

Grundlagen Programmierung

(Praxis Programmierung mit C++)

# **Inhalte Teil I**

* Datentypen & Variablen
* Kontrollstrukturen
  + Verzweigungen ( if … else … )
  + Schleifen ( while … do … )
* Komplexe Datentypen
* Design
* Programmablaufplan
* Nassi-Shneiderman Diagramm
* Pseudo Code

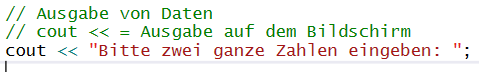
# **Inhalte Teil II**

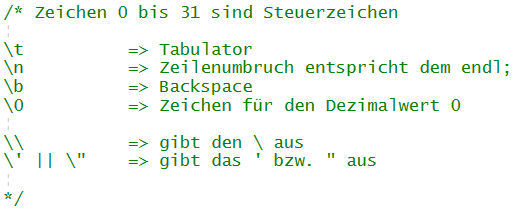
* Funktionen
* Bibliotheken
* Fehlerbehandlung

# **Klausur**

Termin: 10.12.2020  
Dauer: ca. 3 Std.  
Themen: Inhalt Teil I & Teil II

# **Kommentare**

**// Zeilen Kommentar**Nach dem Kommentar endet die Zeile & der Kommentar  


**/\* Block Kommentar \*/**  


# **Trennzeichen**

Trennzeichen dienen dazu Zeichenketten zu trennen oder zu verknüpfen.  
Auch schließen diese eine Zeile oder ein Codeabschnitt ab.

* Leerzeichen
* Semikolon ; | Kommando Ende
* Kommentare
* Operatoren
  + Z.b. + - \* / (Mathematische Operatoren)
* Komma

# **Bezeichner**

Die Bezeichner sind die Namen von Klassen, Funktionen oder Variablen.  
Sie können Alphanumerisch sein, müssen aber bestimme Regeln einhalten.

* Ziffern, Buchstaben & Unterstriche sind erlaubt  
  Sonderzeichen, Umlaute sind nicht erlaubt
* Das erste Zeichen darf keine Nummer sein   
  int 0Null ist nicht erlaubt  
  int Null0 ist erlaubt  
  int \_0Null ist erlaubt
* Groß- & Kleinschreibung wird unterschieden  
  int Null und int null sind zwei verschiedene Variablen
* Bezeichner dürfen keine Schlüsselwörter sein

# **Programmiersprachen**

Es gibt viele verschiedene Programmiersprachen. Jede Programmiersprache hat seine eigenen Schlüsselwörter und Syntax. Die Schlüsselwörter und Syntax sind teilweise in verschiedenen Programmiersprachen ähnlich.

|  |  |
| --- | --- |
| **Übersetzte Programmiersprachen** | **Interpretierte Programmiersprachen** |
| C / C++, C#, Fortan, Cobol, Visual Basic | PHP, JavaScript, PowerShell, Python,  Visual Basic Script (VBS) |
| Explizierte Datentypen   * Datentypen müssen FEST zugeordnet werden   Int, char, float, long, string, …. | Implizierte Datentypen   * Datentypen werden anhand des Wertes zugeordnet   123 = int 12.3 = double “123“ = text |

# **Zusammensetzung eines Programms**

**\*.h**  
**\*.cpp**  
Header Datei   
Quellcode / Programmcode

**COMPILER  
Verwandelt Header & Quellcode  
in ein Objekt um**

**\*.obj**

**BINDER  
(Deutsch: Linker)**

**\*.lib**    
Bibliotheken

**\*exe**  
Ausführbare Datei

# **Datei Aufbau**

Eine Datei ist immer in mehreren “Abschnitte“ unterteilt.

Compiler Anweisungen

\*cpp  
Quellcode / Programmcode

Definitionen  
Deklarationen

Klassen  
Hauptfunktionen

**Compiler Anweisung**

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

**Definition / Deklarationen**

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

**Klassen / Hauptfunktionen**

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

# **Nassi-Shneiderman Diagramm**

Ein Nassi-Shneiderman-Diagramm ist ein Diagrammtyp zur Darstellung von Programmentwürfen im Rahmen der Methode der strukturierten Programmierung.

