INF1
Entscheidungen

- Ablaufstrukturen
- Logische Ausdrücke
- Auswahl: *if ... else ...*
- Mehrfachauswahl: *switch ...*
- Konstanten und Aufzähltypen

Ablaufstrukturen

Entscheidungen

- Bis jetzt: Sequenz
  - Anweisungen werden sequentiell abgearbeitet
- Jetzt: Auswahl
  - Anweisungsfolgen, die nicht grundsätzlich, sondern abhängig von bestimmten Entscheidungskriterien durchgeführt werden
  - Entscheidungskriterien werden in Form von logischen Ausdrücken formuliert

Entscheidungen

- Um einen Fahrzeug zu starten, müssen folgende Kriterien erfüllt sein:
  - Die Batteriespannung muss min. 12 Volt betragen
  - Der Tank darf nicht leer sein
  - Es muss ausgekuppelt sein

Ablaufstrukturen

- Notwendig: Planung und genaue Beschreibung des Programmablaufs
- Anforderungen:
  einfach, verständlich, präzise, mächtig, übersichtlich
- Alternativen:
  - Pseudocode
  - Flussdiagramm
  - Struktogramm

Pseudocode

- Textbasiert
- *Eindeutig, aber einfach zu lesen*, auch für Leser mit minimalen Programmierkenntnissen
- Auch in Pseudocodeprogrammen sollte in Unterprogramme gegliedert werden (Übersicht!)
- Fördert das strukturierte Programmieren
- Verwandtes Konzept: *„literate programming“*
  Programme so schreiben, dass sie vor allem für Menschen gut lesbar sind

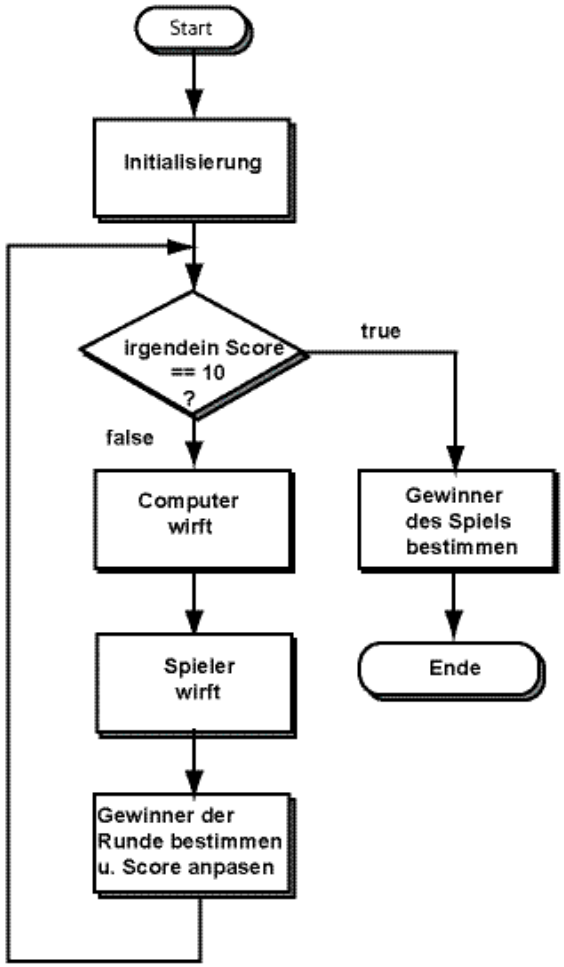
Pseudocode: Beispiel

- Prozedur: **euklid**
- Zweck: Euklidischer Algorithmus zur Berechnung des größten gemeinsamen Teilers
- Parameter: natürliche Zahlen m, n

```
Falls m > n, dann m und n miteinander vertauschen
Jetzt gilt m  < n
Solange m ≠ n wiederhole:
   Setze n = n – m
   Falls m > n, dann m und n miteinander vertauschen
   Jetzt gilt m < n
Ergebnis: n
```

