Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. K. D. Müller-Glaser
Prof. Dr.-Ing. J. Becker
Prof. Dr. rer. nat. W. Stork

Vorlesung: Informationstechnik (IT)

Prof. Dr.-Ing. K.D. Müller-Glaser

Institut für Technik der Informationsverarbeitung (ITIV)

Darstellung von Programmstrukturen

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

www.kit.edu

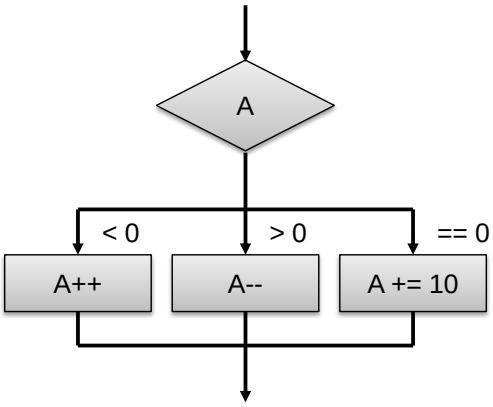
Übersicht

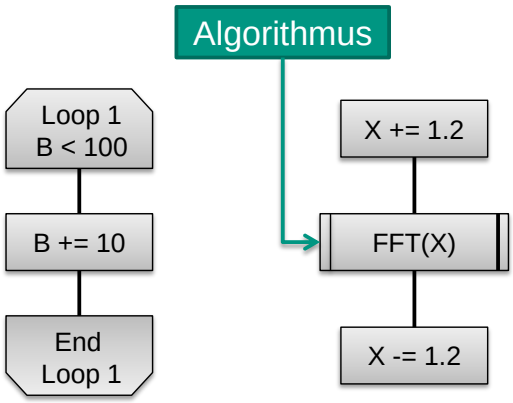
Karlsruher Institut für Technologie

- Programmablaufpläne
- Nassi-Shneiderman-Diagramme
- Gegenüberstellung

Programmablaufplan (PAP)

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





Bei technischen Abläufen der industriellen Produktion gestallten sich Abläufe, etwa die Entstehung eines Produkts, meist sequentiell und mit einfachen Auswahlkriterien. Zur Darstellung werden hierbei seit längerem Flussdiagramme eingesetzt.

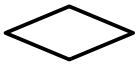
Für die Datenverarbeitung verabschiedete das Deutsche Institut für Normung 1969 einen Standard, um Programmabläufe zu dokumentieren: DIN 66001, Programmablaufplan.

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.



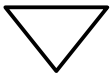
Entscheidung
Linien von den Ecken kennzeichnen die verschiedenen möglichen Programmzweige.

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.

Diese Folien sollen vorab einen kurzen Überblick über einen Teil der verwendeten Blöcke geben. Es handelt sich dabei um zum Teil sehr grundlegende Konzepte der imperativen und einfachen prozeduralen Programmierung.

Im ursprünglichen Standard finden sich aber auch noch viele Symbole für konkrete Geräte, die Sie heute allerdings meist nicht mehr in der praktischen Anwendung finden werden, Beispiel Lochkartenleser.

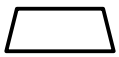
Flussdiagramm - Basiselemente

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.