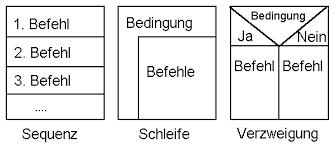


Abbildung | Nassi Shneiderman Blöcke

# **Programmablaufplan (PAP)**

Ein Programmablaufplan ist ein Ablaufdiagramm für ein Computerprogramm, das auch als Flussdiagramm oder Programmstrukturplan bezeichnet wird. Es ist eine grafische Darstellung zur Umsetzung eines Algorithmus in einem Programm und beschreibt die Folge von Operationen zur Lösung einer Aufgabe.

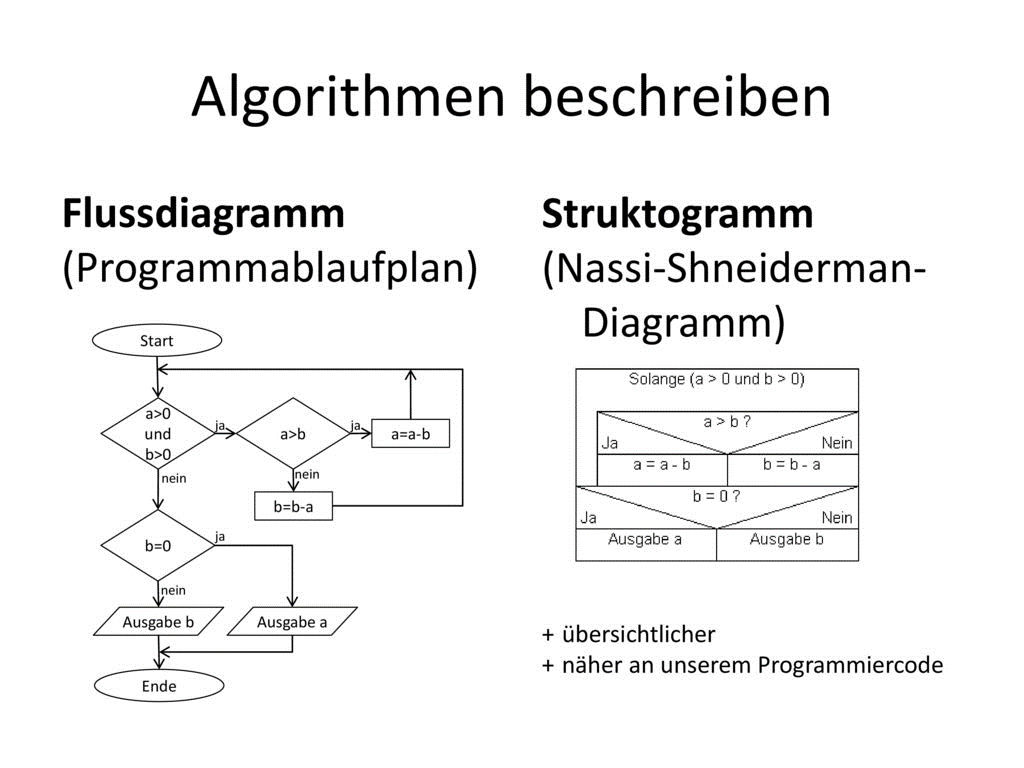


Abbildung | Links PAP - Rechts Nassi-Shneidman Diagramm

# **Datentypen**

Datentypen definieren den Typ von Daten (Werten) der gespeichert werden kann.

|  |  |  |
| --- | --- | --- |
| Name | Wert | Größe |
| bool | true / false | 1 Byte |
| char | -128 bis 127 | 1 Byte |
| Ganze Zahlen z.B. 123 | | |
| short | −32.768 bis 32.767 | 2 Byte |
| long | –2.147.483.648 bis 2.147.483.647 | 4 Byte |
| long long | –9.223.372.036.854.755.808 bis 9.223.372.036.854.755.807 | 8 Byte |
| Komma Zahlen z.B. 1.23 | | |
| float | 1.2E-38 bis 3.4E+38 | 4 Byte |
| double | 2.3E-308 1.7E+308 | 8 Byte |
| C++ Only | | |
| long double | 3.4E-4932 1.1E+4932 | 12 Byte |

# **Modifizierer**

Dienen dazu Datentypen zu bearbeiten

|  |  |
| --- | --- |
| Name | Beschreibung |
| const | Konstante | der Wert des Datentyps kann nicht verändert werden |
| signed | Datentyp mit Vorzeichen | Standard |
| unsigned | Datentyp ohne Vorzeichen |

# **Variablen**

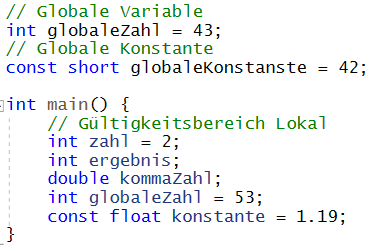
Syntax:

*Modifizierer + Datentyp + Bezeichner = Wert;*

Modifizierer und Wert sind Optional. Sollte der Modifizierer **“const“** angeben werden, so ist der Wert zwingend erforderlich.

