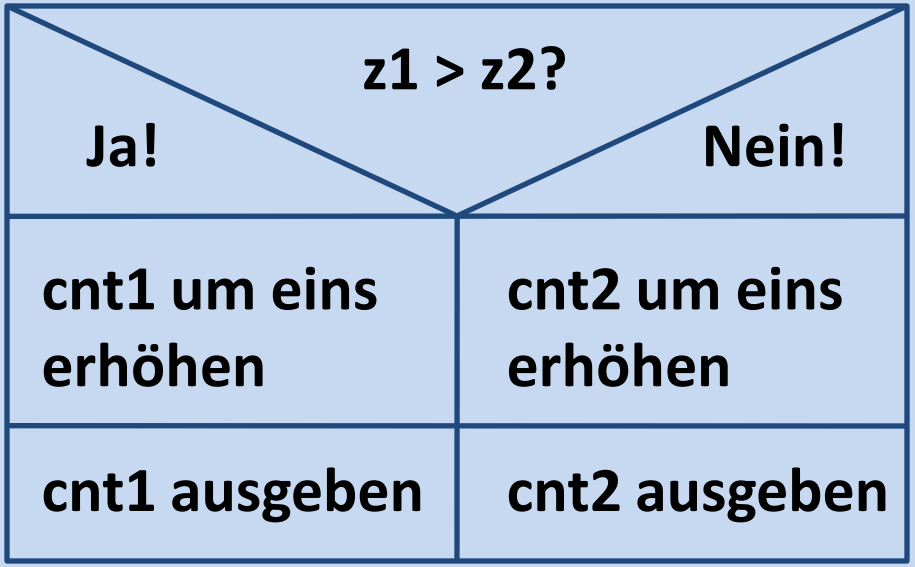
- Sequenz von Anweisungen, eingeschlossen von { }
- Hinter den einzelnen Anweisungen im Anweisungsblock stehen Strichpunkte, nicht aber hinter den geschweiften Klammern { }
- Enthält ein Anweisungsblock lediglich eine einzelne Anweisung, können die geschweiften Klammern { } entfallen

## Beispiel 1:

- Es soll das Maximum von zwei Zahlen bestimmt werden (Variablen: z1, z2 und max)
- Teilaufgabe A: Struktogramm erstellen
- Teilaufgabe B: C-Quelltext erstellen

## Beispiel 2:

- C-Quelltext zum Struktogramm erstellen
- Variablen: z1, z2, cnt1 und cnt2



**Beispiel 3:**

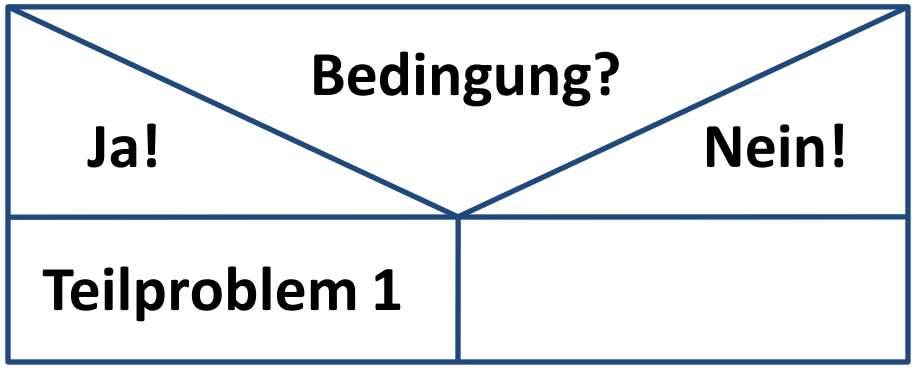
- Es soll das Maximum von drei Zahlen bestimmt werden (Variablen: z1, z2, z3 und max)
- Teilaufgabe A: Struktogramm erstellen
- Teilaufgabe B: C-Quelltext erstellen

**Tipp:**

- Sie können das Problem mit „verschachtelten“ if-else-Anweisungen lösen:
- Fragen Sie mit einer if-else-Anweisung zunächst die erste Bedingung ab; falls diese (nicht) erfüllt ist, fragen Sie mit einer zweiten – verschachtelten – if-else-Anweisung die zweite Bedingung ab.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

