Karlsruher Institut für Technologie

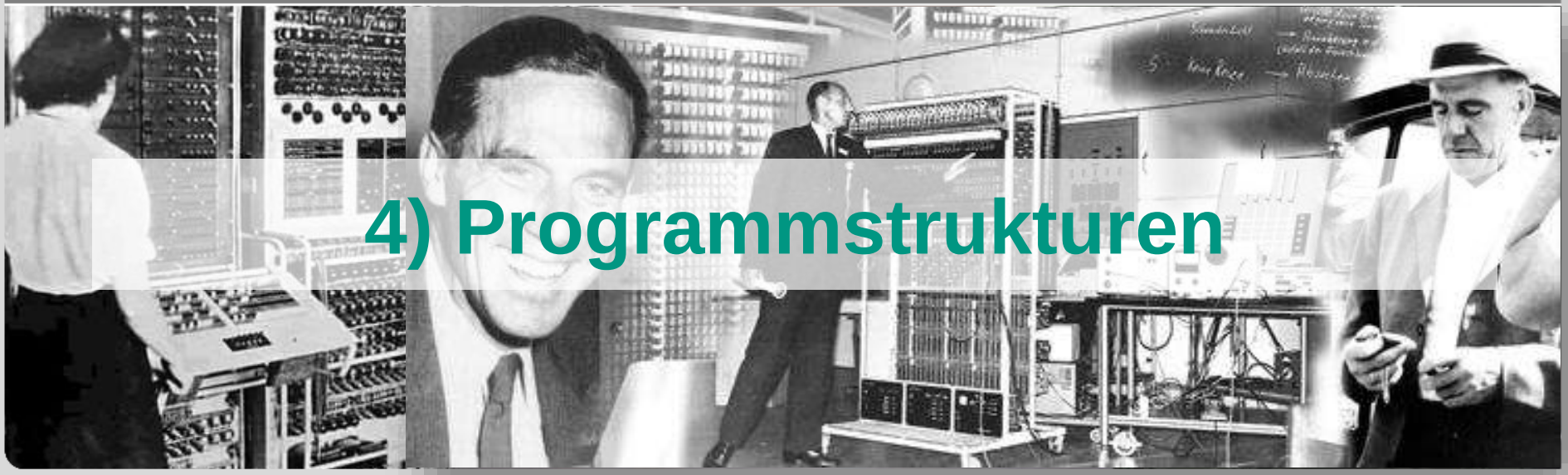
Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Vorlesung Informationstechnik (IT)

Sommersemester 2018

Institut für Technik der Informationsverarbeitung (ITIV)



4) Programmstrukturen

IT Inhalt

3. Weg zum ausführbaren Programm

3.1 Vom Code zum Programm (make)

3.2 Ablauf Erstellung eines Programms

3.3 Die Integrierte Entwicklungsumgebung

3.4 Teile der Entwicklungsumgebung

Programmstrukturen

4.1 Programmablaufpläne

4.2 Nassi-Shneiderman-Diagramme

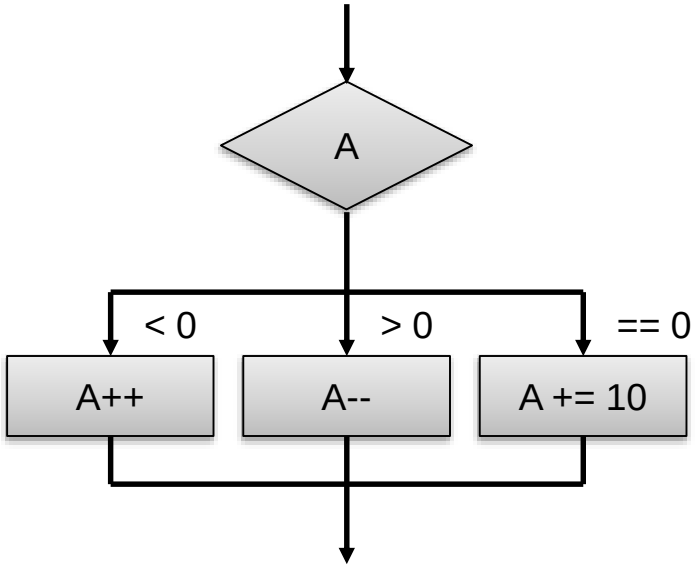
4.3 Gegenüberstellung

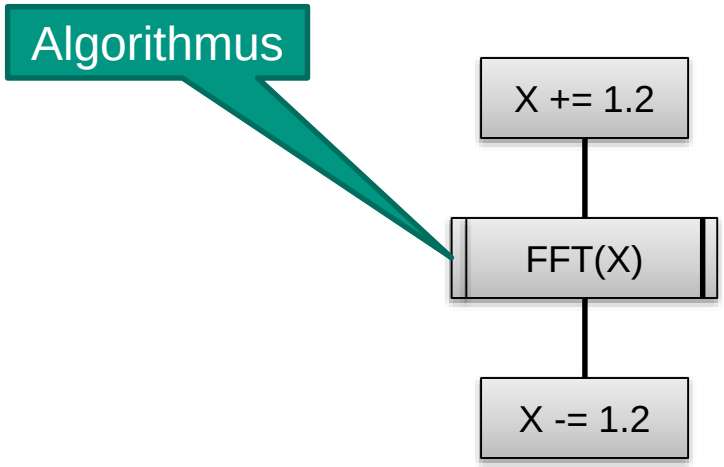
.

Programmablaufplan (PAP)

dargestellt als Flussdiagramm

- Beschreibung Kontroll-Strukturen
- Graphische Notation
- Standardisiert seit 1969 in DIN 66001



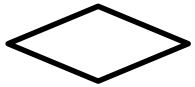


Flussdiagramm - Basiselemente

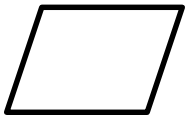
Start/Ende
Markiert Start bzw. Ende des Programmablaufs. Wird üblicherweise mit Start/Ende beschriftet.

Prozess
Ein Kasten kann entweder einen kleinen Schritt oder einen ganzen Unterprozess bezeichnen.

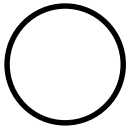
Dokument
Ein gedrucktes Dokument, ein Bericht etc.



Entscheidung
Linien von den Ecken kennzeichnen die verschiedenen möglichen Programmzweige.



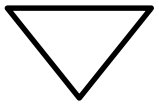
Ein-/Ausgabe
Kennzeichnet Material oder Informationen, die in das System kommen oder es verlassen, etwa eine Bestellung (Eingabe) oder ein Produkt (Ausgabe).



Verbindungspunkt
Weist darauf hin, dass der Ablauf da weitergeht, wo ein anderes, mit dem selben Buchstaben gekennzeichnetes Symbol steht.

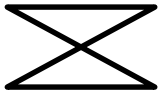
Flusslinie
Linien bestimmen den sequentiellen Ablauf der Ablaufschritte.

Verzögerung
Weist auf Wartebedingungen oder -zeiten im Ablauf hin.

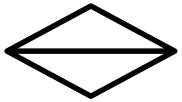


Vereinigung
Kennzeichnet einen Schritt, wo zwei oder mehrere Unterprozesse oder Teillisten zusammengeführt werden.

Flussdiagramm - Basiselemente



Zusammenstellen
Kennzeichnet das Ordnen von Informationen in ein spezifisches Format



Sortieren
Kennzeichnet das Ordnen einer Liste anhand bestimmer Kriterien (Größe, Umfang etc)

Subroutine/Algorithmus
Kennzeichnet einen komplexeren Ablauf, der in das aktuelle Flussdiagramm eingebettet ist. Dieser Ablauf könnte in einem anderen Diagramm spezifiziert sein.

