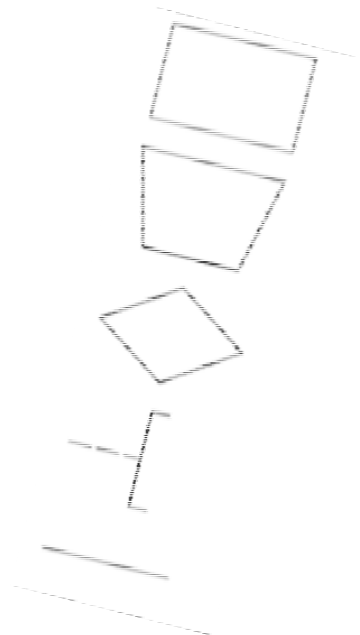


# Vorlesung Programmieren

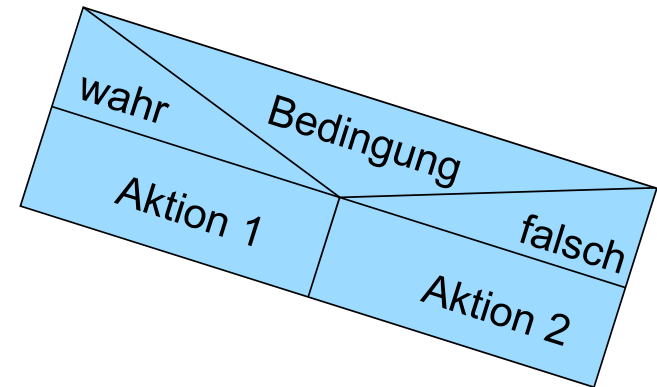
## Thema 2: Programmiersprachen

### Olaf Herden

Fakultät Technik  
Studiengang Informatik



$$L(\text{Bsp}) = \{a(b|c|bd|cd|d)^ne\}$$



Stand: 11/2023

- Grundbegriffe
- Algorithmen und ihre Beschreibung
- Beschreibung von Programmiersprachen
- Programmentwicklung

- **Programmierung:**
  - Erstellung von Computerprogrammen
- **Softwareentwicklung:**
  - Methoden zum Lösen von Problemen mit Hilfe eines Rechners
- **Algorithmus:**
  - Arbeitsanleitung für einen Computer
- **Programmiersprache:**
  - Computerverständliche Notation zur Formulierung von Algorithmen
- **Programm:**
  - In einer Programmiersprache formulierter Algorithmus
- **„Programmieren im Kleinen“:**
  - Lösen kleiner Probleme, typischerweise alleine
- **„Programmieren im Großen“:**
  - Lösen komplexer Probleme, typischerweise in einem Team

- **Software-Engineering/Softwaretechnik:**
  - Vorgehensmodelle, Entwicklungsmethoden, Entwicklungsumgebungen, Projekt-, Qualitäts- und Konfigurationsmanagement
- **Programmierer\*in:**
  - Entwickler\*in von Programmen
- **Programmcode, Quellcode, Sourcecode:**
  - Programmbeschreibung auf einer bestimmten Abstraktionsebene
- **Ausführbares Programm:**
  - Programm in maschinenverständlicher Form
- **Programmaufruf:**
  - Ausführung eines ausführbaren Programms

- Grundbegriffe
- Algorithmen und ihre Beschreibung
- Beschreibung von Programmiersprachen
- Programmentwicklung

- **Arbeitsanleitungen:**
  - Kochrezepte, Bastelanleitungen, Partituren, Spielregeln
- **Aufbau:**
  - Menge von Anweisungen
- **Charakteristika:**
  - Anweisungssequenzen
  - Bedingte Anweisungen
  - Anweisungsschleifen
  - Zutaten / Voraussetzungen
  - Zum Teil „schwammige“ Formulierungen

- Definition Algorithmus:
  - Arbeitsanleitung zum Lösen eines Problems bzw. einer Aufgabe, die so präzise formuliert ist, dass sie von einem Computer ausgeführt werden kann.
- Beschreibung von Algorithmen kann auf verschiedene Arten geschehen:
  - Umgangssprachlich
  - Programmiersprache
  - Programmablaufpläne
  - Struktogramme

