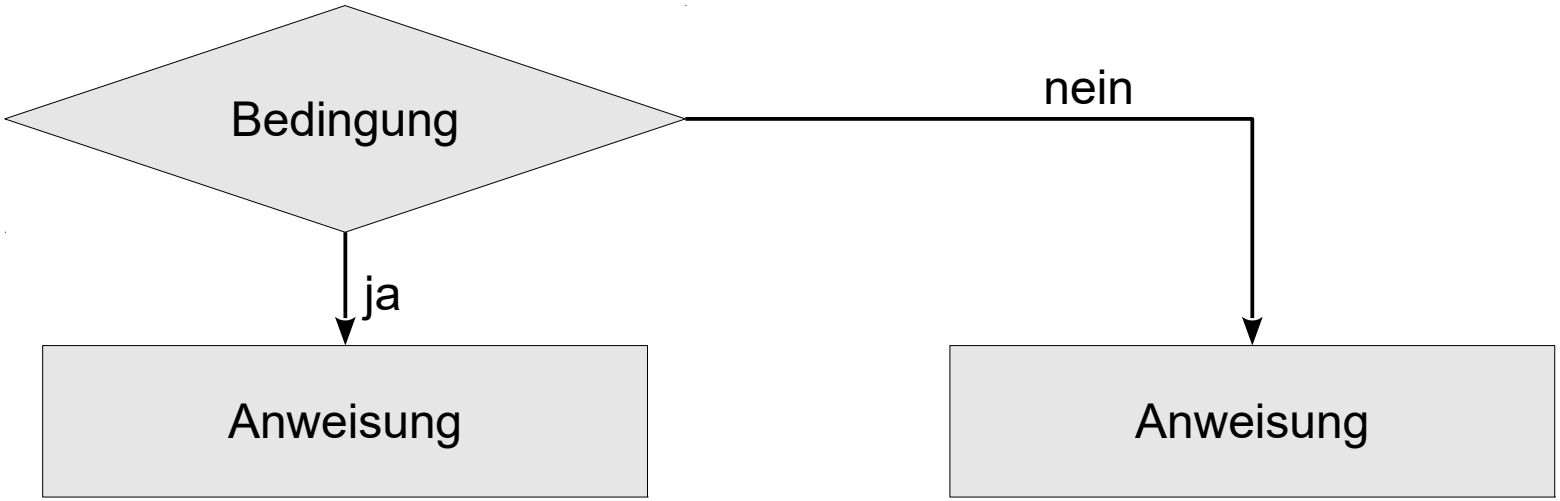
C++ - Einführung in die Programmiersprache

„Bedingte Anweisungen“



Anweisungen

- Befehle für den Präprozessor am Anfang der Quellcode-Datei. Bearbeitung des Quellcodes durch den Präprozessor.
- Befehle in der Programmiersprache C++. Abbildung von Arbeitsschritten in einem Prozess.

Anweisungen in C++

```
#include <iostream>
```

Präprozessor-Anweisungen

```
int ergebnis;
ergebnis = 5 + 5;
std::cout << ergebnis << std::endl;
return 0;
```

Befehle in der Programmier-sprache

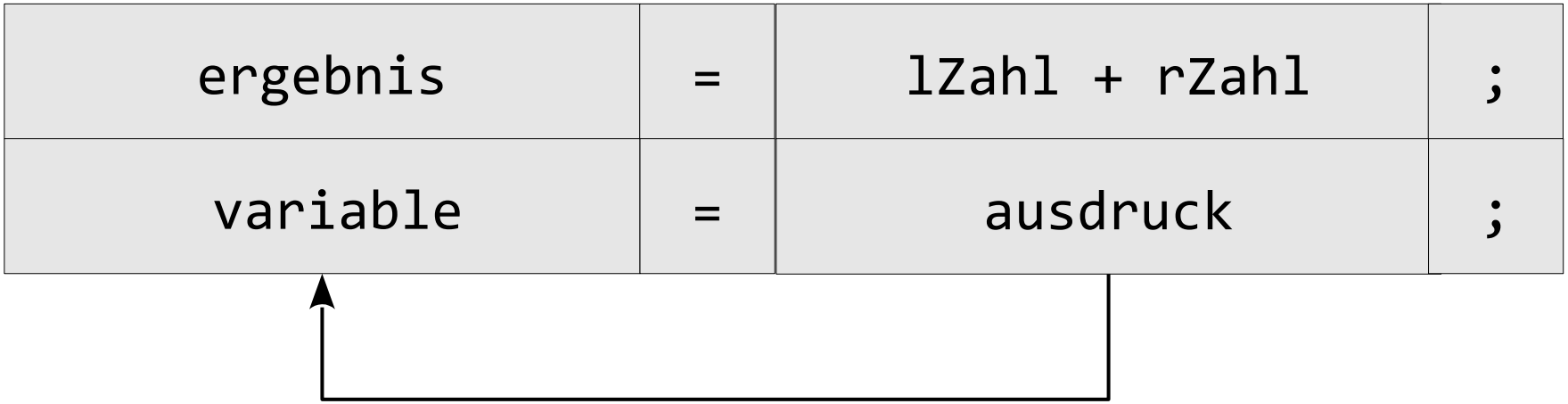
Befehle in der Programmiersprache C++

- Formulierung einer Aktion entsprechend der Syntax der gewählten Programmiersprache.
- Befehle, die von einem Compiler interpretiert werden. Falls ein Fehler in der Syntax ist, wird die Interpretation abgebrochen.
- Beendigung mit einem Semikolon.
- Abarbeitung des Quellcodes von oben nach unten.
- Zusammenfassung mit Hilfe der geschweiften Klammern.