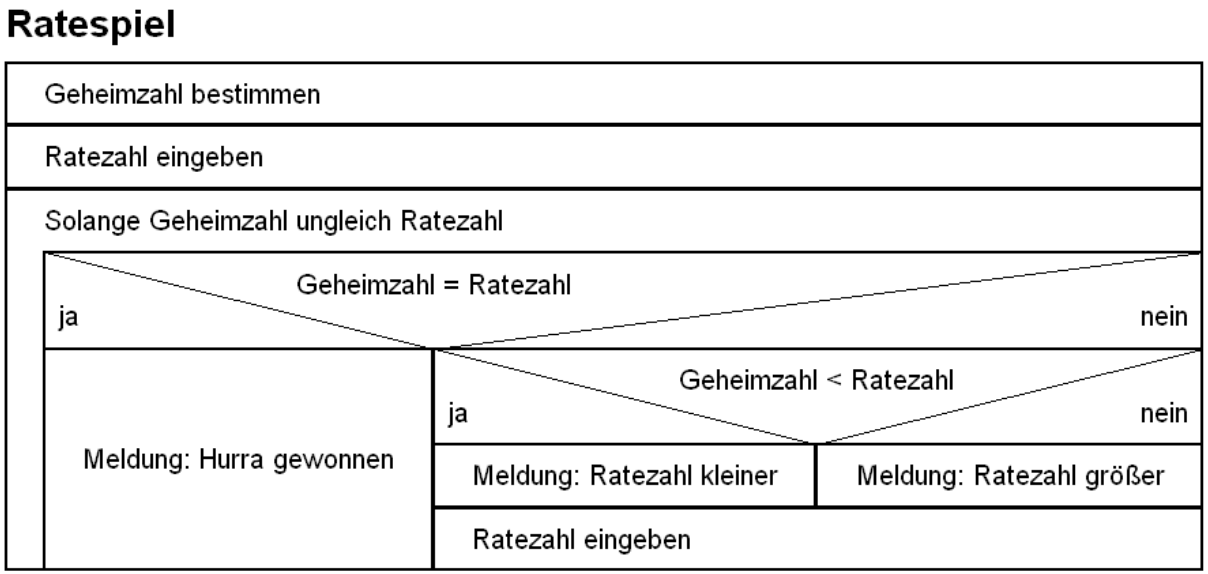
Flussdiagramm

- Auch als Programmablaufplan (PAP) bekannt
- Graphische Darstellung
- Einfach zu lesen, präzise, mächtig
- Schnell unübersichtlich
- Verleitet zur unstrukturierten Programmierung (Sprünge)

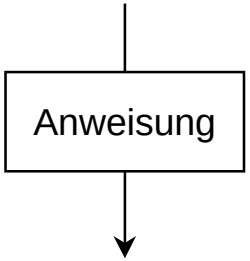


Struktogramm

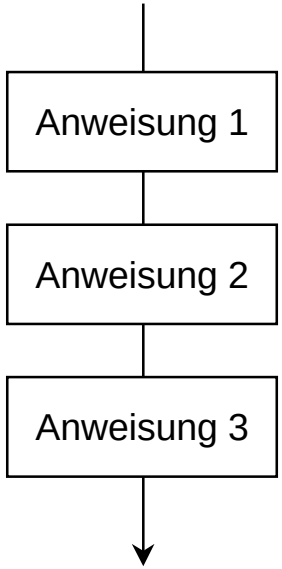
- Nassi-Schneidermann, DIN 66261
- Graphische Darstellung
- Einfach zu lesen, präzise, mächtig
- Vor allem für kleinere Programme – wird schnell sehr umfangreich
- Erzwingt das strukturierte Programmieren



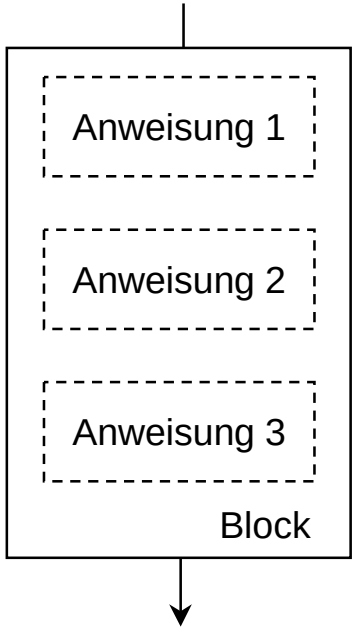
Anweisung, Sequenz und Block



Anweisung



Sequenz



Block

Der Block

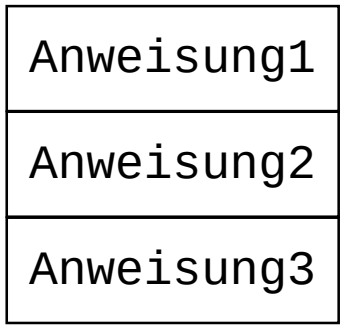
- Grundbaustein einer Ablaufstruktur
- Enthält beliebig viele (auch gar keine!) Anweisungen
- Genau ein Eingang und ein Ausgang
- Block in C:
  - Umgeben von geschweiften {} Klammern, aneinandergereiht
  - Ein Block wird wie eine **einzelne Anweisung** behandelt
  - Bei einem Block gilt die schliessende Klammer als Abschluss (ohne Semikolon)
  - In einem Block können lokale Variablen deklariert werden, die nur bis zum Blockende gültig sind