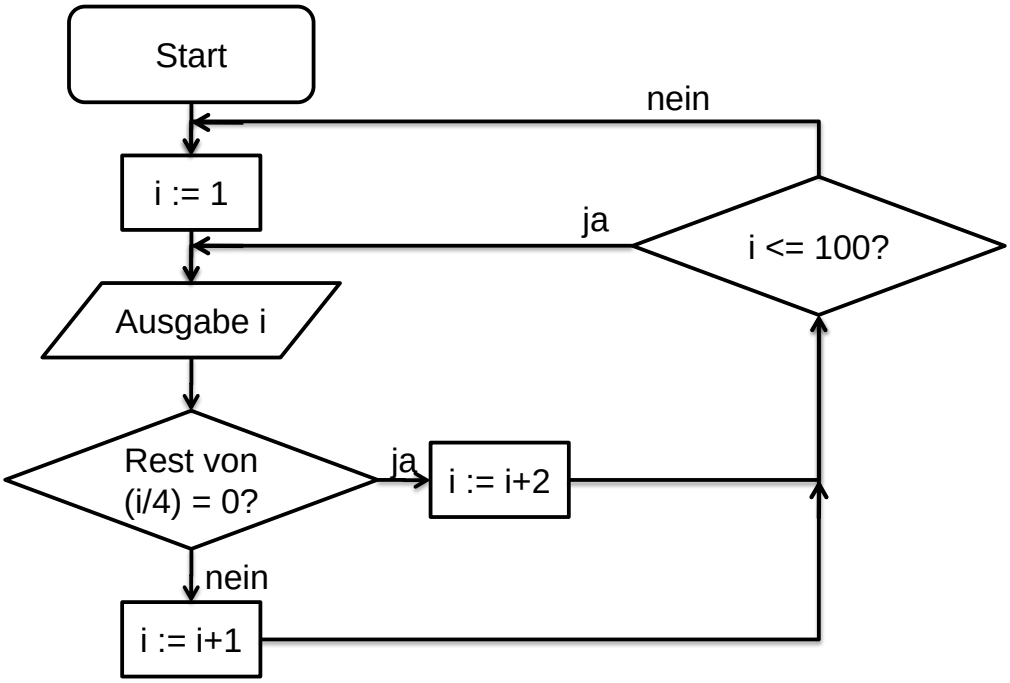


Schleifenende
Kennzeichnet den Punkt, an dem eine Schleife abbrechen soll

Datenspeicher
Kennzeichnet einen Speicher von Daten.

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

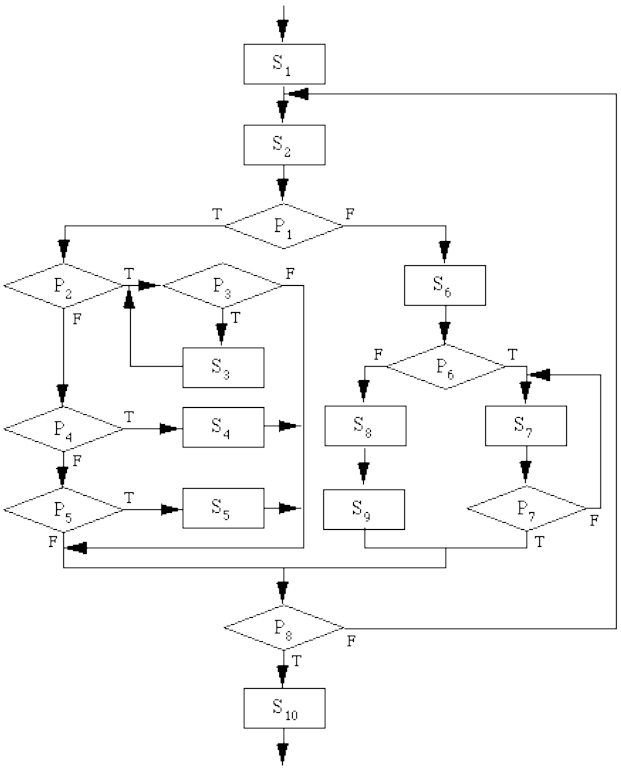
PAP-Beispiel



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

Ist der gezeigte PAP korrekt bezüglich dieser Anforderung?