Möglichkeiten

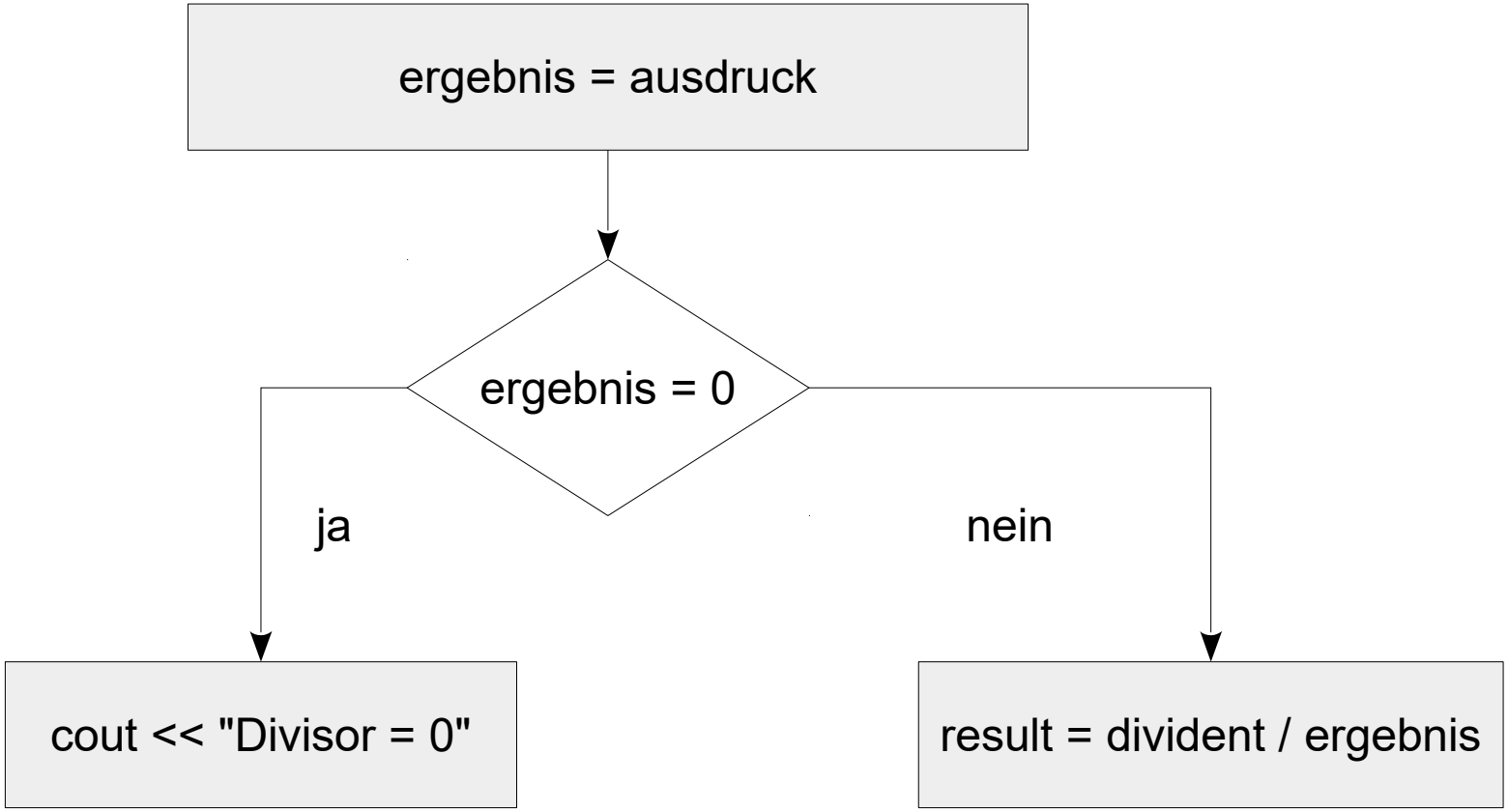
`const int EXPONENT = 2;`	Deklarations-anweisungen
`int ergebnis;`	
`ergebnis = lZahl + rZahl;`	Ausdrucks-anweisungen
`ergebnis = pow(basis, EXPONENT);`	
`std::cout << "3^2 = " << ergebnis;`	
`return 0;`	Sprung-anweisungen

Ausdrucksanweisungen



- In einer Ausdrucksanweisung weist der Zuweisungsoperator einer Variablen einen Wert zu.
- Dieser Wert wird mit Hilfe eines Ausdrucks berechnet.

Bedingte Anweisung



Beispiel in C++

```
double divident  = 5;
double divisor   =  divident % 2;


    if (divisor = 0){
        std::cout <<  " Divisor gleich 0 ";
    }
    else {
        std::cout << divident << " / " << divisor << " = "
                << (divident / divisor);
    }
```

Erläuterung

`if (divisor = 0){`	Definition der Bedingung

`{`
`  std::cout <<  " Divisor gleich 0 ";`
`}`
	Bedingung trifft zu → Codeblock

`else {`
`  std::cout << divident`
`  << " / " << divisor << " = "`
`  << (divident / divisor);`
`}`
	Bedingung trifft nicht zu → Codeblock

Bedingungen

- Ausdruck, der einen boolschen Wert zurückgibt.
- Wahre oder falsche Aussagen. Zum Beispiel: „Das Fahrrad ist blau. Ja / Nein“.
- Vergleiche von Werten. Zum Beispiel: a größer b.
- Beantwortung von Ja / Nein-Fragen. Zum Beispiel „Ist das dein Buch?“.