Sequenz

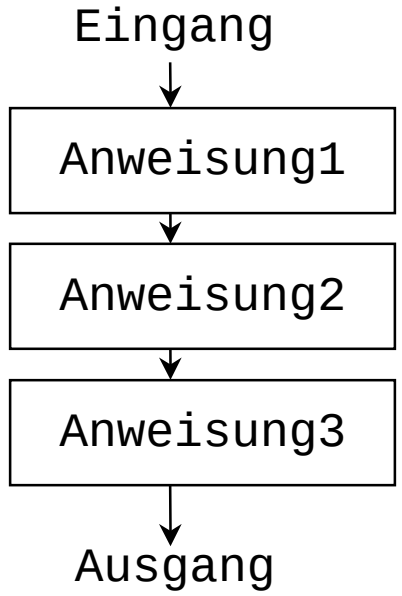
- Einfachste Ablaufstruktur
- Folge von Anweisungen (oder Blöcken), die nacheinander ausgeführt werden
- Pseudocode:

```
Anweisung1;
Anweisung2;
Anweisung3;
```

Struktogramm:



Flussdiagramm:



Verschiedene Ablaufstrukturen

- Durch Benutzung der Sequenz können wir nur vorbestimmte Abläufe programmieren
- Während das Programm läuft, kann kein Einfluss auf den Ablauf genommen werden
- Um komplexere Abläufe zu beschreiben, brauchen wir zusätzliche Strukturen
- Wie viele verschiedene Strukturen braucht es, um alle möglichen Programme darzustellen?

?

Verschiedene Ablaufstrukturen

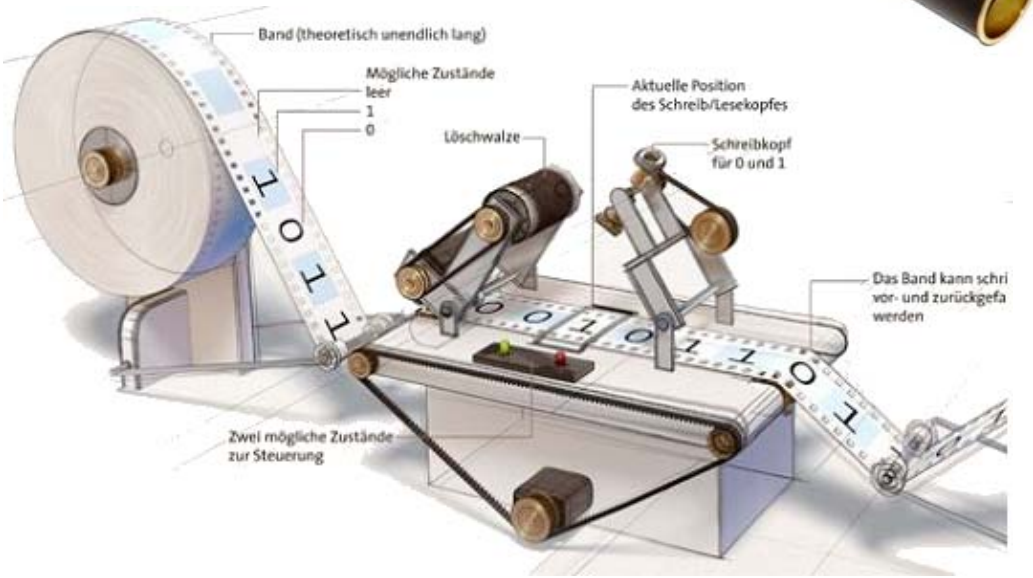
- ANTWORT: Nur 3!
- Sequenz
  Folge von Anweisungen
- Auswahl (diese Vorlesung)
  Anweisungen, die unter bestimmten Bedingungen ausgeführt werden
- Schleife (nächste Vorlesung)
  Wiederholung von Anweisungen

- Turing-Maschine von Alain Turing 1936 vorgestelltes Rechner**modell**

Es wurde anschliessend ***bewiesen***, dass alle deterministischen Rechner Turing Maschinen äquivalent sind