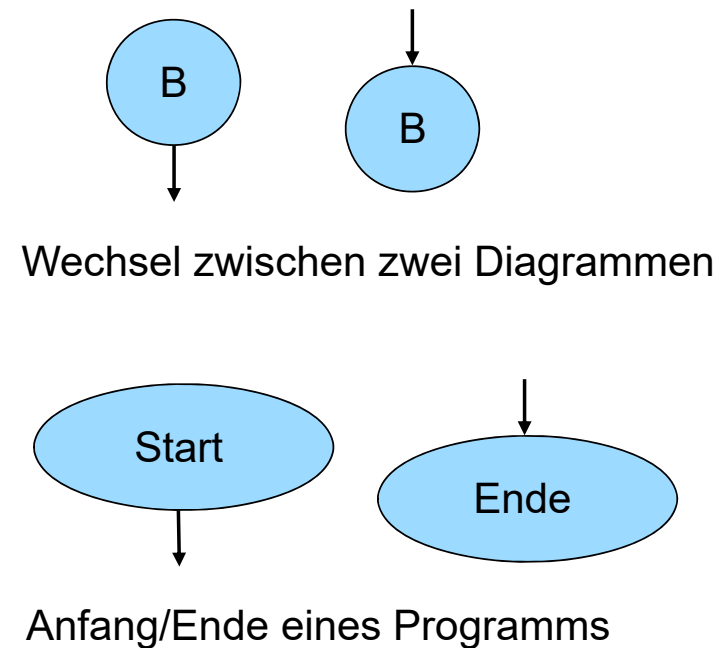
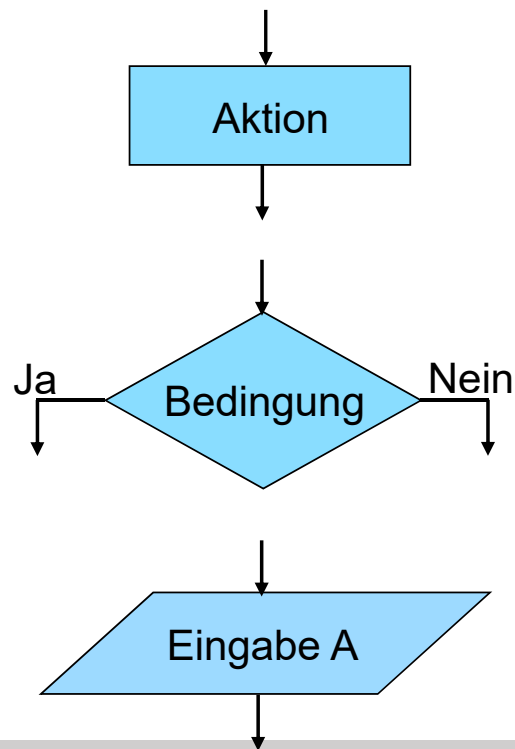
- Beispiel: Berechnung der Summe aller Zahlen von 1 bis n
- Umgangsprachliche Formulierung:
  - Addiere für eine vorgegebene natürliche Zahl n die Zahlen von 1 bis n
  - Dies ist das Resultat