Vergleichsausdrücke

```
bool ergebnis;

ergebnis = (3 == 2);
ergebnis = (3 !=  2);
ergebnis = (3 <  2);
ergebnis = (3 <=  2);
ergebnis = (3 >  2);
ergebnis = (3 >=  2);
```

Aufbau

(	ausdruck	)	>	(	ausdruck	)
(	20	)	>	(	5	)
(	6 - 7	)	>	(	0	)

Operanden

- Variablen, die als Platzhalter für einen definierten Wert genutzt werden.
- Konstanten, die einen bestimmten Wert verkörpern. Der Wert kann nicht verändert werden.
- Literale wie zum Beispiel 2, 3.5, 'A'. Die Werte werden direkt in in die Anweisung geschrieben. Literale werden einmalig an einer bestimmten Stelle im Code benötigt.

Operatoren

- Vergleichsoperatoren vergleichen zwei Werte.
- Vergleichsausdrücke werden durch logische Operatoren miteinander verbunden.
- Zuweisungsoperator. Das Ergebnis des Vergleiches wird einer boolschen Variablen zugewiesen.

Relationale Operatoren

==	Ist gleich
!=	Ist ungleich
<	Ist kleiner
<=	Ist kleiner gleich
>	Ist größer
>=	Ist größer gleich

Nutzung von relationalen Operatoren

- Vergleich von zwei Operanden.
- Beantwortung von Fragen, auf die mit Ja oder Nein geantwortet werden kann.
- Haben Eigenschaften die Ausprägung? Zum Beispiel: Die hat Katze eine graue Fellfarbe. Die Aussage kann durch Sehen überprüft werden. Falls die Katze eine schwarze Fellfarbe hat, ist die die Aussage falsch.

Hinweise

- Zusammengesetzte Operatoren wie ==, <= und so weiter dürfen nicht durch ein Leerzeichen getrennt werden.
- Der Zuweisungsoperator = darf nicht mit dem Operator „ist gleich“ == verwechselt werden.
- Mit Hilfe von Klammern kann die Lesbarkeit des Ausdrucks erhöht werden.
- Gleitkommazahlen nähren sich einem Wert an. Aus diesen Grund sollte eine Überprüfung auf Gleichheit vermieden werden.

Verknüpfung von Bedingungen

```
bool ergebnis;
double zahl;
char zeichen;

ergebnis = ((zahl > 1) && (zahl < 10));
ergebnis = ((zeichen == 'f') || (zeichen == 'F'));
ergebnis = (!(zeichen == 'C'));
```

Aufbau

(	ausdruck	)	&&	(	ausdruck	)

- Vergleichsausdrücke können mit Hilfe von logischen Operatoren verknüpft werden.
- Im ersten Schritt wird der linke Ausdruck ausgewertet. Dann wird der rechte Ausdruck ausgewertet. Das Ergebnis der beiden Ausdrücke wird miteinander verknüft.
- Die Verknüpfung gibt immer einen boolschen Wert zurück.

Verknüpfungsoperatoren

&&	und / and
||	oder /or
!	nicht

Und-Verknüpfung

True	=	(True)	&&	(True)	;
False	=	(True)	&&	(False)	;
False	=	(False)	&&	(True)	;
False	=	(False)	&&	(False)	;

Beispiel

```
ergebnis = ((zahl > 1) && (zahl < 10));
```

- Der linke sowohl als auch der rechte Ausdruck müssen wahr sein.
- Falls der linke Ausdruck falsch ist, wird der rechte Ausdruck nicht mehr ausgewertet werden.

Oder-Verknüpfung

True	=	(True)	||	(True)	;
True	=	(True)	||	(False)	;
True	=	(False)	||	(True)	;
False	=	(False)	||	(False)	;

Beispiel

```
ergebnis = ((zeichen == 'f') || (zeichen == 'F'));
```

- Einer der beiden Ausdrücke muss wahr sein.
- Sobald der linke Ausdruck wahr ist, wird der rechte Ausdruck nicht mehr ausgewertet.

Negation

False	=	(	!	(True)	)	;
True	=	(	!	(False)	)	;

Beispiel

```
ergebnis = (!(zeichen == 'C'));
```

- Falsch wird zu wahr und umgekehrt.
- Die Negation kann häufig durch eine „Umkehrung“ der Operatoren ersetzt werden.
- In diesem Beispiel wird der „ist gleich“-Operator durch den „ist nicht gleich“-Operator ersetzt.

Runde Klammern

```
boolean = (Bedingung)

variable = (ausdruck) operator (ausdruck)

int main(int argc, char** argv)
```

- Runde Klammern fassen Ausdrücke zusammen.
- Runde Klammern erhöhen die Lesbarkeit von komplexen Ausdrücken.