Turing Maschine

- Ein unendlich langes Speicherband mit unendlich vielen sequentiell angeordneten zweiwertigen Feldern.
- Ein programmgesteuerter Lese- und Schreibkopf, der sich auf dem Speicherband feldweise bewegen und die Zeichen lesen und verändern kann.
- Mit jedem Schritt
  - liest der Lese-Schreib-Kopf das aktuelle Zeichen.
  - überschreibt dieses mit einem anderen (oder dem gleichen) Zeichen
  - bewegt sich anschliessend ein Feld nach links oder rechts.
  - welches Zeichen geschrieben wird und welche Bewegung ausgeführt wird
    - *hängt von dem an der aktuellen Position vorgefundenen Zeichen*
    - *dem Zustand ab, in dem sich die Turingmaschine gerade befindet.*



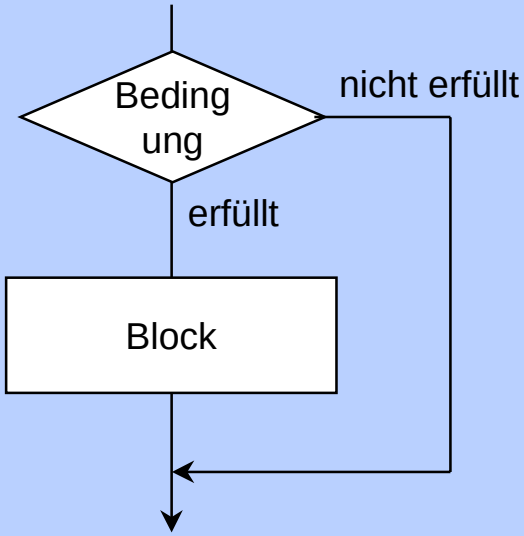
Einfache Auswahl

- Block wird nur *unter einer bestimmten Bedingung* ausgeführt
- Erweiterte Variante: Block A wird ausgeführt, falls die Bedingung erfüllt ist, andernfalls wird Block B ausgeführt

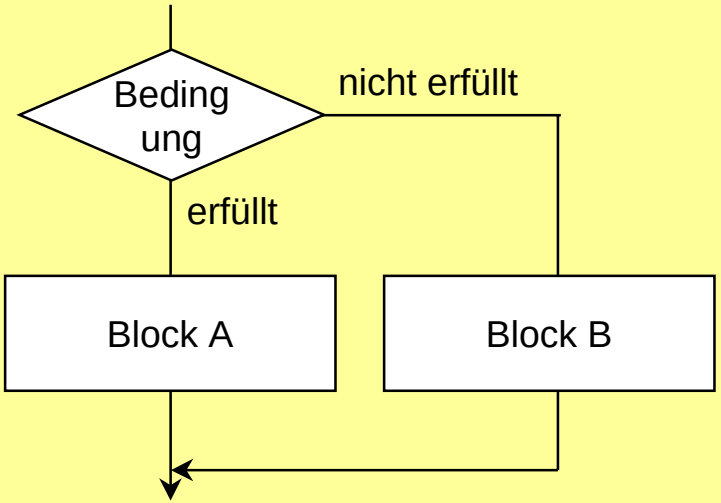
```
if (bedingung) {
    block
}
```

```
if (bedingung) {
    block_a
} else {
    block_b
}
```

Einfache Auswahl



Bedingung	
erfüllt	nicht erfüllt
Block	—



Bedingung	
erfüllt	nicht erfüllt
Block A	Block B

Bedingungen - Logische Ausdrücke

Vergleichsoperatoren

- Auswahlanweisungen enthalten als Bedingungen häufig *Vergleichsoperatoren*
- Ausdrücke mit solchen Operatoren liefern als Resultat einen Wahrheitswert (0 für falsch, sonst wahr)
- C bietet die *Vergleichsoperatoren*:
  - Grösser als `>`
  - Kleiner als `<`
  - Gleich `==` (NICHT: `=`)
  - Nicht gleich `!=` (NICHT: `<>`)
  - Kleiner oder gleich `<=`
  - Grösser oder gleich `>=`

