



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

```
// Ausgabe von Daten
// cout << = Ausgabe auf dem Bildschirm
cout << "Bitte zwei ganze Zahlen eingeben: ";
|
```

/* Block Kommentar */

```
/* Zeichen 0 bis 31 sind Steuerzeichen
:
\t          => Tabulator
\n          => Zeilenumbruch entspricht dem endl;
\b          => Backspace
\0          => Zeichen für den Dezimalwert 0
:
\\          => gibt den \ aus
\' || \"    => gibt das ' bzw. " aus
:
*/
```

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 bestimmte Regeln einhalten.

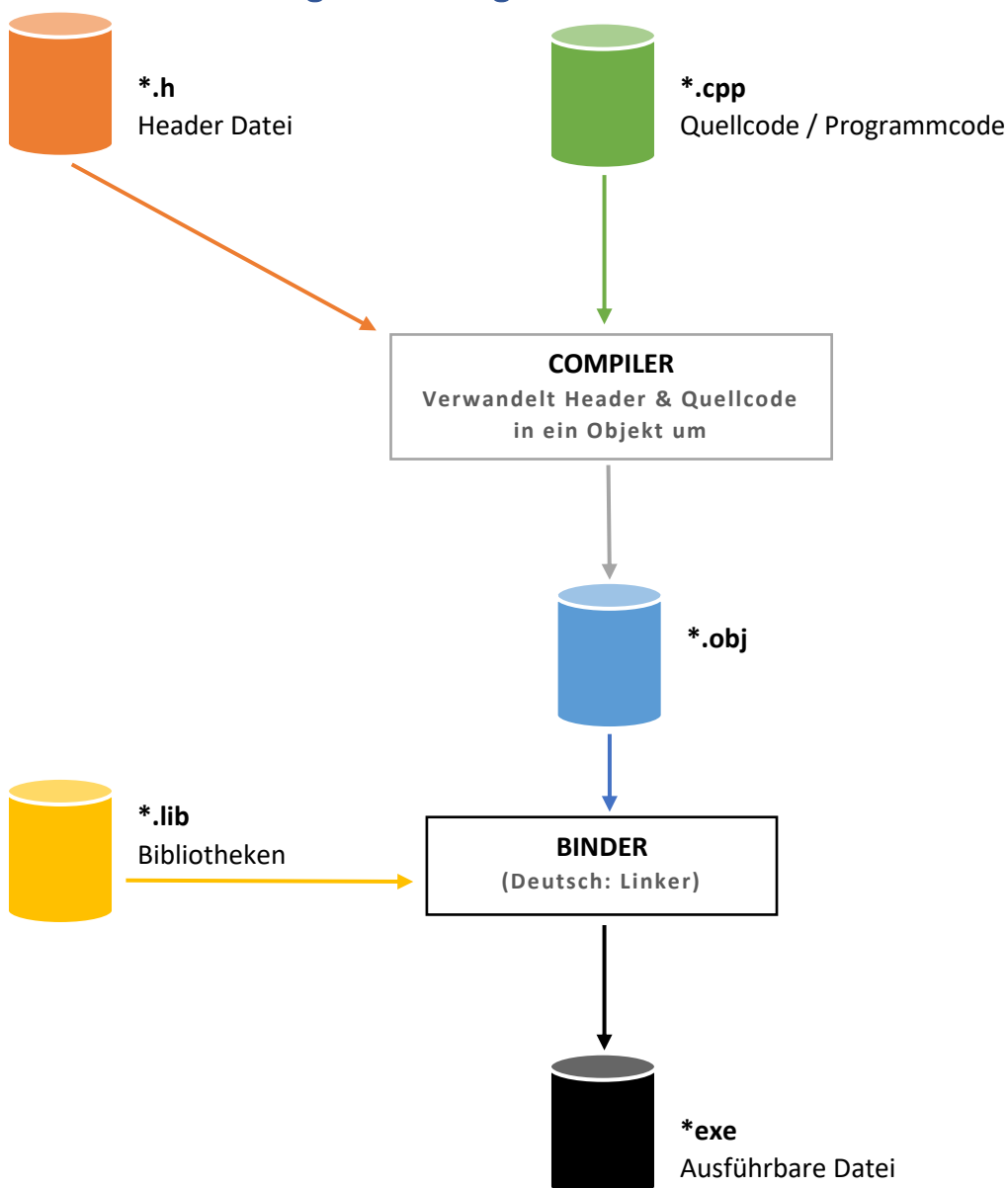
- ⇒ Ziffern, Buchstaben & Unterstriche sind erlaubt
Sonderzeichen, Umlaute sind nicht erlaubt
- ⇒ Das erste Zeichen darf keine Nummer sein
int 0Null ist nicht erlaubt
int Null0 ist erlaubt
int _0Null ist erlaubt
- ⇒ Groß- & Kleinschreibung wird unterschieden
int Null und int null sind zwei verschiedene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