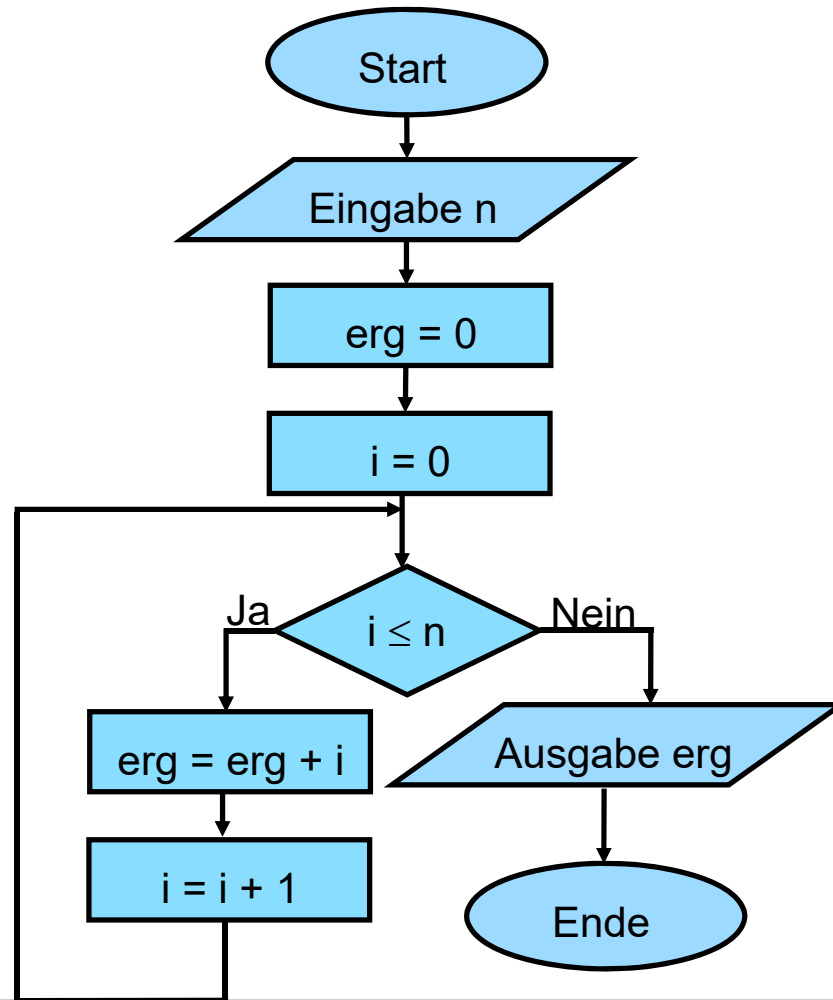
- Programmiersprache bzw. Pseudocode:

```
int n = readInt( );  
int erg = 0;  
int i = 0;  
while (i <= n) {  
    erg = erg + i;  
    i = i + 1;  
}  
printInt(erg);
```


















- Programmablaufplan (PAP) (Synonym: Flussdiagramm, Ablaufplan, Ablaufdiagramm)
- Normierte Methode zur graphischen Darstellung von Algorithmen
- Grundelemente:





Anh. 1

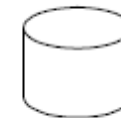
© P. Stahnecht / U. Hasenikamp: Einführung in die Wirtschaftsinformatik, 9. Auflage, Springer-Verlag, Berlin - Heidelberg 1999.  
Das Kopieren auf eine Vortragsfolie bzw. in eine Präsentationsdatei ist gestattet.

Sinnbild	Benennung, Bemerkung	D P	Sinnbild	Benennung, Bemerkung	D P
	Verarbeitung, allgemein (einschließlich Ein-/Ausgabe)	D P		Daten, allgemein	D
	Manuelle Verarbeitung (einschließlich Ein-/Ausgabe)	D P		Daten auf Schriftstück (z.B. auf Belegen, Mikrofilm)	D
	Verzweigung	P		Daten auf Speicher mit nur sequen- tiellem Zugriff	D
	Bemerkung (erläuternder Text)	D P		Daten auf Speicher mit auch direk- tem Zugriff <sup>2)</sup>	D
	Verbindung <sup>1)</sup> : Verarbeitungsfolge bzw. Zugriffsmöglichkeit	P D		Daten im Zentral- speicher	D
	Verbindung <sup>1)</sup> zur Darstellung der Daten- übertragung	D P		Maschinell er- zeugte optische oder akustische Daten	D
	Grenzstelle (zur Umwelt)	D P		Manuelle optische oder akustische Eingabedaten	D
	Verbindungs- stelle	D P			

D = Verwendung im Datenflußplan,  
P = Verwendung im Programmablaufplan

<sup>1)</sup> Bei den Verbindungen gilt die Vorzugs-  
richtung von links nach rechts bzw. von  
oben nach unten. Abweichungen sind  
durch Pfeilspitzen zu kennzeichnen.