Vergleichsoperatoren

Operator	Int	Float	Operation	Beispiel	Ergebnis
<	X	X	Vergleich auf kleiner	x<y	1, wenn x kleiner y, 0 sonst
>	X	X	Vergleich auf größer	x>y	1, wenn x größer y, 0 sonst
<=	X	X	Vergleich auf kleiner oder gleich	x<=y	1, wenn x kleiner oder gleich y, 0 sonst
>=	X	X	Vergleich auf größer oder gleich	y>=x	Wie bei x<=y
==	X	X	Prüfung auf Gleichheit	x==y	1, wenn x und y gleich, 0 sonst
!=	X	X	Prüfung auf Ungleichheit	X!=43	0 wenn x gleich 43, 1 sonst

- Ergeben wie die logischen Operatoren Wahrheitswerte
- Viele Probleme durch Verwechseln von == und =

Logische Operatoren

- Zum Verknüpfen mehrerer Bedingungen

UND-Operator **&&**

	A	B	Z
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

$\longrightarrow Z = A \wedge B$

ODER-Operator **||**

	A	B	Z
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

$\longrightarrow Z = A \vee B$

NICHT-Operator **!**

A	Z
0	1
1	0

$\longrightarrow Z = \bar{A}$

Logische Operatoren

Operator	Int	Float	Operation	Beispiel	Ergebnis
&&	X		Logisches UND	i&&j	1, wenn i und j ungleich 0, 0 sonst
||	X		Logisches ODER	a||b	0 wenn a und b gleich 0, 1 sonst
!	X		Logisches NOT (Negation)	!q	1 wenn q gleich 0, 0 sonst

- Integerzahlen werden als wahr interpretiert, wenn sie ungleich 0 sind, sonst als falsch
- C kennt in der Sprache keinen Datentyp "Boolean" wie viele andere Sprachen -> #include <stdbool.h>
- Die Ergebnisse sind anders, wenn man & oder | verwendet
  (Bit Operationen: gegeben i = 1; j = 2
  i && j = 1 i & j = 0

Bool

- Ein Typ *bool* kann also folgendermassen selber definiert werden:

```
enum bool { false = 0, true = 1 };
```

- Beachten Sie den Unterschied:

```
if (fertig == true) {...} // vergleicht auf 1
if (fertig) {...}         // besser vergleicht auf ungleich 0
```

- In C++ ist der Typ *bool* bereits vordefiniert (aber nicht in allen Compilern ...)
- sonst in stdbool.h definiert -> #include <stdbool.h>

Beispiele

```
25 < 12       ?

25 > 12       ?

25 > 12 || 4 == 2   ?

25 > 12 && 4 == 2   ?

!12    ?

!0 ?

3 < 4 < 2     ?

0.1 + 0.2 == 0.3
```

Wertetest auf Gleichheit

- Ganzzahlen

```
int i = 5;
if (i == 5) // ok
```

- Fliesskommazahlen-Vergleich **nicht**

```
double zielWert = 0.3;
double wert = 0.1 + 0.2;
if (wert == zielWert ) // funktioniert nicht !!!!
```

- Sondern

```
double zielWert  = 0.3;
const double EPS = 1E-10; // ein "genügend" kleiner Wert
if (fabs(0.1 + 0.2 - zielWert) < EPS) // funktioniert
```

Logische Operatoren in arith. Ausdrücken

- Was macht folgender Ausdruck?

```
a = 5 + !!b * 7;
```

… Logische Operatoren in arith. Ausdrücken

- Dieser Ausdruck weist a den Wert 5 oder 12 zu, je nachdem ob b Null ist oder nicht:

```
a = 5 + !!b * 7;
```

Eine solche Programmierung führt zu schlecht lesbaren Programmen !

```
if (b == 0) {
    a = 5;
} else {
    a = 12;
}
```

Präzedenzen & Assoziationen

Operatoren	Präzedenz	Assoziativität
++ -- ~ ! +(Vorzeichen) -(Vorzeichen)	15	rechts
* / %	13	links
+(Addition) -(Subtraktion)	12	links
<< >>	11	links
< <= > >=	10	links
== !=	9	links
&	8	links
^	7	links
|	6	links
&&	5	links
||	4	links
?:	3	links
= += -= *= /= %= <<= >>= &= |= ^=	2	rechts

- Für Klarheit möglichst Klammern verwenden

Auswahl: if ... else ...

