gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

1 Allgemeines

Die 4 Strukturelemente können in verschiedenen Notationen dargestellt werden. Die gebräuchlichsten sind:

- Struktogramm
- Programmablaufplan (PAP)
- Pseudo-Code

2 Struktogramm (STG)

2.1 Allgemeines

- grafische Darstellungsform für den Ablauf eines Programms
- ein STG wird immer von oben nach unten durchlaufen
- von Isaac Nassi und Ben Shneiderman entwickelt, daher auch als Nassi-Shneiderman-Diagramm bezeichnet
- nach DIN 66261 genormt
- ermöglicht optimale grafische Darstellung der Kontrollstrukturen, da keine Sprünge darstellbar sind. Es sind nur „lineare Kontrollstrukturen“ möglich.

2.2 Formale Richtlinien für die Erstellung

- ein Struktogramm hat immer eine rechteckige Form
- jedes Struktogramm hat einen Struktogramm-Kopf mit
  Titel,
  Übergabeparameter („Parameter:“),
  Rückgabewert („Rückgabe:“) und
  Kurzbeschreibung.
  Anmerkung: der "Titel" des Struktogramms muss eindeutig sein und darf nur aus genau einem Wort bestehen.
  Der Kopf steht am Anfang des Struktogramms
- das gesamte Struktogramm wird durch einen Rahmen umgeben.
- Eingaben z.B. von Tastatur werden mit „EINGABE :“ oder „E :“ gekennzeichnet
- Ausgaben z.B. auf den den Bildschirm werden mit „AUSGABE :“ oder „A:“ gekennzeichnet
- ein Struktogramm hat im Allgemeinen höchstens die Länge einer DIN A4-Seite
- die Rückgabe von Werten wird mit „RÜCKGABE:“ oder „R:“ gekennzeichnet
- PARAMETER und RÜCKGABE bilden zusammen mit dem Titel die „*Schnittstelle“* des Struktogramms. Die Schnittstelle kann 0 bis n Parameter und 0 oder 1 Rückgabe enthalten.

2.3 Inhaltliche Richtlinien

Struktogramme sind programmiersprachenunabhängig. Programmiersprachenspezifische Befehlssyntax soll nicht verwendet werden.

Jede Anweisung erhält einen eigenen Strukturblock (siehe 3.).

In Struktogrammen können ***Variablen*** verwendet werden. Eine Variable...

- dient zum **Speichern von Werten**, die ihr zugewiesen werden. Der Wert bleibt so lange erhalten, bis er gelöscht oder durch einen anderen Wert überschrieben wird. Texte werden in Anführungszeichen eingeschlossen, Zahlenkonstanten und Variablennamen nicht.
- braucht einen (eindeutigen) Namen. Er wird bei der ersten Verwendung implizit festgelegt. Eine explizite Deklaration, wie in einer Programmiersprache ist nicht notwendig. Der Name soll „sprechend“ sein.
- erhält ihren Wert durch Zuweisung. Sie wird durch den Zuweisungsoperator „Pfeil nach links“ (←) angezeigt. Das Ziel einer Zuweisung steht links vom Zuweisungsoperator, die Quelle, z.B. eine andere Variable oder ein Rechenausdruck, steht rechts.

Beispiel

Problem: Von Tastatur soll die Eingabe eines Namens angefordert werden. Der eingegebene Name soll anschließend zusammen mit einem Begrüßungstext am Bildschirm ausgegeben werden:

```
Titel:             Eingabe_Name
Parameter:         keine
Rückgabe:          keine
Kurzbeschreibung:  Eingabe eines Namens von Tastatur
                   und Ausgabe am Bildschirm
```

A: "Bitte einen Namen eingeben"
E: name
A: "Guten Tag " name

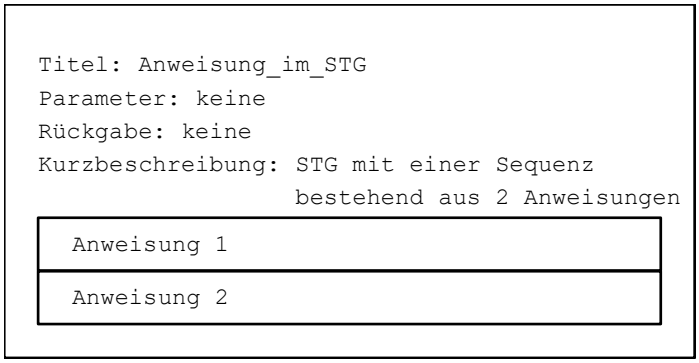
3 Kontrollstrukturen im Struktogramm

3.1 Anweisung oder Sequenz

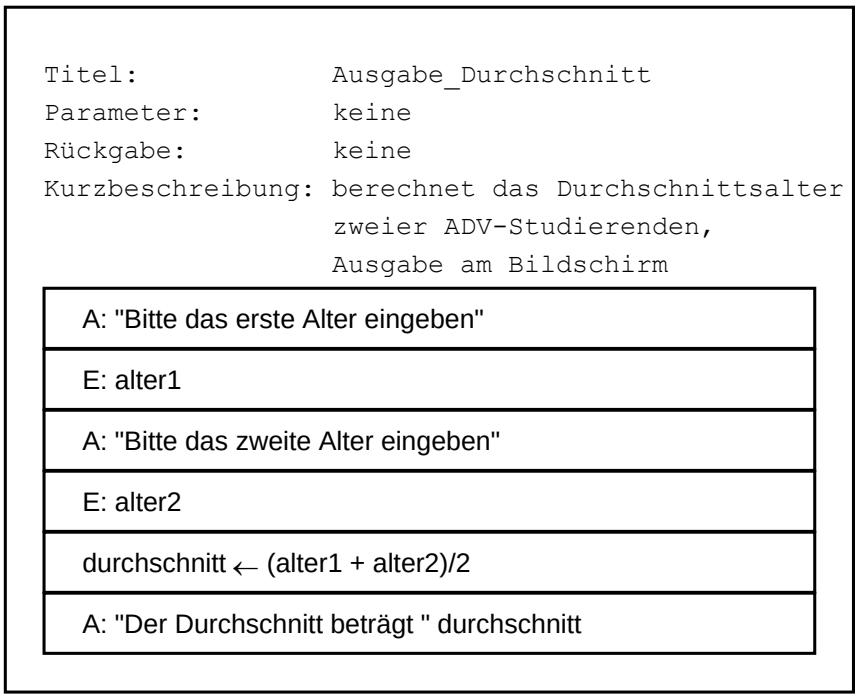
Eine *Anweisung* ist eine einzelne Vorschrift, die im Rahmen des Programms auszuführen ist und dessen Zustand (z.B. Werte von Variablen) verändert. Bei Ausführung einer Anweisung werden Daten oder Adressen verarbeitet.