Rangfolge

Priorität	Operator
1	Postfix-Inkrement ++ Postfix-Dekrement -- Klammerung ()
2	Prefix-Inkrement ++ Prefix-Dekrement -- Vorzeichen + Vorzeichen - Negation !
3	Multiplikation * Division / Modula %
4	Addition + Subtraktion -

Rangfolge

Priorität	Operator
5	Größer > Größer gleich >= Kleiner < Kleiner gleich <=
6	Ist gleich == Ist ungleich !=
7	Und && Oder ||
8	Zuweisung =

Beispiel

```
double dblZahl = 0.3;
int intZahl = 4;
double ergebnis = 0;

ergebnis = dblZahl * dblZahl + 2 * dblZahl * intZahl +
           intZahl * intZahl;


ergebnis = (dblZahl * dblZahl) + (2 * dblZahl * intZahl) +
           (intZahl * intZahl);


ergebnis = (dblZahl * (dblZahl + 2) * dblZahl * intZahl) +
           (intZahl * intZahl);
```

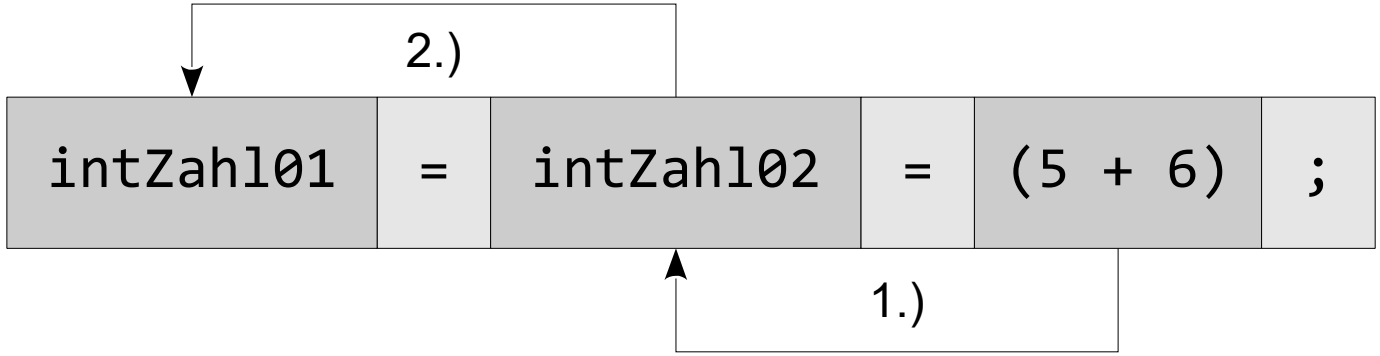
Erläuterung

- Im ersten Beispiel wird der Ausdruck entsprechend der Regel „Punktrechnung vor Strichrechnung“ berechnet. Die Operanden werden entsprechend der Rangfolge der Operatoren verknüpft.
- Im zweiten Beispiel werden die Ausdrücke zur besseren Lesbarkeit mit Hilfe von runden Klammern zusammengefasst.
- Im dritten Beispiel werden die Operatoren und Operanden durch die Klammern zusammengefasst. Durch die Zusammenfassung wird die Rangfolge der Operatoren verändert.

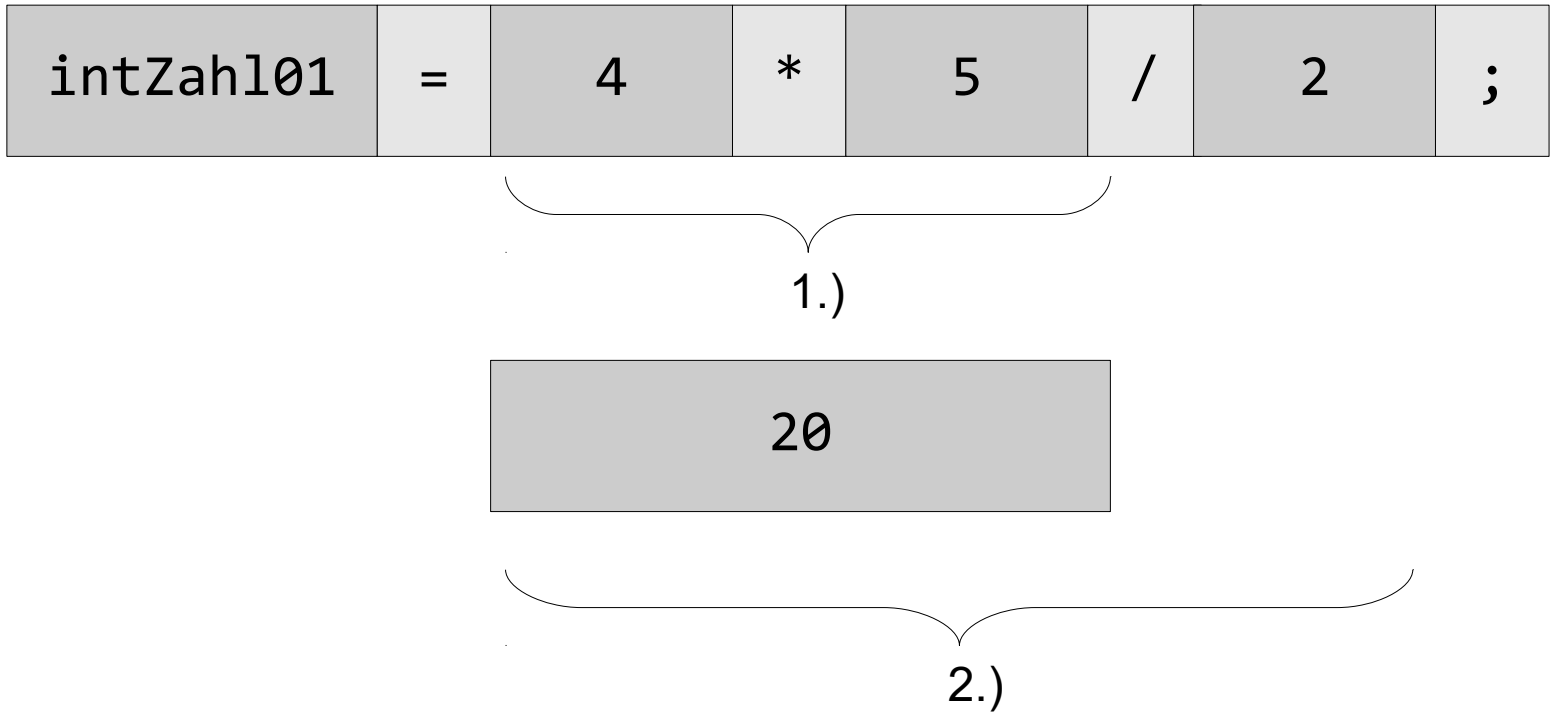
Assoziativität

	Operator
→	Inkrement ++, Dekrement – Multiplikation * Division Modulo % Addition + Subtraktion - Kleiner <, Kleiner Gleich <= Größer >, Größer gleich >= ist gleich ==, ist nicht gleich != Und &&, Oder ||
←	Nicht ! Inkrement ++, Dekrement -- Zuweisung =

Rechtsassoziative Verknüpfung



Linksassoziative Verknüpfung



Abfrage von Attributen mit Hilfe von Funktionen

- Vordefinierte Funktionen in der Standardbibliothek, die mit `is` beginnen, liefern häufig einen boolschen Wert zurück.
- Attribute von Variablen werden überprüft.

Beispiel

```
char zeichen;
bool blnErgebnis;

std::cout << "Bitte geben Sie etwas ein: ";
std::cin >> zeichen;
blnErgebnis = isdigit(zeichen);
std::cout << "\nZahl? " << blnErgebnis;
blnErgebnis = isalpha(zeichen);
std::cout << "\nBuchstabe? " << blnErgebnis;
blnErgebnis = isalnum(zeichen);
std::cout << "\nAlphanumerisch? " << blnErgebnis;

std::cout << std::endl;
```

Hinweise

- Deklaration in der Bibliothek `<cctype>`.
- Überprüfung von Variablen vom Datentyp `char` oder Strings.

Wenn-Dann

- Wenn die Temperatur die Höchstgrenze erreicht hat, dann ...
- Wenn der Kontostand über den Dispo liegt, dann ...
- Wenn die Strecke A doppelt so lang ist wie Strecke B, dann ...
- Wenn die Warenmenge eine Mindestmenge unterschreitet, dann ...

Beispiel

```
double divident;
double divisor = 0.0;

std::cout << "\nBitte geben Sie ein Divident ein: ";
std::cin >> divident;

if(!std::cin.fail()){
    std::cout << "\nBitte geben Sie ein Divisor ein: ";
    std::cin >> divisor;

}
```

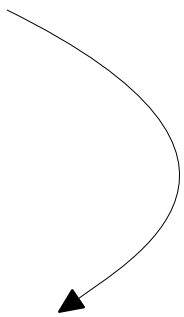
Aufbau

Kopf

Rumpf

`if(!std::cin.fail())`	`if(Bedingung)`

`{`
`  std::cin >> divisor;`
`}`
	`{`
	
	`}`



Wenn das Einlesen keinen Fehler verursacht hat, dann ..

Kopf einer bedingten Anweisung

```
if(Bedingung)
```

- Der Kopf einer bedingten Anweisung beginnt mit dem Schlüsselwort `if`.
- Dem Schlüsselwort folgt eine Bedingung.
- Die Bedingung muss immer geklammert werden. Andernfalls wird der Syntaxfehler „Unexpected token“ angezeigt.

Rumpf der bedingten Anweisung

```
if(Bedingung)
{

}
```

- Der Rumpf beginnt und endet mit den geschweiften Klammern.
- Wenn die Bedingung wahr ist, wird der dazugehörige Codeblock ausgeführt. Der Codeblock wird durch die geschweiften Klammern begrenzt.
- Falls der Anweisungsblock aus nur einer Anweisung besteht, können die geschweiften Klammern weggelassen werden. Aufgrund der besseren Lesbarkeit des Codes sollten diese aber immer gesetzt werden.

Untersuchung von Zuständen

- Wenn die Temperatur über 0°C ist… Wenn die Temperatur unter 0°C ist… Wenn die Temperatur gleich 0°C
- Wenn das Fahrzeug auf der Schiene fährt… Wenn das Fahrzeug im Wasser fährt… Wenn das Fahrzeug auf der Straße fährt...
- Bis zu einer Bestellung von 299 Stück des Produkts A kann kein Rabatt gewährt werden. Ab einer Bestellung von 300 Stück des Produkts A kann ein Rabatt von 2% gewährt werden. Ab einer Bestellung von 1000 Stück des Produkts A und mehr kann ein Rabatt von 5% gewährt werden.

Vollständige bedingte Anweisung

```
double temperatur = 0.6;

if (temperatur > 0.0)
{
    cout << "\nPlus-Temperatur";
}
else if (temperatur < 0.0)
{
    cout << "\nMinus-Temperatur";
}
else
{
     cout << "\nNull-Punkt";
}
```

If - Anweisung

```
if (temperatur > 0.0)
{
   cout << "\nPlus-Temperatur";
}
```

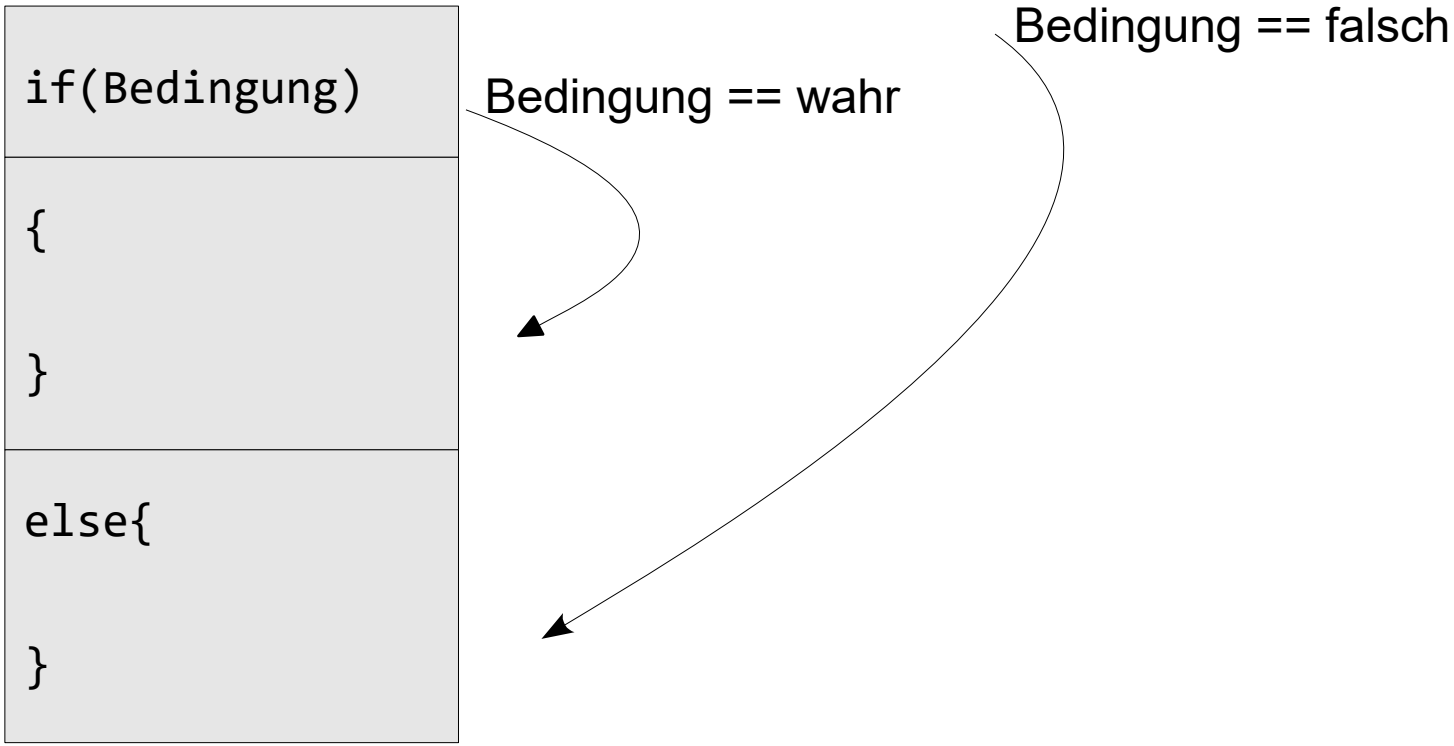
- Wenn (`if`) die Bedingung zutrifft, führe die Anweisungen aus ...
- Wenn die Bedingung wahr ist, wird der dazugehörige Anweisungsblock ausgeführt.

If – else - Anweisung

```
if (temperatur > 0.0)
{
   cout << "\nPlus-Temperatur";
}
else
{
   cout << "\nNull-Punkt";
}
```

- Wenn (`if`) die Bedingung zutrifft, führe die Anweisungen aus ...
- Andernfalls (`else`) führe folgende Anweisungen aus ...

Aufbau



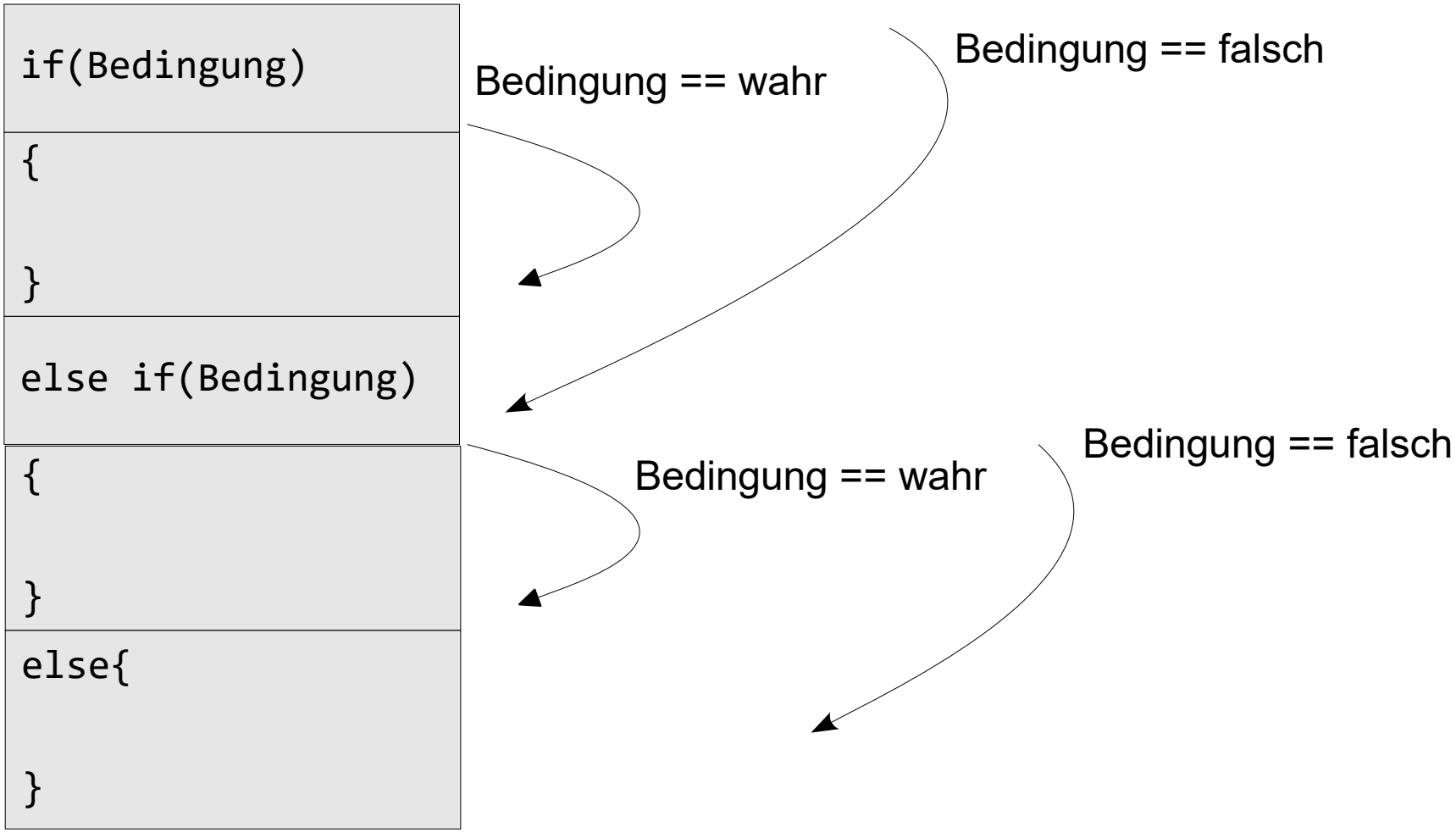
Hinweise

- Die else-Anweisung beschreibt häufig den Standardfall.
- Die else-Anweisung ist optional.
- Eine if-Anweisung benötigt keine else-Anweisung. Aber eine else-Anweisung benötigt immer eine if-Anweisung.

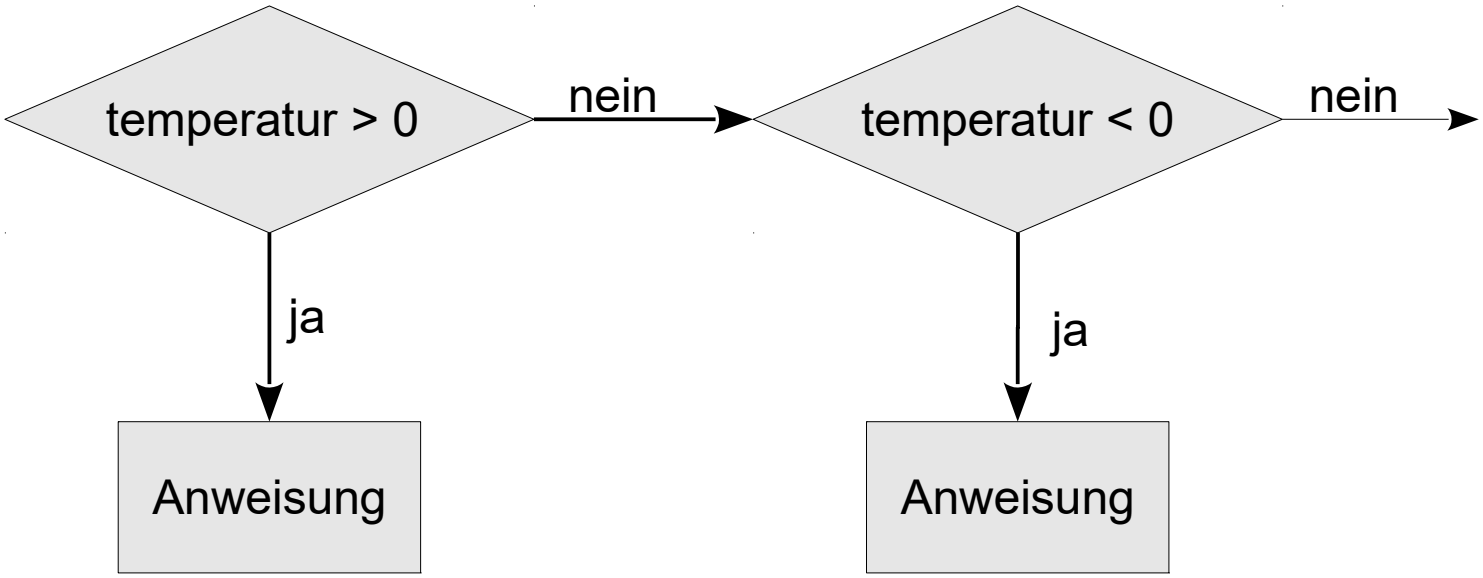
Fallunterscheidung

```
if (temperatur > 0.0)
{
    cout << "\nPlus-Temperatur";
}
else if (temperatur < 0.0)
{
    cout << "\nMinus-Temperatur";
}
else
{
    cout << "\nNull-Punkt";
}
```