PAP-Beispiel

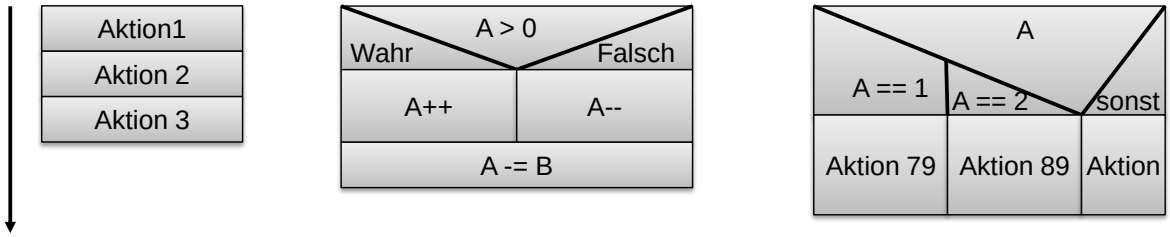


- PAP's werden schon bei kleineren Algorithmen schnell unübersichtlich.
- PAP's verführen zur Verwendung von expliziten Sprunganweisungen (GOTO's) und damit zur Produktion von „Spagetti-Code“
- Für grundlegende Kontrollstrukturen existieren keine Symbole, z. B. für Fallauswahlen, Schleifen
- Schachtelstrukturen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

Forderung nach
strukturierter Programmierung

Nassi-Shneiderman Diagramm

- Struktogramm
  resultierend aus Forderung nach strukturierter Programmierung
- Graphische Notation für Kontroll-Strukturen
- Standardisiert in DIN 66261
- Definiert von Nassi & Shneiderman 1973



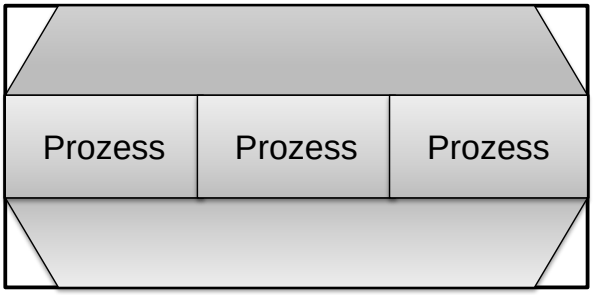
Beim Nassi-Shneiderman-Diagramm handelt es sich ebenfalls um eine graphische Darstellungsform von Programmabläufen. Hierbei wurde aber mehr Wert darauf gelegt, die Blockbildung durch Strukturen hervorzuheben.

Auch dieser Diagrammtyp wurde DIN-normiert. Wie die PAP laufen auch Nassi-Shneiderman-Diagramme sequentiell ab, allerdings werden die Schritte nicht mittels Pfeilen angegeben sondern sind implizit durch die vertikale Reihenfolge festgelegt.

Nassi-Shneiderman: Prozesse

Prozess

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



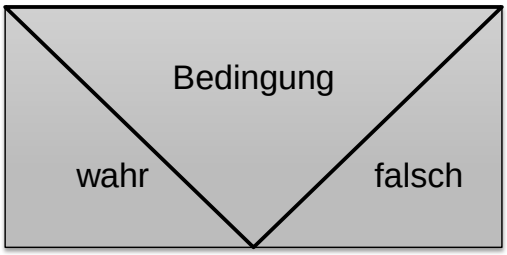
Parallele Prozesse

- Gleichzeitig ausgeführte Prozesse

Ein Prozess ist eine beliebig komplexe Abfolge von Anweisungen, etwa nur eine einzelne Wertzuweisung einer Variablen, bis hin zu einem komplexen Ablauf mit Entscheidungen und Iterationsschleifen.

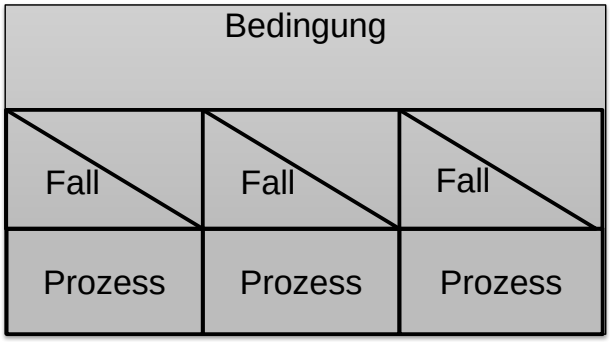
Parallele Prozesse sind zwei oder mehr Prozesse, die gleichzeitig ausgeführt werden.

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

In den meisten Programmiersprachen wird die Entscheidung durch das Schlüsselwort **if** symbolisiert. Diese wird in Nassi-Shneiderman-Diagrammen durch ein Dreieck dargestellt. Um nicht Mehrfachentscheidungen durch das Verschachteln von Einzelbedingungen darstellen zu müssen, gibt es auch hierfür eine Spezialform.