Auswahlanweisung in C

- Auswahl anhand Bedingung
- `if ( Bedingung )`
  `Block`

- Auswahl mit Alternative
- `if ( Bedingung )`
  `Block1`
  `else`
  `Block2`

Beispiel:

```
if ( schulden > 2000 ) {
    printf("Kein Kredit");
} else {
    printf("Kredit ok");
}
```

Auswahlanweisung in C

- Beispiel

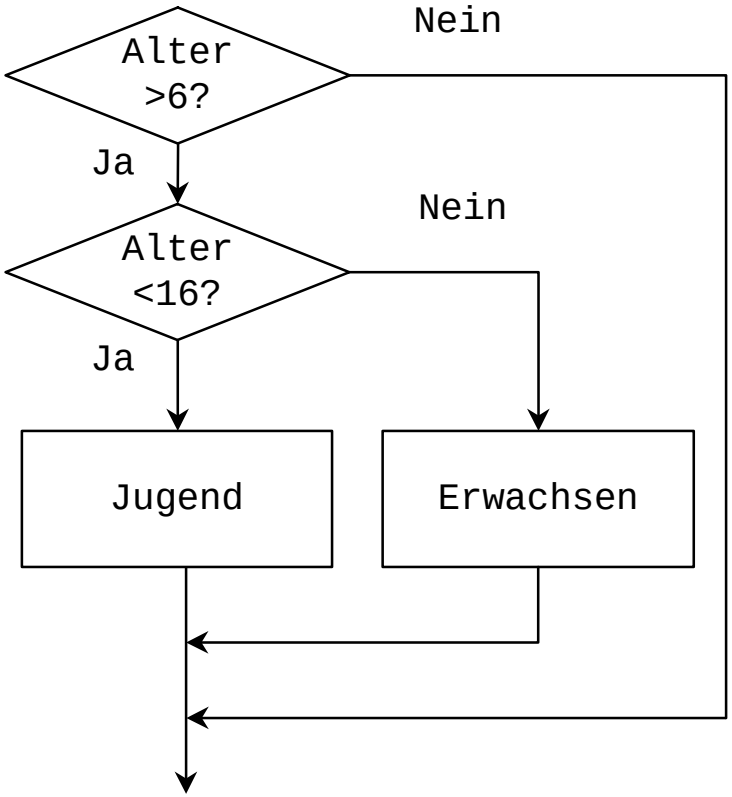
```
if( alter > 6 ){
      if( alter < 16 ){
          printf("Jugendlicher");
      } else {
          printf("Erwachsener");
      }
}
```

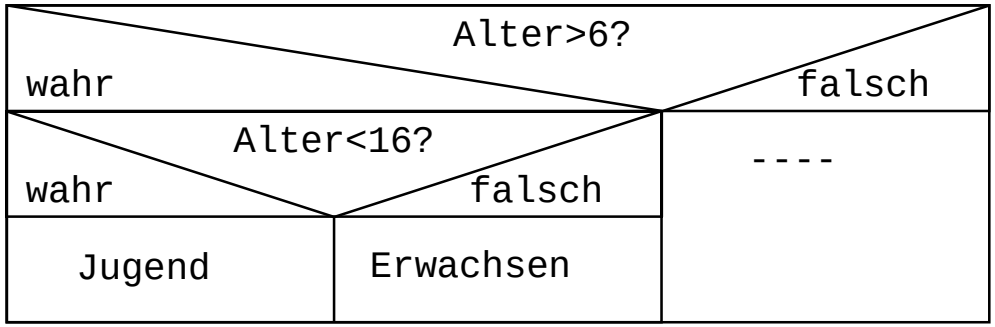
Struktur ohne Klammern nicht immer eindeutig!

Immer Klammern verwenden

Darstellungen der Auswahl

```
wenn alter > 6 :
   wenn alter < 16 :
        Jugend
   sonst :
        Erwachsen
   ende wenn
ende wenn
```





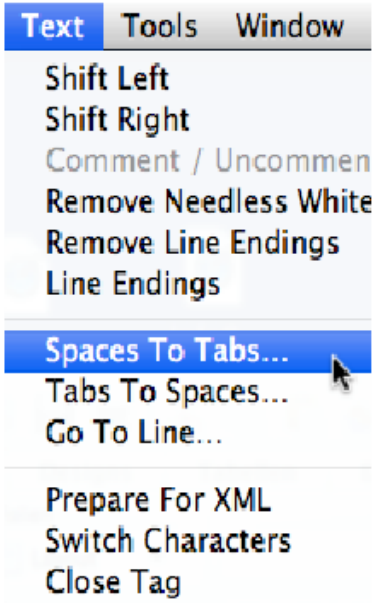
Programmierstil

- Immer auf konsistente Einrückung achten
- Das ist unerlässlich für lesbaren Programmcode
- Ein nicht gut lesbares Programm ist unbrauchbar

```
if (x < y) {
    mach_was();
    if (y > 10) {
        mach_noch_was();
        und_noch_was();
    }
} else {
    mach_was_anderes();
}
```