Eine *Sequenz* ist eine Folge von Anweisungen.

Eine Anweisung wird im Struktogramm als Rechteck dargestellt:



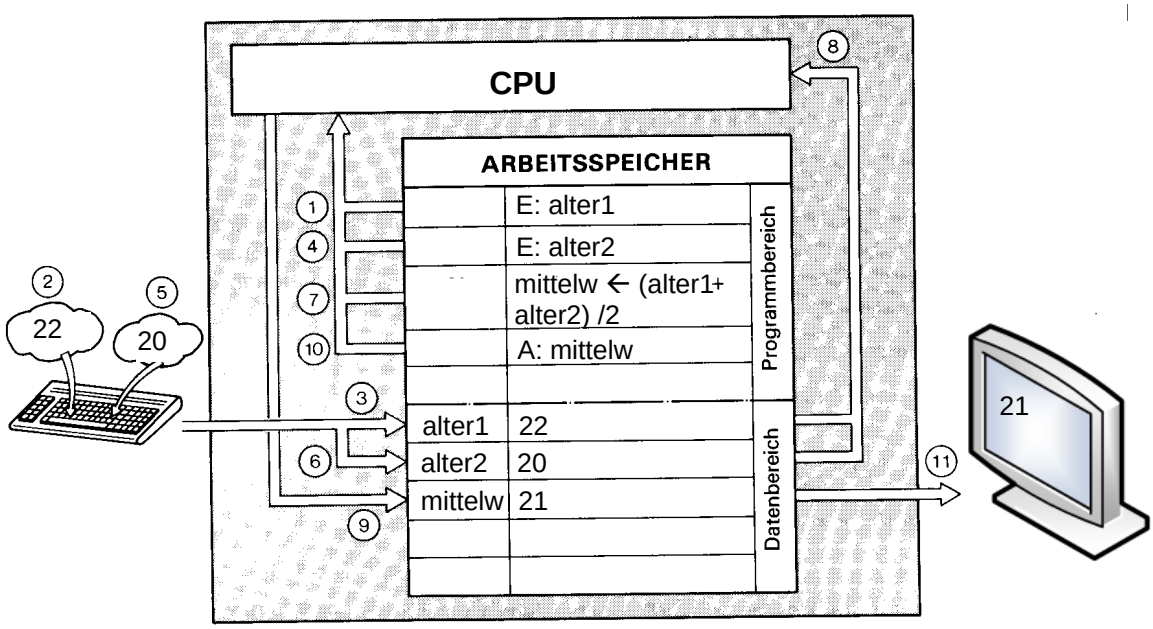
Beispiel: Das durchschnittliche Alter zweier ADV-Studierenden soll ausgegeben werden:



3.2 Einschub: Abarbeitung eines Programms im PC

Da die Arbeitsweise von Computern im Fach HSG zu diesem Zeitpunkt möglicherweise noch nicht besprochen wurde, kommt hier an dieser Stelle ein kurzer Einschub mit einem sehr vereinfachten Modell der Befehlsabarbeitung in einem Computer.

Damit ein Programm im Computer ablaufen kann, muss es zuerst von der Festplatte in den **Arbeitsspeicher** (RAM) geladen werden. Im **Programmbereich** des Arbeitsspeichers sind die Programmbefehle abgelegt und im **Datenbereich** werden die Eingabewerte, die Zwischen- und Endergebnisse des Programms abgelegt.



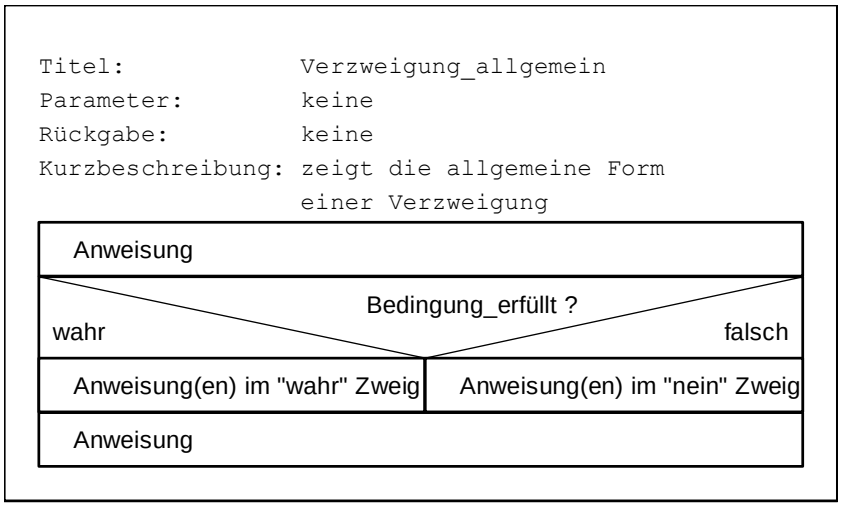
Folgende Schritte laufen (vereinfacht gesehen) ab, wenn das Programm gestartet wird:

1. Die CPU holt sich selbständig den ersten Befehl des Programms *(E: alter1)* aus dem Arbeitsspeicher und veranlasst, dass das Programm anhält, um Daten über die Tastatur zu empfangen.
2. Der Benutzer gibt über die Tastatur das Alter (z.B. 22) ein.
3. Die eingegebenen Daten werden unter der Adresse **alter1** (Variable) im Datenbereich abgespeichert.
4. Durch den nächsten Befehl *(E: alter2)* wartet der Rechner auf eine erneute Eingabe.
5. Der Benutzer gibt den zweiten Wert (z.B. 20) ein.
6. Diese Wert wird unter der Adresse **alter2** (Variable) im Datenbereich abgespeichert.
7. Die CPU liest den nächsten Befehl.
8. Die CPU holt die Variableninhalte von **alter1** und **alter2** und verknüpft beide nach der gegebenen Rechenformel.
9. Das Ergebnis dieser Rechenoperation wird an der Adresse **mittelw** im Speicher abgelegt.
10. Die CPU liest den nächsten Befehl aus dem Arbeitsspeicher *(A: mittelw)*.
11. Dieser Befehl bewirkt, dass der Variableninhalt der Adresse **mittelw** in eine Zeichenkette konvertiert wird und auf dem Bildschirm ausgegeben wird.

3.3 Auswahl oder Verzweigung

Anweisungen werden nur in Abhängigkeit von bestimmten ***Bedingungen*** ausgeführt. Eine Bedingung ist immer entweder „`wahr`“ oder „`falsch`“. Ist die Bedingung `wahr`, so wird der Wahr-Zweig ausgeführt, ist sie `falsch`, wird der Falsch-Zweig ausgeführt.

Darstellung der Verzweigung im Struktogramm



3.3.1 Formulierung von Bedingungen

Bedingungen haben die allgemeine Form

```
<Ausdruck> <Vergleichsoperator> <Ausdruck>
```

<Ausdruck>
ist ein mathematisch sinnvoller Term, der Zahlen, Variablen, mathematische Operatoren und Klammern enthalten kann. Auch Texte können als Ausdruck verwendet werden.

<Vergleichsoperatoren>
sind die in der Mathematik üblichen Symbole für Vergleiche:
`<, <=, >, >=, ==, !=`

Bedingungen können durch **logische Operatoren** miteinander verknüpft werden. Dadurch entstehen ***komplexe oder zusammengesetzte Bedingungen*** der allgemeinen Form

```
<Bedingung> <logischer_Operator> <Bedingung>
```

Logische Operatoren sind

- UND (beide Bedingungen müssen wahr sein)
- ODER (mindestens eine der beiden Bedingungen muss wahr sein) und
- NICHT (negiert den Wahrheitswert einer Bedingung)

Auch komplexe Bedingungen haben nur den Wahrheitswert `wahr` oder `falsch` als Ergebnis.

Es gelten die Wahrheitstabellen:

UND-Verknüpfung

B1	B2	E
f	f	f
f	w	f
w	f	f
w	w	w

ODER-Verknüpfung

B1	B2	E
f	f	f
f	w	w
w	f	w
w	w	w

NICHT-Operator

B	E
f	w
w	f

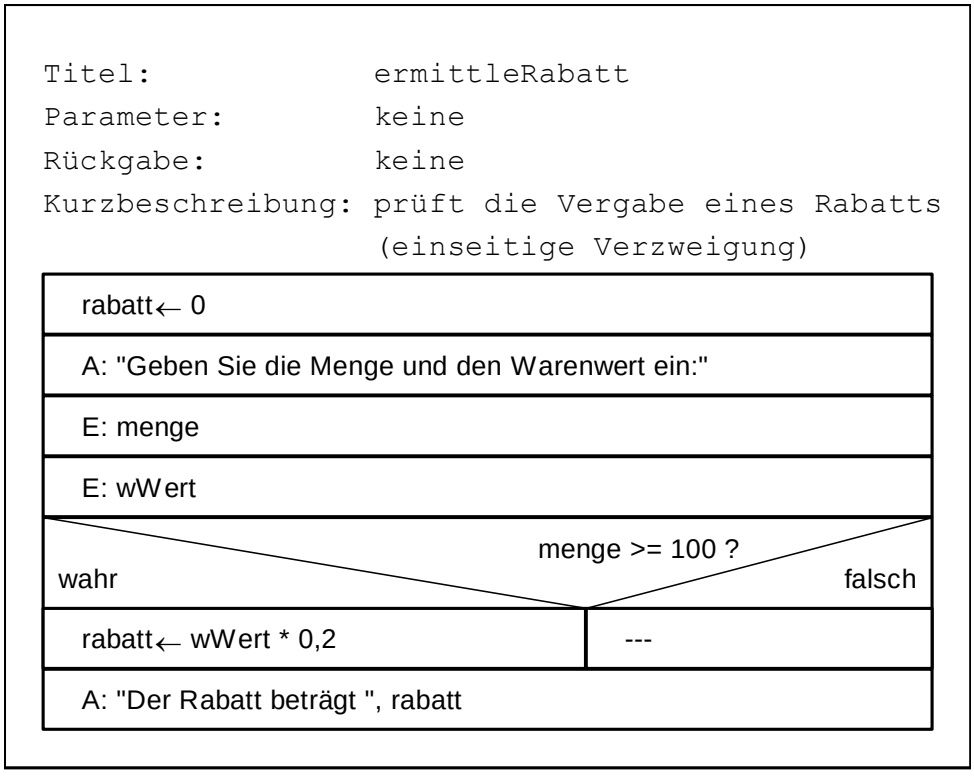
Eine Bedingung *kann* mit einem **Fragezeichen (?)** abgeschlossen werden, um deutlich zu machen, dass die „Frage“ (=Bedingung) nur mit wahr oder falsch beantwortet werden kann.

Man unterscheidet einseitige, zweiseitige und Mehrfach-Auswahl.

3.3.2 Einseitige Auswahl

Nur auf einer Seite befinden sich Anweisungen, die andere Seite (üblicherweise die rechte) bleibt leer.

Beispiel: Bei Abnahme einer Mindestmenge von 100 Stück soll ein Rabatt von 20 % gegeben werden.

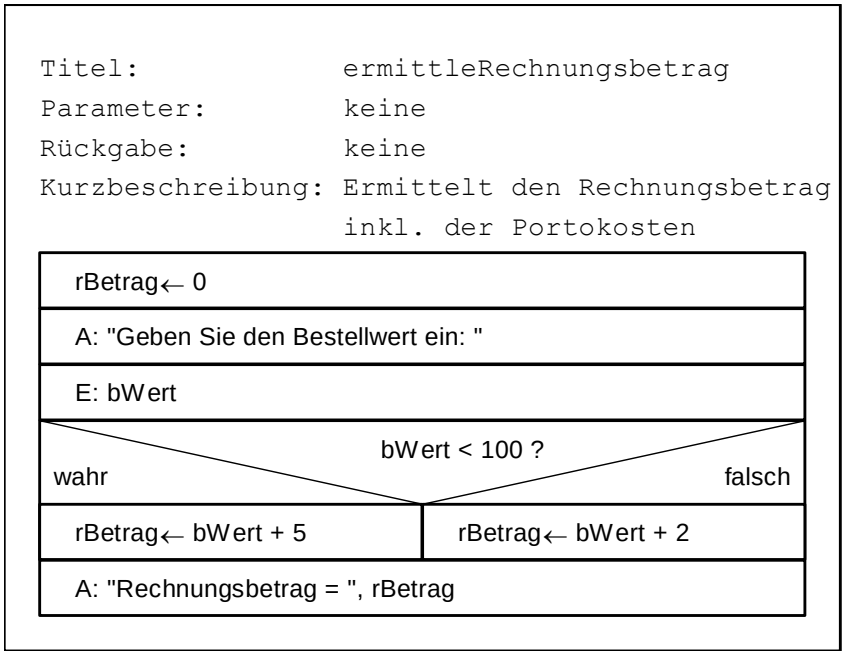


3.3.3 Zweiseitige Auswahl

Auf beiden Seiten befinden sich Anweisungen: sowohl im Wahr-Fall, als auch im Falsch-Fall sind Aktionen durchzuführen.

Beispiel:

Ein Versandhaus berechnet die Höhe des Portos und der Verpackung in Abhängigkeit vom Bestellwert: unter 100 € → 5 €, 100 € und mehr → 2 €:

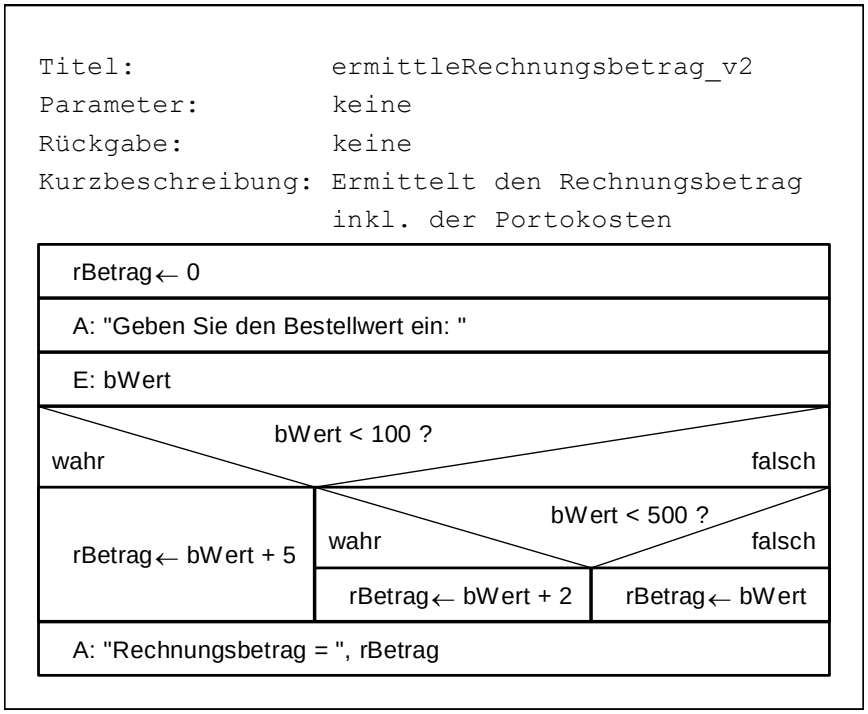


3.3.4 Verschachtelte Verzweigungen

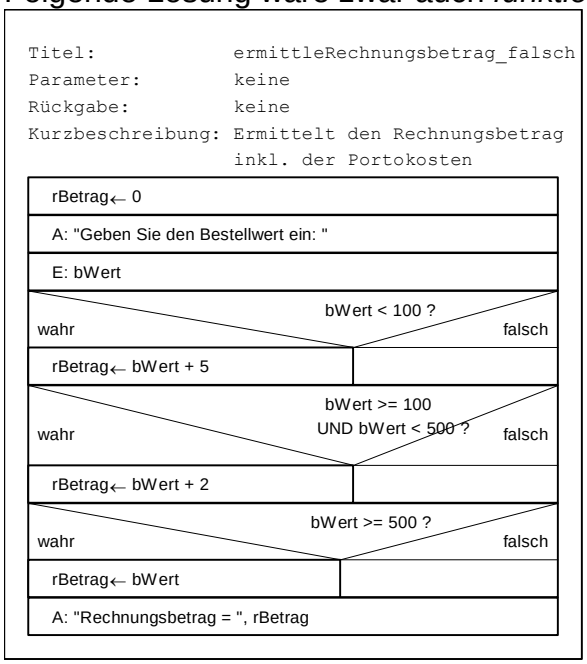
Sollen mehr als 2 Fälle unterschieden werden, so können Verzweigungen auch ineinander verschachtelt werden. Dabei wird im wahr- oder im falsch-Zweig eine weitere Verzweigung eingefügt.

Die Verschachtelungstiefe ist im Prinzip unbegrenzt. Aus Gründen der Übersichtlichkeit sollten jedoch nicht mehr als 4 Stufen geschachtelt werden. Sollten mehr Stufen erforderlich sein, so sollte nach Möglichkeit ein anderer Algorithmus eingesetzt werden.

Beispiel: wie oben, nur bei einem Bestellwert ab 500 € ist das Porto kostenlos:



Folgende Lösung wäre zwar auch *funktionsfähig*, *aber* trotzdem *schlecht*.



Begründung:

- Es wird nicht sofort deutlich, dass die 3 Fälle sich gegenseitig ausschließen.
- Die Bedingungen werden unnötig komplexer (siehe Fall 2).

3.3.5 Mehrfachauswahl

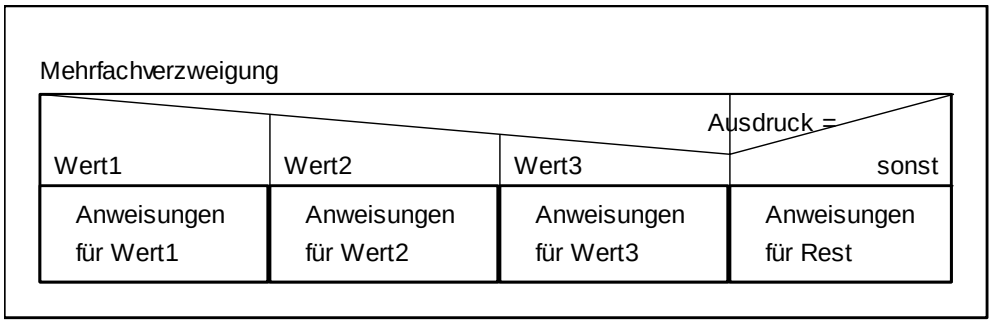
Der Wert eines Ausdrucks wird auf verschiedene Inhalte geprüft.

Werte**bereiche** sind **nicht möglich**, es muss auf einen dedizierten Wert abgeprüft werden.

So gesehen kann eine Mehrfachauswahl immer in eine geschachtelte Verzweigung umgewandelt werden, aber nicht umgekehrt.

Der zu prüfende Ausdruck wird daher nicht mit einem Fragezeichen abgeschlossen, sondern mit einem Gleichheitszeichen (=).

Es gibt immer eine „sonst“-Spalte, die die restlichen Werte abdeckt. Sie sollte nicht für einen der dedizierten Werte benutzt werden.



Anmerkung: manche Autoren erlauben auch die Angabe von Werte*bereichen*. In den meisten Programmiersprachen (wie z.B. Java) muss dies dann jedoch in eine geschachtelte Verzweigung umgewandelt werden, da sie dieses Konzept nicht unterstützen.

Beispiel: In Abhängigkeit von der Kundenkennung werden verschiedene Rabatte gewährt: Großhändler (G) erhalten 20%, Einzelhändler (E) 10% und Endverbraucher 0%.

```
Titel:            ermittleRechnungsbetrag_v3
Parameter:        keine
Rückgabe:         keine
Kurzbeschreibung: Ermittelt Rechnungsbetrag abzüglich eines Rabatts
                  in Abhängigkeit vom Kunden (Mehrfachverzweigung)
```

A: "Geben Sie den Warenwert und die Kundenkennung ein:"			
E: wWert			
E: kKennung			
kKennung =			
'G'	'E'	'V'	sonst
rBetrag ← 0,8 * wWert	rBetrag ← 0,9 * wWert	rBetrag ← wWert	A: "Nur 'G' | 'E' | 'V' zulässig"
A: rBetrag	A: rBetrag	A: rBetrag	

3.4 Qualitätssicherung: Korrektheitsprüfung des Algorithmus durch Schreibtischtest

Der Schreibtischtest ist eine von vielen Möglichkeiten der Software-Qualitätssicherung. Er dient zur Prüfung von Algorithmen auf Korrektheit. Er wird nicht elektronisch, sondern manuell ausgeführt.

Zunächst wird eine Eingabemenge und eine passende Ausgabemenge definiert. Mit dem oder den Werten der Eingabemenge wird dann manuell jede Anweisung des Algorithmus Schritt für Schritt durchgerechnet und die Ergebnisse schriftlich festgehalten. Als sinnvoll hat sich dabei die Tabellenform erwiesen: jede verwendete Variable und jede Ausgabe wird in einer eigenen Spalte festgehalten.

Beispiel: Schreibtischtest zum Struktogramm "Kundenkennung"

Kundenkennung	Warenwert	Rechnungsbetrag	Ausgabe
‘G‘	100	80	80
‘E‘	100	90	90
‘V‘	100	100	100
Alles außer ‘G‘|‘E‘|‘V‘			"Nur ‘G‘|‘E‘|‘V‘ zulässig"

Der Schreibtischtest liefert nur eine Aussage über die Korrektheit des Algorithmus, wenn

- sinnvolle Eingabewerte bekannt sind
- zu den Eingabewerten passende Ausgabewerte bekannt sind
- ein sogenannter deterministischer Algorithmus vorliegt, d.h. es treten nur definierte und reproduzierbare Zustände auf – unter gleichen Voraussetzungen reagiert der Algorithmus immer gleich
- ein terminierender Algorithmus vorliegt, d.h. er kommt in endlicher Zeit zu einem Ergebnis

3.5 Aufruf von Modulen (Unterprogrammen)

In einem Struktogramm wird ein anderes Struktogramm aufgerufen.

Damit werden komplexe Lösungen in kleinere, überschaubare Teillösungen zerlegt. Diese Teillösungen lassen sich dann in unterschiedlichem Zusammenhang immer wieder verwenden.

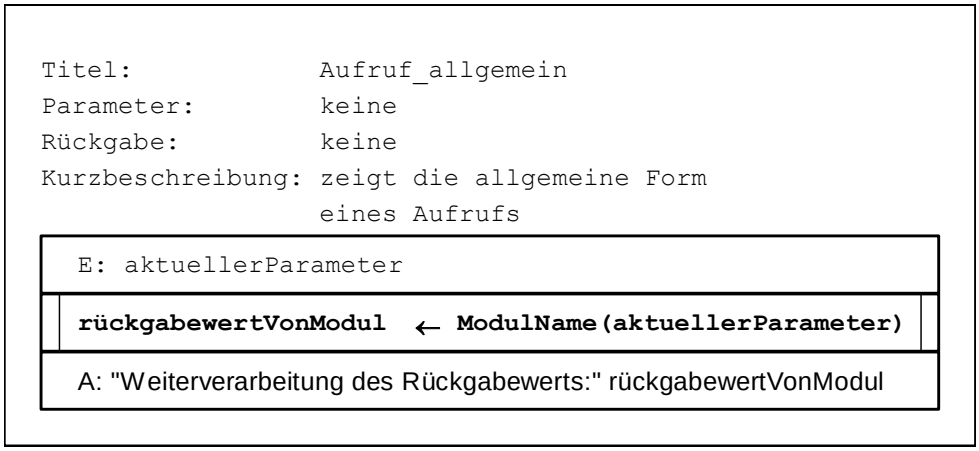
→ **Prinzip der Modularisierung**: Durch Modularisierung wird ein Programm in übersichtliche, funktionierende und möglicherweise wiederverwendbare Teile zerlegt, die über **festgelegte Schnittstellen** miteinander verknüpft werden.

Diese Schnittstelle besteht aus:

- dem **Namen** des Struktogramms (="Titel")
- (optionalen) **Parametern**
- einem (optionalen) **Rückgabewert**

Das Struktogramm des „aufgerufenen“ Moduls muss inhaltlich nicht bekannt sein, lediglich seine Schnittstelle und seine Funktion („Black Box“ Prinzip)

Darstellung des Aufrufs im Struktogramm:



Nach Durchlauf des aufgerufenen Moduls wird zu der aufrufenden Stelle zurückgesprungen, der Rückgabewert kann einer Variablen zugewiesen werden und dann die nächste Anweisung ausgeführt werden.

Anmerkungen zur Bezeichnung des aufgerufenen Moduls

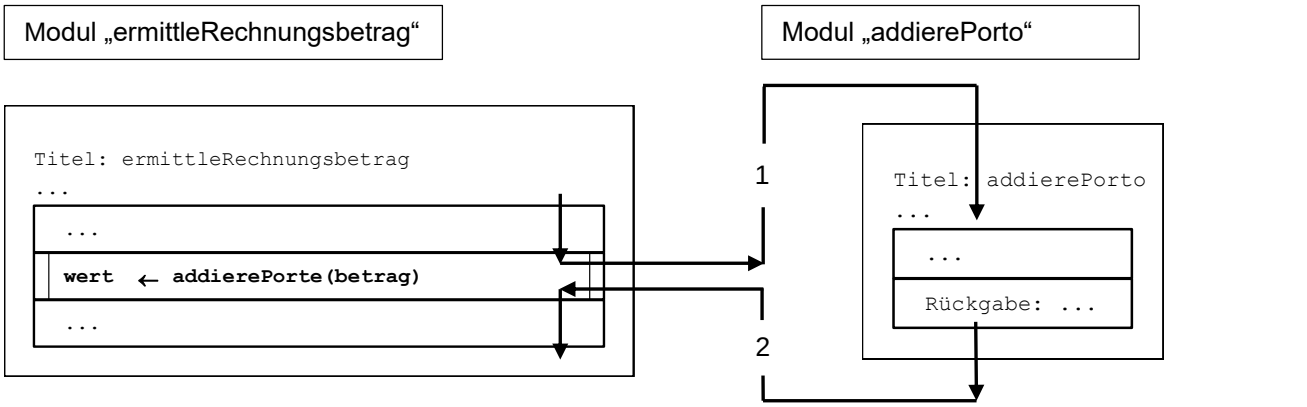
Es existieren unterschiedliche Bezeichnungen für das aufgerufene "Struktogramm". Die häufigsten sind *Unterprogramm, Funktion, Prozedur, Methode, Subroutine, Operation, Modul*. Diese Begriffe sind in ihrer Bedeutung nicht sauber zu trennen und z.T. historisch entstanden. In der objektorientierten Programmierung (wie bei Java) wird im Allgemeinen der Begriff Methode verwendet.

Aus Gründen des einheitlichen Sprachgebrauchs empfehle ich in diesem Zusammenhang – also in der Entwurfs-/Design-Phase - den Begriff "Modul".
Dies gilt jedoch nicht für die spätere Programmierung.

Beispiel: Portoberechnung

Der Benutzer ruft das Modul (das Struktogramm) "Rechnung" auf. Hier wird ein Bestellwert erfasst und anschließend an das Modul "berechnePorto" übergeben. Nach dessen Durchlauf wird der errechnete "Rechnungsbetrag" zurückgeliefert und der Variablen "Rechnungsbetrag" im Struktogramm "Rechnung" zugewiesen.

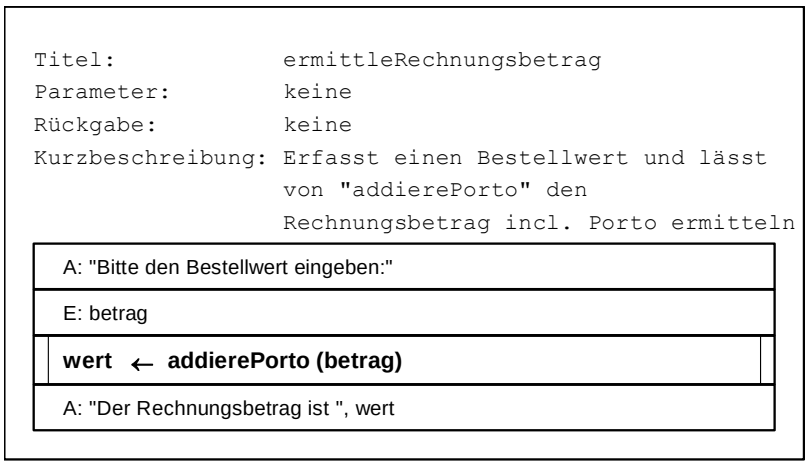
Schematischer Ablauf:

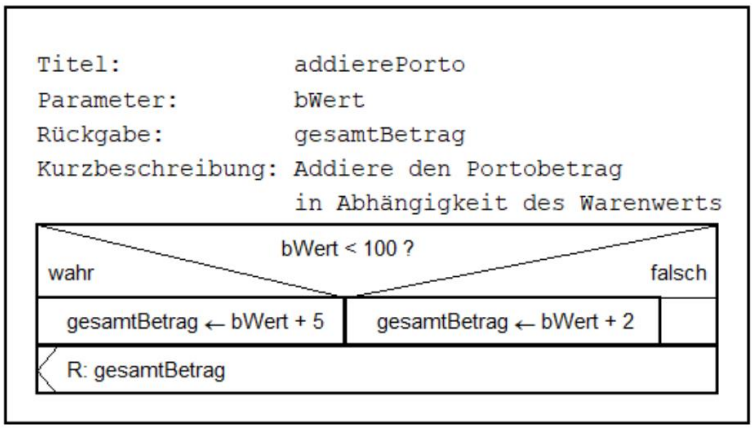


1 – Aufruf und Sprung zum Modul addierePorto mit Übergabe des aktuellen Parameters „betrag“

2 – Rücksprung zum Modul ermittleRechnungsbetrag mit Rückgabe des Gesamtbetrags

Die beiden Struktogramme:





Regeln für die Bildung eines Moduls:

- Gibt es eine sinnvolle (Teil-)aufgabe?
- Kann das Modul die (Teil-)Aufgabe vollständig lösen?
- Ist die Schnittstelle überschaubar?
  - Können oder müssen zur Lösung Parameter übergeben werden?
  - Ist die Anzahl der zu übergebenden Parameter "gering"?
  - Kann das Modul ein Ergebnis liefern?
  - Wird das Modul dadurch flexibler einsetzbar?
- Hat das Modul Wiederverwendungscharakter?
  - Der Wiederverwendungscharakter wird dadurch erhöht, dass ein Modul eine klar umrissene Teilaufgabe erledigt, ohne „nebenbei“ noch andere Funktionalität zu enthalten, die nicht zu der eigentlichen Aufgabe gehört.
    Bsp.: ein Modul, das die Wurzel einer Zahl berechnet, sollte nur das Ergebnis als Rückgabe zurückgeben und nicht vorher noch den Wert am Bildschirm anzeigen. Dies würde die flexible Anwendung stören.
- Wird die Gesamtaufgabe dadurch überschaubarer?

Kann mindestens eine dieser Fragen mit ja beantwortet werden, kann die Bildung eines Moduls sinnvoll sein.

Formale Parameter und aktuelle Parameter:

Die flexible Anwendung von Modulen wird unter anderem dadurch gewährleistet, dass die Namen der Parameter im Modul völlig unabhängig sind von den Namen, die im aufrufenden Modul verwendet werden.

„**Formale Parameter**“ werden die Parameter genannt, die im Kopf des Moduls (Struktogramms) definiert werden. Sie dienen als Variablen („Platzhalter“) für die Parameterwerte, die vom aufrufenden Modul übergeben werden. Diese konkreten Wert nennt man „**aktuelle Parameter**“.

3.6 Wiederholung oder Schleife

Eine oder mehrere Anweisungen sollen in Abhängigkeit von einer Bedingung wiederholt ausgeführt werden oder für eine angegebene Anzahl von Wiederholungen durchlaufen werden.

Grundsätzlich gilt:

- eine Schleife besteht aus dem Schleifenkörper und der Schleifenbedingung
- die Anweisungen des Schleifenkörpers werden ausgeführt, solange die Bedingung erfüllt ist
- die Schleife sollte nur an der Bedingung verlassen werden
- jede Schleife besitzt (mindestens) eine Schleifenvariable, die steuert, ob der Schleifenkörper wiederholt wird. Dazu wird die Schleifenvariable
    - vor der Schleife initialisiert
    - in der Schleifenbedingung als Kriterium verwendet
    - im Schleifenkörper verändert (sonst droht eine Endlosschleife)

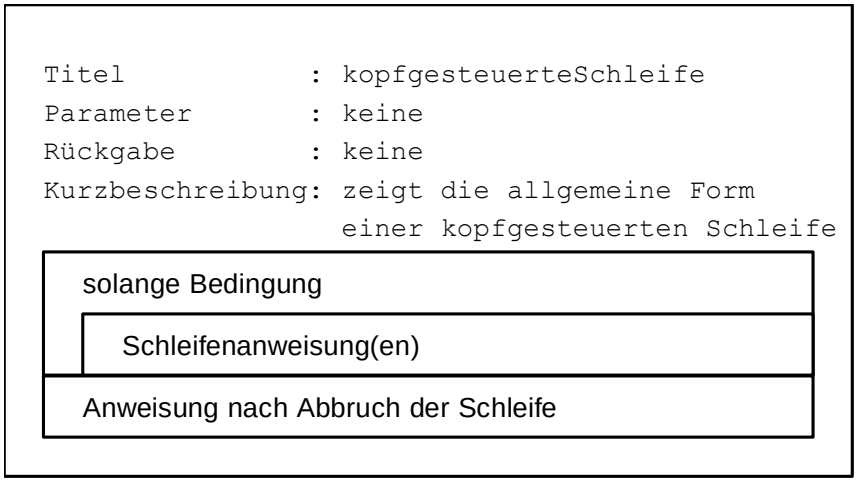
Es gibt 3 verschiedene Wiederholungsanweisungen (Schleifenarten):

- Wiederholung mit Bedingungsabfrage vor jedem Durchlauf (kopfgesteuerte Schleife)
- Wiederholung mit Bedingungsabfrage nach jedem Durchlauf (fußgesteuerte Schleife) Der Schleifenkörper wird also mindestens einmal ausgeführt.
- Wiederholung mit fester Abfragezahl (Zählschleife)

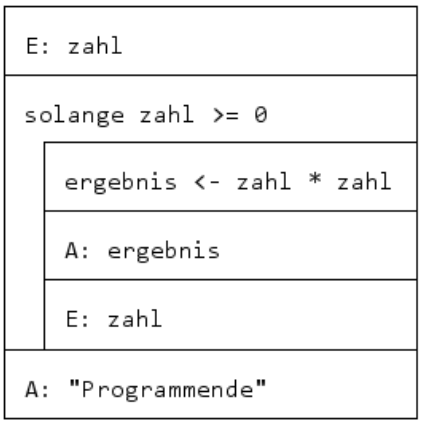
Darstellung der Wiederholung im Struktogramm:

3.6.1 Kopfgesteuert

Bei der kopfgesteuerten Schleife wird zuerst die Bedingung geprüft und wenn diese den boolschen Wert „wahr“ ergibt, anschließend der Schleifenkörper ausgeführt. Im „Falsch-Fall“ wird der Schleifenkörper übersprungen. Eine kopfgesteuerte Schleife kann also auch gar nicht betreten werden.
Der Text in der Bedingung beginnt mit "SOLANGE".



Beispiel: Zahl eingeben, quadrieren und ausgeben, Ende wenn zahl < 0:

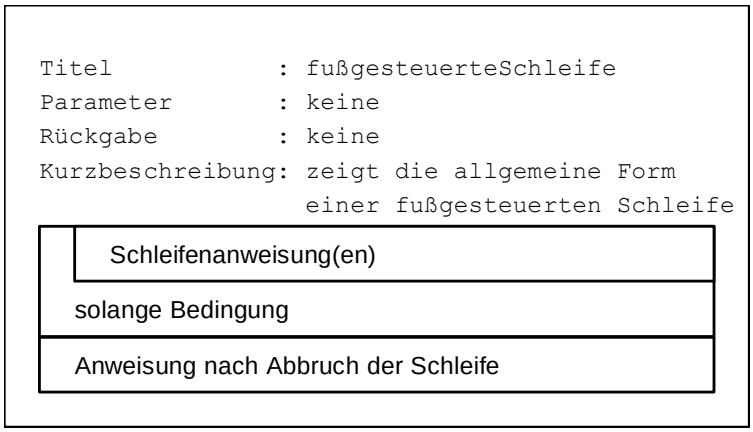


Bestandteile einer Schleife:

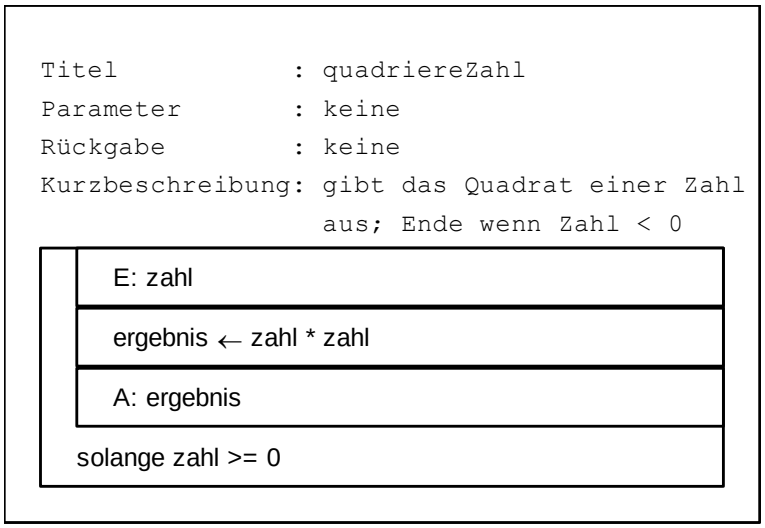
1. Initialisierung
2. Wiederholungsbedingung
3. Schleifenkörper
   (Variable der Bedingung muss geändert werden)

3.6.2 Fußgesteuert

Bei der fußgesteuerten Schleife wird zuerst der Schleifenkörper ausgeführt und anschließend die Bedingung geprüft. Eine fußgesteuerte Schleife wird also mindestens einmal betreten. Der Text in der Bedingung beginnt mit "SOLANGE".



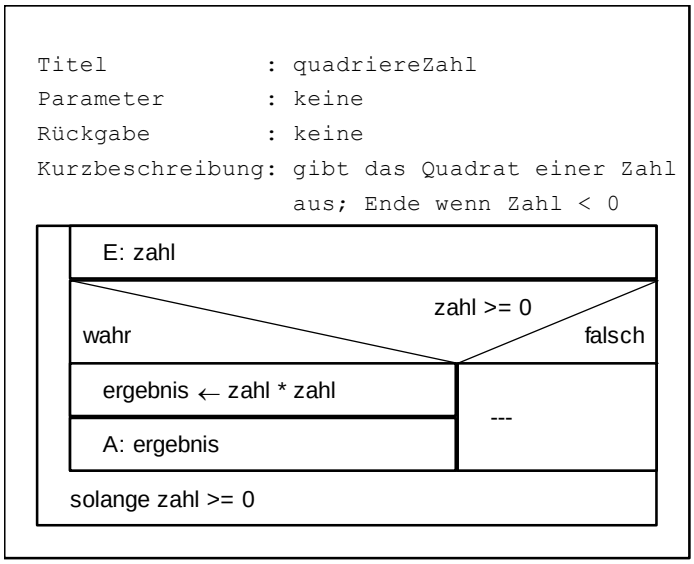
Beispiel: eingegebene Zahl quadrieren, Ergebnis ausgeben, Ende bei zahl < 0



In diesem Beispiel ist die Verwendung der fußgesteuerten Schleife ungünstig, da bei Ersteingabe von -1 das Programm trotzdem die Rechnung ausführt.

Schleifen, die eigentlich eine Abfrage in der Mitte benötigen

Um das vorherige Beispiel korrekt zu lösen, reicht eine reine fußgesteuerte Schleife nicht aus. Man benötigt in der Schleife noch zusätzlich eine Verzweigung.



Diese Art von Problemstellungen treten in der Praxis häufig auf. Sie können prinzipiell auf zwei Arten gelöst werden:

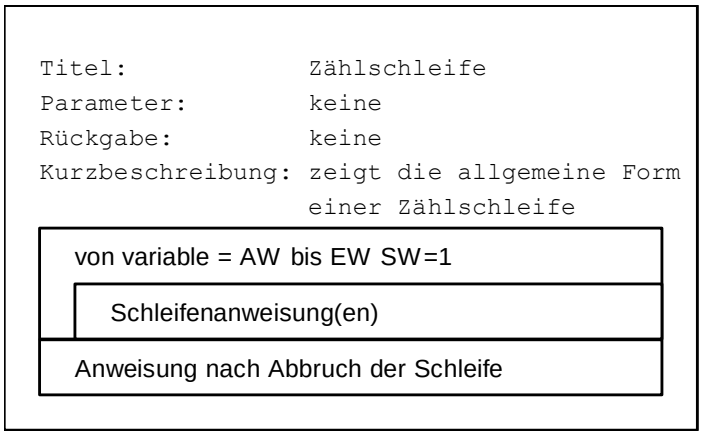
→ *mit einer kopfgesteuerten Schleife*, wobei hierbei eine Anweisung zweimal identisch benötigt wird (s.o. 1. Initialisierung; 3. Schleifenkörper: letzte Anweisung).

oder
→ *mit einer fußgesteuerten Schleife*, wobei hierbei zusätzlich eine Verzweigung benötigt wird (s. Abb. links).

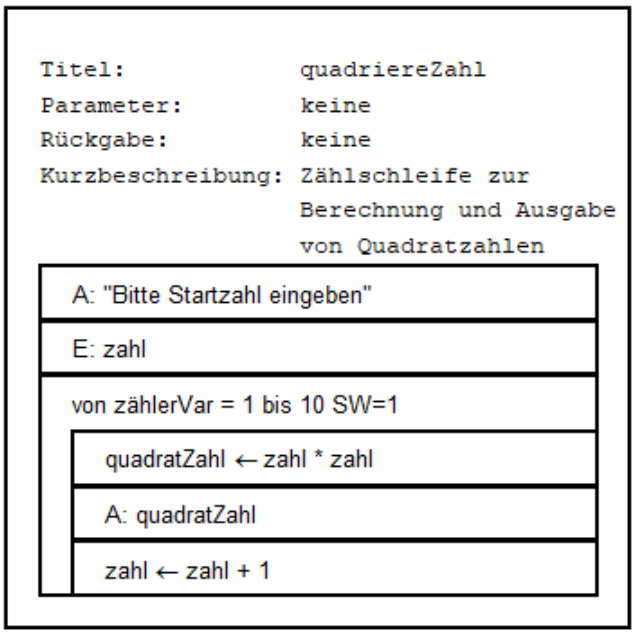
3.6.3 Zählschleife

Die Zählschleife kann verwendet werden, wenn die Anzahl der Schleifendurchläufe bekannt ist. Dementsprechend ist die Formulierung der Bedingung "VON ... BIS [SW...]". Fehlt die Angabe der Schrittweite SW wird standardmäßig SW=1 angenommen.
Bei der Zählschleife entfällt formal die "Veränderung der Schleifenvariablen" als eigene Anweisung. Sie ist implizit in VON...BIS enthalten.

Allgemeine Form



Beispiel:

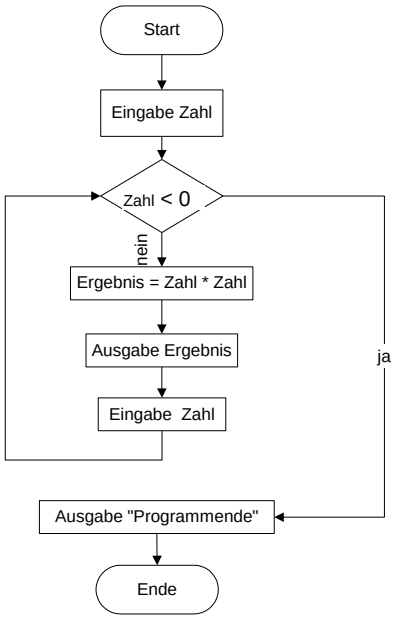


4 Programmablaufplan und Pseudo-Code

4.1 Programmablaufplan (PAP)

- benutzt grafische Symbole, die durch Pfeile miteinander verbunden werden
- genormt nach DIN 66001
- keine eigenen Symbole für grundlegende Kontrollstrukturen (Schleife, Mehrfachauswahl)
- Der Kontrollfluß ist zwar anschaulicher darstellbar, aber es können beliebige Abläufe konstruiert werden, auch solche, die zu einer *unstrukturierten* Programmierung führen (sogenannter „Spaghetti-Code“) →Aus diesem Grund sollten PAP nicht verwendet werden.

Das oben gezeigte Beispiel zum Quadrieren einer Zahl als PAP:



4.2 Pseudo-Code

- textuelle, semiformale Darstellungsform
- Kontrollstrukturen werden in Programmiersprachensyntax dargestellt (z.B. if, while, switch,...), die Anweisungen verbal oder in mehr oder weniger programmiersprachlicher Notation
- nicht genormt, die Syntax kann willkürlich gewählt werden

Das oben gezeigte Beispiel zum Quadrieren einer Zahl im Pseudo-Code:

```
Lies Zahl ein
WHILE Zahl >= 0
  Berechne Ergebnis aus Zahl * Zahl
  Gib Ergebnis aus
  Lies Zahl ein
END_WHILE
Gib Text "Programmende" aus
```