## Kontrollstruktur: bedingte Anweisung




**Bedeutung:**

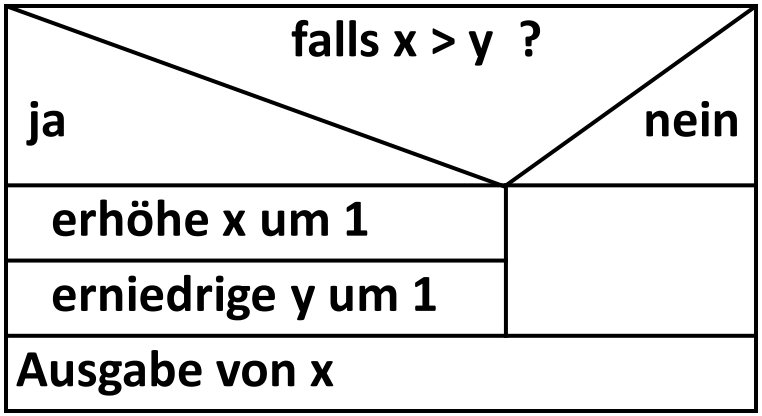
- Ist die Bedingung erfüllt, wird Teilproblem1 bearbeitet
- Andernfalls erfolgt „keine Anweisung“ (leere Anweisung)
- Entspricht einer if-Anweisung ohne „else“

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

## Umsetzung einer bedingten Anweisung in C:

```
if ( Bedingung )
     Anweisungsblock
```

**Beispiel:**




**Umsetzung nach C:**

```
if ( x > y )
{
   x = x + 1;
   y = y - 1;
}
printf("x = %d", x);
```

**Beispiel 4 („Body Mass Index“):**

- Nach Eingabe von Körpergröße l (in Metern) und Körpermasse m (in Kilogramm) soll der „Body Mass Index“ berechnet und ausgegeben werden: $bmi = m / l^2$
- Zusätzlich soll ermittelt werden, in welche der folgenden vier Kategorien der berechnete BMI fällt:

  BMI ≤ 19 → Untergewicht, ansonsten:
  BMI ≤ 25 → Normalgewicht, ansonsten:
  BMI ≤ 30 → leichtes Übergewicht, ansonsten:
  BMI > 30 → Übergewicht.

- Erstellen Sie zunächst ein Struktogramm und dann den C-Quelltext des Programms!

```c
#include <stdio.h>        /* Beispiel 4a: BODY MASS INDEX */
int main(void)
{
    float l, m, bmi;
    printf("Groesse in Metern:  "); scanf("%f", &l);
    printf("Koerpermasse in kg: "); scanf("%f", &m);
    bmi = m / (l*l);
    printf("BMI: %f\n", bmi);
    if(bmi <= 19)
    {
        printf("Untergewicht\n");
    }
    else
    {
        if(bmi <= 25)
        {
            printf("Normalgewicht\n");
        }
        else
        {
            if(bmi <= 30)
                printf("Leichtes Uebergewicht\n");
            else
                printf("Uebergewicht\n");
        }
    }
    return 0;
}
```

# 4.3. Verzweigung - Alternative

Geschachtelte if-else-Anweisungen sind oft unübersichtlich, wenn viele Bedingungen/Fälle überprüft werden müssen – **Ausweg: Mehrfachalternative**

```
if ( Bedingung_1 )
    Anweisungsblock_1
else if  ( Bedingung_2 )
    Anweisungsblock_2
else if
      . . . .
else if  ( Bedingung_n )
    Anweisungsblock_n
else
    Anweisungsblock_x
```

**Bedeutung :**

- Bedingungen werden nacheinander geprüft
- Diejenige Anweisung, bei der die Bedingung **erstmals erfüllt** ist, wird ausgeführt – alle anderen nicht
- Ist keine Bedingung erfüllt, wird die Anweisung nach letztem else ausgeführt
- Letztes else kann aber auch entfallen – entspricht der leeren Anweisung

# 4.3. Verzweigung - Alternative

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>       /* Beispiel 4b: BODY MASS INDEX */
int main(void)
{
    float l, m, bmi;
    printf("Groesse in Metern:  "); scanf("%f", &l);
    printf("Koerpermasse in kg: "); scanf("%f", &m);

    bmi = m / (l*l);
    printf("BMI: %f\n", bmi);

    if(bmi <= 19)
        printf("Untergewicht\n");
    else if(bmi <= 25)
        printf("Normalgewicht\n");
    else if(bmi <= 30)
        printf("Leichtes Uebergewicht\n");
    else
        printf("Uebergewicht\n");

    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

# Operatoren nach Funktionen geordnet:

| Funktion | Operatoren |
|---|---|
| arithmetisch | + - * / % ++ -- |
| relational | > >= < <= == != |
| logisch | && \|\| ! |
| bitorientiert | & \| ^ ~ << >> |
| zeigerorientiert | & * -> |
| zuweisend | = += -= *= /= %= &= ^= \|= <<= >>= |
| sonstige | , ( type ) ( ) [ ] sizeof ?: . |

Bedingungen (zum Beispiel in if-else-Anweisungen) werden mit **relationalen Operatoren** gebildet und mit **logischen Operatoren** verknüpft.

**Beispiele für Verknüpfungen von logischen Bedingungen:**

1. Maximum von 3 Zahlen z1, z2 und z3 bestimmen und der Variablen max zuweisen.
2. Bereichsprüfung: Liegt z im Bereich $1000 \leq z \leq 2000$?
3. Bereichsprüfung: Liegt z im Bereich $1 < z < 9$ und ist zugleich $z \neq 5$?
4. Body Mass Index: Ist $19 \leq BMI < 25$ (Normalgewicht)?
5. Finden Sie eine einfachere Darstellung für: „if ( !(z1 <= z2) )“

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

**Problem:**

Es soll die Summe der Quadrate von 1 bis n berechnet werden. (Dazu existiert zwar auch eine geschlossene Formel, diese soll allerdings nicht benutzt werden!)

**Berechne:**

$$sum = 1^2 + 2^2 + 3^2 + \ldots + n^2 = \sum_{i=1}^{n} i^2$$

Der Rechner kennt allerdings weder ein Summensymbol, noch ein „…“. Man muss dem Rechner daher genau vorschreiben, was er berechnen soll – jeden einzelnen Schritt!

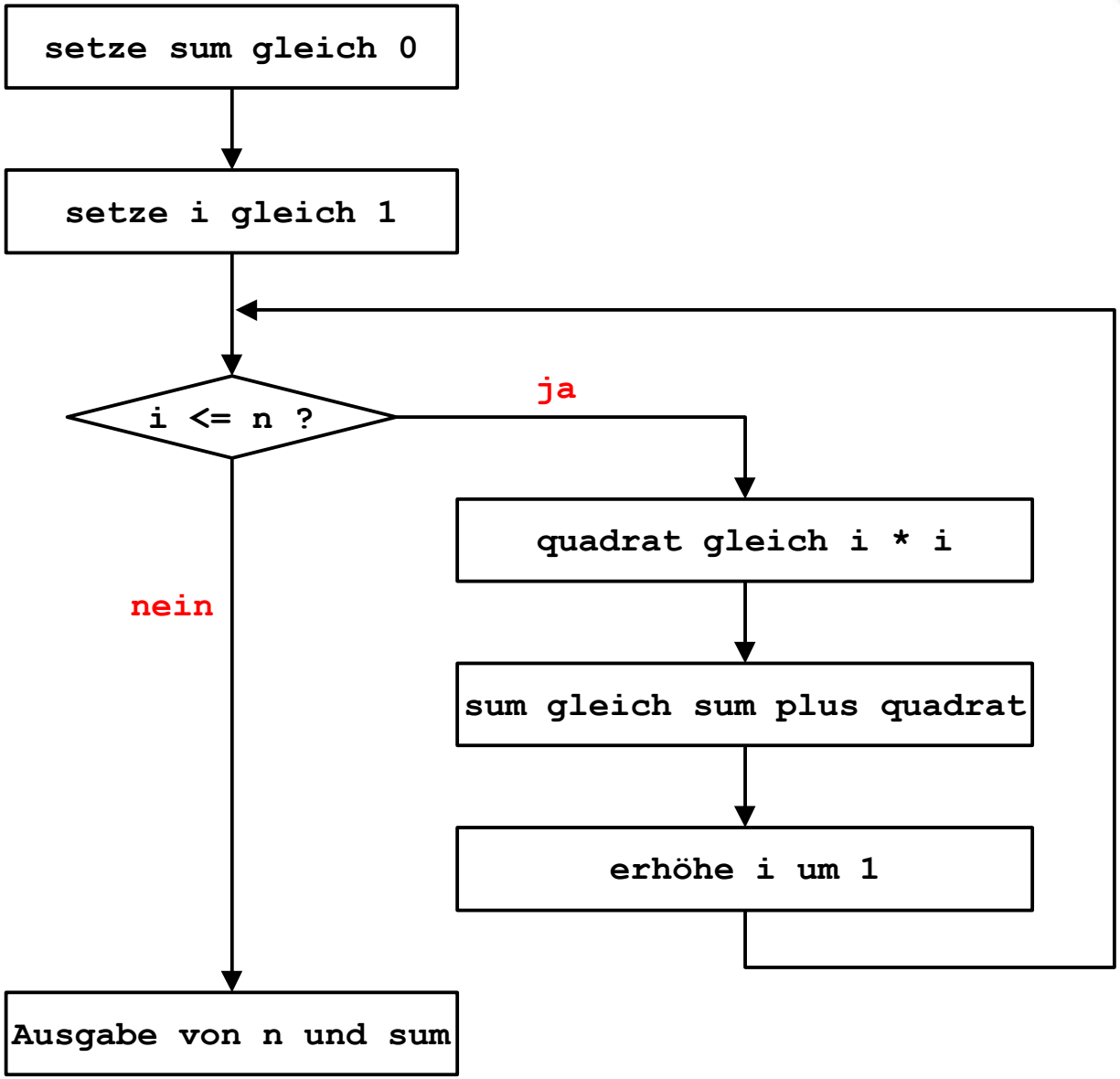
## Algorithmus:

- Variable sum auf 0 setzen
- Zählvariable i auf 1 setzen
- Berechne quadrat = i * i
- Addiere quadrat zu sum
- Erhöhe i um 1
- Prüfe, ob i kleiner gleich n ist; falls ja, Verfahren wiederholen

## Variablen:

| | | |
|---|---|---|
| i | : | Zählervariable |
| n | : | Endwert |
| quadrat | : | Berechnet in jedem Durchlauf das Quadrat von i |
| sum | : | Die berechnete Quadratsumme |

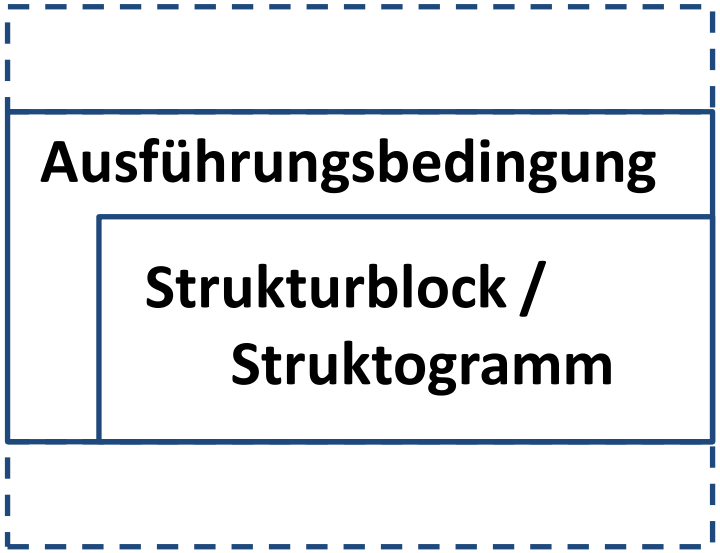
# 4.5. Wiederholungen - Schleifen



```c
#include <stdio.h>          /* Version 1:              */
int main(void)              /* Abweisende Schleife */
{
    int i, n, quadrat, sum;
    n   = 5;
    sum = 0;
    i   = 1;
    while(i <= n)
    {
        quadrat = i * i;
        sum = sum + quadrat;
        i = i + 1;
    }
    printf("Quadratsumme bis %d = %d\n", n, sum);
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kontrollstruktur: abweisende Schleife




## Bedeutung:

Ausführungsbedingung prüfen

- Ist Bedingung **erfüllt**, dann:
  - Strukturblock innerhalb der Schleife ausführen
  - Danach erneut prüfen
- Ist Bedingung **nicht erfüllt**, dann:
  - Schleife verlassen
  - Nächsten Strukturblock nach der Schleife ausführen

## Umsetzung in C:

```
while ( Ausführungsbedingung )
    Anweisungsblock
```

## Nichtabweisende Schleife:

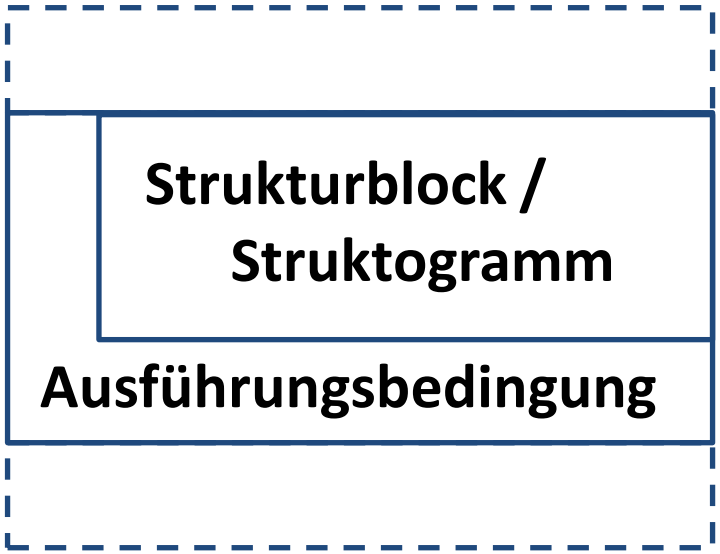
Bei einer **abweisenden Schleife** ist die Bedingungsprüfung am Anfang. Daher ist es ggf. möglich, dass die Schleife überhaupt nicht durchlaufen wird – die Schleife wird komplett übersprungen.

Bei manchen Problemen ist es zweckmäßiger, die Bedingungsprüfung erst am Ende eines Durchlaufs zu machen. Die Schleife wird dann auch bei einer nicht erfüllten Bedingung **wenigstens einmal** durchlaufen → nachfolgende Bedingungsprüfung oder **nichtabweisende Schleife.**

```c
#include <stdio.h>      /* Version 2:                    */
int main(void)          /* Nichtabweisende Schleife */
{
    int i, n, quadrat, sum;
    n = 5;
    sum = 0;
    i = 1;
    do
    {
        quadrat = i * i;
        sum = sum + quadrat;
        i = i + 1;
    }
    while(i <= n);
    printf("Quadratsumme bis %d = %d\n", n, sum);
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kontrollstruktur: nichtabweisende Schleife




## Bedeutung:

1) Strukturblock ausführen
2) Ausführungsbedingung prüfen:
   - Ist Bedingung erfüllt, Struktur-block erneut ausführen (weiter mit Schritt 1)
   - Ist Bedingung nicht erfüllt, Schleife verlassen und mit erstem Struktur-block nach der Schleife fortsetzen

## Umsetzung in C:

```
do
    Anweisungsblock
while ( Ausführungsbedingung );
```

**Problem:**

Es soll eine ganze Zahl x eingelesen werden. Die Eingabe ist so lange zu wiederholen, bis die eingegebene Zahl im Zahlenbereich $50 \leq x < 100$ liegt.

1) Erstellen Sie ein Struktogramm zur Lösung dieser Aufgabe!
2) Erstellen Sie ein entsprechendes C-Programm!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

## Beobachtung:

In Zählschleifen finden sich oft die folgenden drei Komponenten:
**Initialisierung**, **Ausführungsbedingung**, **Veränderungsschritt**

```
#include <stdio.h>
int main(void)
{
    int i;
    i = 1;
    while(i <= 10)
    {
        printf("%d\n", i);
        i = i + 1;
    }
    return 0;
}
```

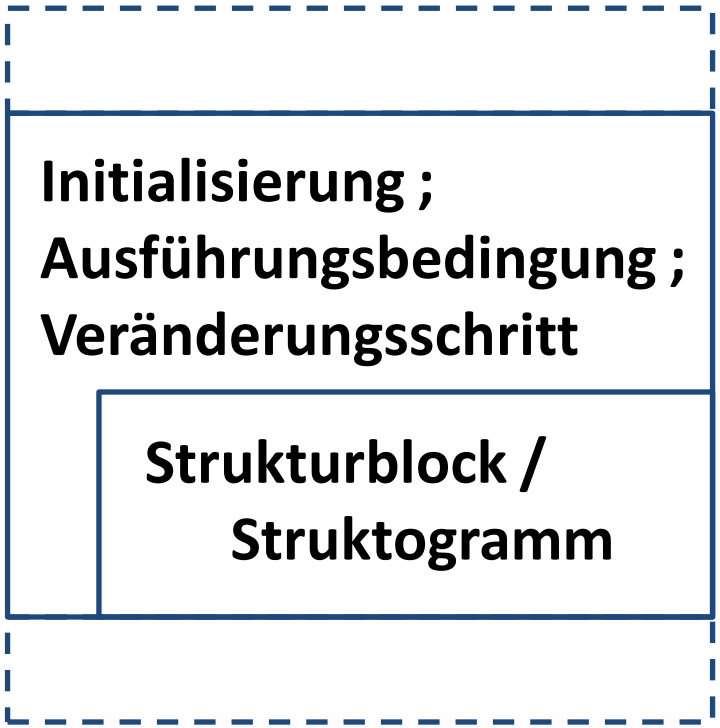
**Initialisierung**
(einmalig, vor Beginn der Schleife)

**Ausführungsbedingung**
(wird vor jedem Durchlauf der Schleife geprüft)

**Veränderungsschritt**
(wird nach jedem Durchlauf ausgeführt)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kontrollstruktur: Zählschleife




## Bedeutung:

- Die Zählschleife ist eine Sonderform der abweisenden Schleife
- Sie bietet eine kompaktere Schreibweise für die Einzelkomponenten **Initialisierung**, **Ausführungsbedingung** und **Veränderungsschritt**

