h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Kap 4. Befehle und Struktogramme

3.1 Elementare Befehle

Befehlstypen

Zuweisung

Verzweigung

Schleife

Block (zusammengesetzt)

Ein-/Ausgabe (eigentlich keiner)

```
Eingabe          cin>>a;
Verzweigung      if ( a<=0 )// unzulaessige Eingabe
Ausgabe                cout <<"\n  a muss > 0 sein ";
                 else
Block-Anfang     {
Zuweisung              x=a;
Schleife-Anfang        do
Zuweisung                  x=x-(x*x-a)/(2*x);
Schleife-Ende          while ( abs (x*x-a)>genauigkeit );
                       //Ergebnis
Eingabe                cout <<"\n  Wurzel von "<<a<<" = "<<x;
Block-Anfang     }
```

Elementare Befehle

Die elementaren "Befehle" sind also

Zuweisung

Berechnung eines Ausdrucks und Speicherung des Ergebnisses

Verzweigung

Fallunterscheidungen

Schleife

Wiederholungen

Block

Ein Block ist aus anderen Befehlen zusammengesetzt

Ein-/Ausgabe

ist in C++ eigentlich kein Befehl sondern ein Ausdruck
cin>>a entspricht Operand Operator Operand

3.2 Grafische Darstellung

Zur Verdeutlichung der Logik eines Programmes sind Grafische Darstellungen sehr hilfreich.

Hierzu gibt es eine Reihe von Möglichkeiten und Tools:

- Flußdiagramme
- Struktogramme (*)
- UML (Unified Modeling Language)

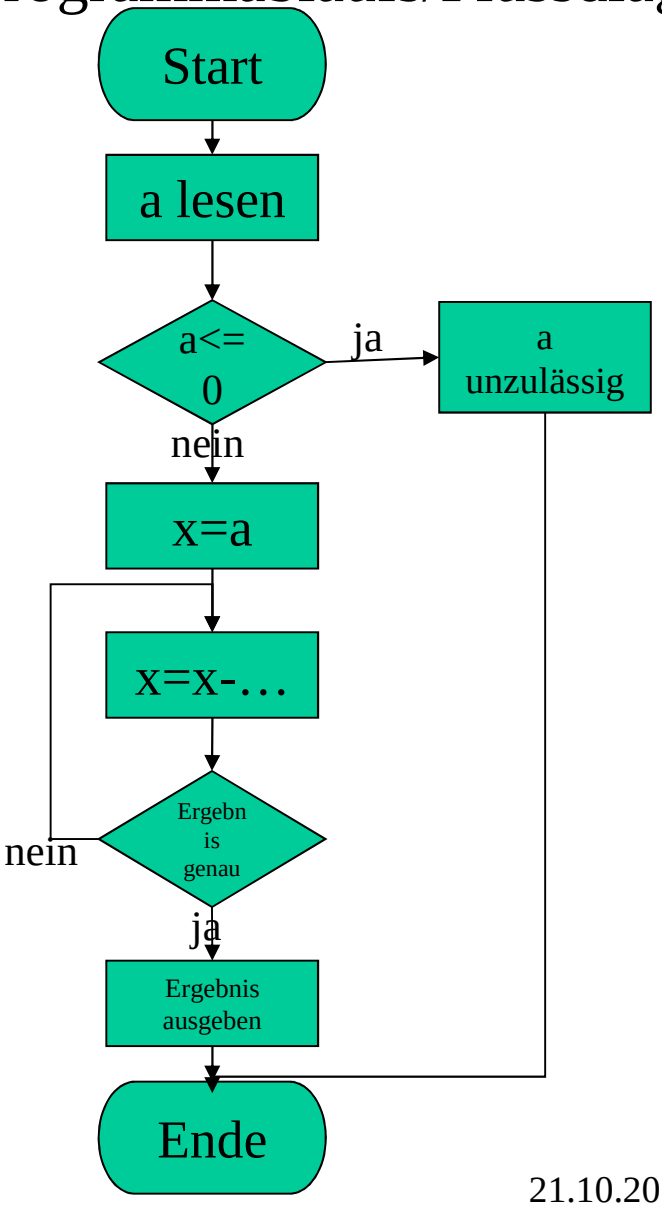
Ziele sind

- eine bessere Darstellung komplexer Abläufe und vor allem die Dokumentation des Programms

Grafische Darstellung des Programmablaufs/Flussdiagramm

Flußdiagramme sind einfach zu verstehen, werden aber schnell unübersichtlich.

Es gibt nur wenige Symbole

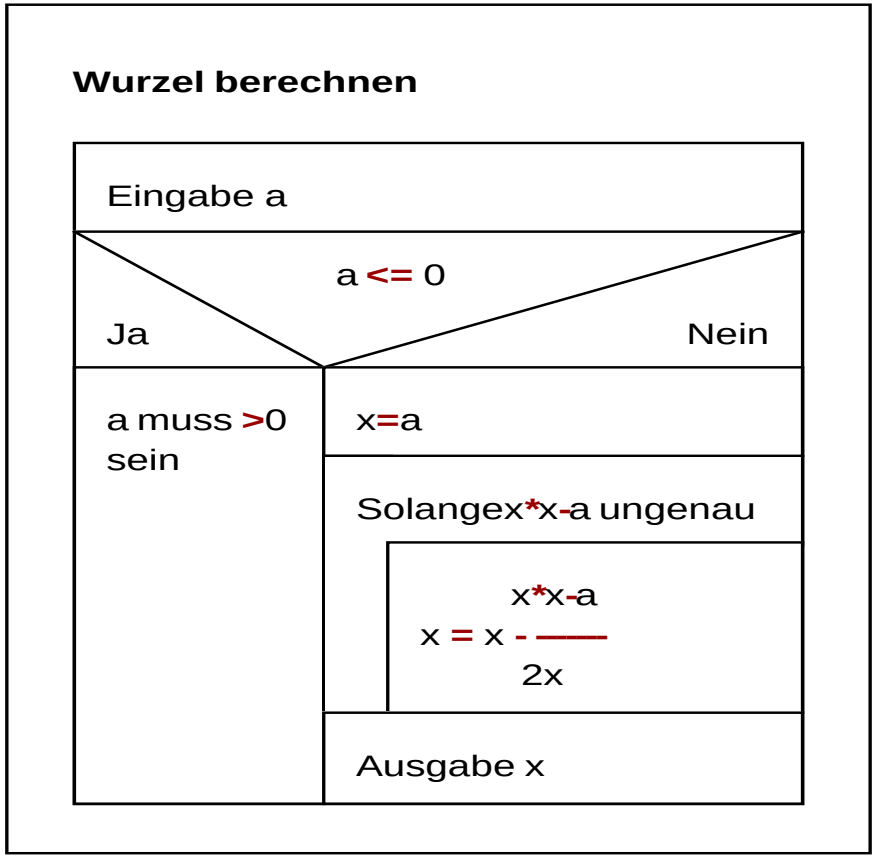


Grafische Darstellung des Programmablaufs/Struktogramm

Struktogramme unterstützen sehr gut die Entwicklung komplexer Abläufe und sind primär ein Mittel zur Unterstützung des Entwurfs.

Es gibt nur wenige Symbole und daher ist diese Technik sehr einfach.

Sie sind ab sofort zu jedem Programm anzufertigen und vor Beginn des Praktikums zusammen mit dem Testplan abzugeben.