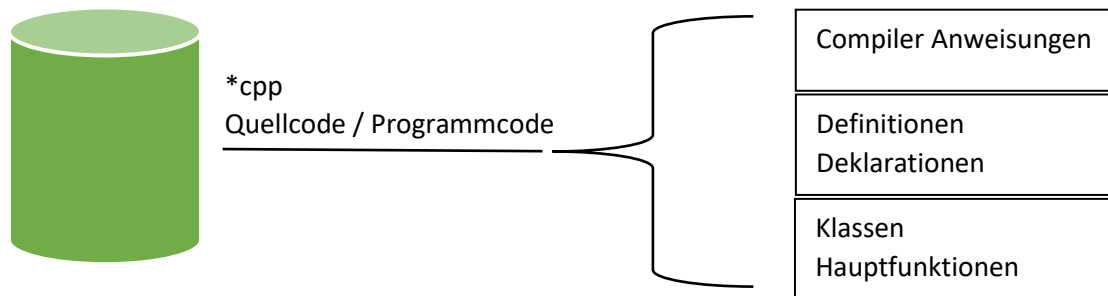
Übersetzte Programmiersprachen	Interpretierte Programmiersprachen
C / C++, C#, Fortan, Cobol, Visual Basic	PHP, JavaScript, PowerShell, Python, Visual Basic Script (VBS)
Explizierte Datentypen ⇒ Datentypen müssen FEST zugeordnet werden Int, char, float, long, string,	Implizierte Datentypen ⇒ Datentypen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

Zusammensetzung eines Programms



Datei Aufbau

Eine Datei ist immer in mehreren "Abschnitte" unterteilt.



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.

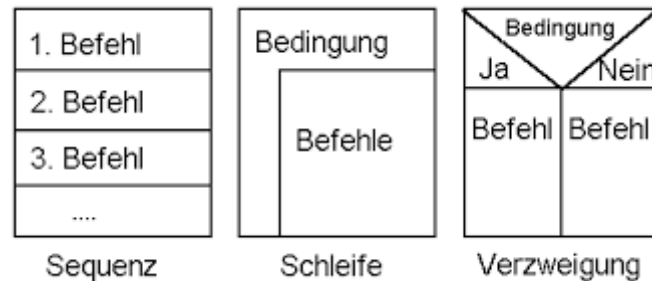
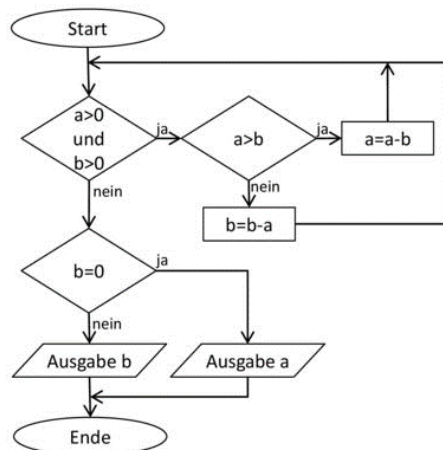


Abbildung 1 | Nassi Shneiderman Blöcke

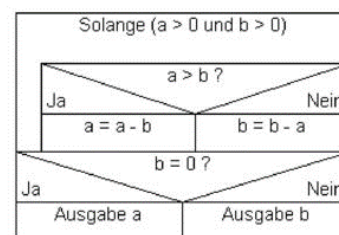
Programmablaufplan (PAP)

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.

Flussdiagramm (Programmablaufplan)



Struktogramm (Nassi-Shneiderman-Diagramm)



- + übersichtlicher
- + näher an unserem Programmiercode

Abbildung 2 | Links PAP - Rechts Nassi-Shneiderman 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 bis 1.7E+308	8 Byte
C++ Only		
long double	3.4E-4932 bis 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”** angegeben 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.

```
// Globale Variable
int globaleZahl = 43;
// Globale Konstante
const short globaleKonstanste = 42;

int main() {
    // Gültigkeitsbereich Lokal
    int zahl = 2;
    int ergebnis;
    double kommaZahl;
    int globaleZahl = 53;
    const float konstante = 1.19;
}
```

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 groß

bool -> char -> short -> ...

Expliziert

von groß auf klein

long double -> double -> ...

long long -> long -> ... -> bool

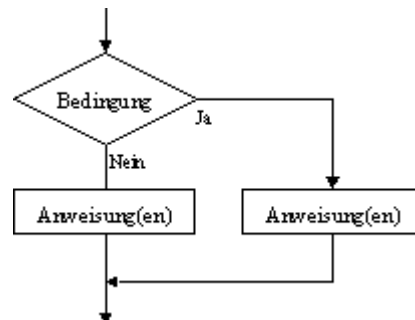
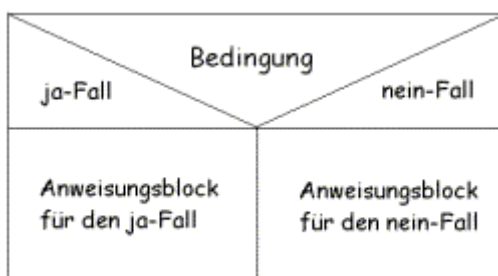
Kontrollstrukturen

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:

```
if (Bedingung) {  
    // TRUE / JA BLOCK  
}else {  
    // OPTIONAL  
    // FALSE / NEIN BLOCK  
}
```

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

```
if (!Bedingung) {  
    // FALSE / NEIN BLOCK  
}else {  
    // OPTIONAL  
    // TRUE / JA BLOCK  
}
```

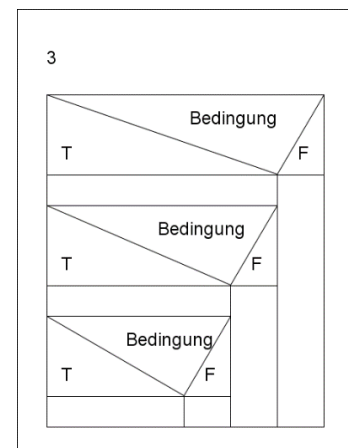
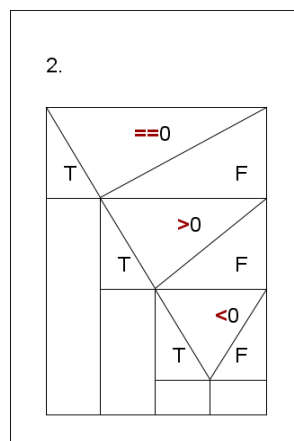
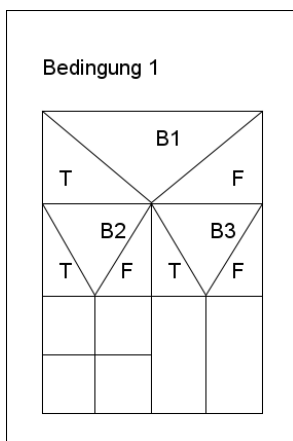

⇒ Komplexe / Mehrfache Verzweigungen

if else

```
if (Bedingung) {  
    // Erste Bedingung erfüllt  
  
    if (Bedingung2) {  
        // Zweite bedingung erfüllt  
    }  
  
} else {  
    // Erste und/oder Zweite Bedingung nicht erfüllt  
  
    if (Bedingung3) {  
        // Dritte Bedingung erfüllt  
    }  
  
}
```

If else if

```
if (Bedingung) {  
    // Erste Bedingung erfüllt  
  
} else if (Bedingung2) {  
    // Zweite bedingung erfüllt  
  
} else if (Bedingung3) {  
    // Dritte bedingung erfüllt  
  
} else if (Bedingung4) {  
    // Vierte Bedingung erfüllt  
  
}
```



Switch

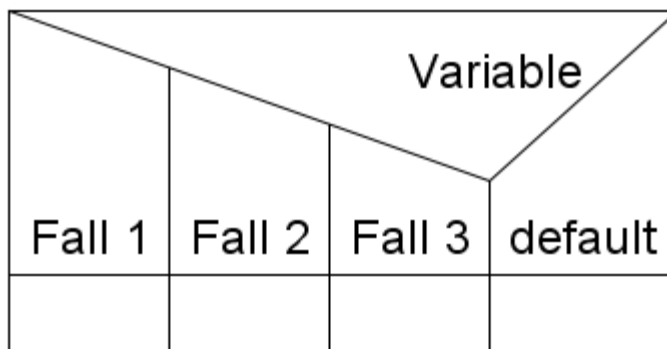
```
int Variable = 0;

switch (Variable) {
case 0:
    // Variable enthält den Wert 0
    break;
case 1:
    // Variable enthält den Wert 1
    break;
default:
    // Variable enthält weder 1 oder 0
    // Im Default wird alles ausgeführt wenn
    // der Wert der Variable nicht abgefragt wird
}
```

Die Variable zur Überprüfung muss Numerisch 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.

Variable == Fall



Schleifen

⇒ Kopfgesteuert

- Zählschleife / for
- so lange bis / while
- über eine Liste / for each

For Schleife

for (Zählervariable; Anfangswert < Endwert; Schrittweite)

ANWEISUNGSBLOCK

Zählschleife for (Initialisierung; Bedingung; Nachbearbeitung) {

Initialisierung: findet nur am Anfang statt

Bedingung: wird nach dem Durchlauf geprüft

Nachbearbeitung: findet immer nach jeden Durchlauf statt
unabhängig von continue

}
*/

```
cout << "FOR SCHLEIFE" << endl;
```

```
for (int durchlauf = 1; durchlauf < 30; durchlauf = durchlauf + 3) {  
    cout << "Durchlauf " << durchlauf << endl;
```

```
    // Bei Durchlauf springt er in die Nachbearbeitung
```

```
    if (durchlauf == 7) {  
        continue;  
    }
```

```
    // Sobald Durchlauf durch glatt 7 Teilbar ist, wird die Schleife beendet
```

```
    if (durchlauf%7 == 0) {  
        break;  
    }
```

```
}
```

```
// Berechnung Dezimalwert Bit-Position
```

```
{
```

```
    int produkt = 1;
```

```
    int summe = 1;
```

```
    int exponent = 7;
```

```
    int basis = 2;
```

```
    for (int i = 1; i <= exponent; i = i + 1) {
```

```
        produkt = produkt * basis;
```

```
        summe = summe + produkt;
```

```
    }
```

```
    cout << "Produkt: " << produkt << endl;
```

```
    cout << "Summe: " << summe << endl;
```

```
// VerschateIt
```

```
for (int i = 1; i <= 10; i = i + 1) {
```

```
    for (int j = 1; j <= 5; j = j + 1) {
```

```
        cout << setw(3) << i << " * " << setw(3)
```

```
            << j << " = " << setw(3)
```

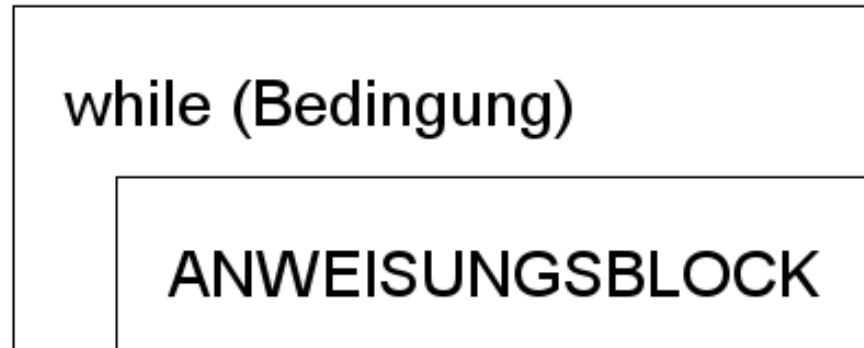
```
            << i*j;
```

```
    }
```

```
    cout << endl;
```

```
}
```

While Schleife



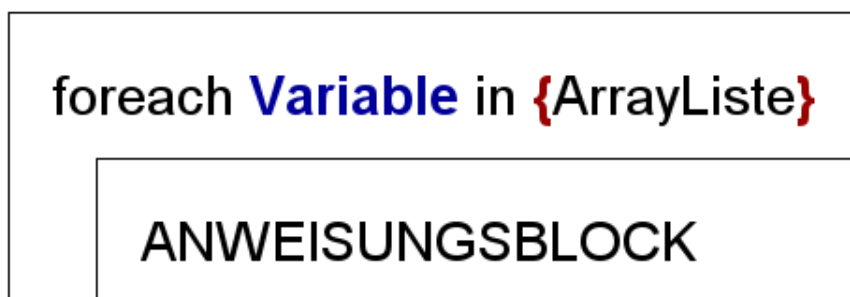
```
/*
Solange Schleife while (Bedingung) {
    Bedingung: wird nach dem Durchlauf geprüft
}
*/
cout << "WHILE SCHLEIFE" << endl;
durchlauf = 1;
while (durchlauf < 35) {
    cout << "Durchlauf " << durchlauf << endl;

    // Bei Durchlauf springt er in die Nachbearbeitung
    if (durchlauf == 7) {
        continue;
    }

    // Sobald Durchlauf durch glatt 7 Teilbar ist, wird die Schleife beendet
    if (durchlauf % 7 == 0) {
        break;
    }

    // Zähler Erhöhung
    // ACHTUNG: Sobald der Wert 7 erreicht hat, wird dieser so nicht mehr erhöht.
    durchlauf = durchlauf + 3;
}
```

For Each Schleife



- ⇒ Fussgesteuert
- so lange bis

Do Schleife

ANWEISUNGSBLOCK

while (Bedingung)

```
/*
FUSSGESTEUERTE SCHLEIFE
do {
    Schleifenrumpf
} while(Bedingung);
Hinter der while Bedingung muss ein ; (Semikolon)
*/

char weiter, eingabe = 'y';
bool bedingung = false;
int durchlauf = 1;

do {
    cout << "Erster Durchlauf " << durchlauf << endl;
    durchlauf = durchlauf + 3;

    bedingung = durchlauf < 30;
} while (bedingung);

/*
FUSSGESTEUERTE SCHLEIFE
do {
    Schleifenrumpf
} while(Bedingung);
Hinter der while Bedingung muss ein ; (Semikolon)
*/

char weiter, eingabe = 'y';
bool bedingung = false;
int durchlauf = 1;

do {
    cout << "Erster Durchlauf " << durchlauf << endl;
    durchlauf = durchlauf + 3;

    // Automatischer Durchlauf bis durchlauf größer 30
    // bedingung = durchlauf < 30;

    // Nach jeder Schleife kann der Benutzer entscheiden
    // ob es weiter gehen soll

    cout << "Weiter ? (y)";
    cin >> weiter;

    bedingung = (weiter == eingabe);
} while (bedingung);
```

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.