**int** *zahl* = 0; **unsigned char** *zeichen* = 255;  
**long** *longzahl* = 0; **bool** *logisch* = true  
**short** *shortzahl* = 0; **float** *kommazahl* = 3.14;  
  
**const** **double** *pi* = 3.14;  
Der Wert *pi* muss direkt angegeben werden.

# **Gültigkeitsbereich**

Der Gültigkeitsbereich gibt einen Bereich an wo eine Variable sichtbar ist.  
Es gibt den Globalen und den Lokalen Gültigkeitsbereich.  
  
**Globale Sichtbarkeit**Eine Variable die außerhalb von { } definiert werden sind Global und von überall in der Datei zur Verwendung frei gegeben.  
  
**Lokale Sichtbarkeit**  
Wird eine Variable innerhalb von { } definiert, so ist diese nur in den vorher bestimmten teil des Quellcodes gültig und verwendbar.  
  


**Hierarchie**Variablen folgen einer bestimmten Hierarchie.  
 **Global -> Lokal**Sollte eine Globale Variable Lokal nochmal neu definiert werden, so zählt die Lokale Variable für den Programm abschnitt.

# **Implizierte / Explizierte Umwandlung**

Bei Umwandlung von Datentypen geht es nur von dem aktuellen Datentyp auf den nächst kleineren (Implizierte Umwandlung) oder auf den nächst größeren (Explizierte Umwandlung) Datentyps  
  
**Impliziert**von klein auf großbool -> char -> short -> …

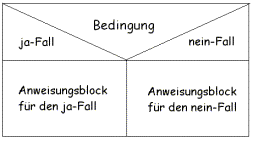
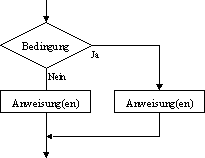
**Expliziert**von groß auf kleinlong double -> double -> …

long long -> long -> … -> bool

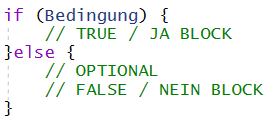
# **Kontrollstrukturen**

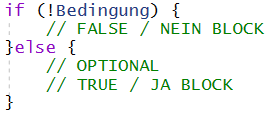
Mit Kontrollstrukturen lassen sich Variablen mit einer Bedingung prüfen. So kann ein Codeblock erst durchgeführt werden, wenn die Überprüfung TRUE oder FALSE ist.  
Eine weitere Kontrollstruktur sind die Schleifen mit denen wir ein Codeblock solange durchlaufen lassen können bis die Bedingung erfüllt wurde.  
  
  
**Verzweigung**

* Einfache Verzweigungen
  + Ja / Nein Entscheidung
  + Operatoren Entscheidung

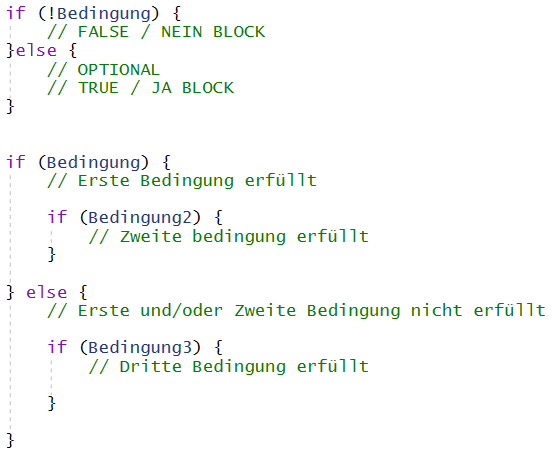
**Syntax:**

  
Um nur den **FALSE / NEIN** Block zu erhalten, muss die Bedingung mit einem **!** negiert werden.