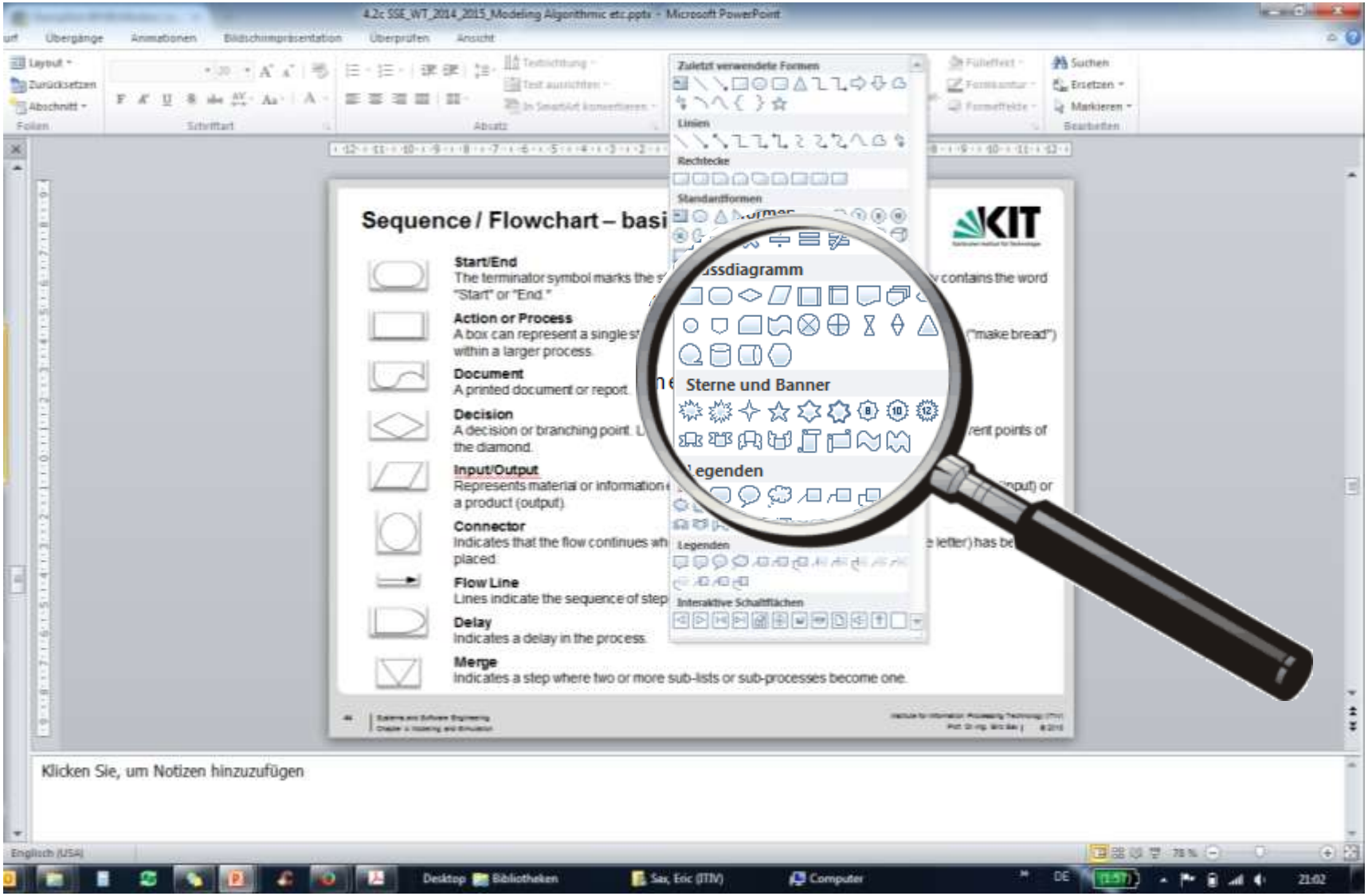
Manueller Eingriff
Kennzeichnet eine Reihe von Anweisungen, die manuell ausgeführt werden müssen.

Schleifenende
Kennzeichnet den Punkt, an dem eine Schleife abbrechen soll

Datenspeicher
Kennzeichnet einen Speicher von Daten.

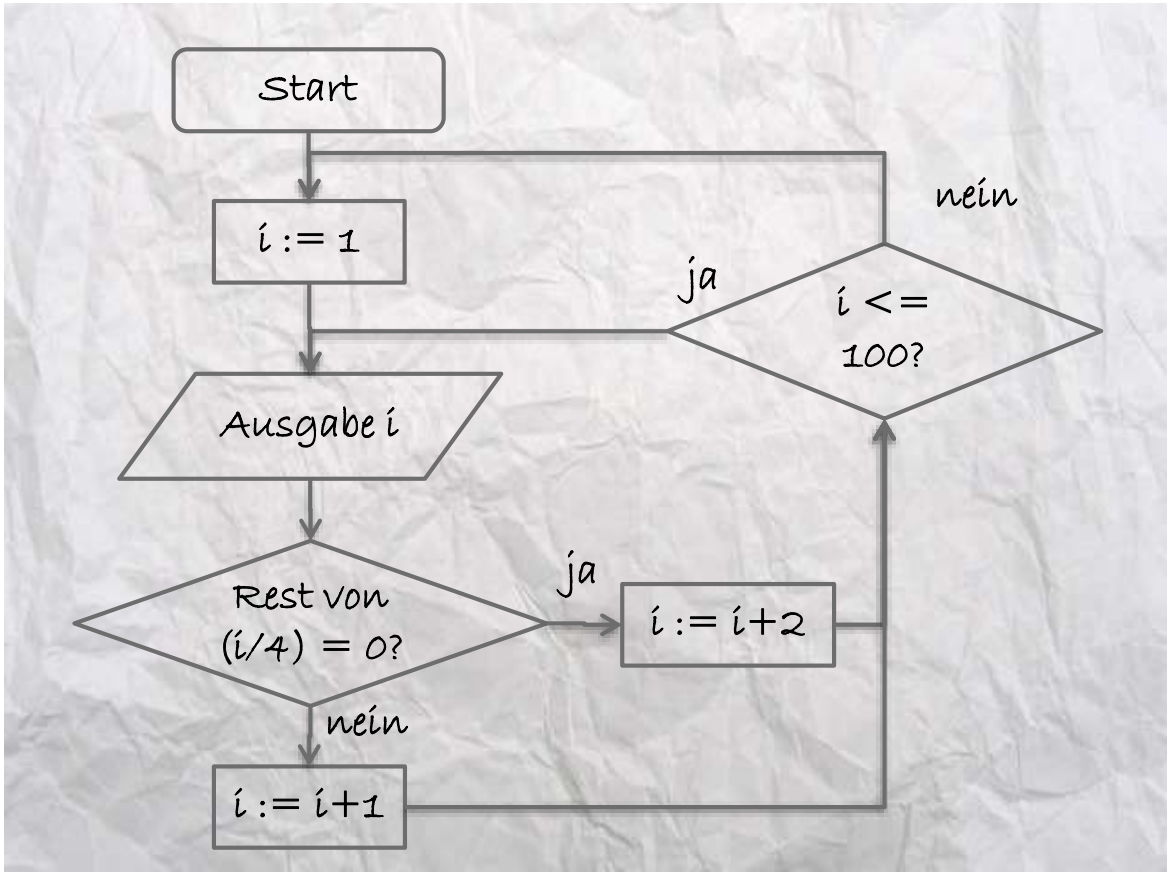
Datenbank (urspr. Magnetplattenspeicher)
Kennzeichnet eine Sammlung von Informationen, die durchsucht und geordnet ausgegeben werden können.

Flussdiagramm „Editor“



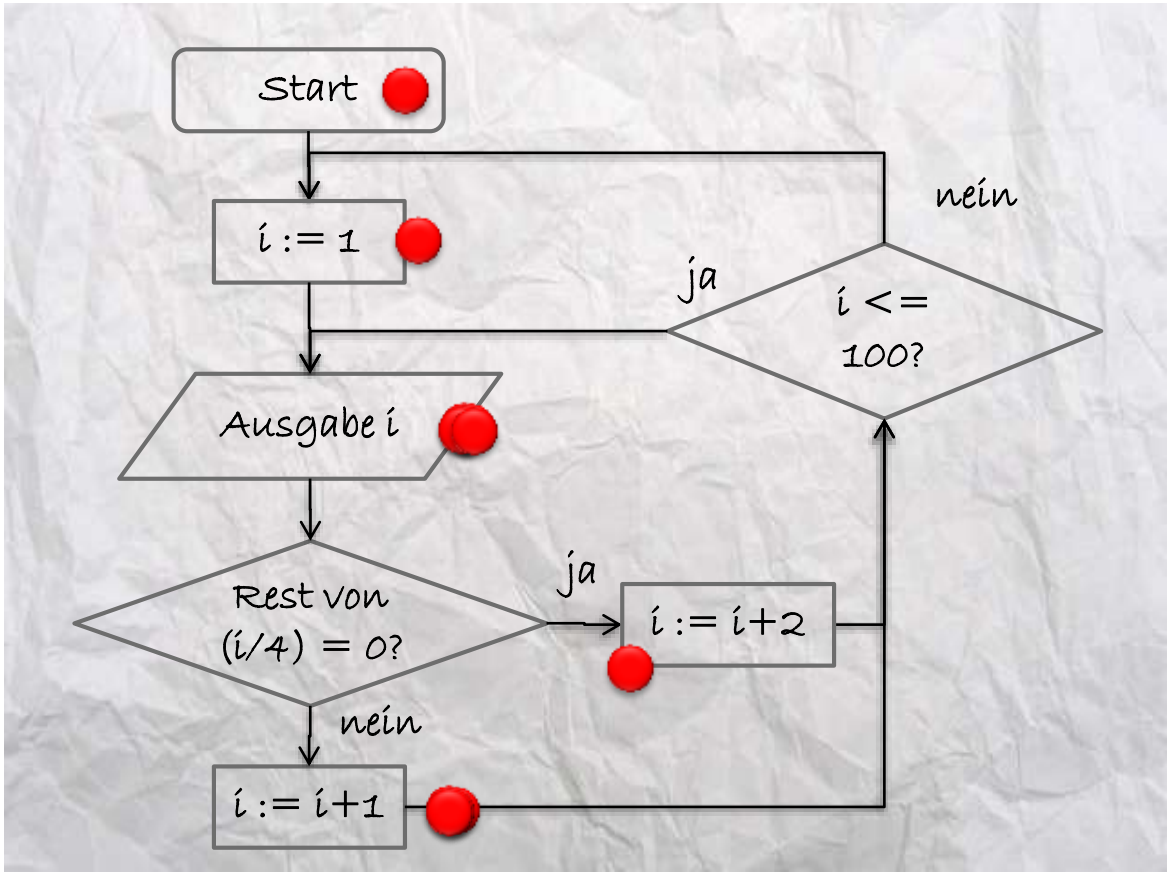
PAP-Beispiel

- Beispiel
  - eine Schleife, die von 1 bis einschließlich 100 zählt und diese Zahlen ausgibt, dabei aber ganzzahlig durch 4 teilbare Zahlen auslässt.