Aufbau


if(Bedingung)
{
}
else if(Bedingung)
{
}
else{
}
Bedingung == wahr
Bedingung == falsch
Bedingung == wahr
Bedingung == falsch

Grafische Darstellung



Erläuterung

- `if (bedingung){ }`. Die am häufigsten vorkommende Ausnahme vom Standardfall. Der am zweit häufigsten vorkommender Status oder Zustand.
- `else if (bedingung){ }`. Falls die am häufigsten vorkommende Ausnahme nicht zutrifft, überprüfe diesen Fall. Falls die Bedingung zutrifft, wenn alle nachfolgenden Fälle nicht mehr behandelt.
- `else{ }`. Standardfall. Der am häufigsten vorkommende Status oder Zustand.

Überprüfung auf „ist gleich“

- Ist die Temperatureinheit Fahrenheit? Ist die Temperatur Kelvin? Ist die Temperatur Celsius?
- Ist das Tier ein Säugetier? Ist das Tier ein Insekt? Ist das Tier ein Vogel?
- Ist das Buch ein Taschenbuch? Ein Hardcover? Oder ein Paperback?

if-then-Anweisung

```
if (einheit == 'c'){
    cout << "\nCelsisus";
}
else if (einheit == 'k'){
    cout << "\nKelvin";
}
else if (einheit == 'f'){
    cout << "\nFahrenheit";
}
else{
    cout << "\nUnbekannnte Maßeinheit";
}
```

Andere Möglichkeit

```
switch(einheit)
{
    case 'c':
        cout << "\nCelsisus";
        break;
    case 'k':
        cout << "\nKelvin";
        break;
    case 'f':
        cout << "\nFahrenheit";
        break;
    default:
        cout << "\nUnbekannnte Maßeinheit";
}
```

switch - Anweisung

```
switch(variable)
{
}
```

- Dem Schlüsselwort `switch` folgt in runden Klammern eine Variable.
- Der Status der Variablen in den runden Klammern wird im Rumpf der Anweisung untersucht.
- Welchen Zustand hat der angegebene Schalter?

Fallunterscheidung

```
switch(einheit)
{
   case 'c':
        cout << "\nCelsisus";
        break;
}
```

- Jede Fallunterscheidung beginnt mit dem Schlüsselwort `case`.
- Die Fallunterscheidung `case status:` ist ein Lesezeichen für einen bestimmten Codeabschnitt in einer switch-Anweisung.
- Hinweis: Der Codeblock der switch-Anweisung wird nicht bei Erreichen eines neuen Lesezeichens automatisch verlassen.

Kopf einer Fallunterscheidung

`case`	`'c'`	`:`
`case`	`zustand`	`:`

- Jede Fallunterscheidung beginnt mit dem Schlüsselwort `case`.
- Dem Schlüsselwort folgt ein bestimmter Zustand oder Status.
- Der Kopf der Zustandsbeschreibung endet mit einem Doppelpunkt.

Vorzeitiger Abbruch

```
switch(einheit)
{
   case 'c':
        cout << "\nCelsisus";
        break;
}
```

- Durch das Schlüsselwort `break` kann ein Codeblock vorzeitig verlassen werden.
- Jeder Codeabschnitt, der durch ein Lesezeichen gekennzeichnet ist, kann vor dem nachfolgenden Lesezeichen verlassen werden.

Vorteil

```
switch(einheit)
{
    case 'C':
    case 'c':
        cout << "\nCelsisus";
        break;

    case 'K':
    case 'k':
        cout << "\nKelvin";
        break;
}
```

Standardfall

```
switch(einheit)
{
   case 'c':
        cout << "\nCelsisus";
        break;

        default:
            cout << "\nUnbekannnte Maßeinheit";
}
```

- Das Schlüsselwort default: kennzeichnet den Standardfall.
- Wenn kein Fall zutrifft, wird dieses Lesezeichen genutzt.
- Der Standardfall wird am Ende der Anweisung definiert.
- Der Fall ist optional.

Schachtelung von Anweisungen

```
if(!std::cin.fail()){
  std::cout << "\nTemperatureinheit ein: ";
  std::cin >> einheit;

  switch(einheit){
       case 'c':
       case 'C':
          std::cout << "\n" << temperatur << " in Celsius";
          break;
        case 'F':
           std::cout << "\n" << temperatur << " in Fahrenheit";
            break;
        case 'K':
            std::cout << "\n" << temperatur << " in Kalvin";
   }
}
```

Erläuterung

- Beliebige Schachtelung von if- und switch-Anweisungen in beliebiger Tiefe ist möglich.
- Durch Einrückungen werden die Rümpfe der Anweisungen einer if- oder switch-Anweisungen zugeordnet. Einrückungen erhöhen die Lesbarkeit von Code.