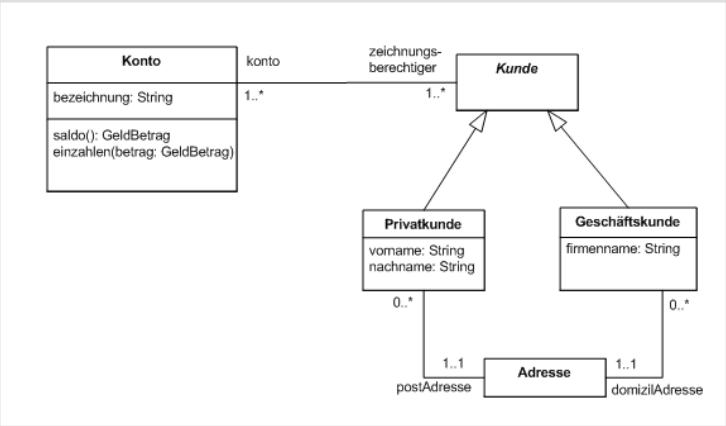


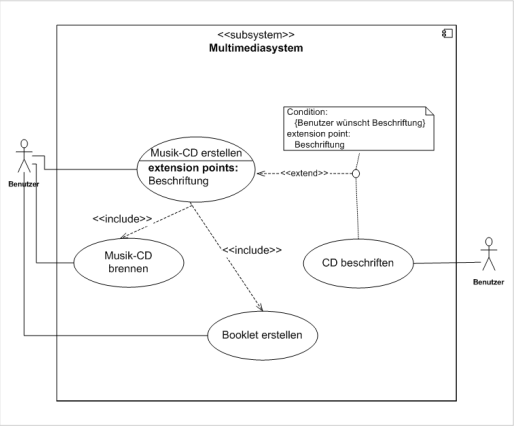
Grafische Darstellung des Programmablaufs/UML

UML Unified Modeling Language

Ist eine Sprache für für Modellierung von Programmen mit grafischen Symbolen für die verwendeten Begriffe.

UML ist wesentlich komplexer (es gibt 13 Diagrammtypen) als die bisher genannten Bespiele und wird in späteren Vorlesungen behandelt. Beispiele siehe unten (Quelle: Wikipedia)





3.3 Blöcke und Gültigkeitsbereiche

Ein **Block** ist die Zusammenfassung mehrerer Befehle mittels eines Paars geschweifter Klammern:

```
{
   hier kommen die Befehle des Blocks
}
```

Ein Block ist damit wie **ein Befehl.**
Er wird immer dort verwendet, wo nur ein Befehl stehen soll, wo man aber mehrere Befehle ausführen möchte.

Dies wird dann bei den Schleifen und Verzweigungen in der Regel der Fall sein

Gültigkeitsbereich

In einem Block dürfen auch Variable deklariert werden.
Unter dem **Gültigkeitsbereich** einer Variablen versteht man den Programmabschnitt, wo die Variable bekannt ist und wo daher auf sie zugegriffen werden kann.
Der Bereich beginnt mit der Stelle innerhalb des Blocks, wo sie deklariert wird und endet am Ende dieses Blocks. Nach Beendigung des Blocks existiert diese Variable nicht mehr. Ihr Speicherplatz ist freigegeben.
Derartige Variable nennt man **lokale Variable**.

```
{
        Deklaration z.B. int i
        Gültigkeitsbereich
}
```

Gültigkeitsbereich/globale Variable

Globale Variable müssen außerhalb von geschweiften Klammern deklariert werden, also vor dem Programm.

Damit sind sie überall bekannt, d.h. in jedem Block.

```
// lokal/global
#include <iostream>
using namespace std;
int i=2;
void main()
{
   int j=1;
   {
      int k=2;
    }
}
```

Globale Variable

Lokale Variable

Weil globale Variable überall bekannt sind und überall damit geändert werden können sind stellen sie ein Risiko dar und sollten üblicherweise nicht verwendet werden.
Ausnahme etwa: globale Konstanten

Gültigkeitsbereich/Beispiel

```
// lokal/global
#include <iostream>
using namespace std;
int i=2;
void main()
{
        int j=1;
        {
                int j=2;
                cout << i << j << endl;
        }

        cout << i << j << endl;
}
```

Der Name der Variablen kann auch dem einer Variablen gleich sein, die schon vorher deklariert wurde. Damit "überdeckt diese Variable die vorher deklarierte, solange man sich in diesem Block befindet.

Was wird ausgegeben?

3.4 Zuweisungen

Zuweisung sind aus Kapitel 3 bekannt.
Sie stellen eigentlich einen Ausdruck dar !!!

Beispiel flaeche=r*r*3.14159;
i = i+1;

Bzw. auch i+=1;//zusammengesetzte Zuweisung

Aufbau

Variable = Ausdruck bzw. Variable op= Ausdruck

Zuweisungen/Struktogramm

flaeche = r*r*3.14159

Symbol: Rechteck !!!!!!!

flaeche = $r^2*\pi$

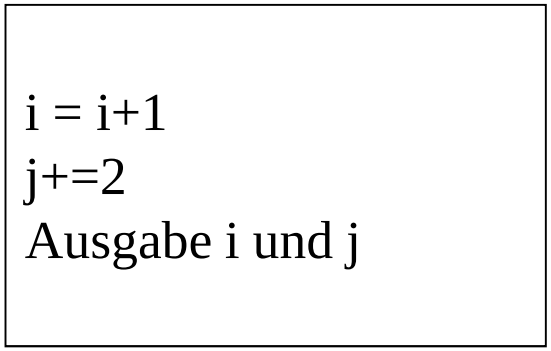
Inhalte sind beliebig zu beschreiben.
Es muss kein C++ Code sein.

$$x = x - \frac{x^2 - a}{2x}$$

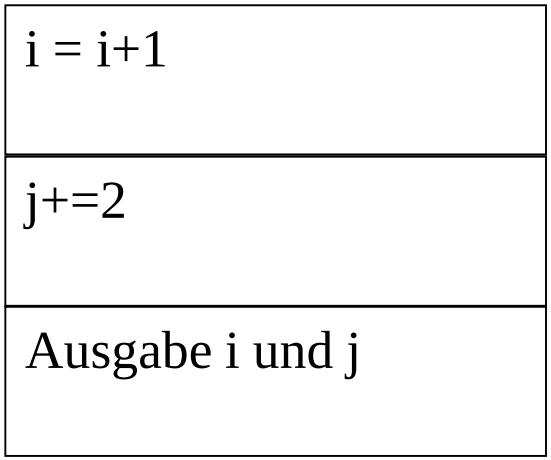
C++ Code sollte aber daraus direkt ableitbar sein

Anmerkung: Das Rechtecksymbol ist das allgemeine Symbol innerhalb der Struktogramme. Es kann jeder Befehl hineingeschrieben werden.

Sequenz



Es dürfen auch mehrere Befehle in ein Rechteck.



Mehrere Rechtecke stehen direkt untereinander.
Dies wird dann auch als Sequenz bezeichnet.

3.5 Verzweigungen

Verzweigungen dienen zur Fallunterscheidung,
d.h. in Abhängigkeit von einer Bedingung soll zwischen verschiedenen Möglichkeiten der weiteren Befehlsausführung entschieden werden wie z.B.
- beim Wurzelziehen (Radikand <0)
- beim Suchen (Vergleich mit dem gesuchten Element)
- bei der Eingabe (zulässig ja/nein)
-- der Anwender kann aus einer Reihe von Aktionen auswählen (Menue).

Es gibt 3 Arten von Verzweigungen

Verzweigungen mit if-else

switch

Auswahloperator

Verzweigung mit if – else : Beispiele

```
if ( x<y )
    minimum=x;
else
    minimum=y;

if ( x<y )
{
    minimum=x;
    cout<<" das Minimum ist x"<<endl;
}
else
{
    minimum=y;
    cout<<" das Minimum ist y"<<endl;
}
cout<< " Der Wert des Minimums ist "<<minimum<<endl;
```

Einrücken und Klammern
if/else stehen in derselben Spalte.