PAP-Beispiel

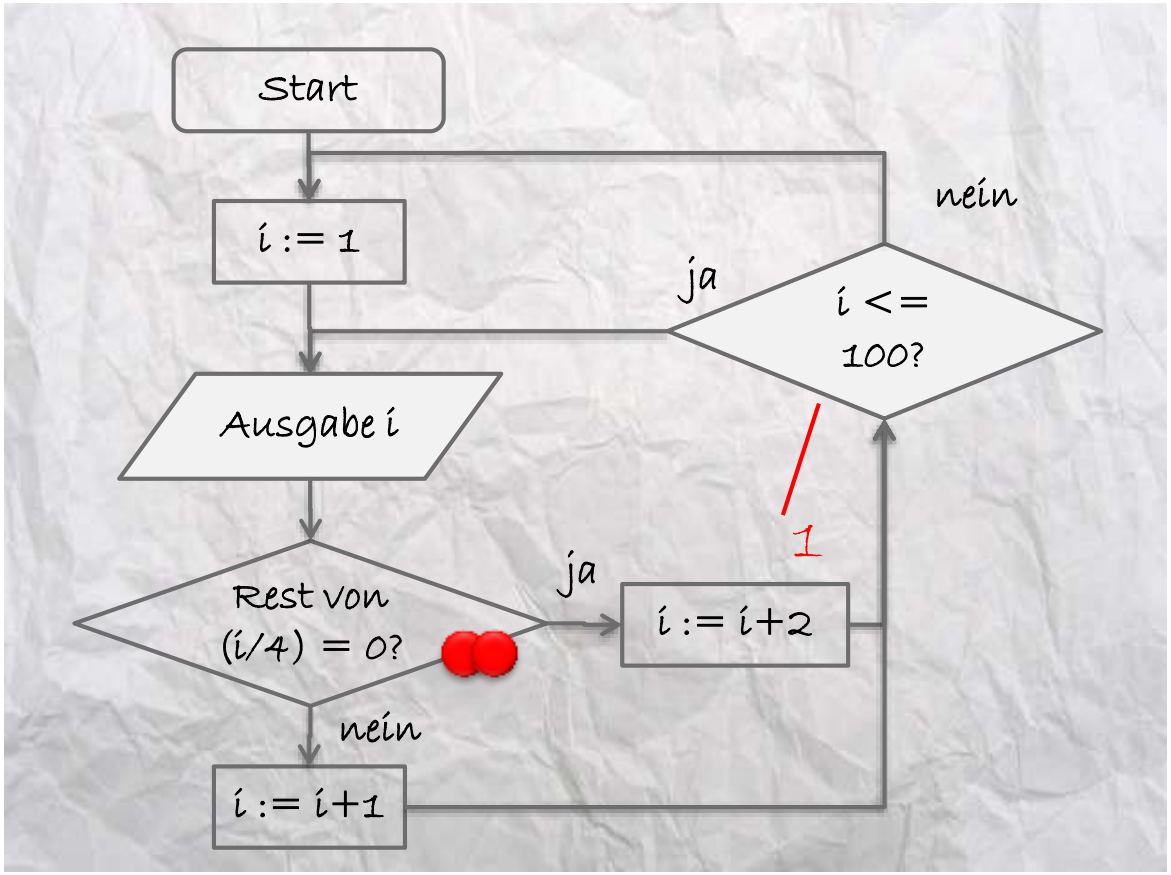
- Beispiel
  - eine Schleife, die von 1 bis einschließlich 100 zählt und diese Zahlen ausgibt, dabei aber ganzzahlig durch 4 teilbare Zahlen auslässt.

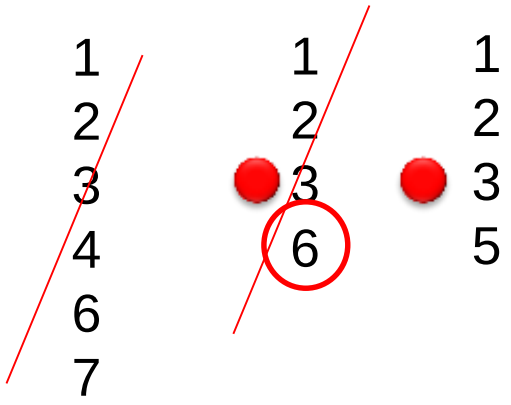


1
2
3
4
6
7

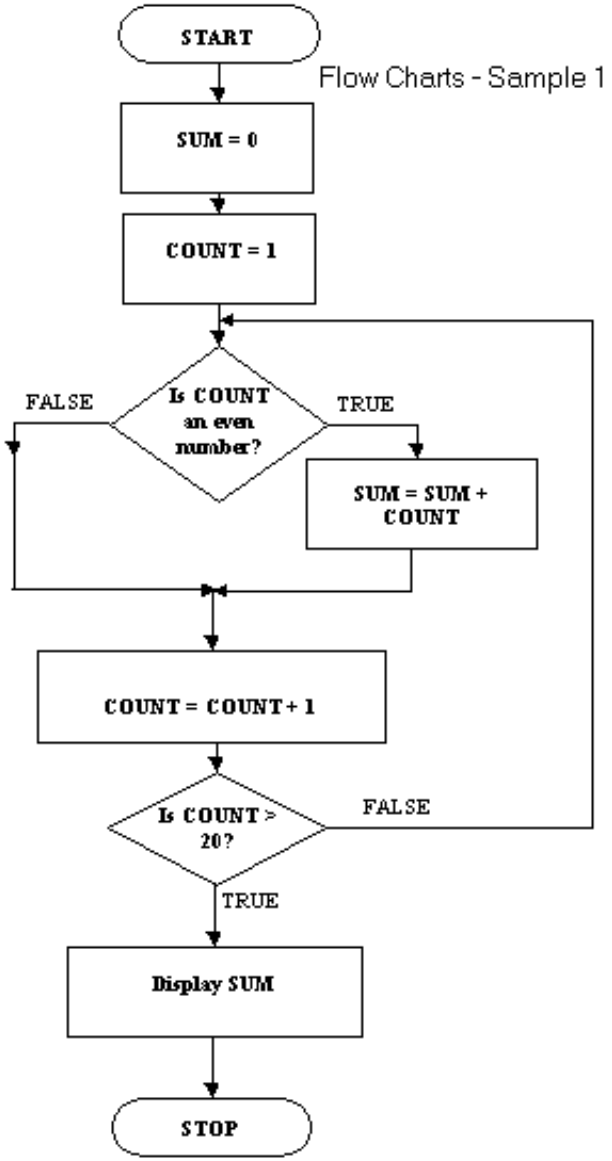
PAP-Beispiel

- Beispiel
  - eine Schleife, die von 1 bis einschließlich 100 zählt und diese Zahlen ausgibt, dabei aber ganzzahlig durch 4 teilbare Zahlen auslässt.



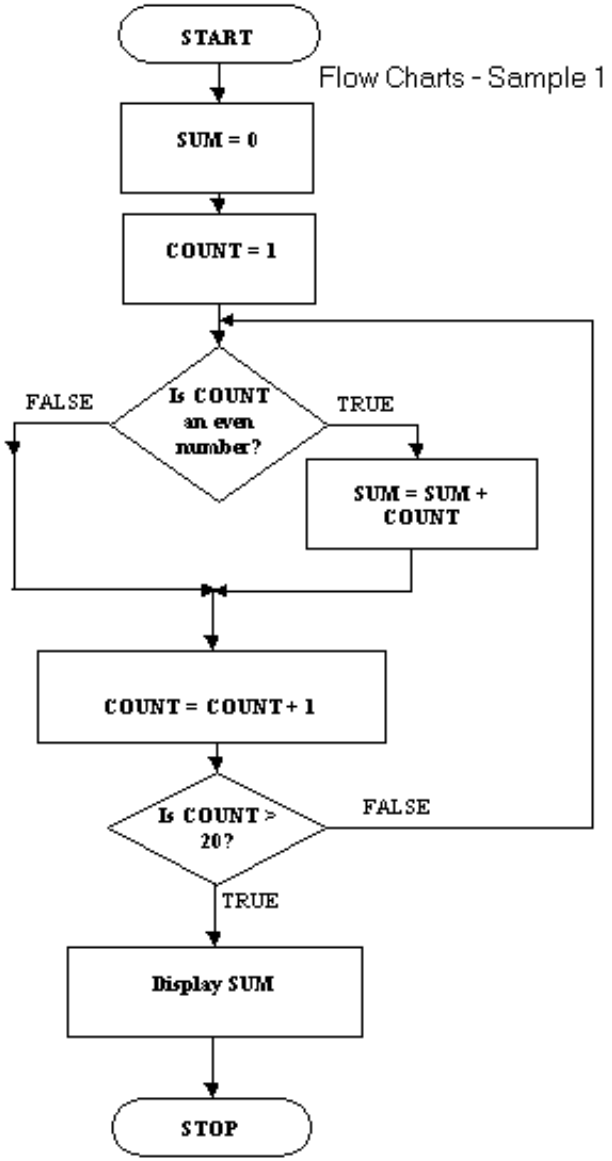


Flussdiagramm Beispiel vs. Pseudo Code



- Der Algorithmus summiert alle geraden Zahlen zwischen 1 und 20 (inklusive) und zeigt dann die Summe an.