<sup>2)</sup> In der Praxis wird dieses  
Symbol häufig um 90°  
gedreht gezeichnet:

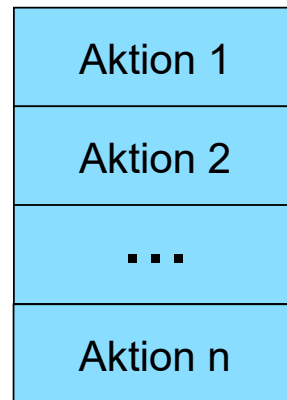


Sinnbilder nach DIN 66001

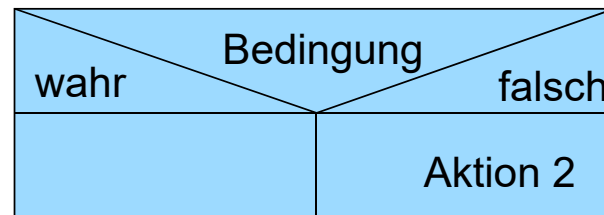
- Struktogramme/Nassi-Shneiderman-Diagramme:
  - Alternative graphische Notation zum Darstellen von Programmen
  - Neuere Notation als Flussdiagramme
  - Durch Strukturierung bessere Programme (vor allem Vermeidung von Sprüngen)
- Notationselemente (DIN 66261):



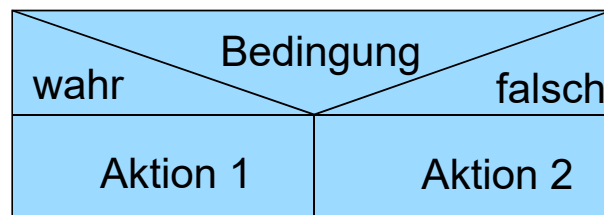
Strukturblock



Folge (Aneinanderreihung von Strukturblöcken)

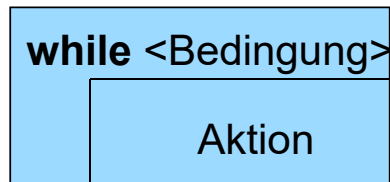


Bedingter Block

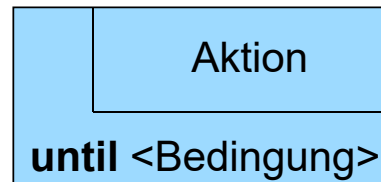


Alternative Blöcke

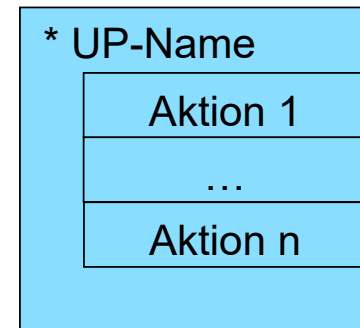
- Notationselemente (DIN 66261) Fortsetzung:



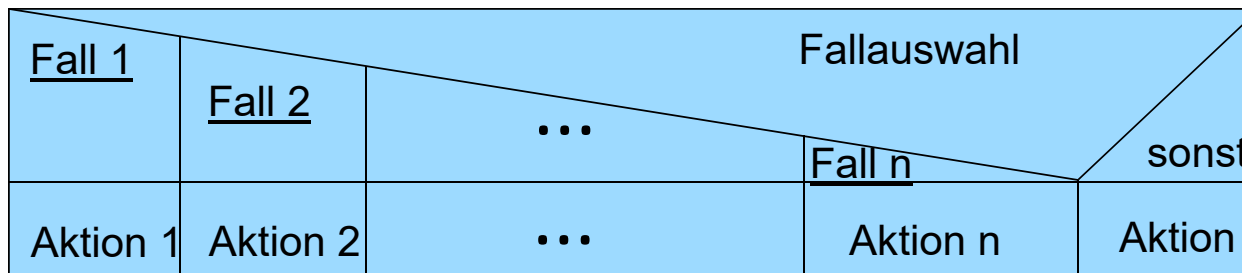
Abweisende Schleife



Nicht-Abweisende Schleife

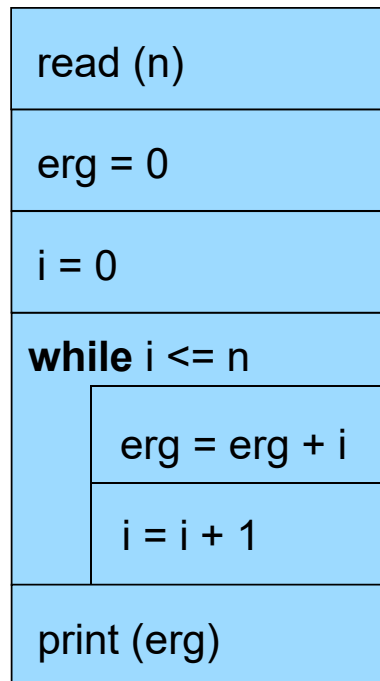


Unterprogramm



Fallunterscheidung

- Beispiel: Summe der ersten n Zahlen



- PAP:
  - enthält explizit bedingte Verzweigungen
  - schließt implizit unbedingte Verzweigungen nicht aus
  - D.h.: Programmablauf ist verschieden vom Steuerfluss (zeitliche Reihenfolge der Abarbeitung der Anweisungen)
- Struktogramme:
  - Unbedingte Verzweigungen sind ausgeschlossen
  - Dies garantiert Übereinstimmung von Programmablauf und Steuerfluss
  - Relative Länge der Strukturblöcke repräsentiert Steuerfluss

- **Eindeutigkeit:**
  - Beschreibung muss präzise genug sein, um Doppeldeutigkeiten auszuschließen
- **Abstraktion/Parametrisierbarkeit:**
  - Algorithmus löst i.A. Klasse von Problemen
  - Wahl eines einzelnen Problems erfolgt über Parameter
- **Ausführbarkeit:**
  - Algorithmus ist auf einer existierenden Maschine umsetzbar und danach ausführbar
- **Terminierung:**
  - Algorithmen, die für jede mögliche Eingabe nach endlich vielen Schritten anhalten, heißen terminierend, ansonsten nichtterminierend
  - Beide Klassen sind interessant (abhängig vom Problem)

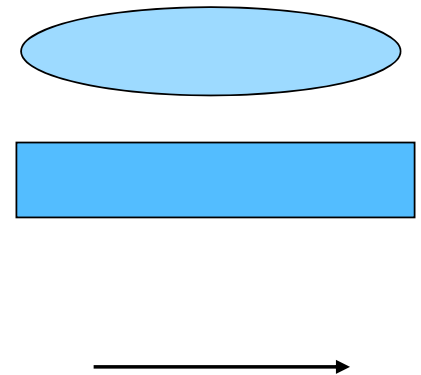


- **Determinismus:**
  - Zu jedem Zeitpunkt der Ausführung existiert höchstens eine mögliche Fortsetzung
  - Hat Algorithmus zu einem Zeitpunkt mehrere Möglichkeiten, aus denen er wählen spricht man von Nicht-Determinismus
- **Determiniertheit:**
  - Mit gleichen Eingabewerten und Startbedingungen liefert Algorithmus stets dasselbe Ergebnis
  - Manchmal Abweichung bei Verwendung von Heuristiken
- **Korrektheit:**
  - Algorithmus liefert für jeden möglichen Eingabewert das richtige Ergebnis
- **Vollständigkeit:**
  - Algorithmus ist auf alle (zugelassenen) Eingabewerte anwendbar
- **Effizienz (bezgl. Zeit und/oder Speicher):**
  - Speicher und Zeit sind i.A. knappe Ressourcen
  - Schonender Einsatz wird als Effizienz bezeichnet

- **Erweiterbarkeit:**
  - Algorithmus ist so konzipiert, dass er an (in der Zukunft möglicherweise auftretende) weitere Anforderungen leicht angepasst werden kann
- **Wiederverwendbarkeit:**
  - Algorithmus ist so aufgebaut, dass er leicht in anderen Anwendungen verwendet werden kann
- **Portabilität:**
  - Algorithmus ist so aufgebaut, dass er leicht auf andere Plattformen oder in neue Versionen einer Sprache übertragen werden kann
- **Verständlichkeit:**
  - Algorithmus ist so aufgebaut, dass er für einen Außenstehenden leicht nachzuvollziehen ist