Üblich: Tabulator oder 2/4 Leerzeichen

Kann in vielen Editoren ineinander umgewandelt werden



Beispiel von oben

- Um einen Benzinmotor (heute) automatisch zu starten, müssen folgende Kriterien erfüllt sein:
  - Falls VW: Bordcomputer muss überprüft haben, ob nicht gerade ein Abgastest läuft
  - Die Batteriespannung muss min. 12 Volt betragen
  - Der Tank darf nicht leer sein
  - Es muss ausgekuppelt sein

Umsetzung in C

```
if (vBatt >= 12 && tankInhalt > 0 && bKupplung == FALSE) {
    bStartFreigabe = TRUE;
}
```

```
bStartFreigabe = (vBatt >= 12) && (tankInhalt > 0)
                               && (!bKupplung);
```

Unterschied ??

Variante 3

```
bStartFreigabe = TRUE;

if(vBatt < 16) {
    bStartFreigabe = FALSE;
    error(1);
}
else if(tankInhalt == 0) {
    bStartFreigabe = FALSE;
    error(2);
}
else if (bKupplung) {
    bStartFreigabe = FALSE;
    error(3);
}
```

Aufgabe

```
if( alter > 6 ) {
    if( alter < 16 ) {
        printf("Junior");
    } else {
        printf("Erwachsen");
    }
}
```

- Aufgabe:
  Schreiben Sie das obige Beispiel mit Hilfe logischer Operatoren

Lösung

```
if( (alter > 6) && (alter < 16) ) {
   printf("Junior");
}
else if (alter >= 16 ) {
   printf("Erwachsen");
}
```

- Vermeidet Schachtelungen
- Keine der beiden Methoden ist „eindeutig“ besser (Programmierstil)
- && verbindet nur vollständige Bedingungen:
  if (alter > 6 && < 16) funktioniert nicht

Ternärer Operator

- Für eine einfache Auswahl innerhalb eines Ausdrucks gibt es eine Kurzform:

```
min = a < b ? a : b ;
```

- Das ist in diesem Fall eine Kurzform für:

```
if (a < b ) {
    min = a;
} else {
    min = b;
}
```

Ternärer Operator

- Allgemein:
  <bedingung> **?** *<wert wenn true>* **:** *<wert wenn false>*
- Während *if..else* eine Anweisung ist, kann der ternäre Operator in einem Ausdruck vorkommen:

```
printf("%s", ergebnis > 100 ? "gross" : "klein");
```

Nur verwenden, wenn nicht auf Kosten der Lesbarkeit!

Mehrfachauswahl: switch ...

Beispiel

- Ein double-Wert soll auf ein, zwei oder drei Nachkomma-Stellen gerundet werden
- Idee: Wert mit 10, 100, oder 1000 multiplizieren, auf die nächste ganzen Zahl runden und dann wieder durch 10, 100 oder 1000 dividieren

Erster Versuch

```
if (anzStellen == 1) {
    wert = wert * 10.0;
    wert = round(wert);
    wert = wert / 10.0;
}
if (anzStellen == 2) {
    wert = wert * 100.0;
    wert = round(wert);
    wert = wert / 100.0;
}
if (anzStellen == 3) {
    wert = wert * 1000.0;
    wert = round(wert);
    wert = wert / 1000.0;
}
```