Die Befehle im if- bzw. else-Zweig sollten stets eingerückt werden:
z.B. 3 Positionen

Die Klammern sollten auf Höhe von if / else stehen

Verzweigung mit if-else : Allgemein

```
if ( Bedingung )
    Anweisung1
else
    Anweisung2
Anweisung3//naechste Anweisung nach dem if-else
```

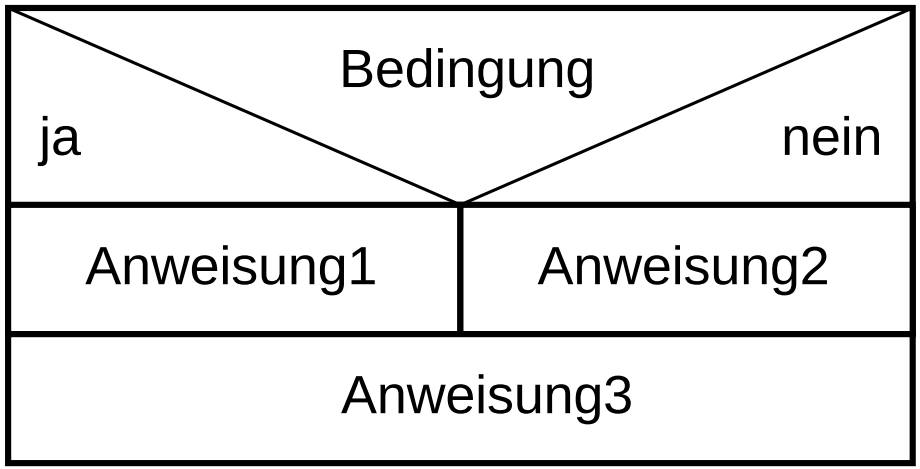
Wirkungsweise

Auswertung der Bedingung = boolescher Ausdruck
falls true : Führe Anweisung1 aus, dann kommt Anweisung3
falls false: Führe Anweisung2 aus, dann kommt Anweisung3

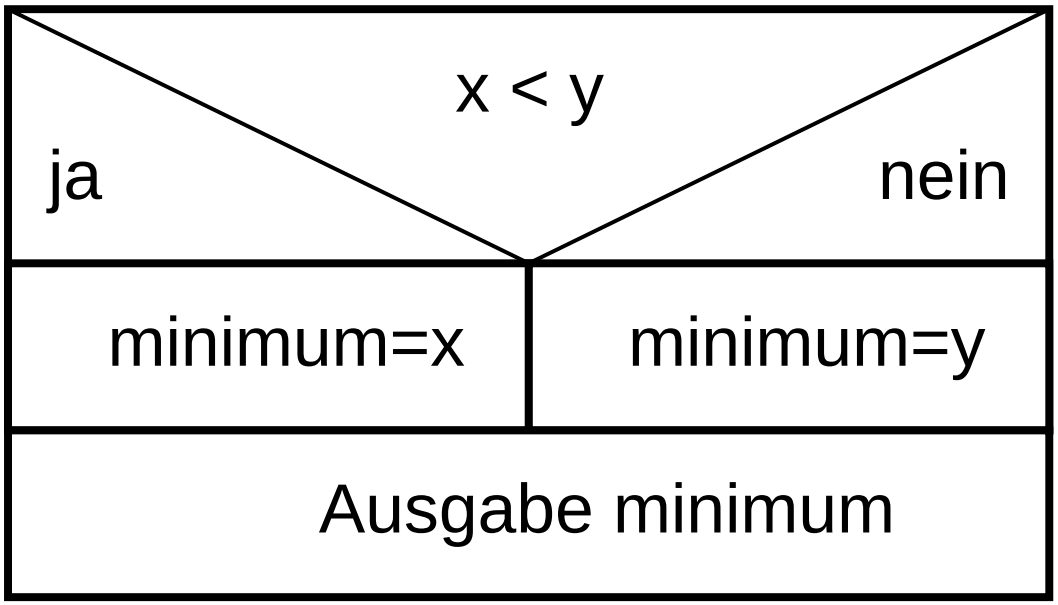
Anmerkung:
-Mit Anweisung ist jeweils auch ein Block gemeint (siehe Musterprogramm).
-Der else-Zweig kann weggelassen werden.

```
if ( Bedingung)
    Anweisung1
Anweisung3//naechste Anweisung
```

Struktogramm



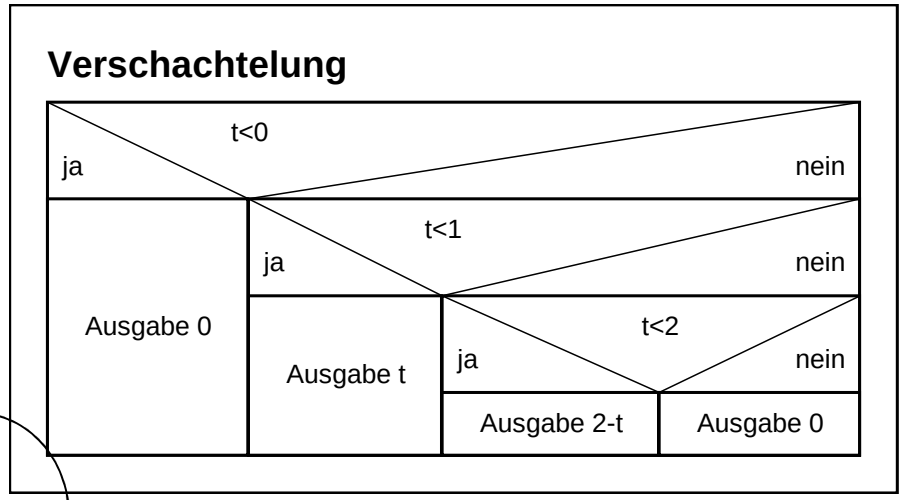
Beispiel



Sequenz: 2 Rechtecke untereinander

Verschachtelung

```
// Werte Stromimpuls
#include <iostream>
using namespace std;
void main()
{
  double t;
  cin >> t ;
  if ( t < 0 )
    cout << " Wert ist " << 0  <<endl;
  else
  {
    if ( t < 1 )
      cout << " Wert ist " << t <<endl;
    else
    {
       if ( t < 2 )
         cout << " Wert ist " << 2-t <<endl;
       else
         cout << " Wert ist " << 0 <<endl;
     }
  }
}
```



if-else Befehle müssen immer vollständig
im if oder else Teil eines „übergeordneten“
if-else Befehls enthalten sein.

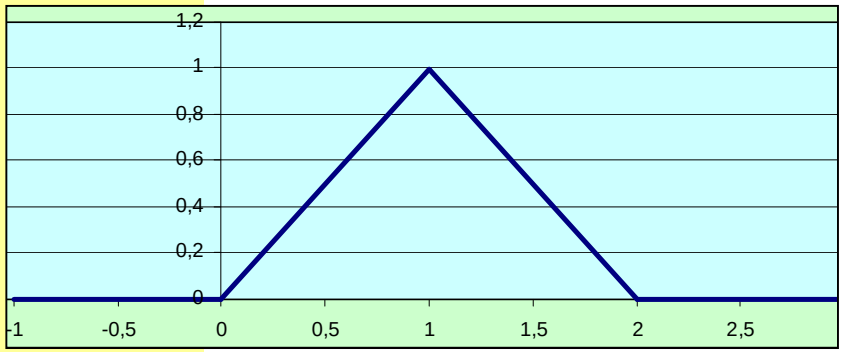
Übliche Vorgehensweise:
Von außen nach innen.

Alternative

```
// Werte Stromimpuls
#include <iostream>
using namespace std;
void main()
{
        double t;
        cin >> t ;
        if ( t < 0 )
                cout << " Wert ist " << 0  <<endl;
        else if ( t < 1 )
                cout << " Wert ist " << t <<endl;

        else if ( t < 2 )
                cout << " Wert ist " << 2-t <<endl;
        else
                cout << " Wert ist " << 0 <<endl;

}
```



if - else if - else if - … else **else if Kette**

else if Kette

```
if ( Bedingung1)
        Anweisung1
else if (Bedingung2)
        Anweisung2
else if ....

else
        Anweisungm
Anweisungn
```

Wirkungsweise

Auswertung von Bedingung1

falls true : Führe Anweisung1 aus,
dann Anweisungn

falls false:

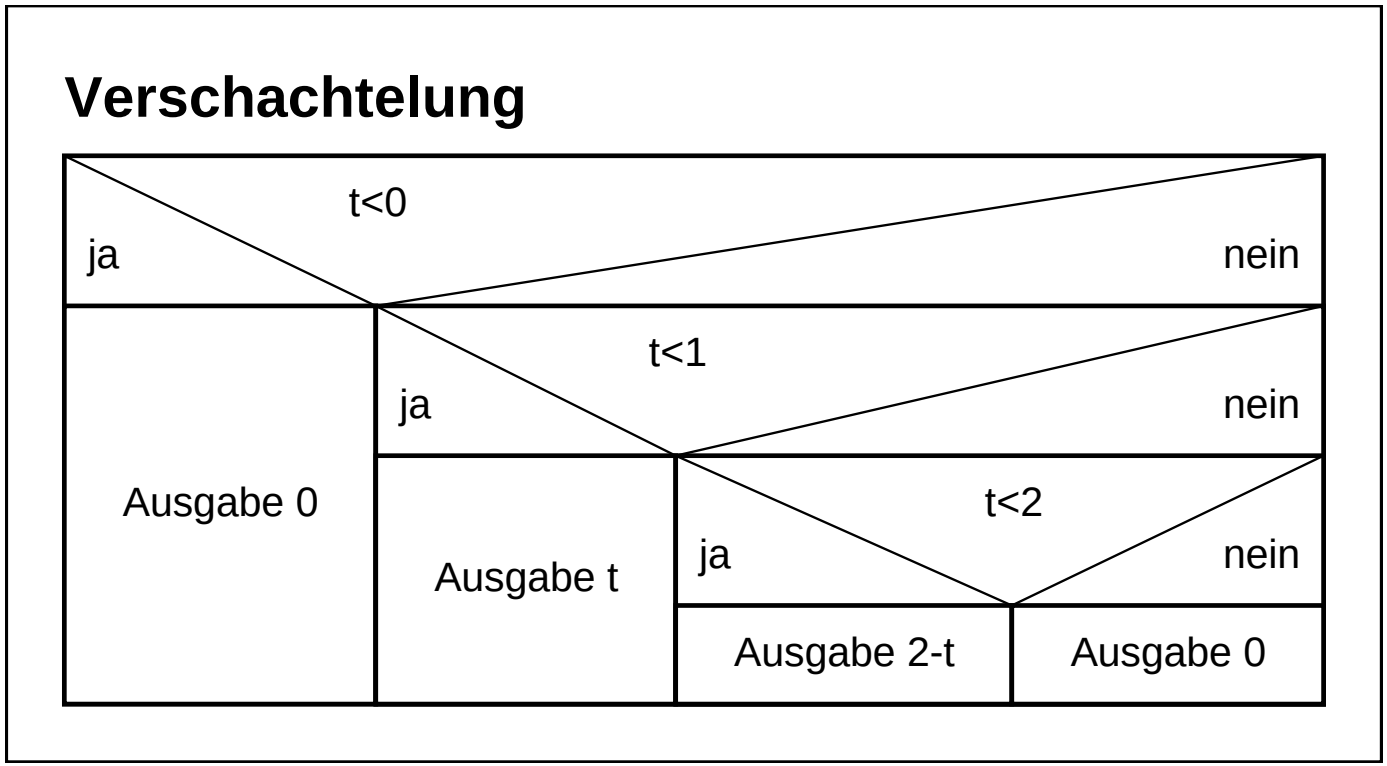
Auswertung von Bedingung2

falls true : Führe Anweisung2 aus,
dann Anweisungn

falls false:

.....

Struktogramm



Wie Verschachtelung

Verzweigung mit Switch: Beispiel

Rechner für Grundrechenarten und %

```
// Kleinrechner/switch
#include <iostream>
using namespace std;
void main()
{
        int i,j;
        char op;
        cin >> i >> op >> j;
        switch (op)
        {
                case '+' :
                        cout << i << op << j <<" = " << i+j << endl;
                        break;
                case '-' :
                        cout << i << op << j <<" = " << i-j << endl;
                        break;
                case '*' :
                        cout << i << op << j <<" = " << i*j << endl;
                        break;
                case '/' :
                        cout << i << op << j <<" = " << i/j << endl;
                        break;
                case '%' :
                        cout << i << op << j <<" = " << i%j << endl;
                        break;
                default :
                        cout << " unzulässiger Operator " << endl;
        }
}
```

switch

```
switch ( ganzzahliger Wert )
{
   case konstante1 :
         Befehl;Befehl;...Befehl;
         break;
   case konstante2 :
         Befehl;Befehl;...Befehl;
         break;
    .....

   default :
         Befehl;Befehl;...Befehl;
}
```

Einrücken/Klammern
Analog Verzweigung

Kann weggelassen werden
Nicht empfehlenswert !!!!!

switch

Wirkungsweise

Bestimmung des Werts (muss ganzzahliger Typ sein)

Vergleich mit den ganzzahligen Konstanten (alle verschieden)

Bei Übereinstimmung mit einer Konstanten->Verzweigung zur case – Marke.
Ausführung der Befehle bis zum break, dann Verzweigung zum Ende
(nach dem switch).

Fehlt der break-Befehl wird mit dem darauf folgenden Befehl fortgesetzt !!!!!

Stimmt der Wert mit keiner Konstanten überein wird bei default fortgesetzt,
Sofern vorhanden (wenn nicht, findet keine Aktion statt).

Struktogramm

op					
+	-	*	/	%	sonst
Ausgabe i + j	Ausgabe i - j	Ausgabe i * j	Ausgabe i / j	Ausgabe i modulo j	Fehler meldung

Gegenüberstellung

Die else-if Kette ist universeller als switch. Sie kann mit beliebigen Bedingungen formuliert werden.

Switch ist beschränkt auf den Vergleich ganzzahliger Ausdrücke (u.a. Aufzählungstyp !!), ist aber übersichtlicher, insbesondere wenn es viele Fälle sind.

Verzweigung mit dem Auswahloperator

Beispiel

Anstelle von

```
if ( x<y )
    minimum = x;
else
    minimum = y;
```

Form :

(Bedingung) ? wert1: wert2;

Wirkung :

Ist die Bedingung wahr, wird wert1 verwendet, sonst wert2.

kann auch

```
minimum = (x<y)? x : y;
```

mit dem Auswahloperator ? verwendet werden (bedingte Bewertung).

Der Auswahloperator wird bei größeren Ausdrücken schnell unübersichtlich.

3.5 Schleifen

Schleifen dienen dazu, eine Gruppe von Befehlen zu wiederholen etwa bei
- Der Eingabe einer Tabelle von Zahlen
- Berechnung des Durchschnitts einer großen Menge von Zahlen
- Der Bestimmung des Minimums/Maximus einer Menge von Messwerten
- Der Suche eines Elements in einer Tabelle
-Der Anwender wird/kann beliebig oft Aktionen durchführen (Menue)

Diese Wiederholungen finden solange statt, wie eine Wiederholungsbedingung (WB) gilt:
-Noch Daten da
-Noch nicht alle Zahlen untersucht (bei Maximum, Minimum, Suche)

WB ist üblicherweise ein Ausdruck vom Typ bool.

Das Prinzip ist also
….. WB prüfen – Befehle ausführen – WB prüfen – ….
C++ kennt nun 3 verschiedene Arten von Schleifen

Übersicht Schleifen

while-Schleife (siehe Musterprogramm)

while (WB) { Befehle }
Arbeitsweise: WB – Befehle – WB – Befehle – WB …

do-while-Schleife

Einrücken/Klammer
Analog Verzweigung

do { Befehle } while (WB)
Arbeitsweise: Befehle – WB – Befehle – WB – Befehle – WB …

for-Schleife (Verallgemeinerung der While-Schleife)
Übliche Arbeitsweise: Zähler durchläuft Werte von anfang bis ende

While-Schleife

while (WB) { Befehle }
Arbeitsweise: Befehle – WB – Befehle – WB – Befehle – WB …

while (Wiederholungsbedingung)
Anweisung1
Anweisung2//naechste Anweisung

Wirkung
(*) Werte Bedingung aus
falls true : Führe Anweisung1 aus
Mache weiter bei (*)
falls false: Führe Anweisung2 aus

Kopfgesteuerte Schleife

Modellierung einer while-Schleife

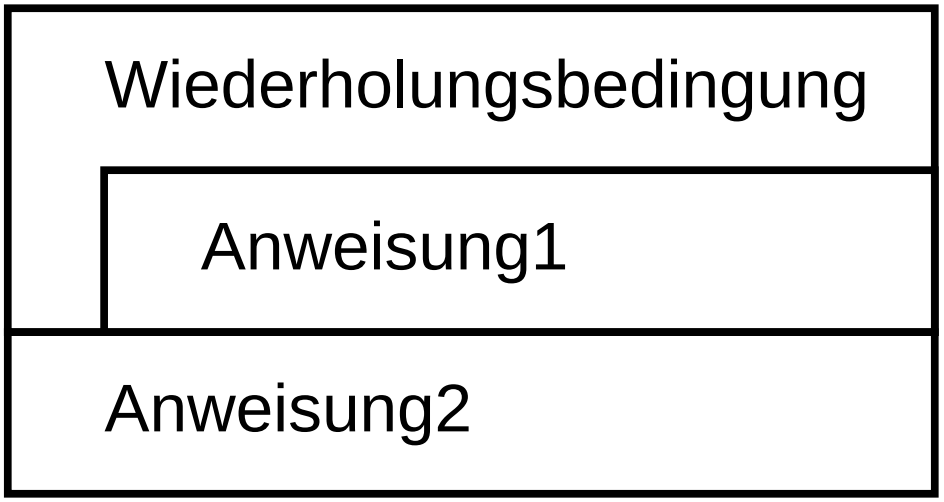
1) Identifizierung der **Wiederholungs**bedingung (WB) der Schleife
(ev. Negation einer Abbruchbedingung)

2) Identifizierung aller Befehle, die wiederholt werden wie z.B.:
nächste Daten lesen – neue Summe berechnen – zählen – ...

3) Ermittlung, ob die Schleife immer mindestens einmal durchlaufen wird
falls ja: *do while -> später*
andernfalls: while

4) Existenz von Befehlen unter 3) prüfen, welche WB ändern.
Ansonsten ist es eine unendliche Schleife.

Struktogramm/Allgemein



Beispiel

Musterprogramm

Aufgabe:
es sollen positive Zahlen eingelesen und deren Summe berechnet werden.
Bei Eingabe einer Zahl <=0 endet die Aufgabe.

Wiederholungsbedingung (WB) : eingelesene Zahl > 0

Befehle in der Schleife : Summe bestimmen, neue Zahl lesen

Mindestens einmal: nein, wenn die erste Zahl <= 0 ist -> while-Schleife

Änderung WB : ja da neue Zahl gelesen wird

Beispiel


```
double summe=0.0, zahl;
cin>>zahl;
while ( zahl>0 )
{
        summe+=zahl;
        cin>>zahl;
}
//summe ist berechnet
```

Übliche Vorgehensweise:

1. Lesen (genauer Leseversuch) dierekt vor der Schleife
2. WB: Kontrolle der Eingabe
3. AM Ende der Schleife: wieder Lesen

do while Schleife

do { Befehle } while (WB)
Arbeitsweise: Befehle – WB – Befehle – WB – Befehle – WB …

do
  Anweisung1
while (Wiederholungsbedingung)
Anweisung2

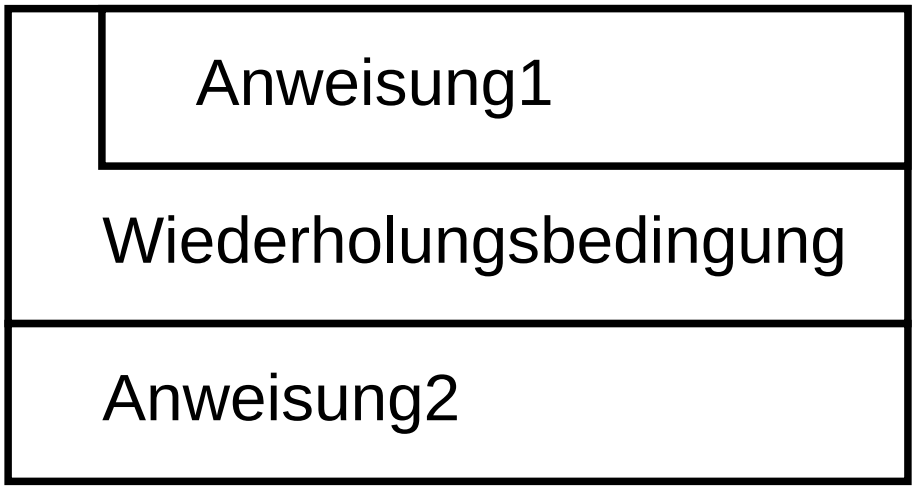
Wirkung
(*) Führe Anweisung1 aus
Werte WB (boolscher Ausdruck) aus
falls true : Mache weiter bei (*)
falls false: Führe Anweisung2 aus

In einer do while Schleife wird die Anweisung in der Schleife mindestens 1 mal ausgeführt !!!!

Fußgesteuerte Schleife

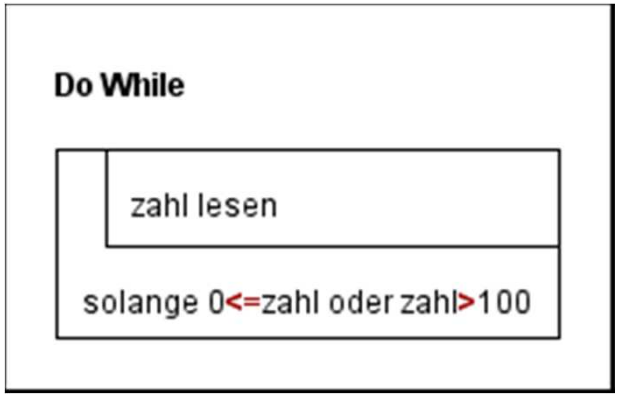
Modellierung analog while-Schleife

Struktogramm



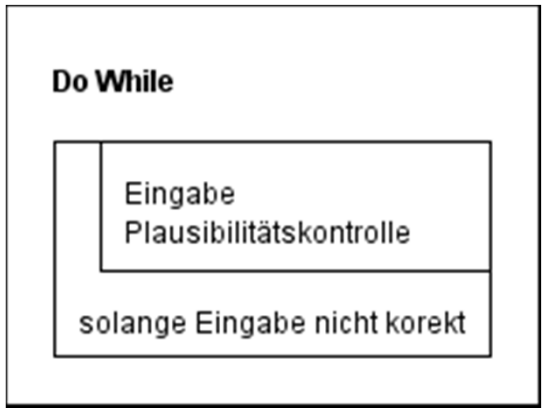
Beispiel: do while

Die do-while schleife ist selten.
Typisch hierfür sind allenfalls Eingaben mit Plausibilitätskontrolle wie z.B.
Es soll eine ganze Zahl größer 0 und kleiner gleich 100 eingelesen werden.




```
do
{
    cin>>zahl;
}
while ( 0<zahl || zahl>100 )
```

Allgemein



for-Schleife- ein Beispiel

Es sollen n ganze Zahlen eingelesen und aufsummiert werden. n ist ebenfalls einzulesen.

```
int n,summe=0,zahl;
cin>>n;
for ( int zaehler=1; zaehler<=n; zaehler++)
{
  cin>>zahl;
  summe+=zahl;
}
```

Einrücken/Klammer analog Verzweigung

Arbeitsweise

Initialisierung(1 mal) : zaehler wird deklariert und 1 gesetzt

* **WB** prüfen: ist zaehler<=n

Befehle: zahl lesen und aufsummieren

Reinitialisierung: zaehler erhöhen

weiter bei *

for-Schleife allgemein

```
for (Anweisung1;Wiederholungsbedingung;Anweisung3)
      Anweisung2
Anweisung4//naechste Anweisung
```

Wirkung

Führe Anweisung1(=**Initialisierung**) aus
(ev. mehrere durch Komma getrennt,
Deklarationen sind üblich)

Beispiel: int zaehler=1

(*) Werte die Wiederholungsbedingung aus
Falls true : Führe Anweisung2 (Schleifenbefehle) aus
Führe Anweisung3 (=**Reinitialisierung**) aus
(ev. mehrere durch Komma getrennt)
Weiter bei (*)

Beispiel: zaehler<=n

Beispiel: zaehler++

Falls false: Weiter mit dem ersten Befehl nach der Schleife
Anweisung4

Modellierung von for-Schleifen (Zählschleifen)

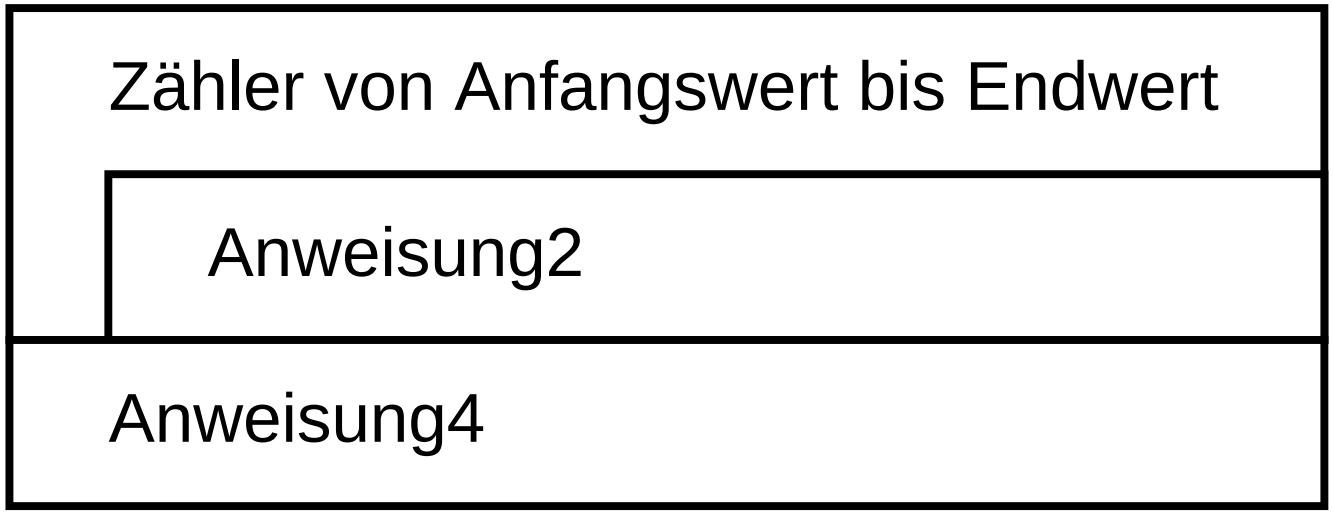
Prinzipielles Vorgehen bei einfachen Schleifen

1. Bestimmung des Zählers und seines Anfangswertes, der Anzahl der Wiederholungen bzw. des Wertes wo die Schleife endet

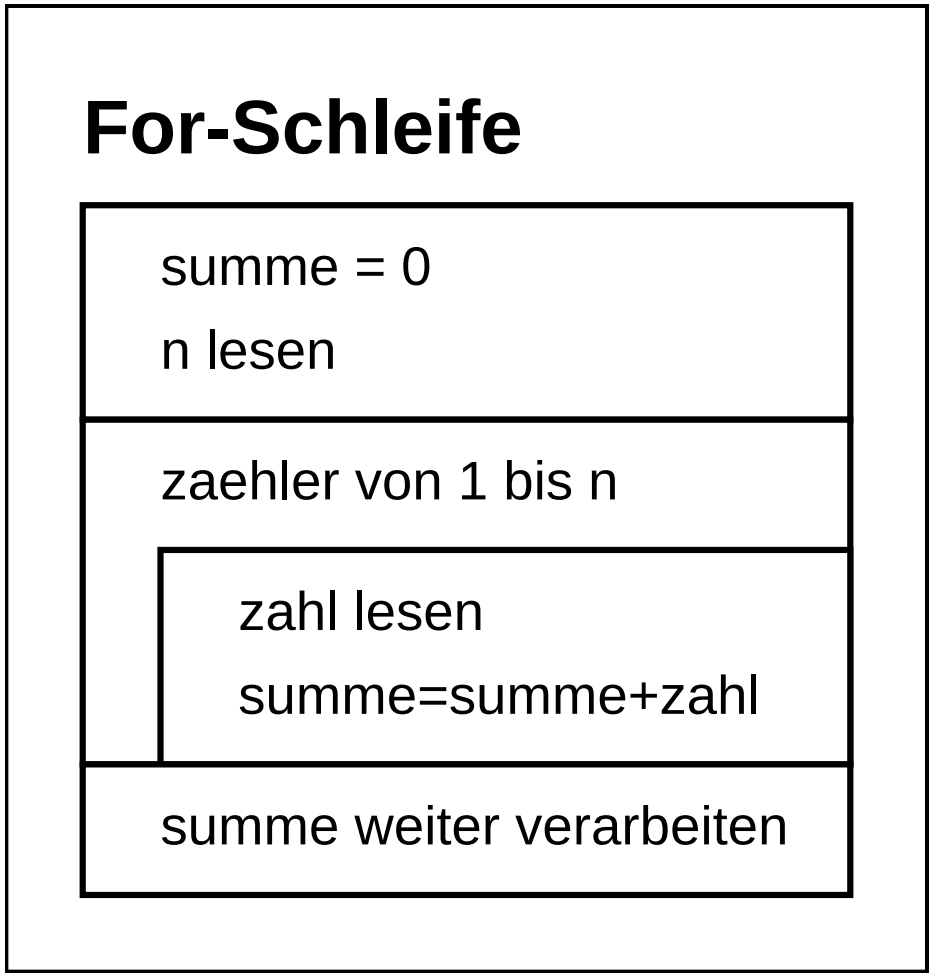
2) Initialisierung: Zähler= Anfangswert
3) Wiederholungsbedingung: Zähler < Endwert
4) Reinitialisierung: Zähler erhöhen
5) Identifizierung aller Befehle, die wiederholt werden wie z.B.: nächste Daten lesen – neue Summe berechnen ...

Die Möglichkeiten einer for-Schleife sind wesentlich weitreichender als hier dargestellt.

Struktogramm/vereinfacht



Struktogramm/Beispiel



For-Schleife: andere Schrittweite

Soll die Zählvariable beispielsweise alle geraden Zahlen durchlaufen, ändert man die Reinitialisierung

```
for (i=0 ;  i<n ; i=i+2)
{
    ....
}
```

For-Schleife/Rückwärtsschleife

Soll die Zählvariable ausgehend von einem hohem Anfangswert dann immer kleiner werden formuliert man beispielsweise wie folgt

```
for (k=m ;  k>0 ; k=k-1)
{
    ....
}
```

For-Schleife/mehrere Zähler

Beispiel für mehrere Initialisierungen bzw. Reinitialisierungen

```
for (i=1,k=100;  i<n && k > 0; i=i+1,k=k-1)
{
    ....
}
```

For-Schleife/ohne Zähler

Die for-Schleife ist ein sehr mächtiger Befehl, der prinzipiell jede Wiederholung modellieren kann. Dies ist dann oft sehr trickreich, jedoch aber oft auch sehr schwer lesbar/verständlich.

```
for (Datei oeffnen, hole erste Daten ; kein Dateiende ;  lese naechste Daten)
{
  verarbeite Daten
}
```

Initialisierung:	Datei öffnen, ersten Datensatz lesen (Versuch)
Wiederholungsbedingung:	Leseversuch geglückt, Daten waren da
Schleifenbefehle:	Daten verarbeiten
Reinitialisierung:	nächsten Datensatz lesen (Versuch)

3.6 Ein-/Ausgabe

Die Ein- und Ausgabe in C++ erfolgt üblicherweise mittels cin und cout. Dies sind jedoch keine Befehle sondern Objekte von Stream-Klassen.

Die Klassen als wesentlicher Bestandteil der Objektorientierung wurden jedoch wegen der Fülle des Stoffes für diese einsemestrige Vorlesung weggelassen.

Weiterhin werden die aus C stammenden Funktionen printf/scanf nicht betrachtet.

Ein-/Ausgabe mit cin/cout

Es können ein aber auch mehrere Daten mit einem Aufruf bearbeitet werden. Formatierungen sind mittel I/O-Manipulatoren möglich.

```
cin>>x;
cin>>a>>b>>c;
cout<<i;
cout<<"Ergebnis "<<i<<endl;
cout<<setw(10)<<setprecision(5)<<fixed<<d<<endl;//mit Manipulatoren
```

Weitere Ein-/Ausgabebefehle

put schreibt ein Zeichen (char)

cout.put(c1)

get liest das nächste Zeichen, welches auch ein Whitespace-Zeichen sein kann.

cin.get(c1)

read/write Lesen und Schreiben von character-arrays (später)

cin.read(c,10) bzw. cout.write(c,10) für jeweils 10 Zeichen

getline Lesen von Zeichen in eine Variable vom Typ String bis zum Auftreten eines Begrenzungszeichens.

getline(cin,name) Stringvariable name / Begrenzungszeichen newline

getlinie(cin,name,'#') Stringvariable name / Begrenzungszeichen #

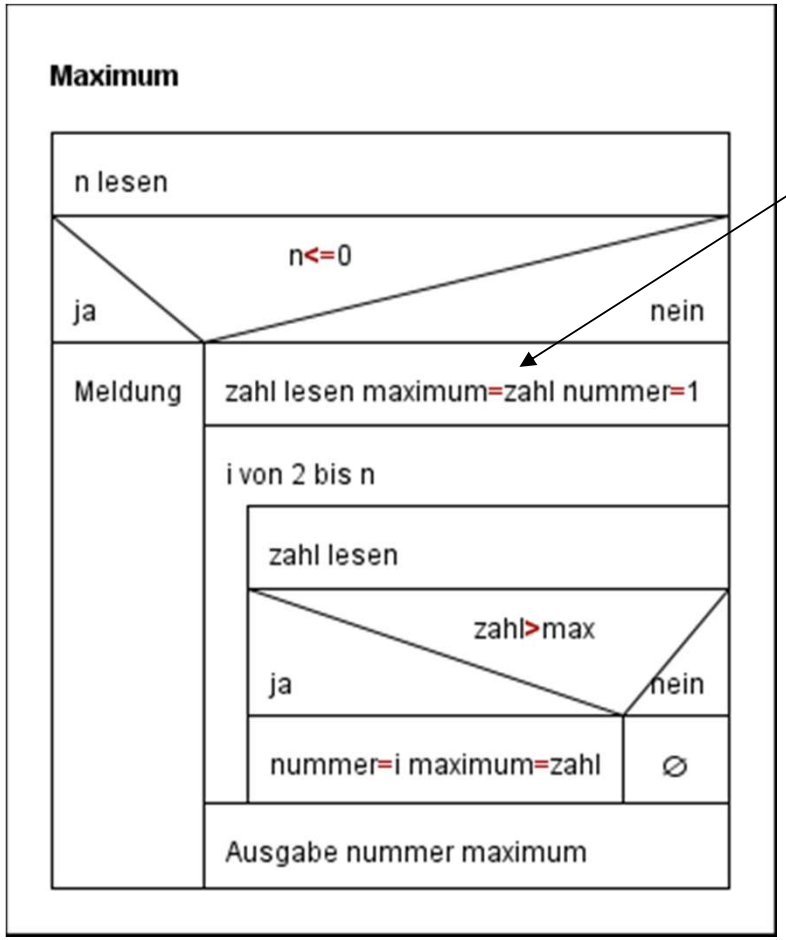
3.7 Sonstige Befehle

Abbruchbefehle

break	Verzweigung ans Ende eines Switch-Befehls oder einer Schleife. Weiter mit dem darauf folgenden Befehl.
continue	Beginne mit der nächsten Wiederholung einer Schleife while/do-while: Wiederholungsbedingung for: Reinitialisierung
return x-5	Beendigung der Funktion / Wert ist x-5
exit(n)	Beendigung des Programms :Fehlercode n (ganze Zahl)
goto marke	Verzweigung zu einer Marke / soll nicht verwendet werden „schlechter Programmierstil“

3.8 Beispiel for

Es sollen n Zahlen eingelesen und ihr Maximum berechnet werden. Zusätzlich ist die Nummer der eingelesenen Zahl zu bestimmen. n ist zu lesen.



Achtung: bei Maximum/Minimumsuche als Anfangswert immer das erste Element wählen

Testplan

Nr.	Fall	Eingabe n	Zahlen	Ergebnis Nummer	Max
1	Keine Zahl	0		Meldung	
2	Eine Zahl	1	5	1	5
3	Normal/Anfang	3	8 5 1	1	8
4	Normal/Mitte	3	5 8 1	2	8
5	Normal/Ende	3	5 1 8	3	8

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Beispiel for

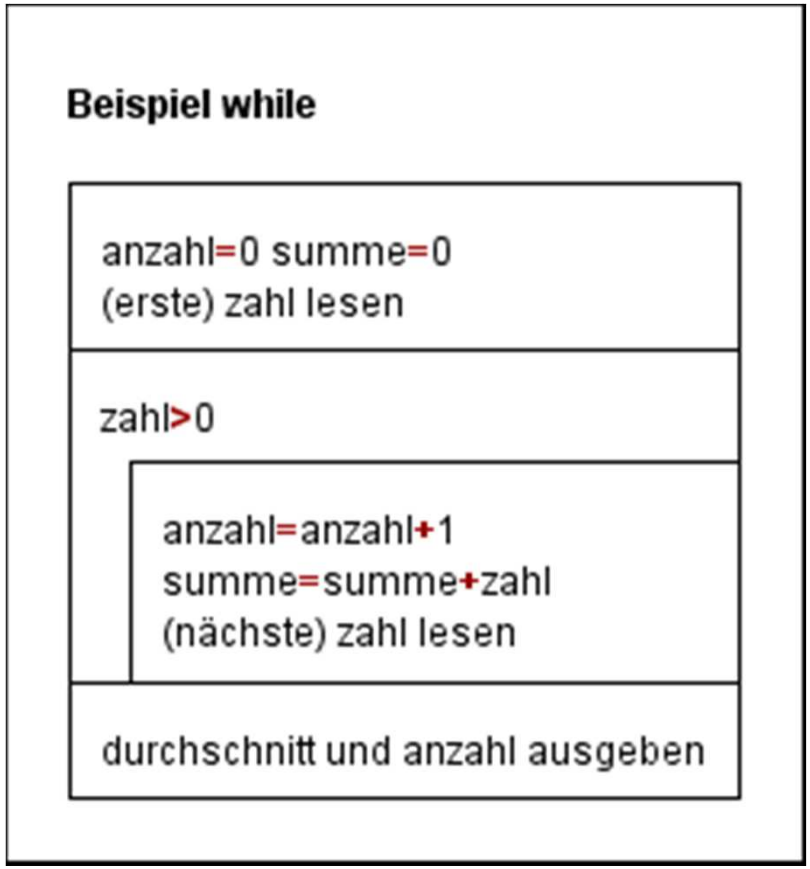
```
void main()
{
   int n;
   cout<<"n einlesen ";
   cin >>n;
   if ( n<=0 )
        cout<<" keine Zahlen gelesen "<<endl;
   else
   {  int zahl, nummer=1;
      cin>>zahl;
      int maximum=zahl;
      for ( int i=2 ; i<=n ; i++ ) // Zaehlvariable in der Schleife deklarieren
      {
            cin>>zahl;
            if ( zahl>maximum )
            {
                 maximum=zahl;
                 nummer=i;
            }
      }
      cout<<"Maximum "<<maximum<<" = "<<nummer<<". Zahl"<<endl;
   }
}
```

Initialisierung : i=2 (Zählvariable auf den Anfangswert setzen)
Wiederholungsbedingung: i<=n (Zählvariable abfragen)

Reinitialisierung: i=i+1 (Zählvariable hochzählen

3.9 Beispiele/while

In einem Programm sollen positive ganze Zahlen eingelesen und deren Durchschnitt berechnet werden. Das Programm endet, wenn eine negative Zahl oder 0 eingegeben wird. Danach sind Durchschnitt und Anzahl der gelesenen Zahlen am Bildschirm auszugeben.



Tipp:
Bei derartigen Aufgaben sollte man stets die erste Zahl vor der Schleife lesen, die anderen dann am Ende der Schleife. Die Wiederholungsbedingung überprüft die erfolgreiche Eingabe.

Beispiele/while

```
// Lesen von positiven Zahlen
#include <iostream>
using namespace std;
void main()
{
        int anzahl=0,eingabezahl;
        float summe=0.0;
// Erste Zahl eingeben
        cout << " Bitte positive Zahl eingeben " << endl;
        cout << " Abbruch mit 0 " << endl;
        cin >> eingabezahl;
//Schleife
        while (eingabezahl>0)
        {
                anzahl=anzahl+1;
                summ=summe+eingabezahl;
                cout << " Bitte naechste Zahl eingeben/Abbruch mit 0 " << endl;
                cin >> eingabezahl;
        }
//Ergebnis ausgeben
        cout << " Der Durchschnitt ist : "      << summe/anzahl << endl;
        cout << " Es wurden "<<anzahl<<" Zahlen gelesen"<<endl;
}
```

Welche Testfälle sollte man wählen?
Welche Schwächen hat dieses Programm?

Beispiele/while

Ein guter Testplan hätte die Schwächen des Programms aufgedeckt.
Der Fall n=0 sollte noch nach der Schleife abgefangen werden.

Testplan

Nr.	Fall	Eingabe Zahlen	Ergebnis Durchschnitt	Anzahl
1	Keine Zahl	0	?????????	0
2	Eine Zahl	2,2 0	2,2	1
3	Normalfall	1 0,3 2 0	1,1	3

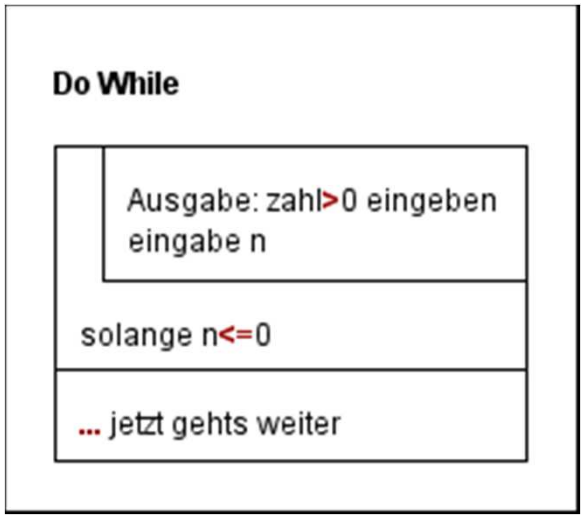
Beispiele/while

```
// Lesen von positiven Zahlen
#include <iostream>
using namespace std;
void main()
{
                int anzahl=0,eingabezahl;
                float summe=0.0;
// Erste Zahl eingeben
                cout << " Bitte positive Zahl eingeben " << endl;
                cout << " Abbruch mit 0 " << endl;
                cin >> eingabezahl;
//Schleife
                while (eingabezahl>0)
                {
                                anzahl=anzahl+1;
                                summ=summe+eingabezahl;
                                cout << " Bitte naechste Zahl eingeben/Abbruch mit 0 " << endl;
                                cin >> eingabezahl;
                }
//Ergebnis ausgeben
               if ( anzahl==0 )
                  cout<<" keine Zahlen gelesen"<<endl;
                else
                {
                  cout << " Der Durchschnitt ist : "          << summe/anzahl << endl;
                  cout << " Es wurden "<<anzahl<<" Zahlen gelesen"<<endl;
                }
}
```

Durch die Berücksichtigung des Testfalls "keine Zahlen" (Testplan !) ergibt sich ein geänderter Programmcode.

3.10 Beispiel do while

Der Anwender soll eine Zahl n>0 eingeben.




```
void main()
{
  int n;
  do
  {
      cout<<endl;
      cout<<" Zahl > 0 eingeben ";
      cin>>n;
  }  while(n<=0)
  // jetzt geht's weiter
}
```

Nachteil:
Der Anwender bekommt keinen Hinweis, dass seine letzte Eingabe fehlerhaft war.

Aber: Do While Schleifen kommen selten vor.

3.11 Übung Schleife und switch

Es soll in einem Auswahlmenue der Anwender gefragt werden, welche Aktion er durchführen will bzw. ob er die Durchführung von Aktionen abbrechen will.

Als Aktionen sollen der Einfachheit die Aktionen

Ausgabe " Aktion A"

…

Ausgabe " Aktion F"

vorgegeben sein.

Erstellen Sie ein entsprechendes Programm

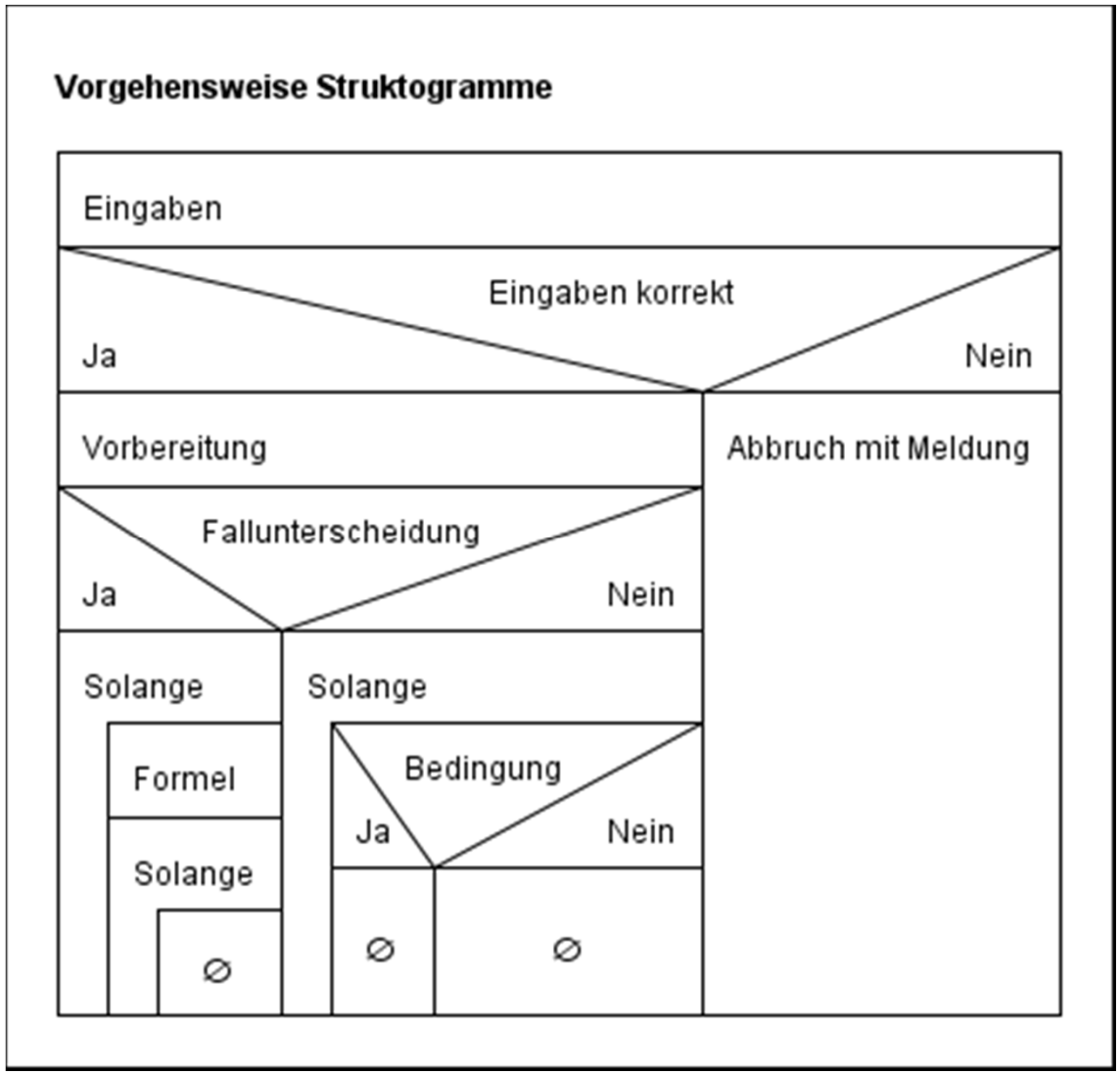
Übung Schleife und switch

Testplan

Nr.	Fall	Eingabe	Ergebnis
1	Wahl A	A	Aktion A: Ausgabe A
2	Wahl B	B	Aktion B: Ausgabe B
3	Wahl C	C	Aktion C: Ausgabe C
4	Wahl D	D	Aktion D: Ausgabe D
5	Wahl E	E	Aktion E: Ausgabe E
6	Wahl F	F	Aktion F: Ausgabe F
7	Ende	X	Programmende

Struktogramm und Programm zur Übung

3.12.Vorgehensweise Struktogramme



Von außen nach innen

TOP DOWN

Einzelne Rechtecke erst mit Stichwort zusammenfassend beschreiben und dann detaillieren