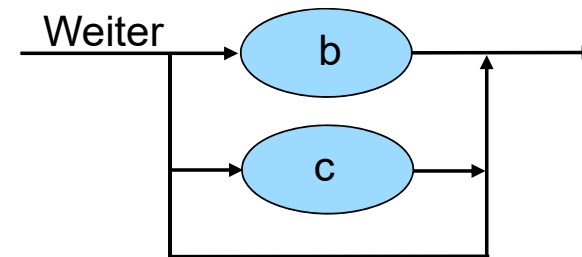
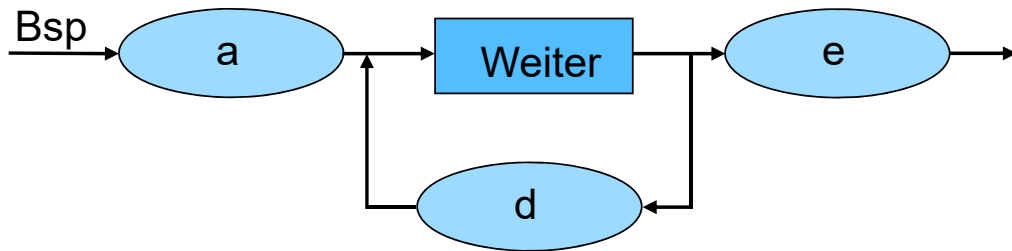
- Grundbegriffe
- Algorithmen und ihre Beschreibung
- Beschreibung von Programmiersprachen
- Programmentwicklung

- Programmiersprachen werden durch kontextfreie Grammatiken beschrieben (Siehe Informatik-VL)
- Eine wichtige graphische Notation sind dazu die Syntaxdiagramme
- Notationselemente:
  - Knoten
    - Ellipsen (Terminal-Symbole, nicht weiter ableitbar)
    - Rechtecke (Nichtterminal-Symbole, können weiter abgeleitet werden)
  - Kanten
    - Knotenverbindende Pfeile
    - Eintretender Pfeil (Eingangskante)
    - Austretender Pfeil (Ausgangskante)



- Interpretation:
  - Durchläuft man ein Syntaxdiagramm von der Eingangs- zur Ausgangskante entlang den Pfeilen, dann ist die Folge der Knoteninhalte, die dabei „aufgesammelt“ werden, aus dem Syntaxdiagramm ableitbar.
- Strukturregeln:
  - Jedes Syntaxdiagramm (SD) besitzt eine Bezeichnung
  - Elemente eines Syntaxdiagramms sind Knoten (Ellipsen, Rechtecke) und Kanten (Pfeile)
  - Rechtecke enthalten die Bezeichnung eines (anderen) Syntaxdiagramms
  - Ellipsen enthalten Token (Elemente der beschriebenen Sprache)
  - In jeden Knoten führt genau ein Pfeil hinein
  - Aus jedem Knoten führt genau ein Pfeil hinaus
  - Pfeile dürfen sich aufspalten bzw. zusammengeführt werden
  - Jedes SD besitzt genau eine eintretende Kante (kein Ausgangsknoten)
  - Jedes SD besitzt genau eine austretende Kante (kein Eingangsknoten)

- Gegeben sei folgendes Syntaxdiagramm:



- Syntaktisch korrekt:

ae  
 abe  
 abdce  
 abdbde

- Syntaktisch nicht korrekt:

be  
 ad  
 abce  
 adae

- Menge syntaktisch korrekter Ausdrücke nennt man die vom Syntaxdiagramm SD erzeugte Sprache  $L(SD)$
- Notation:
  - Das mehrfache Auftreten eines oder mehrerer Zeichen wird durch ein hochgestelltes  $n$  ausgedrückt
  - Das alternative Erscheinen eines oder mehrerer Zeichen durch  $(Alternative_1 | Alternative_2 | \dots | Alternative_n)$
  - Im Beispiel erzeugte Sprache  $L(Bsp) = \{a(b|c|bd|cd|d)^ne\}$