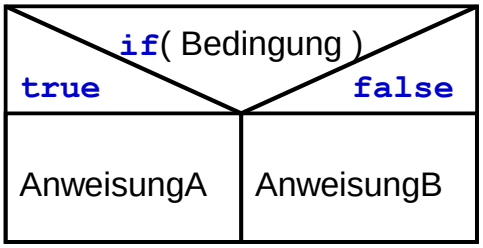
Verzweigungen, Entscheidung

- Festlegen, welche Anweisung als nächstes ausgeführt wird

- Syntax:

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

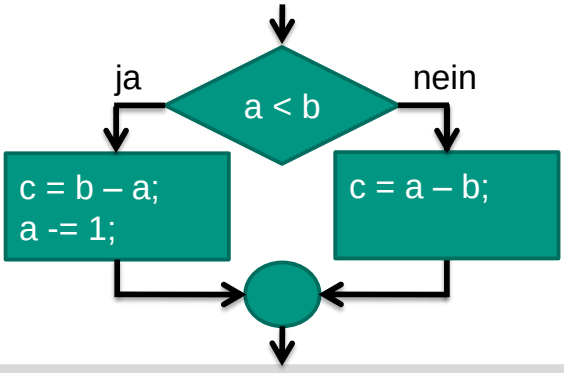
Optional



- `else`-Verzweigung ist optional

- Beispiel:

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



Verzweigung, Mehrfachentscheidung

- Festlegen, welche Anweisung als nächstes ausgeführt wird

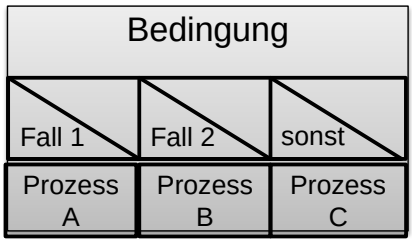
switch case anstelle geschachtelter if else

- Syntax:

```
switch( Ausdruck ) {
   case 1:
   AnweisungA;
   break;
   case 2:
   AnweisungB;
   break;
      ...
   default:
   AnweisungC;
}
```

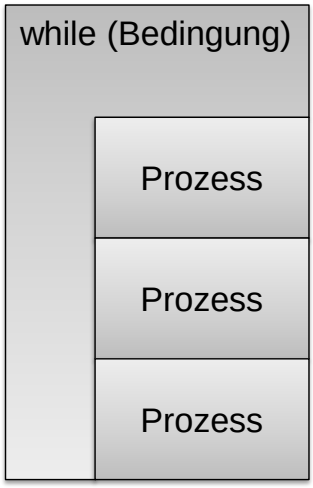
Optional

Ausdruck muss vom Typ ganzzahlig oder char sein!




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

Nassi-Shneiderman: Schleifen

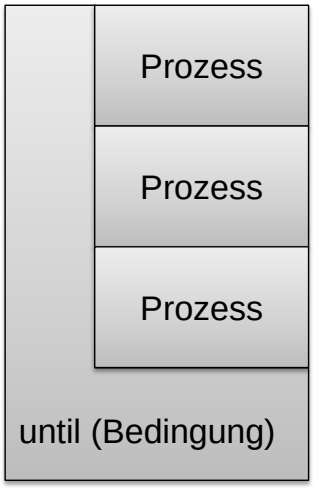


While-Schleife

- Wiederholt „umklammerte“ Blöcke solange, wie Bedingung zutrifft

Until-Schleife

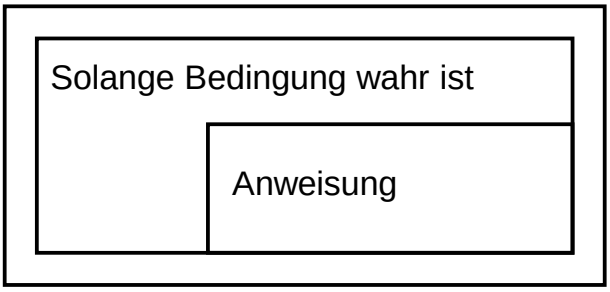
- Wiederholt „umklammerte“ Blöcke solange, bis Bedingung zutrifft



While - Schleife

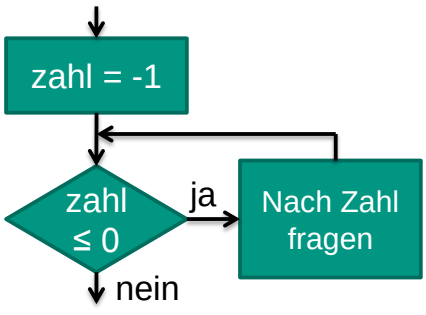
- Viele Anweisungen müssen mehrfach wiederholt werden
  - Feste Anzahl Wiederholungen oder abhängig von einer Bedingung
- **`while`**-Schleife - Kopfgesteuert
  - Bedingung **`true`** oder **`false`**
- Syntax:

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



- Beispiel:

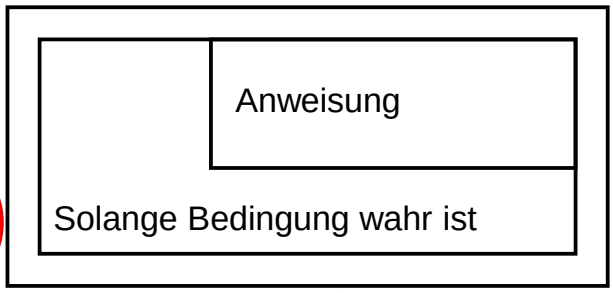
```
int zahl = -1;
while( zahl <= 0 ) {
  cout << "Bitte geben Sie eine";
  cout << "Zahl größer Null ein: ";
  cin >> zahl;
}
```



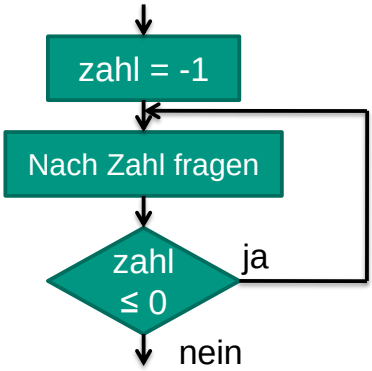
Do-While - Schleife

- Auch Fußgesteuerte Schleife genannt
  - Wird mindestens einmal ausgeführt (Unterschied zur **`while`**-Schleife)

- Syntax:
```
do {
      Anweisung
} while( Bedingung );
```



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

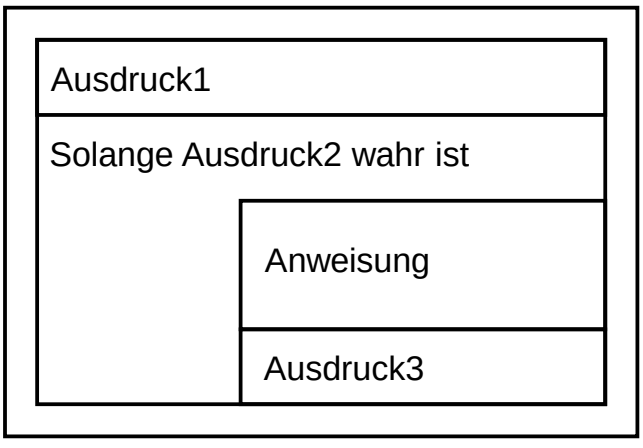


For – Schleife(1)

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

- Syntax:
```
for( Ausdruck1; Ausdruck2; Ausdruck3 ) {
      Anweisung;
}
```

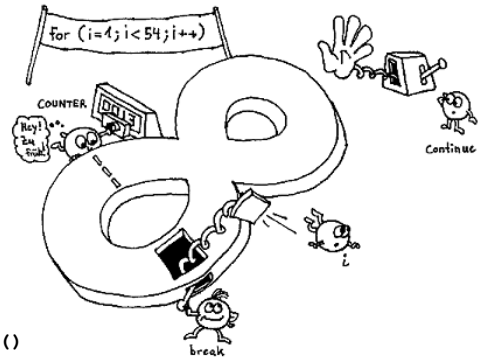
- Ausdruck1 = Initialisierung
  - Zu Beginn der Schleife einmalig ausgeführt
- Ausdruck2 = Bedingung
  - Schleife läuft, solange Bedingung wahr
- Ausdruck3 = Veränderung
  - Am Ende jedes Schleifen-Durchlaufs ausgeführt



For – Schleife(2) und break;

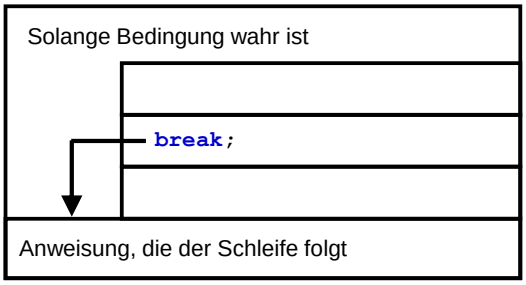
Um eine Schleife sofort zu verlassen
kann der Befehl break verwendet werden

Nützlich für besondere Ereignisse und Fehler-Behandlung



- Beispiel:

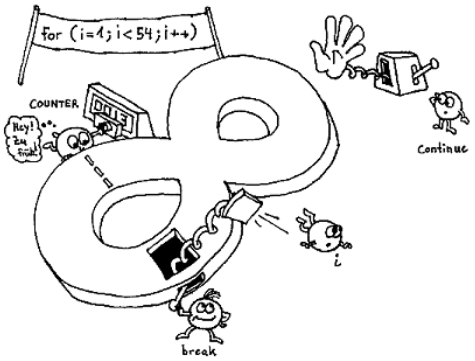
```
for( int i = 1; i < 54; i++ ) {
    cout << i << " " << i * i;
    cout << endl;
    if( i * i > 1000 ) {
      break;
    }
}
```



- Beispiel:

```
int main ()
{
   char inp;
   ...
   for (;;)     // Endlos-Schleife
   {
      ...
      // Ein Auswahlmenü ausgeben
      cout << "Auswahl? ";
      // Antwort einlesen
      cin >> inp;
      if (inp == 'E')       // Falls Auswahl E
          break;            // Schleife verlassen
      ...
   }
   ...
}
```

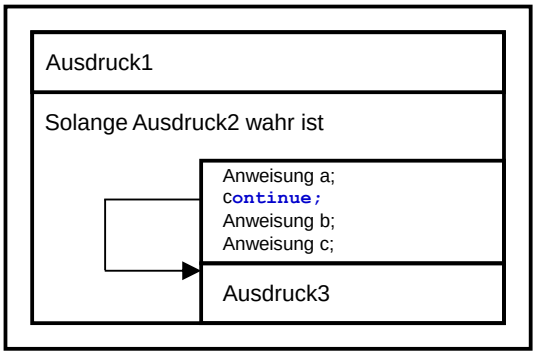

Schleifen und continue;



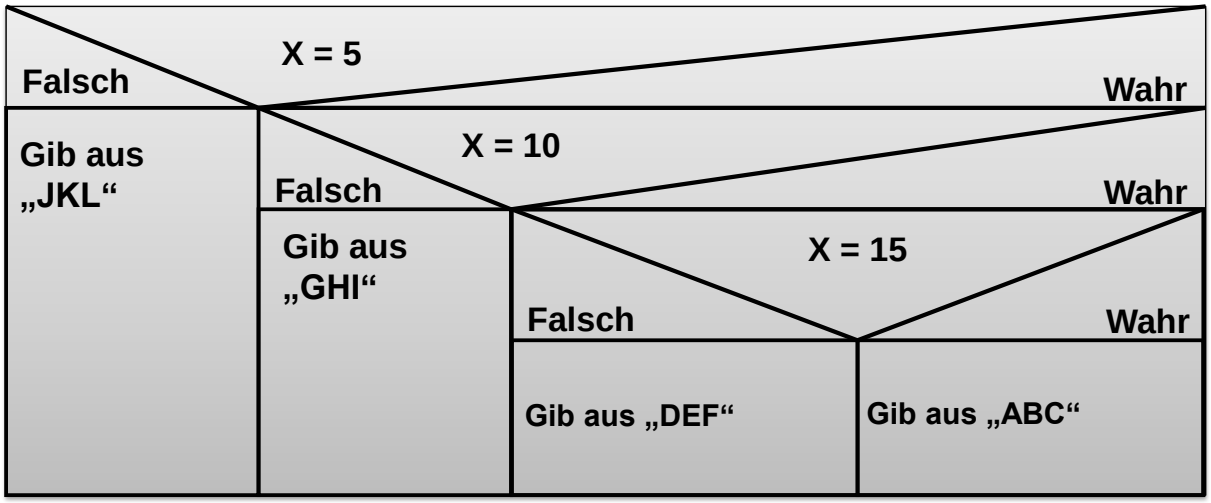

```
int main ()
{
   ...
   for (int index=0; index<10; index++)
   {
      ...
      if (index == 5)   // Falls index gleich 5
          continue;      // Rest der Schleife überspringen
      ...               // Weitere Anweisungen wenn index ungleich 5
      ...
   }
   ...
}
```

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.



Diagramm

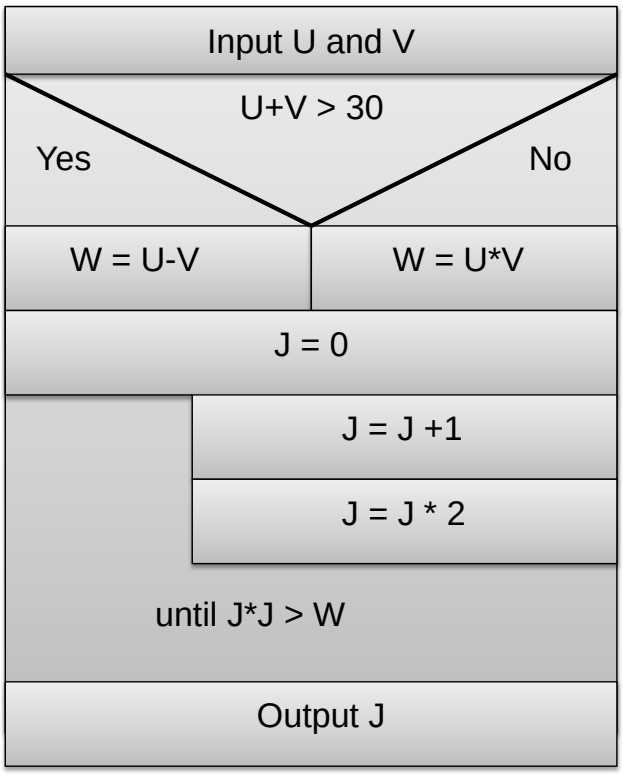



```
IF X == 5
    THEN IF X == 10
         THEN IF X == 15
              THEN DISPLAY „ABC“
              ELSE DISPLAY „DEF“
         ELSE DISPLAY „GHI“
    ELSE DISPLAY „JKL“
ENDIF
```

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

Wie erkenntlich schützt auch das graphische Notieren von Programmabläufen nicht vor logischen Fehlern.

Diagramm



Nassi-Shneiderman vs. PAP

Nassi-Shneiderman	Programmablaufplan
•repräsentiert gut das strukturierte Programmierparadigma •Planung mittels N-S 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

In der Vorlesung über Programmierparadigma haben Sie bereits von der strukturierten Programmierung gehört.

Da Nassi-Shneiderman-Diagramme schwer zu zeichnen sind, sollten Zeichenwerkzeuge verwendet werden.

- Microsoft Visio (Groß und mächtig, aber nur eingeschränkt brauchbar)
- Structorizer (http://structurizer.fisch.lu/) (kostenlos)
- SmartDraw (http://www.smartdraw.com/specials/flowchart.asp)

Zwischenübung01: 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
```