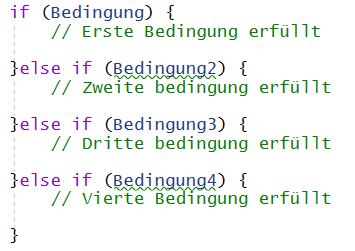


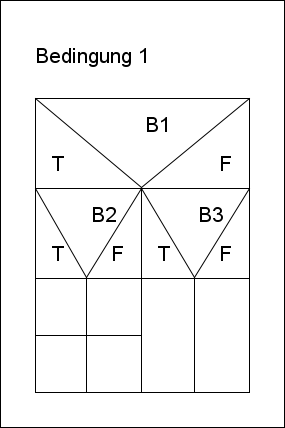
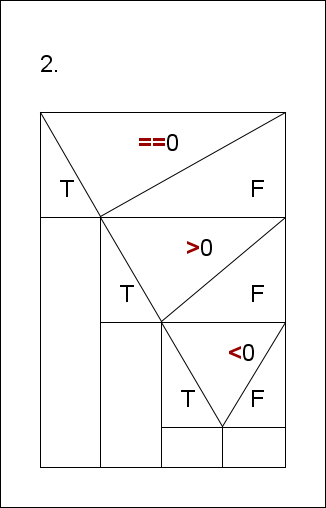
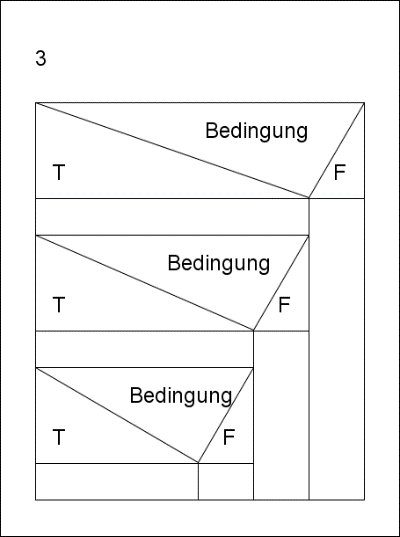
* Komplexe / Mehrfache Verzweigungen

**if else**

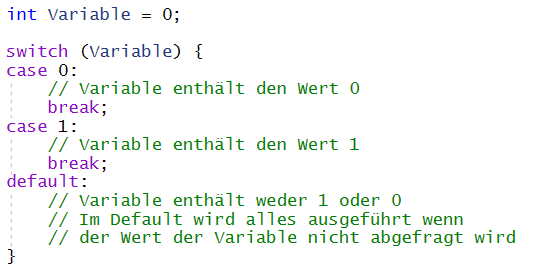


**If else if**

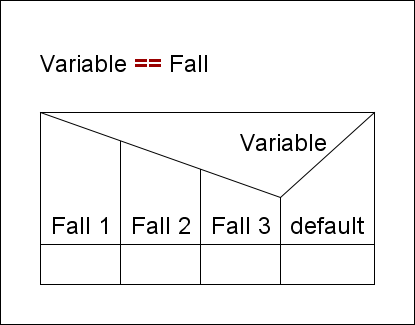


**Switch**

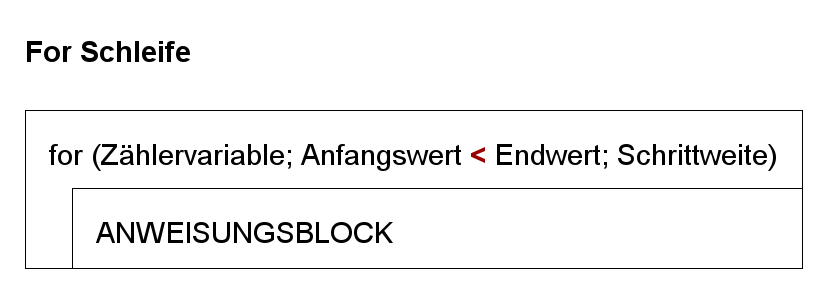


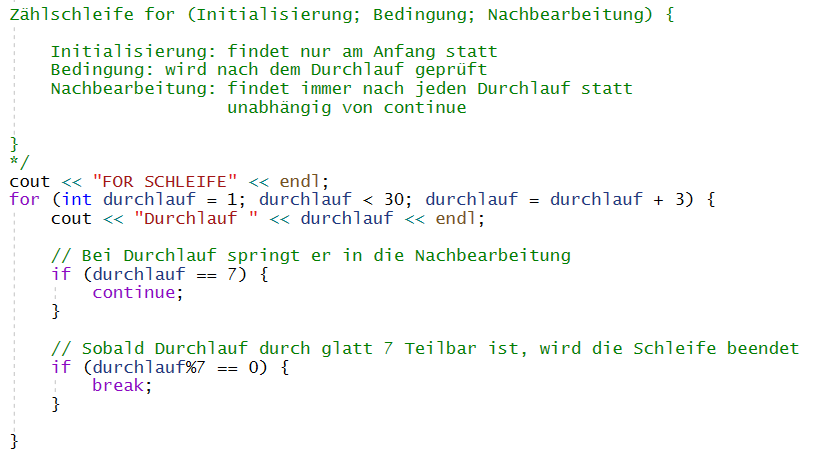
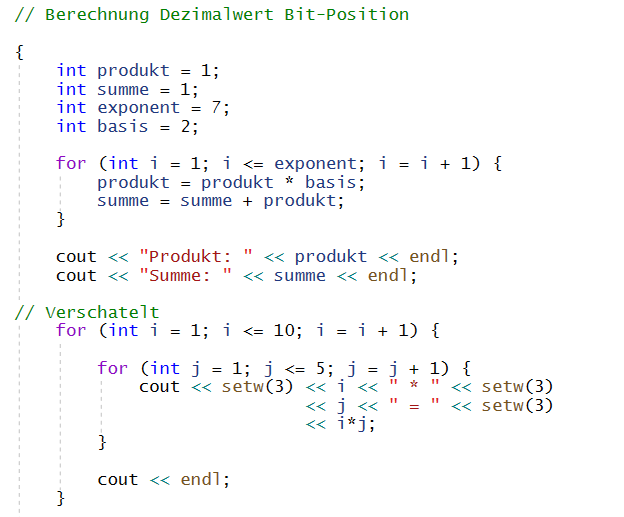
Die Variable zur Überprüfung muss Nummerisch sein, damit diese im **CASE** (Fall) abgefragt werden kann. Sollte die Variable mit dem des **CASE** übereinstimmen, wird der Codeblock bis zum **BREAK**; ausgeführt.  
Es wird immer bis zum nächsten **BREAK**; abgearbeitet, auch wenn dieser erst im nächsten **CASE** steht.

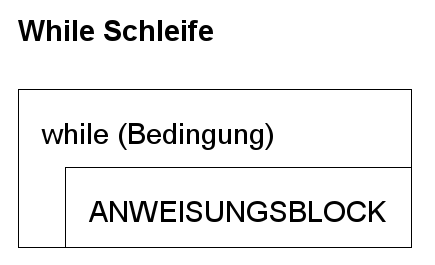
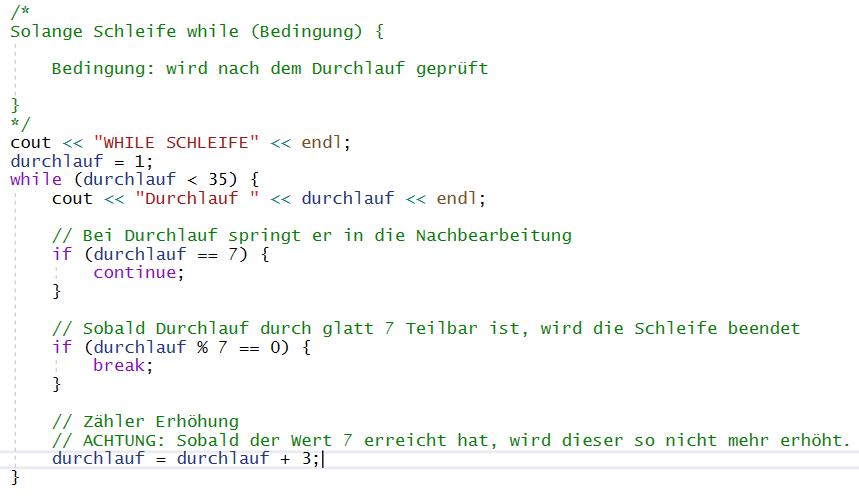


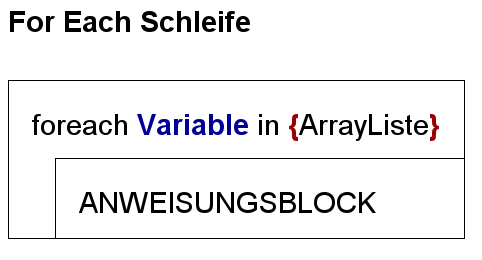
**Schleifen**

* Kopfgesteuert
  + Zählschleife / for
  + so lange bis / while
  + über eine Liste / for each

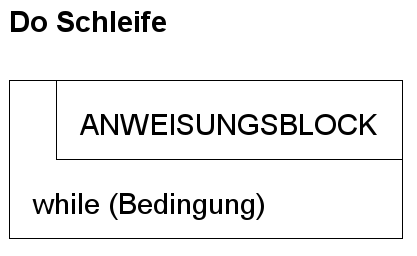


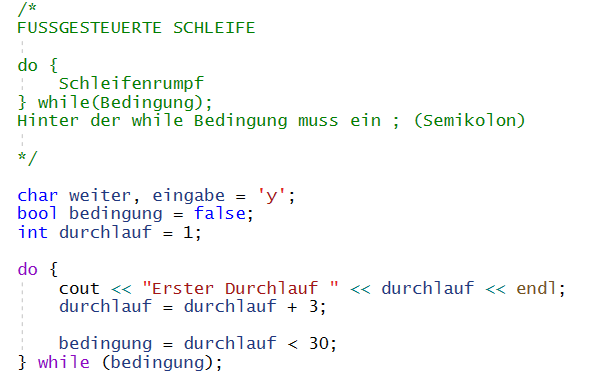
  


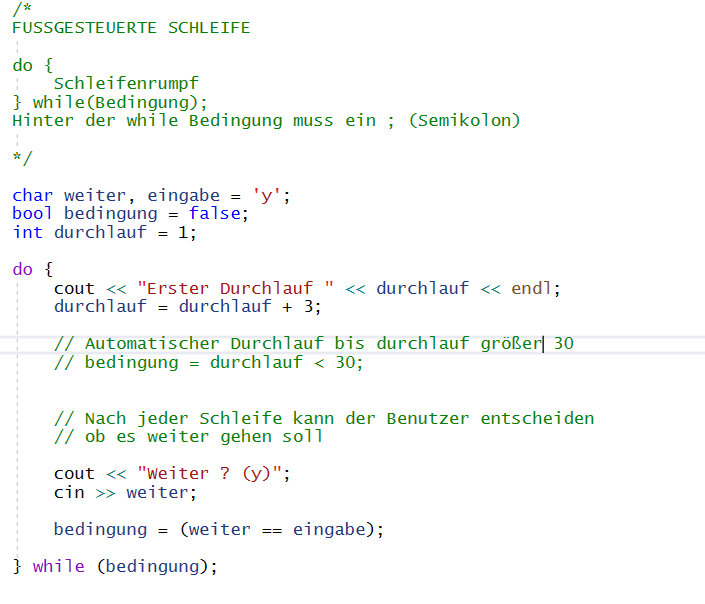
  




* Fussgesteuert
  + so lange bis







**Bedingungen**Die Bedingungen werden genutzt um Entscheidung bei den Verzweigungen & Schleifen fest zu legen.

* Einfache Bedingung
  + Wert Operator Vergleichswert  
    3 < 4 || “Hallo“ == “Guten Tag“  
    Gibt TRUE oder FALSE zurück

**Komplexe Bedingungen**Komplexe Bedingungen werden mit Hilfe eines Operators “verknüpft“. So kann man Prüfen ob die eine Bedingung erfüllt ist oder eine andere oder ob alle Bedingungen erfüllt oder nicht erfüllt wurden.

* Logische Operatoren
  + **||** ODER
  + **&&** UND
  + **!** NICHT

**Syntax:**

Bedingung1 Operator Bedingung2

**Operatoren**Damit die Bedingung auch weiß was und wie sie etwas vergleichen soll, gibt es die Operatoren.

* Einfache Vergleichsoperatoren
  + **<** größer als
  + **>** kleiner als
  + **<=** größer oder gleich als
  + **=>** kleiner oder gleich als
  + **==** genau gleich
  + **!=** nicht gleich

**Konditionaloperator**Es ist eine vereinfachte **if-Anweisung** die aber direkt einer Variablen zugewiesen oder im Output verwendet werden kann.





Die Ausgabe hier wäre: *Wert ist: gleich 0*  
Wäre die Variable nicht 0 würde in der Ausgabe stehen: *Wert ist: ungleich 0  
  
  
TO DO*