- BNF: Backus-Naur-Form:
  - Technik zur textuellen Darstellung kontextfreier Grammatiken
  - Verwendung von Ersetzungsregeln (Produktionen)
  - Besitzen linke und rechte Seite
  - Linke Seite: Nichtterminalsymbol
  - Nichtterminalsymbol: durch  $\langle \rangle$  gekennzeichnet
  - Alternativen: durch  $|$  gekennzeichnet
  - $\varepsilon$  (Epsilon): leere Alternative
- EBNF:
  - Erweiterung der BNF (Abkürzungsmöglichkeiten)
  - $[ \dots ]$  bedeutet: Symbole in Klammern können auch wegfallen
  - $\{ \dots \}$  bedeutet: Symbole in Klammern können beliebig oft wiederholt werden



- BNF:

$\langle \text{Bsp} \rangle ::= a \langle \text{Weiter-und-mehr} \rangle$

$\langle \text{Weiter-und-mehr} \rangle ::= \langle \text{Weiter} \rangle d \langle \text{Weiter-und-mehr} \rangle$   
 $\quad \quad \quad | \langle \text{Weiter} \rangle e$

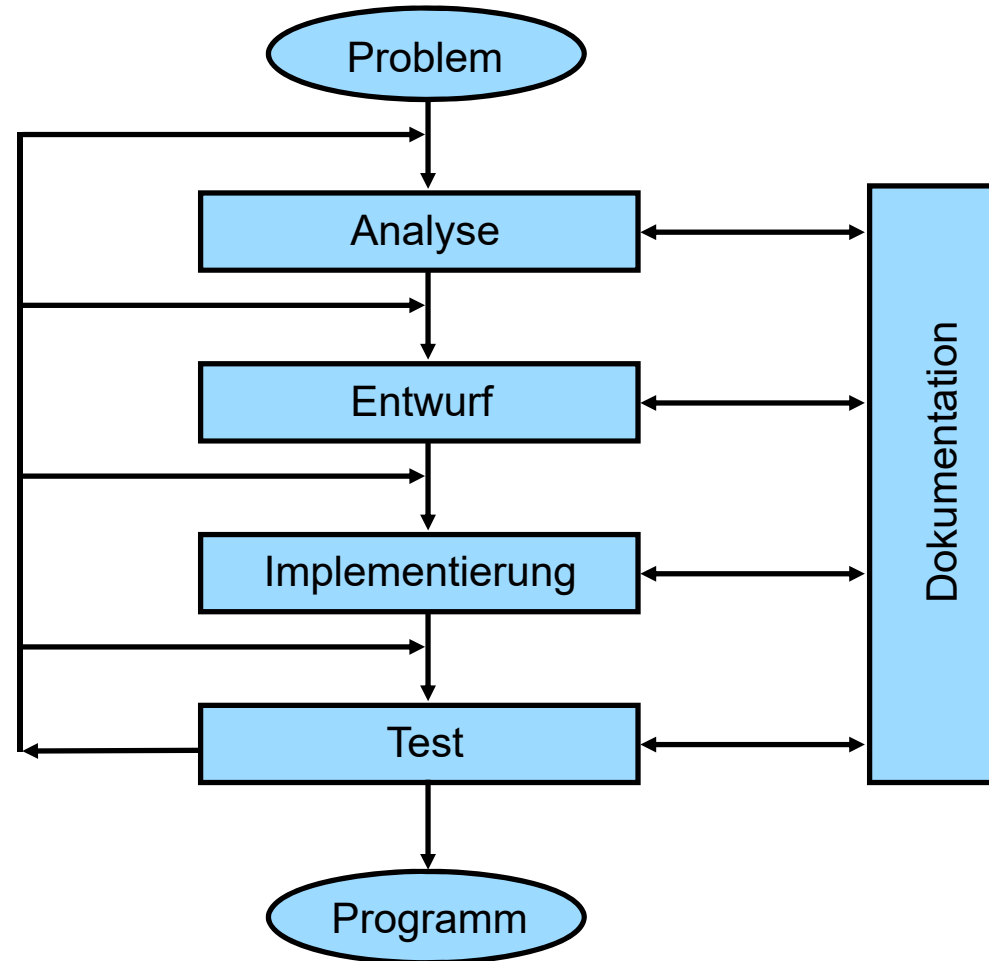
$\langle \text{Weiter} \rangle ::= b | c | \varepsilon$

- EBNF:

$\langle \text{Bsp} \rangle ::= a \langle \text{Weiter} \rangle \{ d \langle \text{Weiter} \rangle \} e$

$\langle \text{Weiter} \rangle ::= b | c | \varepsilon$

- Grundbegriffe
- Algorithmen und ihre Beschreibung
- Beschreibung von Programmiersprachen
- Programmentwicklung



- Untersuchung des Problems und des Problemumfelds
- Diskussion mit anderen Personen
- Fragestellungen / Tätigkeiten:
  - Problemstellung exakt und vollständig
  - Gegebene Initialzustände und Eingabeparameter
  - Gewünschte Endzustände und Ausgabewerte
  - Randbedingungen, Constraints

- Entwurf:
  - Entwicklung des Algorithmus
  - Kreativer Prozess (Auffassungsgabe, Intelligenz, Erfahrung)
  - Fragestellungen / Tätigkeiten:
    - Existierende Lösungen für vergleichbare Probleme
    - Allgemeinere Probleme
    - (Rekursive) Aufteilung des Problems in Teilprobleme
    - Durchführung des Entwurfsprozess für Teilprobleme
    - Zusammensetzen der Lösungen der Teilprobleme zur Lösung des Gesamtproblems

- Übertragung des Entwurfs in eine Programmiersprache
- Fragestellungen / Tätigkeiten:
  - Editieren
  - Compilieren
  - Einige Fehler beheben

- Überprüfung des Programms auf logische und technische Fehler
- Man kann nur die Existenz von Fehlern nachweisen, nicht die Abwesenheit!
- Fragestellungen / Tätigkeiten:
  - Korrektheit
  - Vollständigkeit
  - Debugging
- Teststrategien:
  - Andere Personen testen lassen
  - Testmengen konstruieren (Randfälle/Grenzwerte finden)
  - Nach Fehlerbeseitigung erneut testen

- Exakte Problemstellung
- Beschreibung der generellen Lösungsidee
- Beschreibung des Algorithmus
- Programmcode
- Beschreibung der Testmengen
- Protokolle der Testläufe
- Aufgetretene Probleme
- Alternative Lösungsansätze



- Einführung
- Effizienzverbesserung
- Wartung
- Erweiterung
- Portierung

- Grundbegriffe
  - Programmieren
  - Software Engineering
  - Algorithmus
  - Programm
- Algorithmen und ihre Beschreibung:
  - Umgangssprachlich
  - Programmiersprache/Pseudocode
  - Programmablaufplan
  - Struktogramm
  - Eigenschaften von Algorithmen

- Beschreibung von Programmiersprachen
  - Syntaxdiagramm
  - E(BNF)
- Programmentwicklung
  - Analyse
  - Entwurf
  - Implementierung
  - Testen

- **Aufgabe 1**

Was bedeuten die beiden Eigenschaften Determinismus bzw. Determiniertheit von Algorithmen?

- **Aufgabe 2**

Es soll ein Algorithmus entworfen werden, der zu einem eingelesenen Wert die Fakultät berechnet. Stellen Sie diesen als Flussdiagramm und als Struktogramm dar!

## • Aufgabe 3

a) Geben Sie ein Syntaxdiagramm an, das die Sprache  $L = \{abnc \text{ mit } n > 3\}$  erzeugt!

b) Welche Sprache erzeugt die folgende EBNF-Grammatik?

```
S ::= NatZahl | Addition
```

```
NatZahl ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

```
Addition ::= "(" S "+" S ")"
```

c) Stellen Sie die EBNF-Regeln aus Teilaufgabe b) als Syntaxdiagramme dar!

## • Aufgabe 4

Gegeben sei der folgende Programmablaufplan. Stellen Sie diesen als Struktogramm dar!