Flussdiagramm Beispiel vs. Pseudo Code

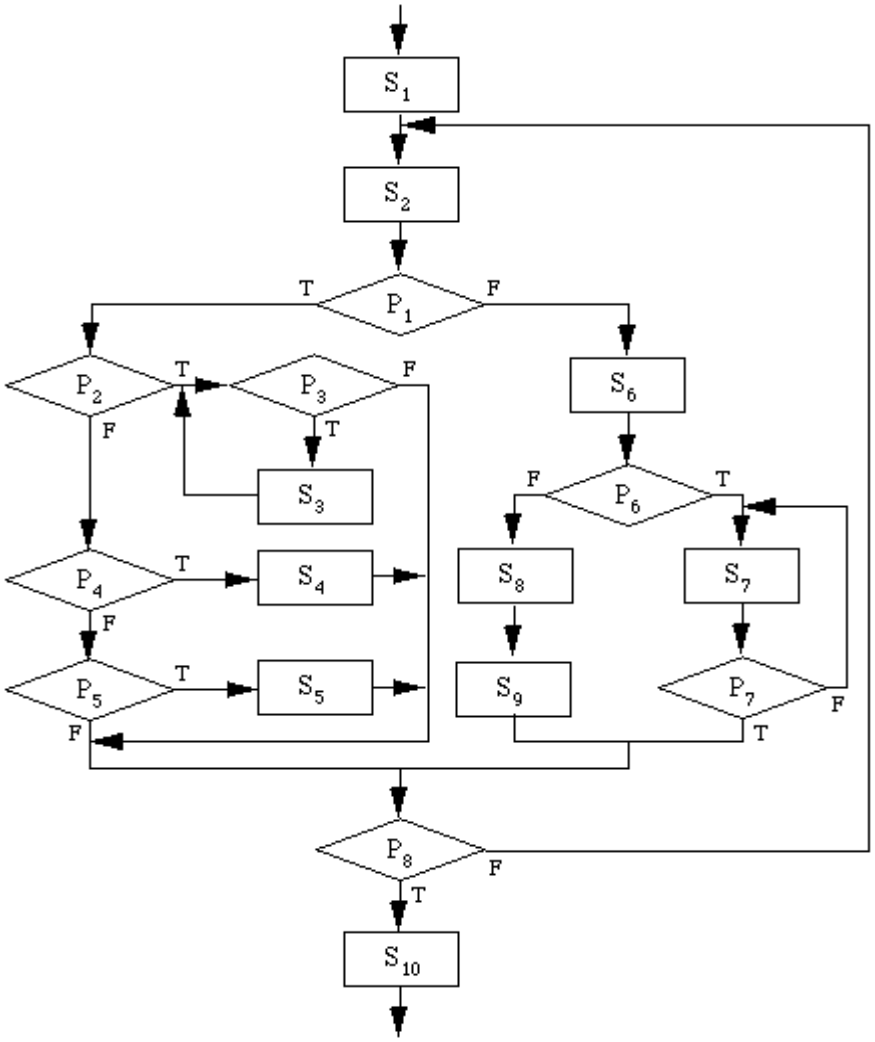


- Der Algorithmus summiert alle geraden Zahlen zwischen 1 und 20 (inklusive) und zeigt dann die Summe an.
- Der entsprechende Pseudocode ist:

```
sum = 0
count = 1
REPEAT
 IF count is even
  THEN sum = sum + count
 count = count + 1
UNTIL count > 20
DISPLAY sum
```

- Verglichen mit dem PAP deutlich weniger Aufwand.
- Wozu dann das Flussdiagramm?
  - Mehr standardisiert
  - Einfacher lesbar

Kritische Anmerkungen



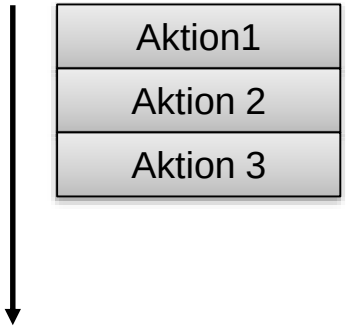
- PAP's werden schon bei kleineren Algorithmen schnell unübersichtlich.
- PAP's verführen zur Verwendung von expliziten Sprunganweisungen (GOTO's) und damit zur Produktion von „Spaghetti-Code“
- Für grundlegende Kontrollstrukturen existieren keine Symbole, z. B. für Fallauswahlen, Schleifen
- Schachtelstrukturen sind im PAP nicht gut zu erkennen.
- Forderung nach strukturierter Programmierung

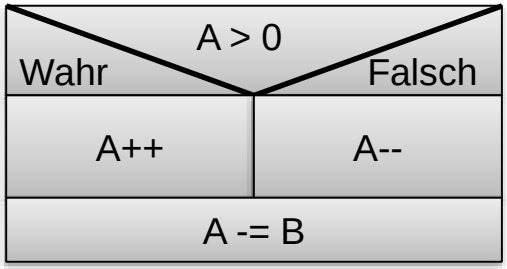
Nassi-Shneiderman-Diagramm

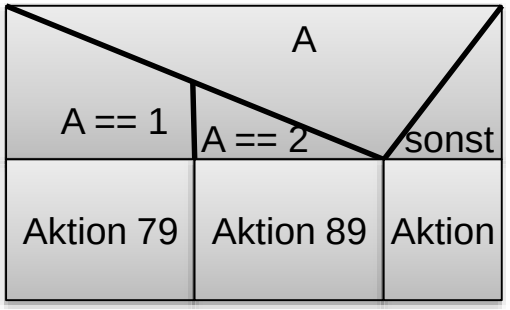
- Ein Nassi-Shneiderman-Diagramm ist ein Diagrammtyp zur Darstellung von Programmentwürfen im Rahmen der strukturierten Programmierung.
  - Struktogramme
- 1972/73 von Isaac Nassi und Ben Shneiderman entwickelt
- Genormt in der DIN 66261
- Die Methode zerlegt das Gesamtproblem in immer kleinere Teilprobleme, bis schließlich nur noch elementare Grundstrukturen wie Sequenzen und Kontrollstrukturen zur Lösung des Problems übrig bleiben.
- *Remark: Vorgehensweise entspricht der sogenannten Top-down-Programmierung, in der zunächst ein Gesamtkonzept entwickelt wird, das dann durch eine Verfeinerung der Strukturen des Gesamtkonzeptes aufgelöst wird.*

Nassi-Shneiderman-Diagramm

- Ein Nassi-Shneiderman-Diagramm ist ein Diagrammtyp zur Darstellung von Programmentwürfen im Rahmen der strukturierten Programmierung.
  - Struktogramme
- 1972/73 von Isaac Nassi und Ben Shneiderman entwickelt
- Genormt in der DIN 66261



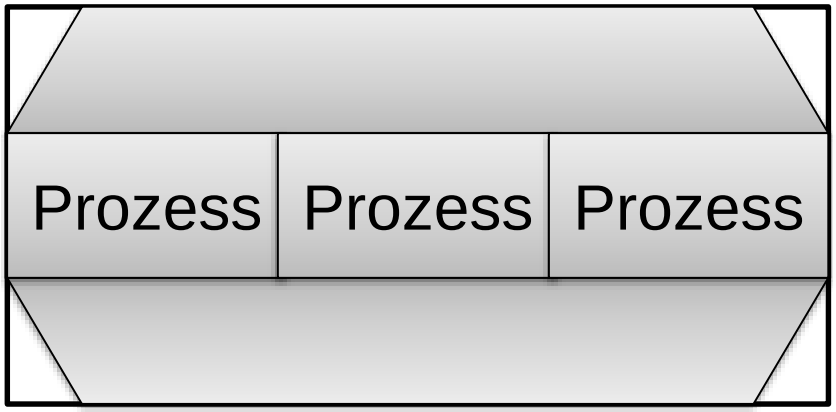




Nassi-Shneiderman: Prozesse

Prozess

- abgeschlossene Programmfunktionalität
- Prozesse untereinander anordnen, um eine Sequenz darzustellen

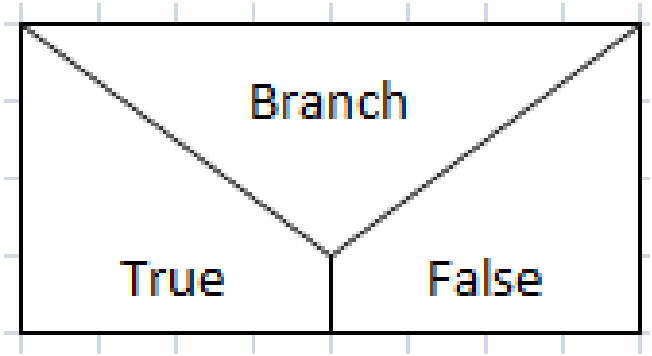


Parallele Prozesse

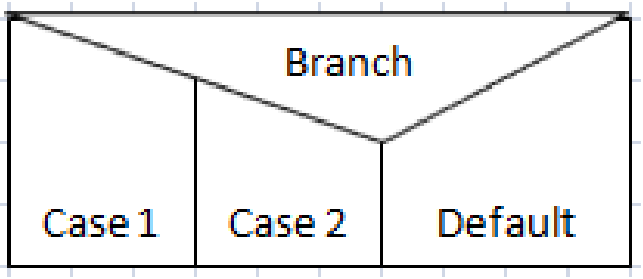
- Gleichzeitig ausgeführte Prozesse

Nassi-Shneiderman: Verzweigung

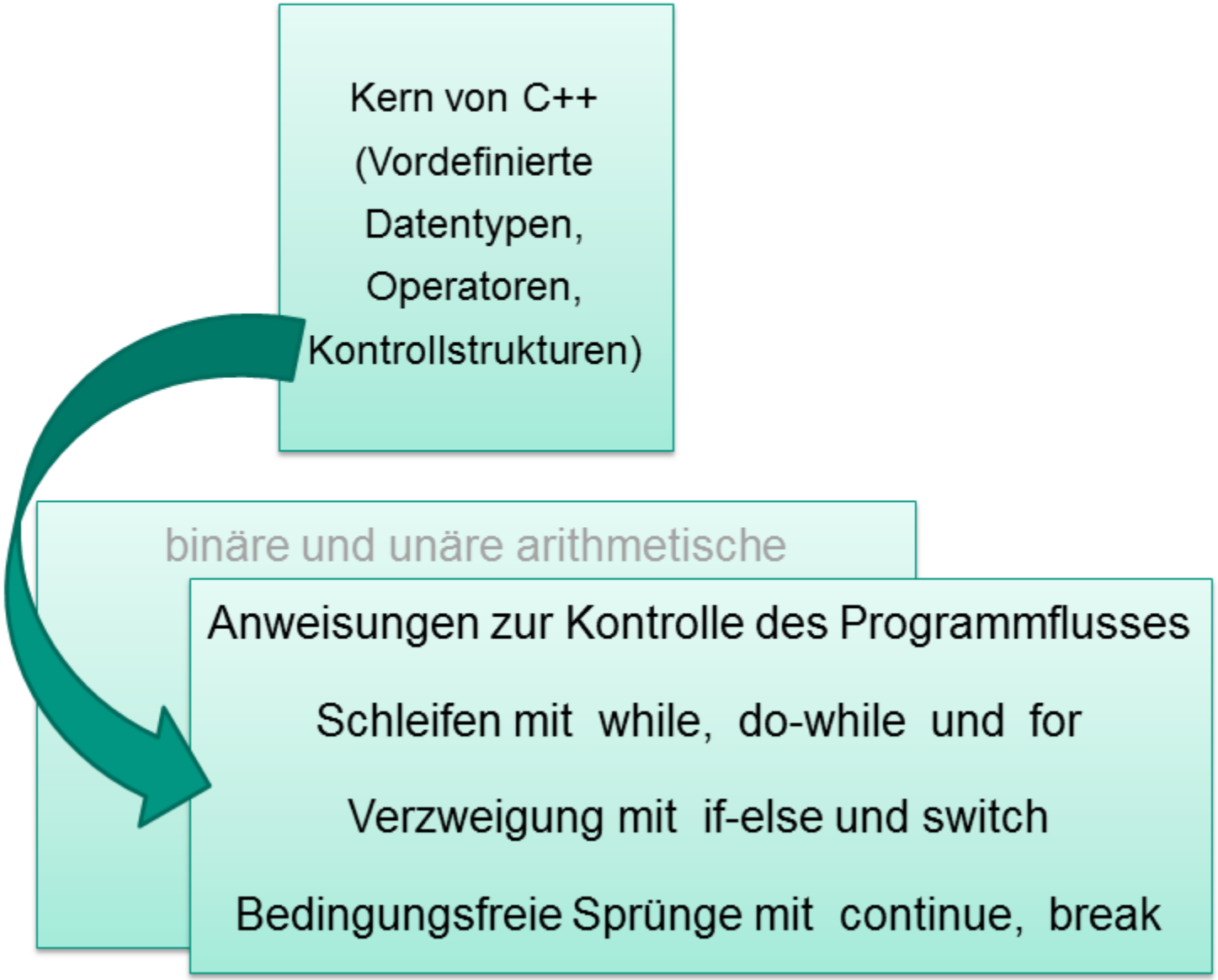
- Entscheidung
  - Überprüft Bedingung
  - Ausführung der unter der richtigen Antwort stehenden Blöcke



- Mehrfachentscheidung
  - Mehrere mögliche Fälle definiert
  - Ausführung der Blöcke unter zutreffendem Fall



C++ Operatoren

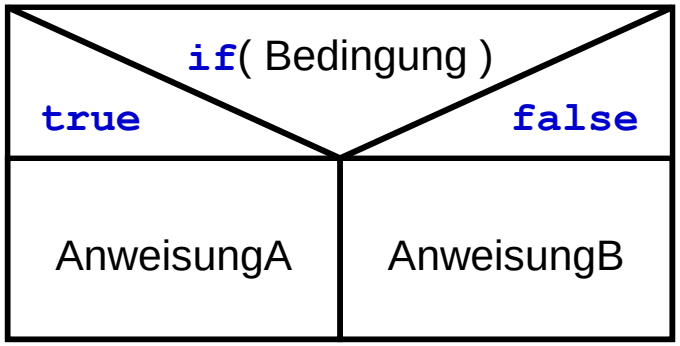


Verzweigungen, Einfach-Entscheidung

- Festlegen, welche Anweisung als nächstes ausgeführt wird
- Syntax:

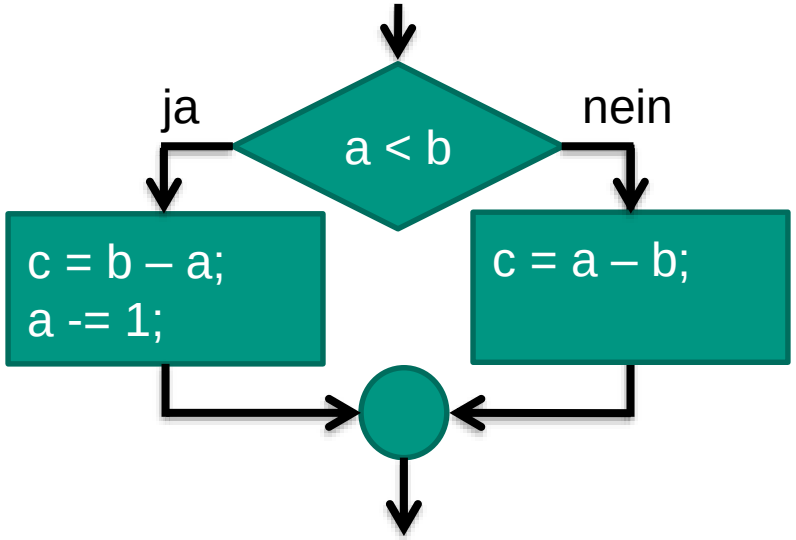
```
if( Bedingung ) {
          AnweisungA;
  } else {
          AnweisungB;
        }
```

Optional



- `else`-Verzweigung ist optional
- Beispiel:

```
if( a < b ) {
  c = b - a;
  a -= 1;
} else {
  c = a - b;
}
```



Verzweigung, Mehrfach-Entscheidung

- Festlegen, welche Anweisung als nächstes ausgeführt wird
  - `switch case` anstelle geschachtelter `if else`
- Syntax:

```
switch( Ausdruck ) {
        case 1:
        AnweisungA;
        break;

        case 2:
        AnweisungB;
        break;
      ...
        default:
        AnweisungC;
 }
```

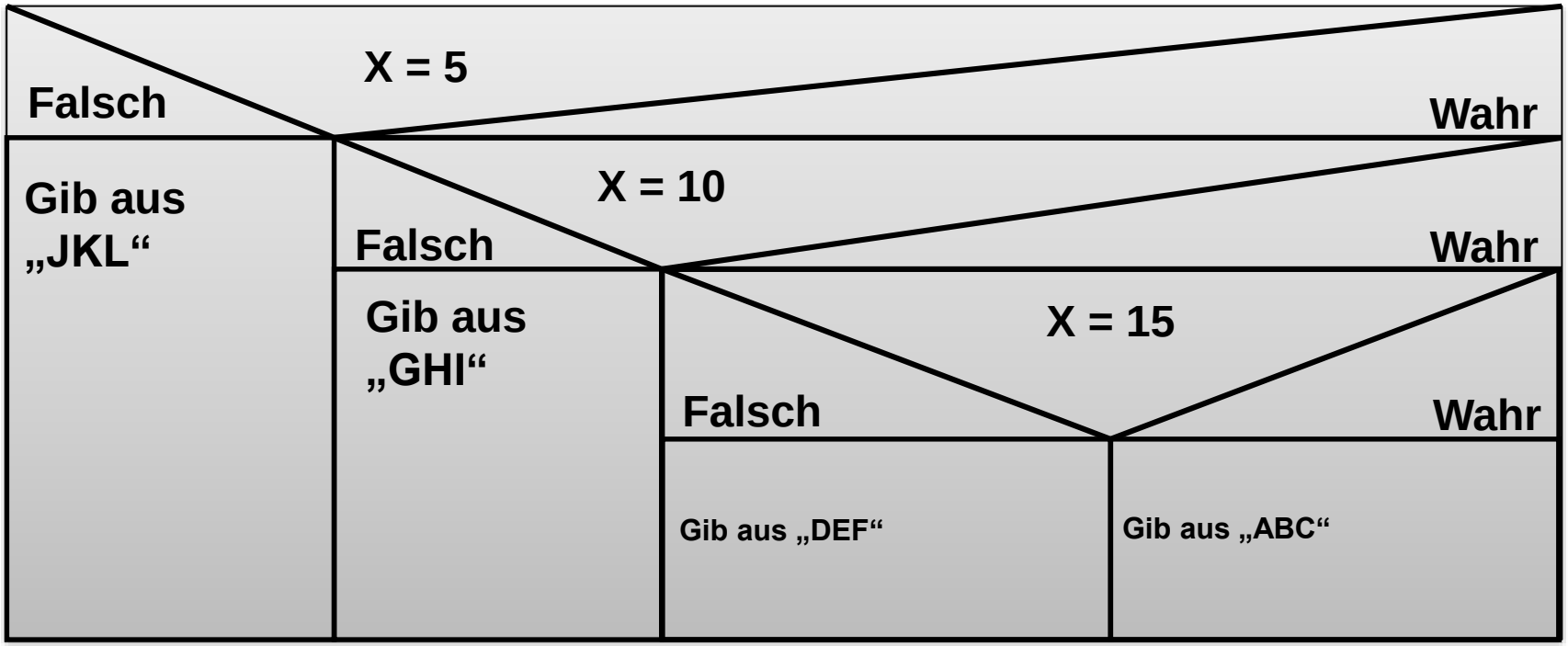
Optional

Beispiel:

```
char alpha;
...
switch
(alpha)
{
case 'e':
case 'E':
   Aktion1;
break;
case 'a':
case 'A':
   Aktion2;
}
```

  - „Ausdruck“ muss vom Typ ganzzahlig oder char sein!

Diagramm



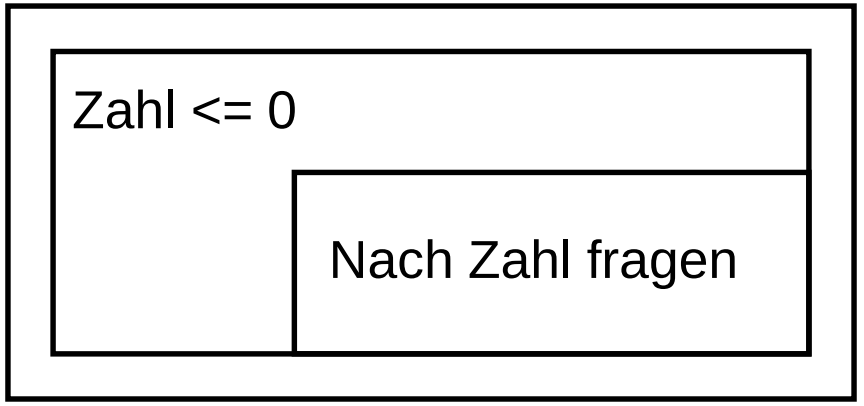

```
IF X == 5
   THEN IF X == 10
        THEN IF X == 15
           THEN DISPLAY „ABC“
           ELSE DISPLAY „DEF“
        ELSE DISPLAY „GHI“
   ELSE DISPLAY „JKL“
ENDIF
```

Merke:
Syntaktisch richtig notiertes Programm ergibt noch keinen logischen Programmablauf.

While - Schleife

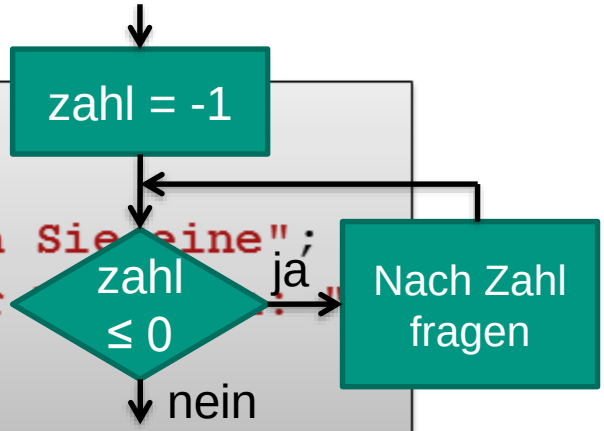
- Viele Anweisungen müssen mehrfach wiederholt werden
  - Feste Anzahl Wiederholungen oder abhängig von einer Bedingung
- **`while`**-Schleife - Kopfgesteuert
  - Bedingung **`true`** oder **`false`**
- Syntax:

```
while( Bedingung ) {
        Anweisung
      }
```

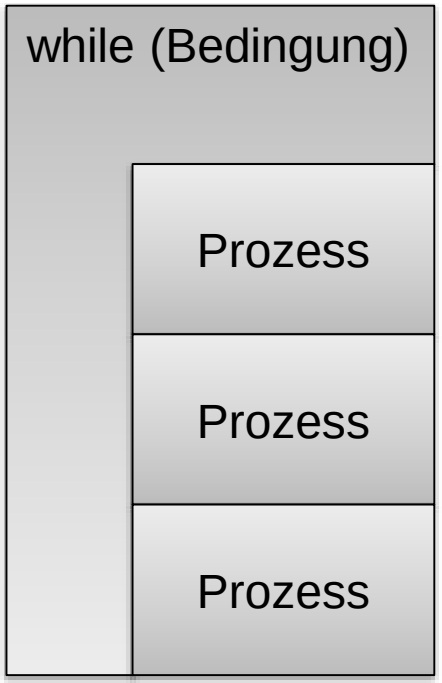


- Beispiel:

```
int zahl = -1;
      while( zahl <= 0 ) {
        cout << "Bitte geben Sie eine";
        cout << "Zahl größer
        cin >> zahl;
      }
```



Nassi-Shneiderman: Schleifen



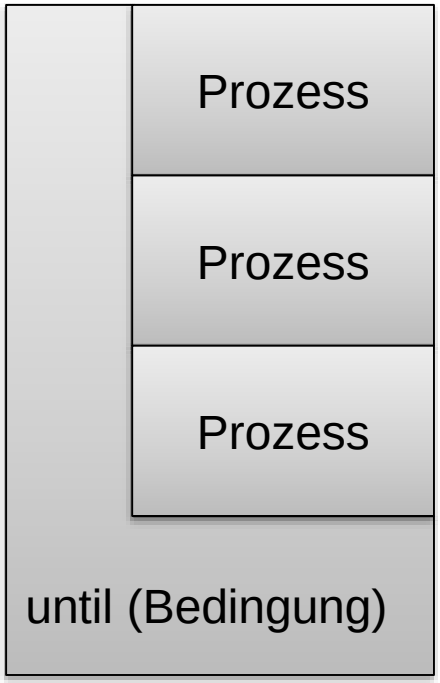
„While“-Schleife

- Wiederholt „umklammerte“ Blöcke solange, wie Bedingung zutrifft

„Do While“ bzw.

„Repeat Until“-Schleife

- Wiederholt „umklammerte“ Blöcke solange, bis Bedingung zutrifft

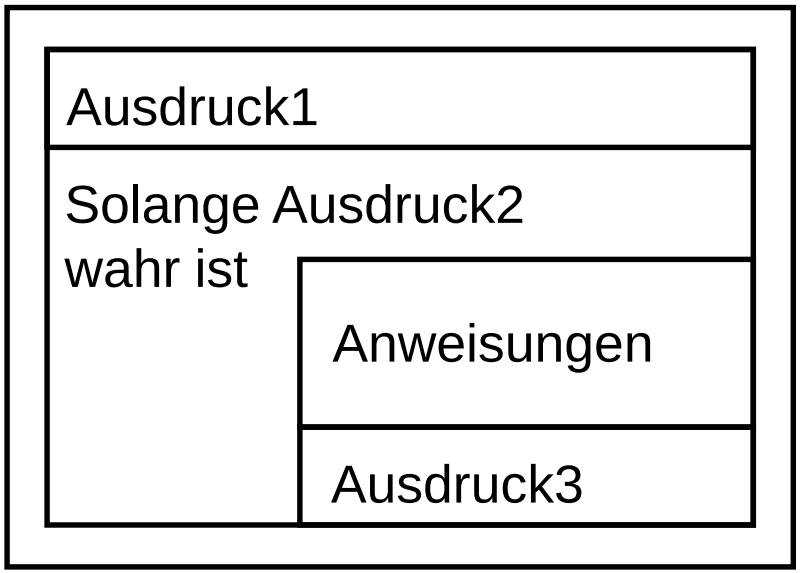


For – Schleife(1)

- Oft verwendet bei fester Anzahl an Durchläufen oder wenn eine Zählvariable benötigt wird
- Syntax:

```
for( Ausdruck1; Ausdruck2; Ausdruck3 ) {
        Anweisung;
    }
```

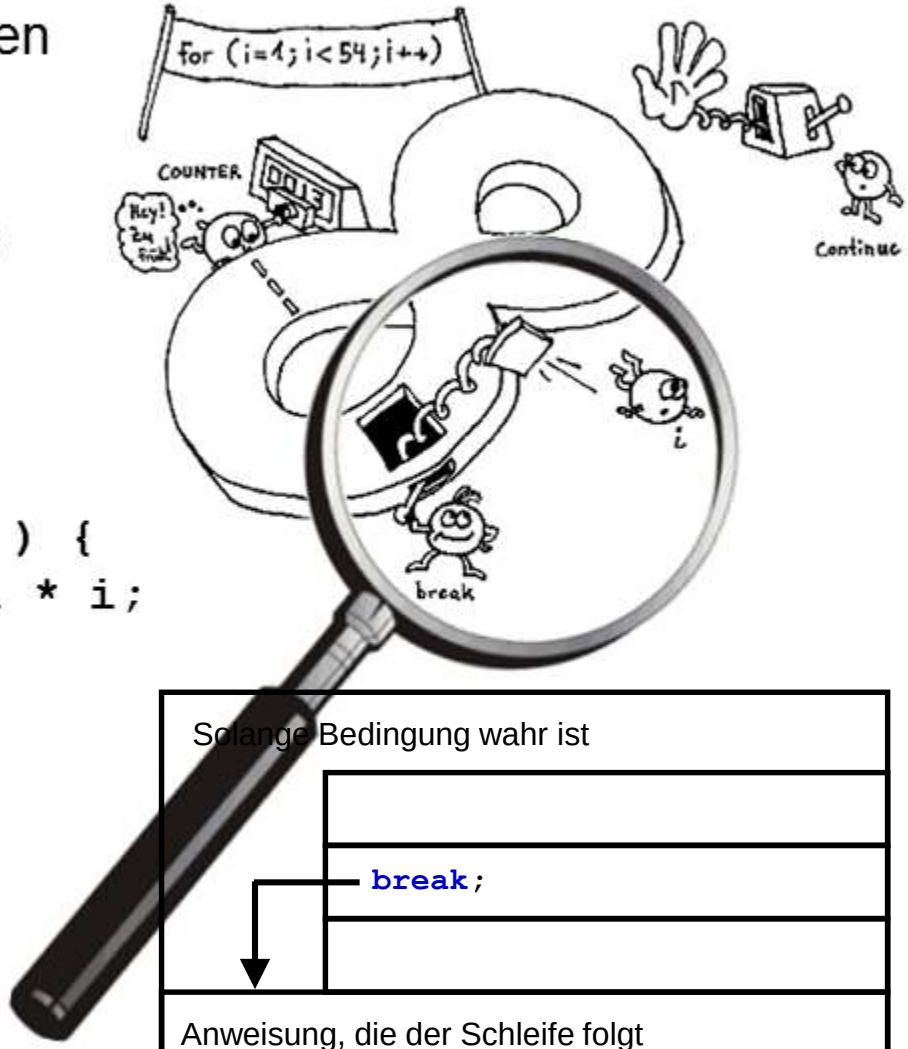
- Ausdruck1 = Initialisierung
  - Zu Beginn der Schleife einmalig ausgeführt
- Ausdruck2 = Bedingung
  - Schleife läuft, solange Bedingung wahr
- Ausdruck3 = Veränderung
  - Am Ende jedes Schleifen-Durchlaufs ausgeführt



For – Schleife(2) und break;

- Um eine Schleife sofort zu verlassen kann der Befehl break verwendet werden
  - Nützlich für besondere Ereignisse und Fehler-Behandlung
- Beispiel:

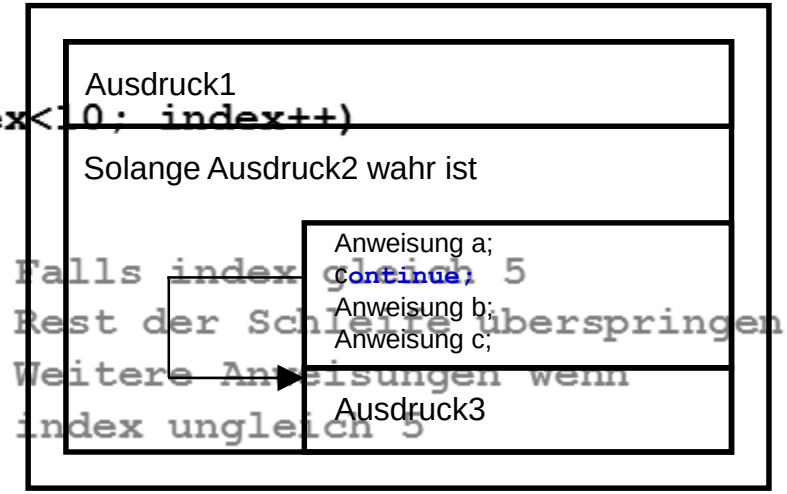
```
for( int i = 1; i < 54; i++ ) {
        cout << i << " " << i * i;
        cout << endl;
        if( i * i > 1000 ) {
          break;
        }
}
```



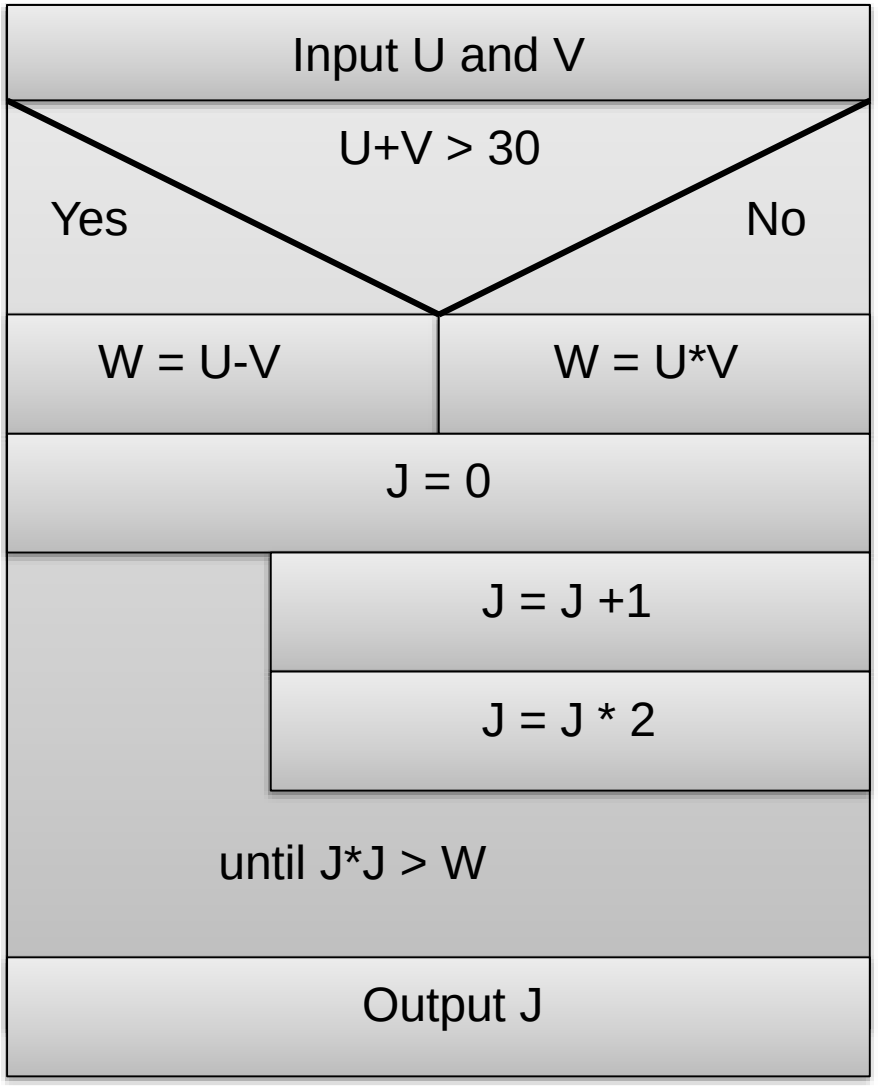
Schleifen und continue;

- Die continue-Anweisung ist nur innerhalb einer for- oder while-Schleife erlaubt.
- Sie bewirkt, dass die restlichen, der continue-Anweisung folgenden Anweisungen, übersprungen werden; die Schleife selbst wird aber nicht verlassen.
- Bei einer for-Schleife wird nach der continue-Anweisung mit der Auswertung des letzten Ausdrucks in der for-Klammer (Aktion pro Schleifendurchlauf) fortgefahren.
- Beispoiel:

```
int main ()
{
    ...
    for (int index=0; index<10; index++)
    {
        ...
        if (index == 5)  // Falls index gleich 5
            continue;    // Rest der Schleife überspringen
        ...              // Weitere Anweisungen wenn
                         // index ungleich 5
        ...
    }
    ...
}
```



Diagramm



$U = 5, V = 3$

$U + V = 8$

No

$W = 15$

$J = 0$

$J = 1$

$J = 2$

$J = 3$

$J = 6$

„6“

IT Inhalt

3. Programmstrukturen

- Programmablaufpläne
- Nassi-Shneiderman-Diagramme
- Gegenüberstellung

4. SW-Entwicklungsprozess

- Qualität von Software
- Entwicklungsablauf
- Verifikation & Validierung und Test

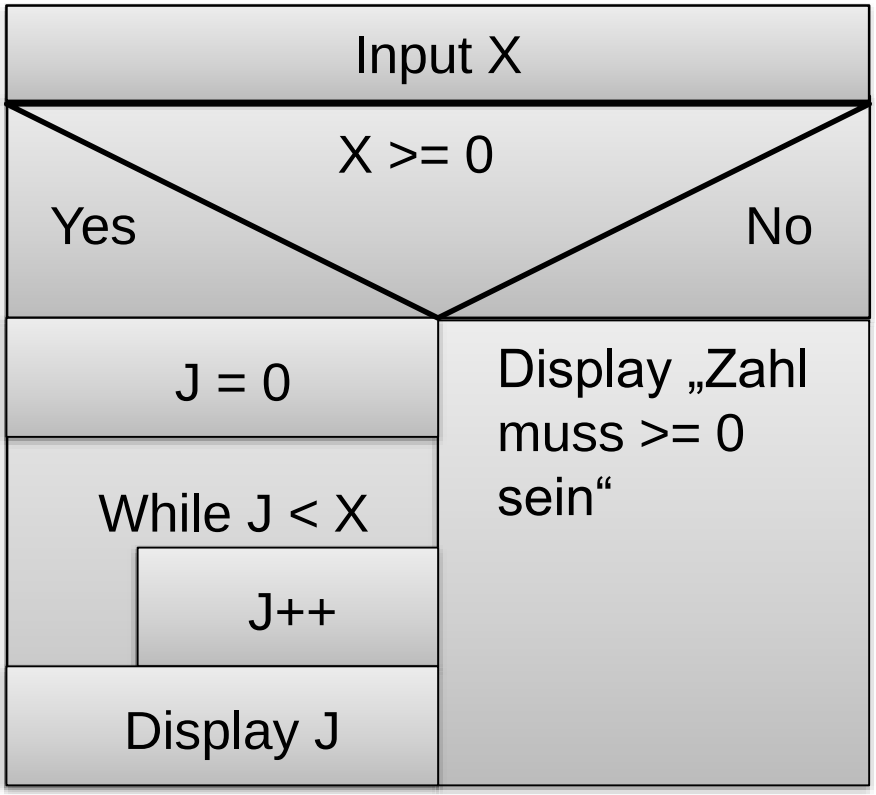
5. Entwicklungsumgebung

- Die Integrierte Entwicklungsumgebung
- Teile der Entwicklungsumgebung
- Vom Code zum Programm
- Ablauf Erstellung eines Programms

Beispiel: Nassi-Shneiderman

- Skizzieren Sie für folgenden Pseudo-Code ein Nassi-Shneiderman Diagramm:

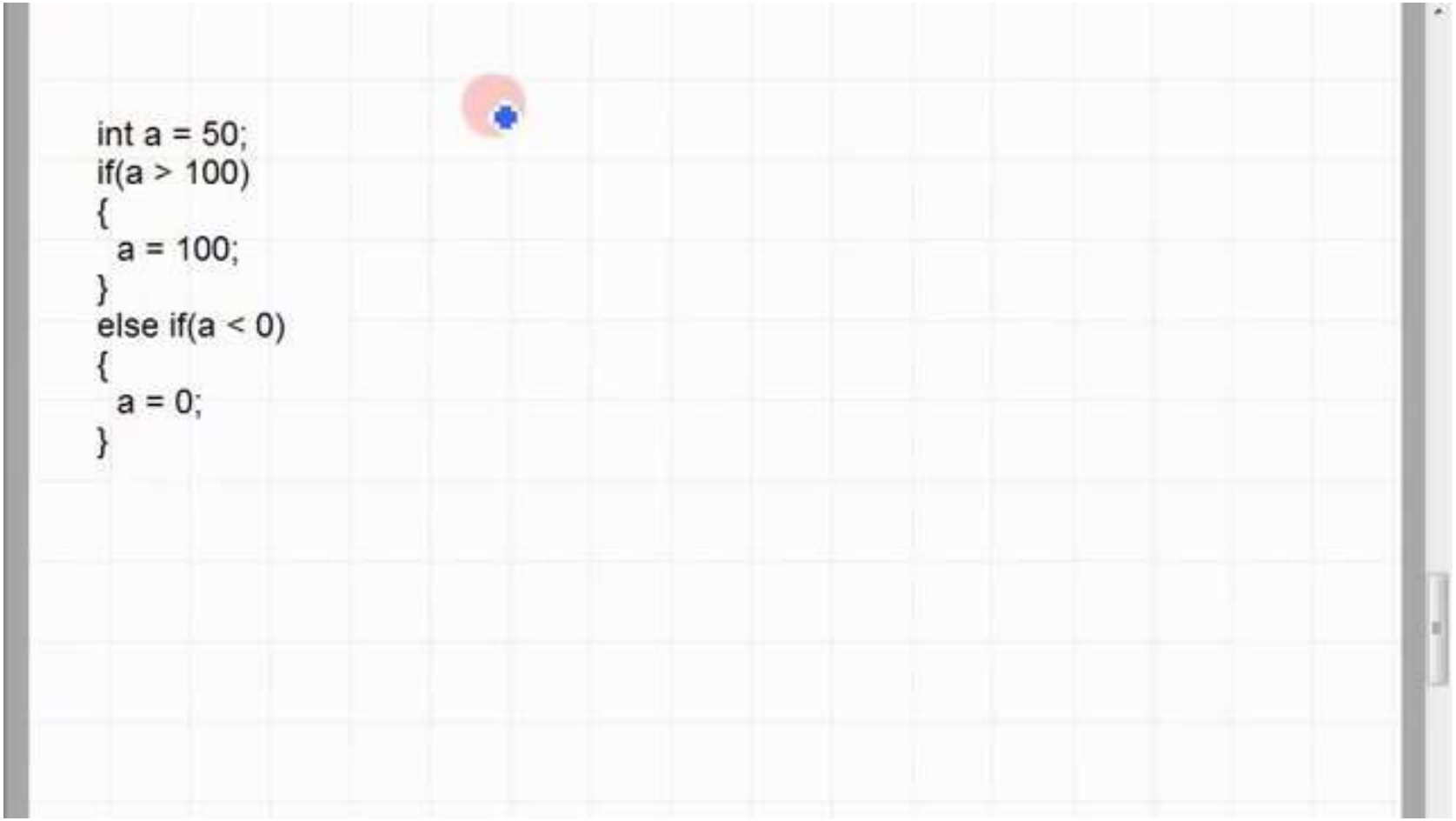
```
INPUT X
IF X >= 0
   THEN
      J = 0
      WHILE J < X
         J++
      DISPLAY J
   ELSE DISPLAY „Zahl muss >= 0 sein!“
ENDIF
```



Nassi-Shneiderman vs. PAP

Nassi-Shneiderman	Programmablaufplan
▪ repräsentiert gut das strukturierte Programmierparadigma ▪ Strukturierte Planung verhindert Sprungbefehle ▪ Blöcke fassen Schleifen eindeutig zusammen	▪ einfach zu zeichnen ▪ Symbole für viele konkrete Einsatzzwecke ▪ sehr leicht nachvollziehbarer Kontrollfluss
▪ von Hand schwer zu zeichnen ▪ alle konkreten Aufgaben sind nur Prozesse ▪ Blockform führt oft zum Eindruck der Überfülltheit	▪ Impliziert Programme, die mit Sprungbefehlen arbeiten ▪ Verschachtelte Schleifen nicht erkenntlich

Nassi-Shneiderman-Diagramm



- Nassi-Schneiderman Diagramme
  - Prozesse, sequentiell und parallel
  - if und switch case
- Schleifen
  - While-Schleife
  - Do-While-Schleife
  - For-Schleife
  - break, continue

Clever

Hiding Techniques


Backup
Shift