Bedingung ? ja : nein;

```
int Variable = 0;  
cout << "Wert ist: " << (Variable == 0 ? "" : "un") << " gleich 0";
```

Die Ausgabe hier wäre: *Wert ist: gleich 0*

Wäre die Variable nicht 0 würde in der Ausgabe stehen: *Wert ist: ungleich 0*

Zeichenketten / Listen

Listen enthalten Variablen/Elemente des gleichen Datentyps.

Syntax:

Modifizierer Datentyp Bezeichner[maxAnzahl] = { Werteliste }

Position in Liste

0 – MAXANZAHL -1

- ⇒ maxAnzahl = maxAnzahl an Elemente in der Liste
bsp.: maxAnzahl = 8 | Maximal 8 Elemente in der Liste

Element in der Liste

Anzahl der Werte darf nicht größer sein als maxAnzahl

"" = Zeichenkette

" " = Zeichen

ELEMENTE							
1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7
POSITION							

Werteliste

Werteliste { Kommagetrennt }

- ⇒ = { } | alle Elemente werden mit 0 erstellt
- ⇒ = { 1, 2 } | Element 1 und 2 bekommen den Wert 1 und 2, der Rest bekommt 0

Zugriff

ANZEIGEN EINES LISTENELEMENTS

Syntax:

- ⇒ Bezeichner[Pos]

Bsp.:

- ⇒ cout << Bezeichner[Pos] << endl;
- ⇒

ANZEIGEN VON ALLEN LISTENELEMENTEN

Bsp.:

- ⇒ for (int pos = 0; pos < maxAnzahl, pos = pos + 1) {
 cout << Bezeichner[pos];
}

Ändern

ÄNDERN EINES LISTENELEMENTS

Syntax:

- ⇒ Bezeichner[Pos] = 7;

Felder / Listen

Multidimensionale Felder & Listen.

Datentyp Bezeichner[maxAnzahl][maxAnzahl] [maxAnzahl] = { Werteliste };

Syntax:

```
char Monatsnamen[12][12] = {"Januar","Februar","Maerz","April","Mai","Juni"};
```

- ⇒ Erstellt eine char List mit dem Bezeichner Monatsnamen
- ⇒ Die erste Liste hat maxAnzahl = 12
- ⇒ Die zweite Liste hat maxAnzahl = 12
- ⇒ Die erste Liste wird mit der Zeichenkette der Monatsnamen befüllt
- ⇒ " | Wird benutzt für Zeichenketten
- ⇒ ' | Wird benutzt für einzelne Zeichen

Speicherabbild:

Datentyp Bezeichner [2][2] = { {1,2} {3,4} };

0	1	2
1	3	4
	0	1

Einlesen von Numerischen Werte:

```
// Konstante MaxElemente für die Liste
const int MAXELEMENT = 4;

// Numerische Liste
// { 1, 2 } Rest wird mit 0 belegt
// { 1, 2, 3, 4 } Jedes Element bekommt seinen eigenen Wert
// { 0 } Jedes Element bekommt den Wert 0
int listNumber[MAXELEMENT] = {0};

// Einlesen der Liste per UserInput
for (int i = 0; i < MAXELEMENT; i = i + 1) {
    cin >> listNumber[i];
}
```

Einlesen von Zeichenketten:

```
char Buchstabenkette[30] = "AB";

// Einlesen von maximal 29 Zeichen ohne Leerzeichen
// cin.width(MAXANZAHL)
cin >> Buchstabenkette;
cin.width(30);
// Ausgabe der Zeichenketten
cout << "Liste: " << Buchstabenkette << endl;

// Einlesen von maximal 29 Zeichen mit Leerzeichen
// cin.getline(LISTE,MAXANZAHL)
cin.getline(Buchstabenkette, 30);
// löscht alle zuvor gesetzten Errorbits.
cin.clear();
cin.sync();

// Auslesen der Anzahl an eingebenden Zeichen
cout << "Zeichen Eingeben: " << cin.gcount() << endl;
// Ausgabe der Zeichenketten
cout << "Liste: " << Buchstabenkette << endl;
```


Ändern von Werten:

Datentyp Bezeichner [2][2] = { {1,2} {3,4} };

Bezeichner[1][1] = 5;

0	1	2
1	3	5
	0	1

ÄNDERUNG IM
SPEICHERABBILD

// Ändern von Werten

```
listNumber[2] = 42 * listNumber[1];
```

```
listNumber[2] = 4 + listNumber[1];
```

```
Buchstabenkette[0] = 'A';
```

Ausgabe von Werten:

// Ausgabe ALLER Numerischen Listeneinträge

```
for (int Element : listNumber) {  
    cout << "ELEMENT: " << Element << " ";  
}
```

// Zeichen/Ketten Liste

```
char Buchstaben[2] = { 'A', 'B' };
```

```
char Buchstabenkette[30] = "AB";
```

// Ausgabe von Buchstabenlisten

```
cout << "Liste Buchstabenkette: " << Buchstabenkette << endl;
```

```
cout << "Liste Buchstaben: " << Buchstaben << endl;
```

Zählen von Elementen:

// Zählen der Elemente in Liste mit Schleife

```
int Laenge = 0;
```

```
while (Buchstabenkette[Laenge] != '\0') {  
    Laenge = Laenge + 1;  
}
```

```
cout << "Laenge von Buchstabenkette: " << Laenge << endl;
```

Verändern der Ausgabe von Elementen:

// Rückwärts Ausgeben von Elementen

```
cout << "Liste Rueckwärts ausgeben\n";
```

```
for (int i = Laenge; i >= 0; i = i - 1) {  
    cout << Buchstabenkette[i];  
}  
cout << endl;
```

// Alternative

```
for (int i = 0; i < Laenge; i = i + 1) {  
    cout << Buchstabenkette[Laenge - 1 - i];  
}  
cout << endl;
```

Vertauschen von Elementen

// Vertauschen von Elementen

```
for (int i = 0; i < Laenge / 2; i = i + 1) {  
    char temp = Buchstabenkette[i];  
    Buchstabenkette[i] = Buchstabenkette[Laenge - 1 - i];  
    Buchstabenkette[Laenge - 1 - i] = temp;  
}  
cout << Buchstabenkette << endl;
```

Verknüpfen von Zeichenketten

```
// Verknüpfen von Zeichenketten
char Text1[40] = "Hallo, ";
char Text2[5] = "Welt";

for (int i = 0; i <= 5; i = i + 1) {
    Text1[7+i] = Text2[i];
}
cout << Text1 << endl;
```

Kopieren von Zeichenketten

```
// Kopieren von Zeichenketten
for (int i = 0; i <= 5; i = i + 1) {
    Text1[i] = Text2[i];
}
cout << Text1 << endl;
```

Multidimensional

```
char Monatsnamen[12][12] = { "Januar", "Februar", "Maerz", "April" };
```

Ausgabe

```
// Ausgabe = MAERZ
cout << Monatsnamen[2] << endl;
// Gibt nur den 4ten Buchstaben aus
cout << Monatsnamen[2][3] << endl;

// In der For-Schleife werden erst alle 12 Inhalte von der ersten Liste geloopt
// In der zweiten For-Schleife werden jetzt nur die ersten 2 Buchstaben ermittelt
// und Ausgegeben
for (int i = 0; i < 12; i = i + 1) {
    for (int j = 0; j < 2; j = j + 1) {
        cout << Monatsnamen[i][j];
    }
    cout << endl;
}
```

Ändern

```
// Ändert bei Maerz das große M durch ein kleines m
Monatsnamen[2][0] = 'm';
cout << Monatsnamen[2] << endl;
```