## Umsetzung in C:

```
for (initialisierung; bedingung; veränderung)
    Anweisungsblock
```

**Problem:**

Berechne die folgende Summe für ein gegebenes n:

$$\text{sum} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

## Umsetzung in C:

```
float sum; int i; int n = 5;
sum = 0.0;
for(i = 1; i <= n; i = i + 1)
{
    sum = sum + 1.0 / i ;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

**Problem:**

Schreibe ein Programm, das einen Radius r einliest und wahlweise den Kreisumfang, die Kreisfläche oder das Kugelvolumen berechnet und ausgibt.

**Variablen:**

| | | | |
|---|---|---|---|
| auswahl | : | 1 = Kreisumfang | Typ int |
| | : | 2 = Kreisfläche | |
| | : | 3 = Kugelvolumen | |
| | | andere Werte sind ungültig | |
| r | : | Radius | Typ float |
| erg | : | Ergebnis | Typ float |

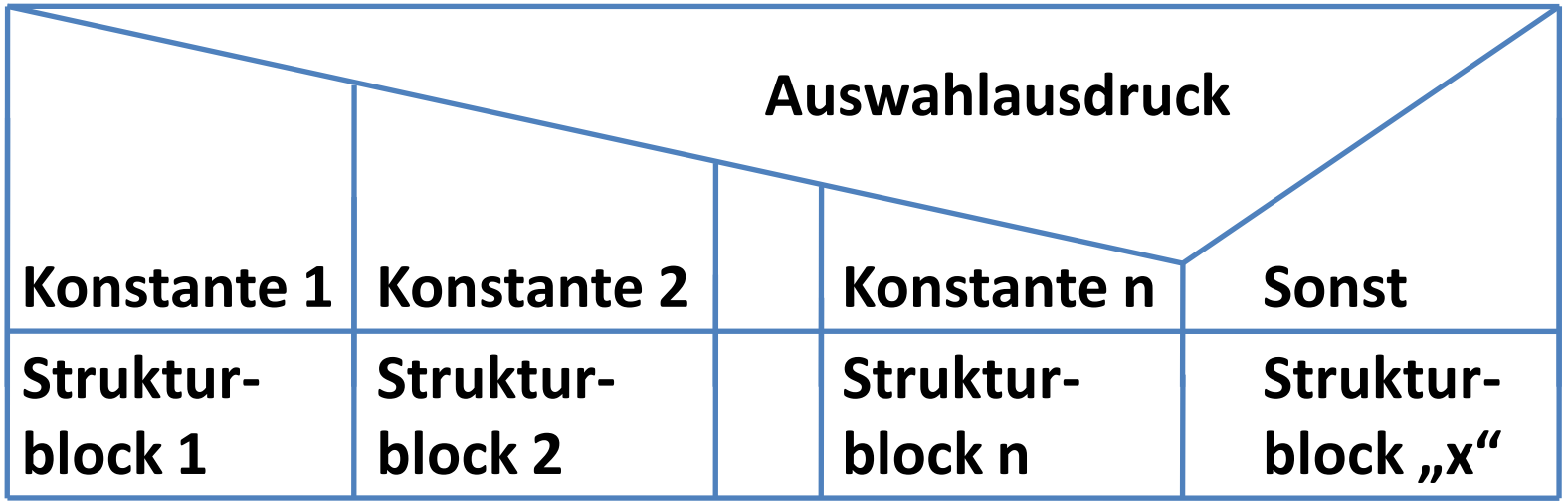
# 4.6. Mehrfachauswahl

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```c
#include <stdio.h>    /* Mehrfachauswahl:  Kreis, Kugel berechnen */
int main(void)
{
    int wahl; float r, erg;
    printf("Umfang(1)-Flaeche(2)-Volumen(3)?"); scanf("%d", &wahl);
    printf("Radius ?\n"); scanf("%f", &r);
    switch(wahl)
    {
    case 1:  erg = 2.0*3.14*r;
             printf("Umfang = %f", erg);
             break;
    case 2:  erg = 3.14*r*r;
             printf("Flaeche = %f", erg);
             break;
    case 3:  erg = 4.0/3.0*3.14*r*r*r;
             printf("Volumen = %f", erg);
             break;
    default: printf("Auswahl falsch");
    }
    printf("\nProgrammende!");
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kontrollstruktur: Mehrfachauswahl




**Bedeutung:**

1) Auswahlausdruck (Fallabfrage) auswerten – ergibt einen Integerwert
2) Strukturblock ausführen, bei dem die Konstante mit dem Wert des Auswahlausdrucks übereinstimmt
3) Tritt keiner der aufgeführten Fälle ein, „Sonst-Block“ ausführen

Der „Sonst-Block“ kann entfallen; dann erfolgt keine Aktion, wenn der Auswahlausdruck mit keiner Konstanten übereinstimmt

**Achtung: Konstante 1, Konstante 2... müssen Integer-Konstanten sein, also keine Gleitkommazahlen und auch keine Variablen!**

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

## Umsetzung in C:

```
switch ( Auswahlausdruck )
{
      case Konstante_1 : Anweisungen_1
                         break;
      case Konstante_2 : Anweisungen_2
                         break;

      . . . .