Alternative: Mehrfachauswahl

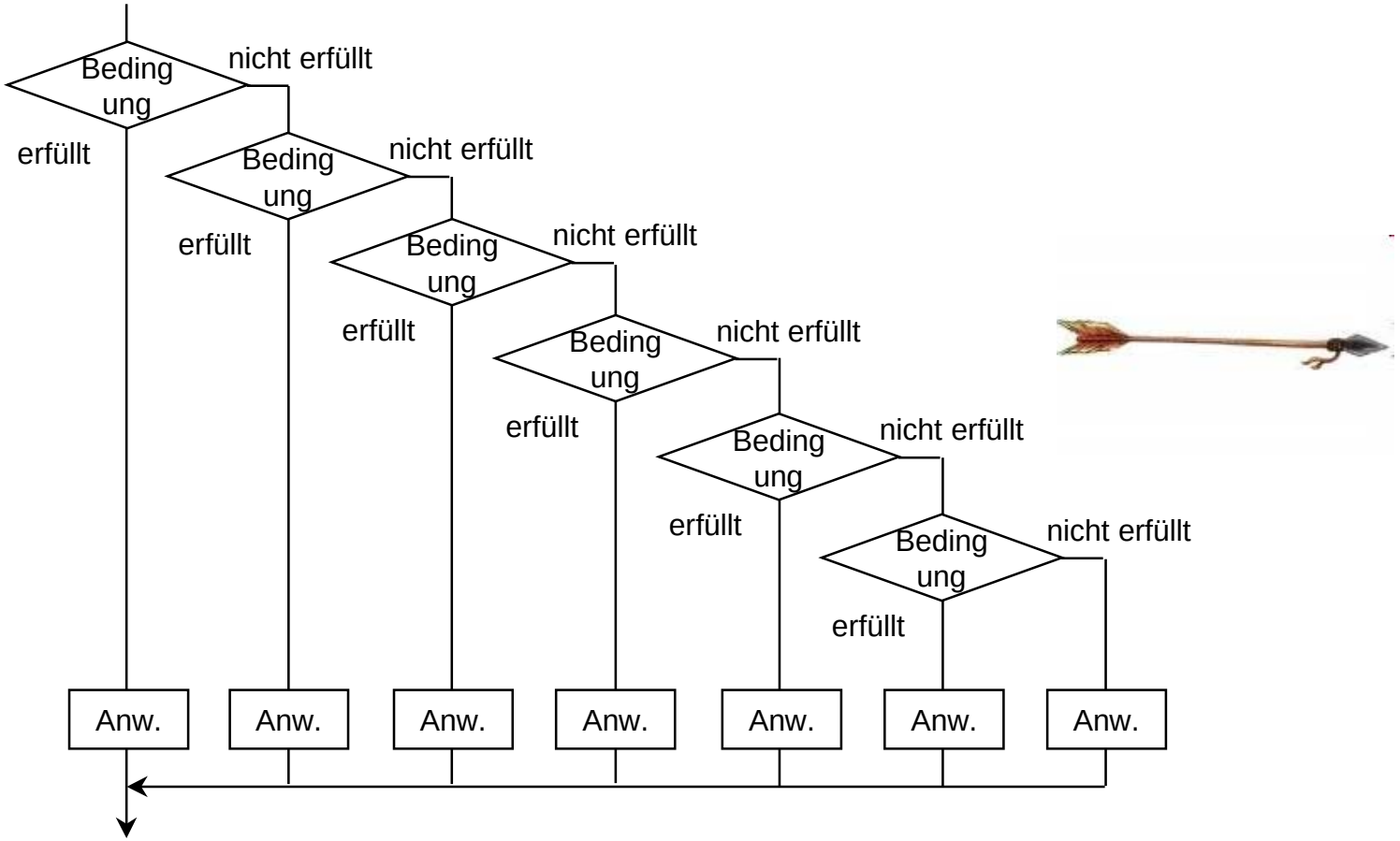
```
switch (anzStellen) {
    case 1: wert = wert * 10.0;
         wert = round(wert);
         wert = wert / 10.0;
         break;

    case 2: wert = wert * 100.0;
         wert = round(wert);
         wert = wert / 100.0;
         break;

    case 3:  wert = wert * 1000.0;
          wert = round(wert);
          wert = wert / 1000.0;
          break;

    default:  wert = wert * 100.0;
          wert = round(wert);
          wert = wert / 100.0;
}
```

Mehrfachverzweigungen



`switch` - Anweisung

Mehrfachauswahl

- Die Mehrfachauswahl wird in C durch die switch-Anweisung realisiert.
- Je nach Wert der Variablen springt das Programm zum entsprechenden case-Label und fährt dort weiter
- Achtung: ohne break-Befehl werden auch nachfolgende case-Labels ausgeführt (kann für Bündelung verwendet werden)
- Das default-Label steht für alle Fälle, die noch nicht mit case-Labels abgedeckt wurden (analog zu „else“)

Mehrfachauswahl (Beispiel)

```
int day = 3;
switch( day ) {
  case 1: printf("Montag");
          break;
  case 2: printf("Dienstag");
          break;
  case 3:
  case 4: printf ("Mittwoch oder Donnerstag");
          break;
  …
  default: printf("kein Tag");
}
```

break

- