Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Grundlagen der Programmierung & Algorithmen und Datenstrukturen

Einführung in die strukturierte Programmierung – Teil 2

Die Inhalte der Vorlesung wurden primär auf Basis der angegebenen Literatur erstellt. Darüber hinaus orientiert sich diese an der Vorlesung von Prof. Dr.-Ing. Faustmann (ebenfalls HWR Berlin).

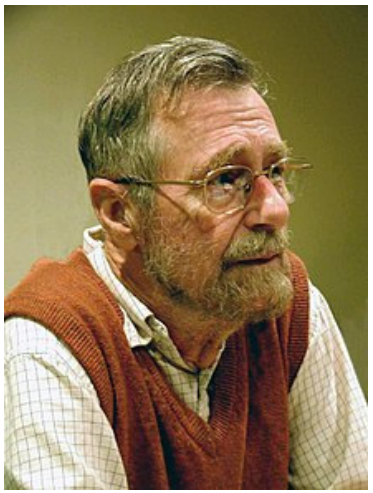
Inhalt des Parts

- Strukturierte Programmierung
- Kontrollstrukturen
- Darstellungsformen zur Strukturierung von Algorithmen
  - Programmablaufplan (Flussdiagramme)
  - Struktogramm (NASSI-SHNEIDERMA-Diagramm)
  - Pseudocode
- Entwurfsprinzipien & Entwurfsmethodiken

Strukturierte Programmierung

Strukturierte Programmierung

- Methodisches Vorgehen beim Entwurf von Programmen.
- Verzicht auf unbedingte Sprünge (GOTO-Problem).
- Ziele - Verbesserung qualitativer Eigenschaften.
  - Erhöhung der Lesbarkeit
  - Vereinfachung von Wartung und Pflege
  - Erhöhung der Zuverlässigkeit
- Modularisierung des Gesamtproblems (Arbeitsteilung).



Quelle der Abbildung: https://de.wikipedia.org/wiki/Edsger_W._Dijkstra

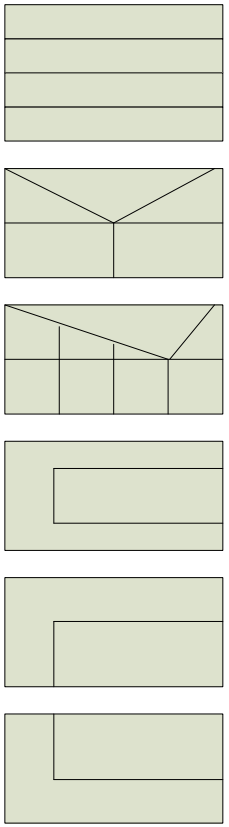
Strukturierte Programmierung

- Man unterscheidet einfache Anweisungen und Kontroll-strukturen, die die Ausführung von Anweisungen steuern.
- Jede Problemlösung lässt sich nach [Böhm/Jacopini 1966] durch folgende Kontrollstrukturen realisieren:
  - *Sequenz*
    - Ausführung mehrerer Anweisungen hintereinander
  - *Auswahl*
    - Bedingte Verzweigung
  - *Wiederholung*
    - Bedingte Wiederholung
  - *Aufruf anderer Algorithmen*
    - Algorithmus (Name und dessen Parameter)

Quelle: Böhm C., Jacopini G., FlowDiagrams, turing machines and languages with only two formation rules, in: Communications of the ACM, 9, 1966

Überblick Kontrollstrukturen

- Sequenz *(mehrere Anweisungen hintereinander ausführen)*
- Auswahl/Verzweigung *(bedingte Ausführung von Anweisungen)*
  - Einseitige Auswahl
  - Zweiseitige Auswahl
  - Mehrfachauswahl
- Wiederholung *(wiederholte Ausführung von Anweisungen)*
  - Zählergesteuerte Wiederholung
  - Kopfgesteuerte Wiederholung
  - Fußgesteuerte Wiederholung
- Aufruf *(eines anderen Algorithmus)*

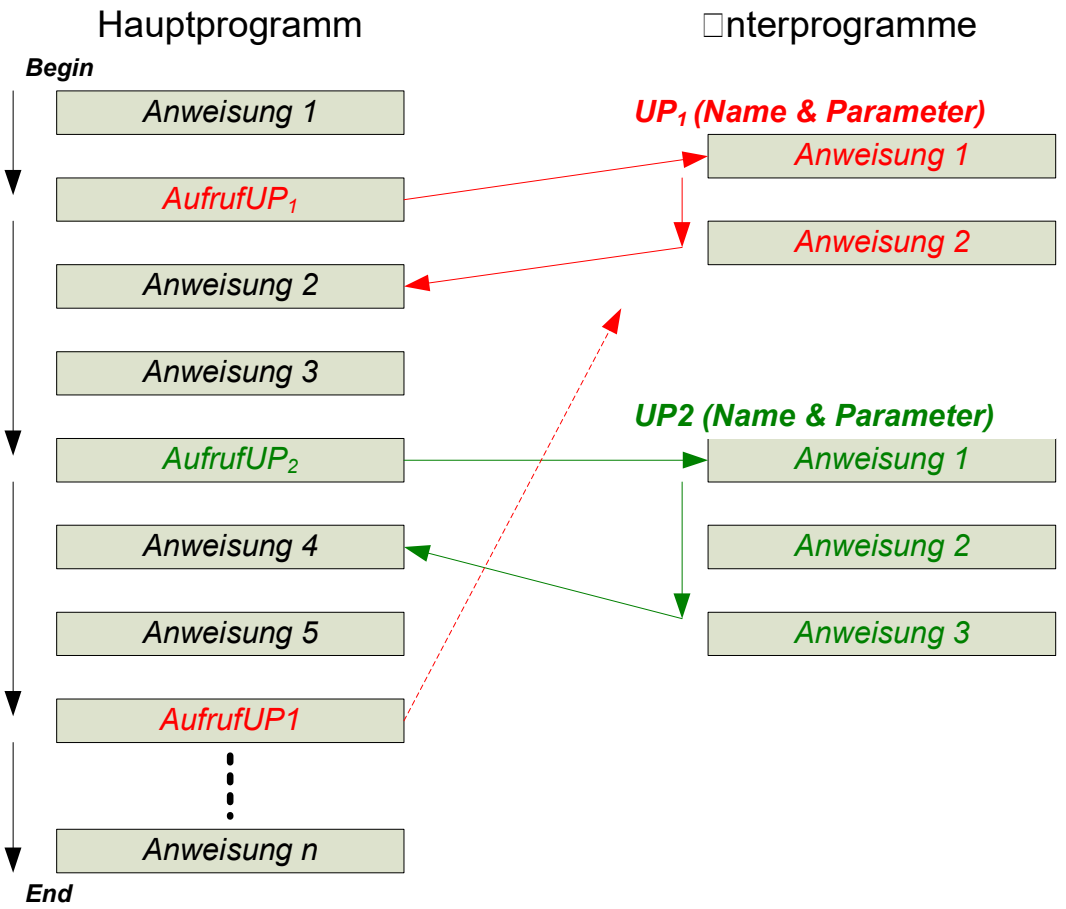


Strukturierte Programmierung

- Kontrollstrukturen im Kontext der strukturierten Programmierung besitzen genau einen Eingang und einen Ausgang.
- Zwischen dem Eingang und dem Ausgang gilt das Lokalitätsprinzip, d.h. der Kontrollfluss verlässt den durch Ein- und Ausgang definierten Kontrollflussbereich nicht.
- Aus makroskopischer Sicht verläuft der Kontrollfluss linear durch den Algorithmus, d.h. streng sequentiell vom Anfang bis zum Ende. Daher spricht man auch von linearen Kontrollstrukturen.
- Vorteile durch die Einhaltung der Lokalität und Linearität:
  - Erleichtert den Korrektheitsbeweis eines Algorithmus.
  - Kombination komplexerer Kontrollstrukturen durch „sequenzielle“ und „innere“ Verkettung erfolgen.

Quelle: Quelle: Balzert, H.: Lehrbuch der Softwaretechnik, Spektrum Akademischer Verlag

Strukturierte Programmierung



In Anlehnung an: http://gdp.wiinf.uni-wuerzburg.de/MMLLS/kurs_programmierung/1_17_GA_Strukturiert.html

Übung

- Recherchieren Sie nach Programmiersprachen die über eine GOTO-Anweisung (unbedingter Sprung) verfügen?
- Welche Nachteile sind mit der GOTO-Anweisung verbunden, existieren ggf. auch Vorteile?
- Durch welche Kontrollstrukturen kann der GOTO-Befehl (z.B. Überfügung in andere Programmiersprache) ersetzt werden?

→ Gruppenübung 15 min.

→ Gemeinsame Diskussion und Übungsblätter

PAP, Struktogramm und Pseudocode

Programmablaufplan

- PAP und Flussdiagramme (seit 1969 im Gebrauch)
- Nutzung grafischer Symbole die durch Linien verbunden sind
- Genormt entsprechend DIN 66001
- PAPs haben entscheidende Nachteile:
  - Sie besitzen keine eigenen Symbole für grundlegende Kontrollstrukturen wie Mehrfachauswahl und Wiederholungen (hier vorgestellte Symboliken sind spätere Erweiterungen!).
  - Sie bieten den Nutzern zu große Freiheiten (Anordnung, Verzweigung).
  - Schachtelstrukturen sind nur schwer erkennbar.
  - Ein Vereinbarungsteil kann nicht in die Symbolik integriert werden.
  - Große Programmsysteme sind kaum darstellbar (geringe Abstraktion)

Struktogramme

- Struktogramm-Notation (Vorschlag von Nassi, Shneiderman 1973)
- Genormt nach DIN 66261
- Struktogramme besitzen folgende Vorteile:
  - Optimale grafische Darstellung linearer Kontrollstrukturen
  - Keine Möglichkeit, Sprünge darzustellen
  - Ausreichend Platz für aussagekräftige Namen
  - Eigener Vereinbarungsteil
  - Darstellung der Auswahl in ablaufadäquater Form,
    - d.h. Alternativen werden vertikal angeordnet
    - dagegen textuelle Darstellungen horizontal

Pseudocode

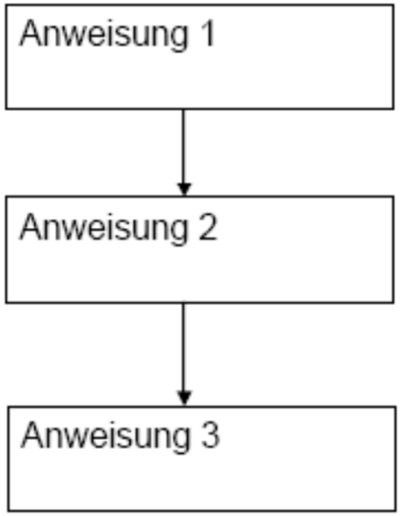
- Die Pseudocode-Notation ist eine semiformale, textuelle Beschreibung eines Programmablaufs.
- Pseudocode lehnt sich an Programmiersprachen an.
- Der Aufbau von Pseudocode ist nicht genormt.
- Aufbau der Notation
  - Für Kontrollstrukturen werden ähnliche Wortsymbole verwendet, wie diese auch in Programmiersprachen vorkommen (z.B. if then else).
  - Für Anweisungen werden verbale Formulierungen (z.B. erhöhe i um 1) oder Anweisungen ähnlich einer Programmiersprache (i := i + 1) genutzt.

Übung

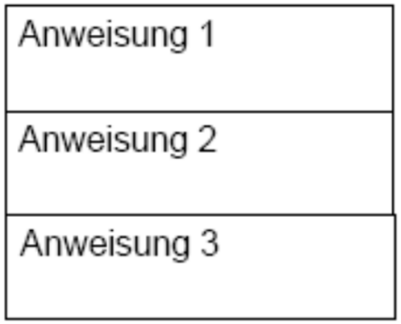
- Machen Sie sich mit der grafischen Notation entsprechend DIN 66261 bzw. EN 28631 vertraut, welche grundlegenden Elemente stehen Ihnen zur Visualisierung zur Verfügung?
- Warum erzwingt die Verwendung von Struktogrammen den Entwurf gut strukturierter Algorithmen?
- Recherchieren Sie nach potentiellen Werkzeugen die eine Erstellung von Struktogrammen unterstützen.
- Experimentieren Sie mit den Werkzeugen und den darin verfügbaren Struktogramm-Elementen bzw. bereitgestellten Beispielen.
  - → Gruppenübung 30 min.
  - → Diskussion und Übungsblätter

Sequenz

Programmablaufplan



Struktogramm

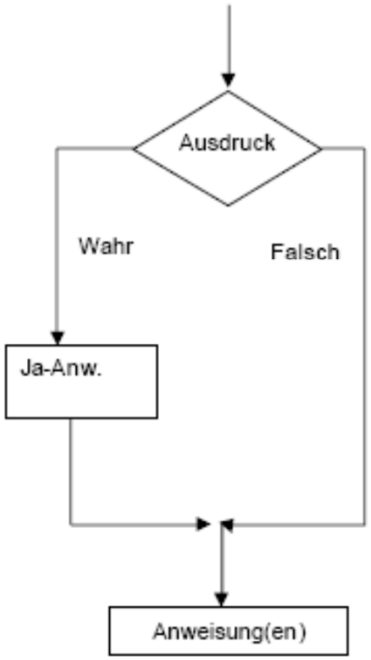


Pseudocode

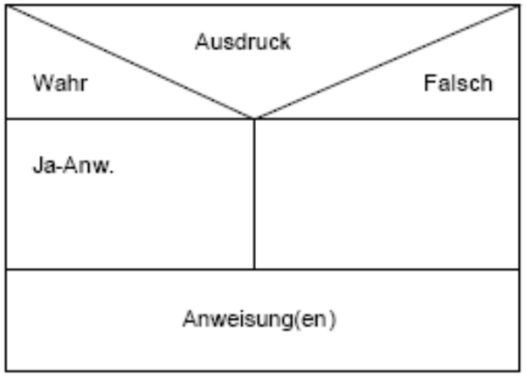
```
Anweisung 1;

Anweisung 2;

Anweisung 3;
```

Einseitige Auswahl

Programmablaufplan



Struktogramm

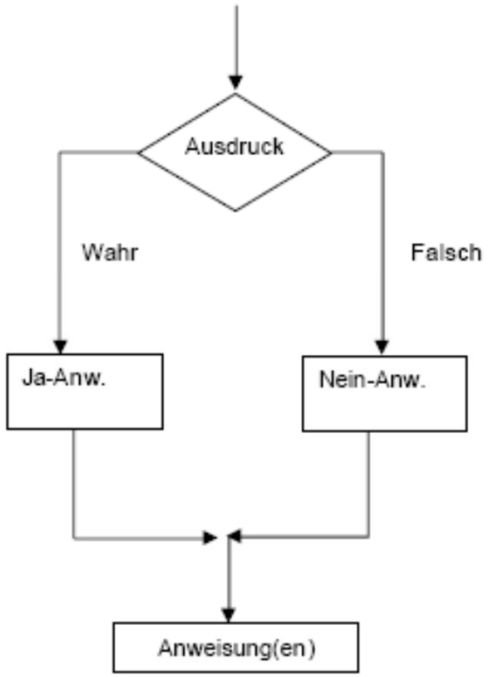


Pseudocode

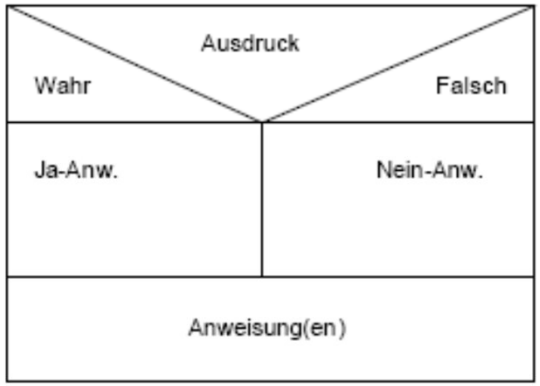
```
if Ausdruck
    then Ja-Anweisung(en)
end if
Anweisung(en)
```

Zweiseitige Auswahl

Programmablaufplan



Struktogramm

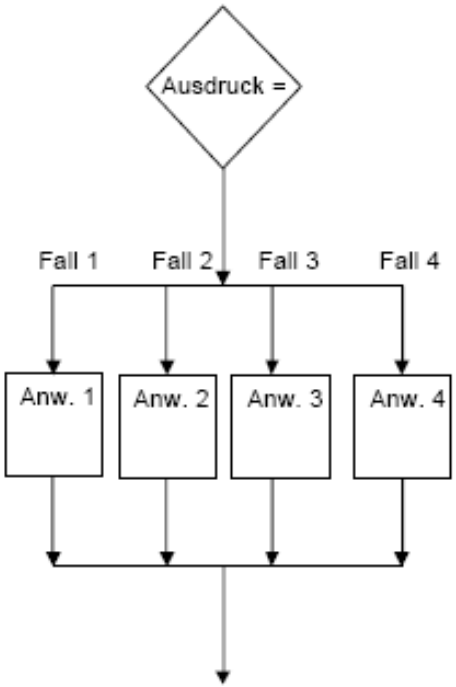


Pseudocode

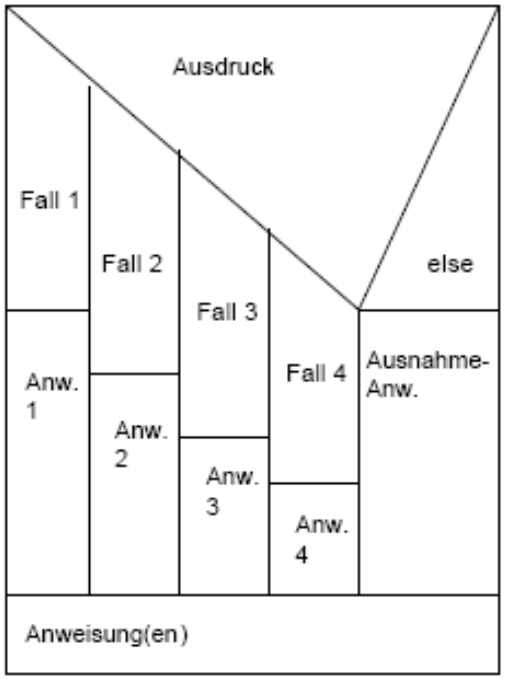
```
if Ausdruck
    then Ja-Anweisung(en)
    else Nein-Anweisung(en)
end if
Anweisung(en)
```

Mehrfachauswahl

Programmablaufplan



Struktogramm



Pseudocode

case Ausdruck **is**

Fall1: Anw. 1

Fall 2: Anw. 2

Fall 3: Anw. 3

Fall 4: Anw. 4

else Anweisung(en)

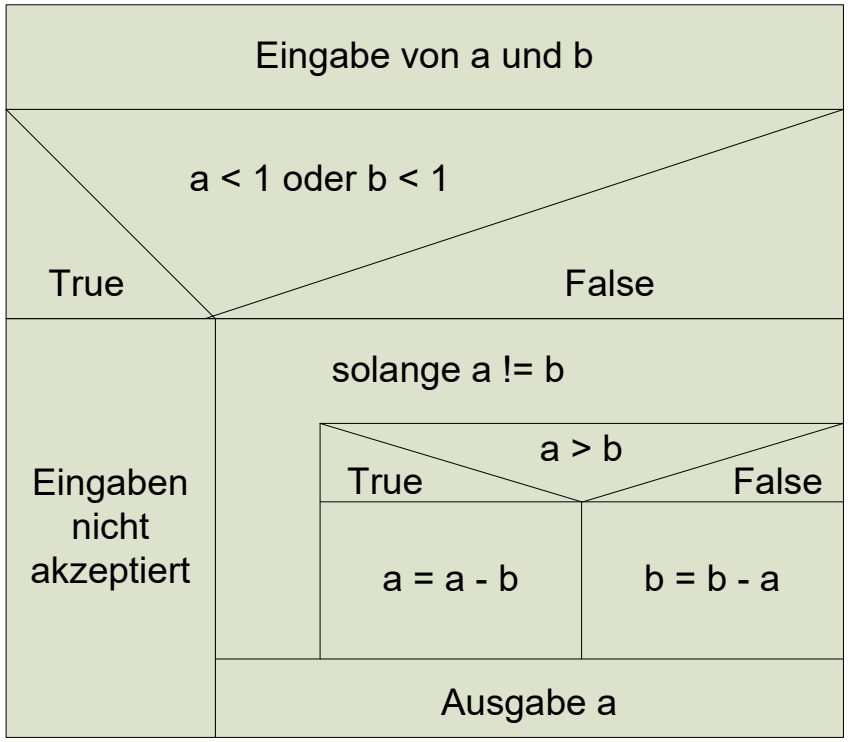
end case

Anweisung(en)

Übung

- Identifizierens Sie die verwendeten Strukturblöcke?
- Überlegen Sie sich allgemeine Testfälle um alle Anweisungen mindestens einmal zu durchlaufen?
- Prüfen Sie ihre Testfälle durch Wertbelegungen für a und b?
- Fachliche Aufgabenstellung des Algorithmus?

  → 20 min. Gruppenübung



Übung

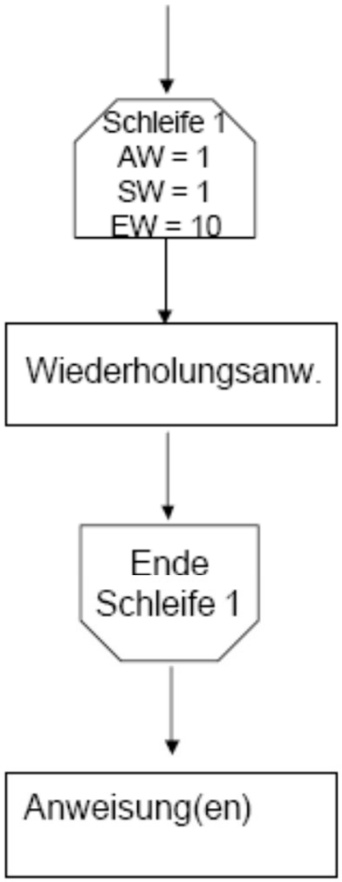
- Erstellen Sie ein Struktogramm und Pseudocode zur Lösung der folgenden Problemstellung:
- Ein Fahrscheinautomat gibt Einzel-, Wochen- und Monatskarten für die Tarifzonen 1 – Kurzstrecke und 2 – Normal aus.
- Der Grundpreis für Tarifzone 1 beträgt 2 Euro, für Tarifzone 2 2,80 Euro.
- Die Preise für Wochen- und Monatskarte ergeben sich aus dem Grundpreis multipliziert mit 6 bzw. 25.
- Der Benutzer gibt jeweils eine Nummer für die Tarifzone und für die Art der Zeitkarte (Einzel = 1, Woche = 2, Monat = 3) ein. Bei Fehleingaben wird eine Fehlermeldung ausgegeben.
  - → Gruppenübung 30 min.
  - → Diskussion und Übungsblätter

Konzept der Wiederholung

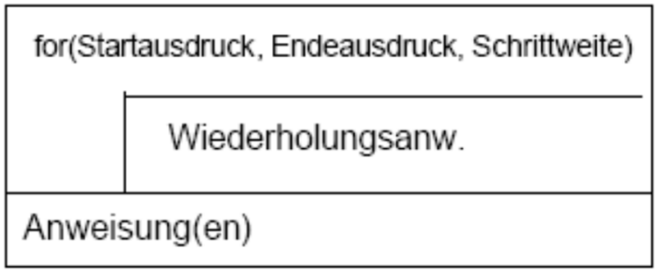
- Wiederholungen bzw. Schleifen oder auch Iterationen:
  - Wiederholungen in Abhängigkeit einer Bedingung
  - Wiederholungen entsprechend einer gegebenen Anzahl
- Wiederholungskonstrukte werden unterschieden:
  - Wiederholung mit Abfrage vor jedem Wiederholungsdurchlauf
  - Wiederholung mit Abfrage nach jedem Wiederholungsdurchlauf
  - Wiederholung mit fester Wiederholungsanzahl (Zählschleife)
- Eine Wiederholung mit Abfrage nach jedem Durchlauf lässt sich auf eine Wiederholung mit Abfrage vor jedem Durchlauf zurückführen, wenn die Schleife so initialisiert wird, dass die Bedingung am Anfang erfüllt ist.

Zählergesteuerte Wiederholung

Programmablaufplan



Struktogramm

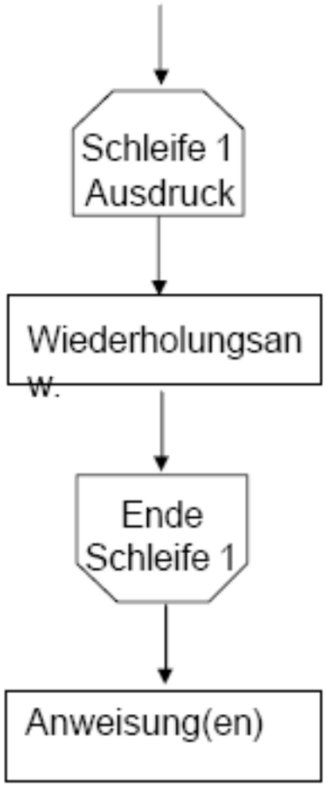


Pseudocode

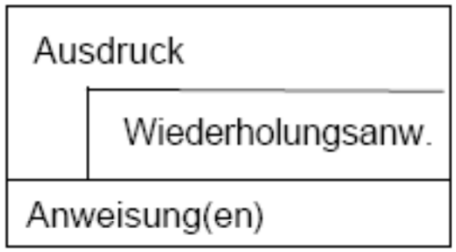
```
for (Startausdruck, Endeausdruck, Schrittweite)
do
    Wiederholungsanw.
end do
Anweisung(en)
```

Kopfgesteuerte Wiederholung

Programmablaufplan



Struktogramm

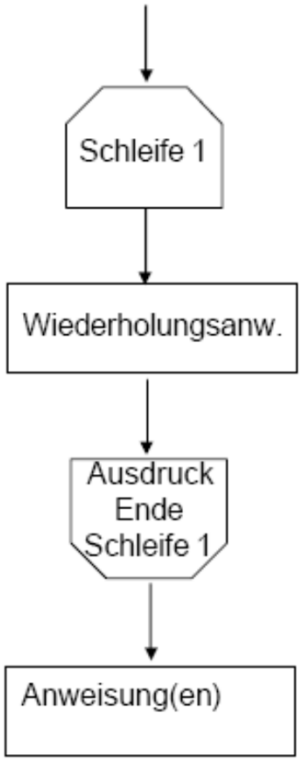


Pseudocode

```
while Ausdruck
do
    Wiederholungsanw.
end do
Anweisung(en)
```

Fußgesteuerte Wiederholung

Programmablaufplan



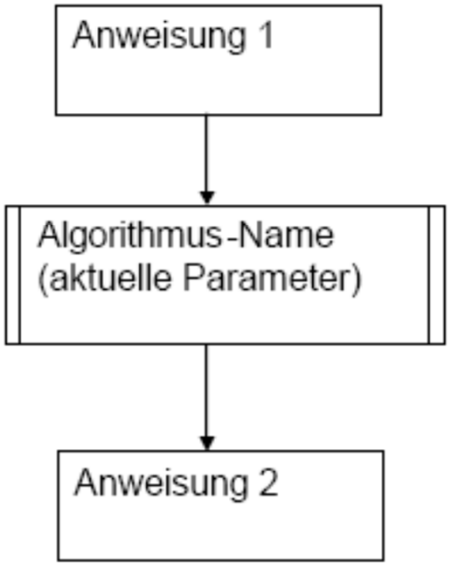
Struktogramm

Pseudocode

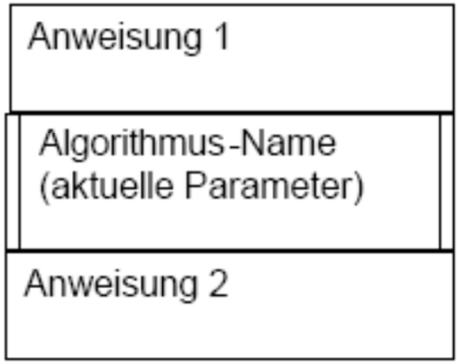
```
do
    Wiederholungsanw.
until Ausdruck
Anweisung(en)
```

Aufruf

Programmablaufplan



Struktogramm



Pseudocode

```
Anweisung 1;

Algorithmus-Name
 (aktuelle Parameter);

Anweisung 2;
```

Übung

- Formulieren Sie in Pseudocode eine Zählschleife, die von n=1 auf 40 mit der Schrittweite 4 zählt.
- Formen Sie diese Schleife in eine funktional äquivalente kopfgesteuerte Schleife um.
- Formen Sie diese Schleife in eine funktional äquivalente fußgesteuerte Schleife um.
- Optional: Geben Sie für alle in Pseudocode erstellten Schleifen die korrespondierenden Struktogramme an.

→ Einzelüberlegung 30 min. (optionale Aufgabe in Heimarbeit)

→ Auswertung siehe Übungsblätter

Übung

- Erstellen Sie ein Struktorgramm zur Berechnung der Fakultät.

  Die Fakultät ist das Produkt der ersten n natürlichen Zahlen: n!

- Geben Sie den korrespondierenden Pseudocode zu diesem

  Algorithmus an.

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times \ldots \times n, \qquad n \in N, \qquad 0! = 1$$

→ Gruppenübung 30 min.

→ Diskussion und Übungsblätter

Entwurfsmethodiken

Problemnäherung

- Ein Problem sollte *in Teilprobleme zerlegt* werden. Diese Teilprobleme sollten proaktiv formuliert werden (Handlungen).
- Dabei ist jeweils zu entscheiden, ob es sich um eine *wiederholende-,* eine in sich *abgeschlossene-* oder eine *fallabhängige* Aktivität handelt.
- Abgeschlossene Aktivitäten können wiederum aus anderen Aktivitäten zusammengesetzt oder atomar sein.
- Somit ergibt sich für einen ersten Entwurf eine Darstellung eines groben Ablaufs, der dann weiter verfeinert wird.
- Die meisten Problemstellungen lassen sich zunächst grob in eine Sequenz von Aktivitäten untergliedern → vgl. EVA-Prinzip.

Problemlösungsstrategien

Top-Down-Strategie

Zerlegung eines Problems in kleinere, leichter lösbare Teilprobleme. Zurückführung der Teilprobleme auf atomare Anweisungen bzw. Kontrollstrukturen bzw. auf bereits gelöste Probleme.

Bottom-□p-Strategie

Es werden bereits vorhandene Komponenten zu einer neuen Lösung zusammengebaut.

Gemischte Strategie

Durch das Herunterbrechen eines gegebenen Problems in Teilprobleme und die Verwendung bereits vorhandener Problemlösungskomponenten kommt es in der Praxis meist zu einer vermischten Strategie von Top-Down- und Bottom-□p-Vorgehen.

Bedingungen

Bedingungen

- Können den Programmablauf beeinflussen.
- Einsatz logischer- und Vergleichsoperatoren.

Beispiele für Prüfungen:

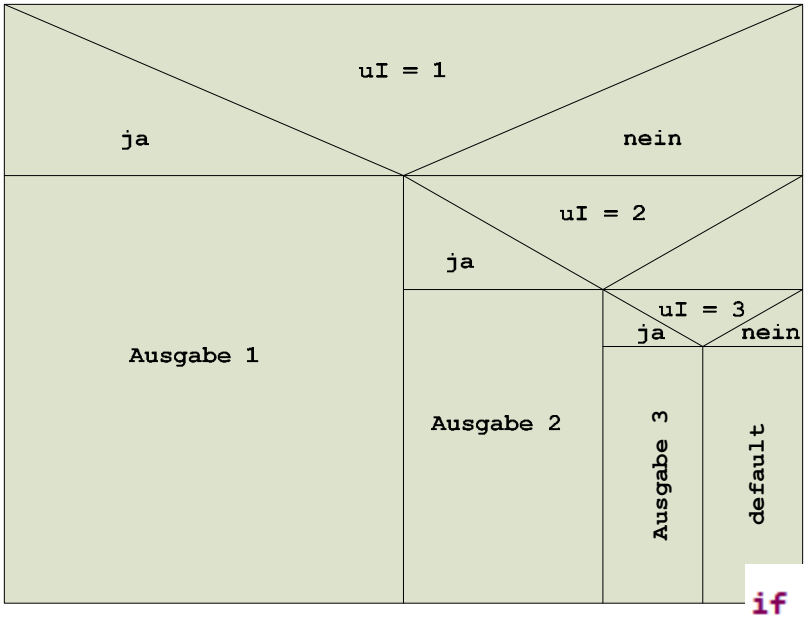
- Zahl gerade?:
  - (zahl mod 2) = 0
- Zahl zwischen 100 und 500?:
  - (zahl>100) and (zahl<500)
- Zeichen ist eine Ziffer?:
  - (zeichen>=‘0‘) and (zeichen<=‘9‘)
- hat Mitarbeiter „Meier“ bereits □rlaubstage verbraucht:
  - (mitarbeiter= „Meier“) and (urlaubstage < 30)

```
public class EvalTest {

    public static void main(String[] args) {

        int zahl = 110;
        int urlaubstage = 24;
        char zeichen = '2';
        String mitarbeiter = "Meier";

        if ((zahl % 2) == 0){
            System.out.println("gerade Zahl");
        }
        if((zahl>100) && (zahl<500)){
            System.out.println("im Wertebereich");
        }

        if((zeichen>='0') && (zeichen<='9')){
            System.out.println("Ziffer");
        }

        if (mitarbeiter.equals("Meier") && (urlaubstage < 30)){
            System.out.println("Urlaubstage verbraucht");
        }
    }
}
```

Geschachtelte Auswahl vs. Mehrfachauswahl

- Eine Mehrfachauswahl (case-Anweisung) kann auch durch die Schachtelung von if-Anweisungen realisiert werden.
- Die Mehrfachauswahl ist sinnvoll, wenn eine Variable auf mehrere Werte abgefragt wird.
- Die Mehrfachauswahl ist *übersichtlicher* als die Schachtelung von if-Abfragen – ab Java 12 auch multiple Mehrfachauswahl.
- Sie kann jedoch häufig nur auf ordinale Datentypen (z.B. Ganzzahlen) angewendet werden – ab Java 1.7 auch Strings.

Geschachtelte Auswahl vs. Mehrfachauswahl




```
if (userInput == 1) {
    System.out.println("Sie Haben 1 gewählt");
} else if (userInput == 2) {
    System.out.println("Sie Haben 2 gewählt");
} else if (userInput == 3) {
    System.out.println("Sie Haben 3 gewählt");
} else {
    System.out.println("Sie etwas anderes gewählt");
}
```

Geschachtelte Auswahl vs. Mehrfachauswahl

```
switch (userInput) {
case 1:
    System.out.println("Sie Haben 1 gewählt");
    break;
case 2:
    System.out.println("Sie Haben 2 gewählt");
    break;
case 3:
    System.out.println("Sie Haben 3 gewählt");
    break;
default:
    System.out.println("Sie etwas anderes gewählt");

}
```

Aufbau einer Mehrfachauswahl:

- Die Fälle nach ihrer Häufigkeit ordnen.
- Alle Auswahlfälle, die in ihrer Häufigkeit vergleichbar sind, werden alphabetisch oder numerisch geordnet.
- Der letzte „else“-Teil (bei Java default) ist für potentielle Fehlerfälle vorbehalten.
- Die Anweisungen innerhalb von Anweisungsblöcken sollten möglichst kurz und einfach gehalten werden.