      case Konstante_n : Anweisungen_n
                         break;
      default          : Anweisungen_x
}
```

***Achtung:***

***An dieser Stelle sind lediglich Integer-Konstanten erlaubt (keine Variablen, auch keine Bereiche)!***

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

**Sprunganweisungen** bewirken, dass das Programm an einer anderen Stelle fortgesetzt wird, sie unterbrechen den linearen Ablauf de Programms.

Ein Beispiel für eine Sprunganweisung, die **break-Anweisung**, findet sich im Beispiel zur Mehrfachauswahl („Kreis, Kugel berechnen“)

- Die break-Anweisung verlässt den Bereich der switch-case-Anweisung und setzt die Programmusführung mit der ersten nachfolgenden Anweisung fort.
- Wird break vergessen, dann werden die nachfolgenden case-Blöcke ebenfalls ausgeführt!

Die break-Anweisung wird aber nicht nur in switch-case-Anweisungen eingesetzt, sondern auch bei der Programmierung von Schleifen:

**Wirkung der break-Anweisung:**

- Die aktuelle for-, while-, do-while-Schleife bzw. switch-Anweisung wird sofort verlassen
- Die Programmausführung wird unmittelbar nach dem verlassenen Block (Schleife/Anweisung) fortgesetzt
- Beachte: Nur die innerste Schleife wird verlassen!

```
int i, j, k;
Anweisungen ...
for(j = 0; j < 100; j++)
{
    Anweisungen ...
    if(j == (k + 23))
        break;
    Anweisungen ...
    for(i = 0; i < 50; i++)
    {
        Anweisungen ...
        if(i == k)
            break;
        Anweisungen ...
    }
    Anweisungen ...
}
Anweisungen ...
```

**Beispiel:**

Verschachtelte Schleifen und break

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

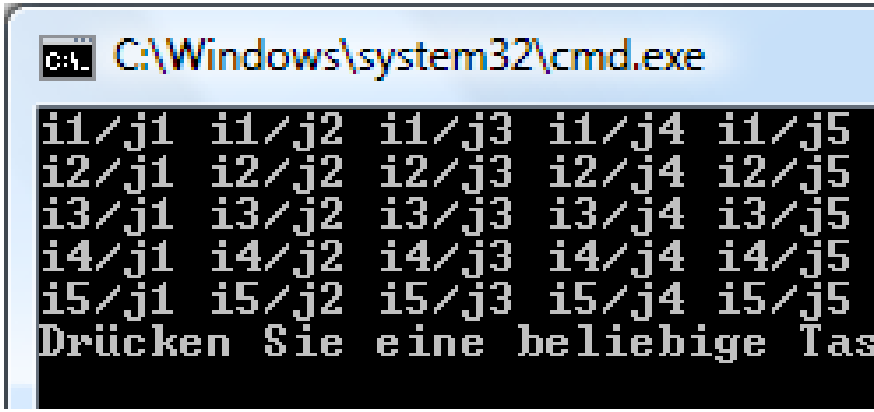
# Kapitel 4 – Strukturierte Programmierung und Kontrollstrukturen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

## Beispiel 1:

- Wenn eine (äußere) Schleife wiederum eine (innere) Schleife enthält, spricht man von verschachtelten Schleifen.
- In jedem einzelnen Durchlauf der äußeren Schleife wird die komplette innere Schleife abgearbeitet.

```
#include <stdio.h>
int main(void)
{
  int i, j;
  for(i = 1; i < 6; i = i + 1)
  {
    for(j = 1; j < 6; j = j + 1)
    {
      printf("i%d/j%d \t", i, j);
    }
    printf("\n");
  }
  return 0;
}
```



## Beispiel 2:

- Ermitteln Sie, ob die Ziffernsumme (Quersumme) einer ganzen Zahl gerade oder ungerade ist.
- Fordern Sie den Anwender zur Eingabe der Zahl auf. Geben Sie die Zahl zur Kontrolle aus. Geben Sie die Zwischenwerte der Quersummenbildung aus.
- Nachdem Sie ausgegeben haben, ob die Quersumme gerade oder ungerade ist, verlangen Sie nach einer neuen Zahl. Durch Eingabe einer 0 soll Ihr Programm abgebrochen werden.
- Erstellen Sie zunächst ein Struktogramm. Schreiben Sie anschließend das zugehörige C-Programm.

***Anwendung zum Beispiel bei Prüfsummen:*** *Prüfen, ob eine vierstellige Artikelnummer gültig ist oder nicht. Dazu wird eine zusätzliche Stelle hinzugefügt, so dass die Quersumme durch 10 teilbar ist.*

**12331 42716 42718**

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

## Teilprobleme:

- Zahl einlesen und wieder ausgeben
- Quersumme bilden
- Prüfen, ob Quersumme gerade oder ungerade ist
- Ergebnis ausgeben
- Neue Zahl einlesen und wieder ausgeben
- Abbruchbedingung prüfen → Abbruch bzw. Wiederholung

**Quersumme bilden:** 4613 → 14

*alle Ziffern addieren – aber wie bekommt man die Ziffern?*

letzte Ziffer: Rest bei Division durch 10 (Modulo-Operator!)

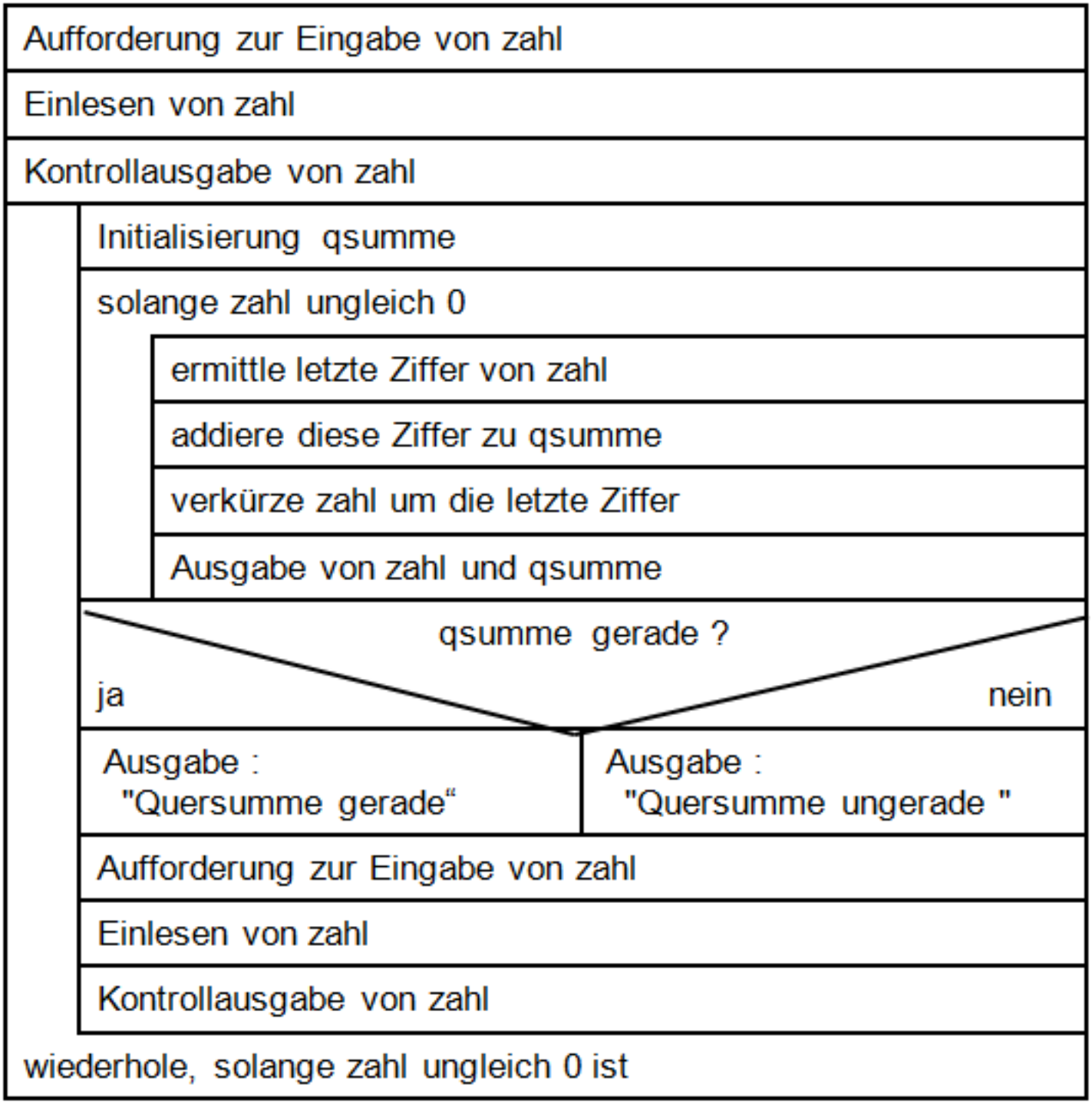
neue Zahl: alte Zahl / 10 (Integer-Division, Rest abschneiden!)

**Quersumme gerade oder ungerade:**

quersumme % 2 == 0, gerade == 1, ungerade

**Variablen:**

zahl (eingelesene Ziffer), qsumme (Quersumme), lziffer (letzte Ziffer)




**Struktogramm**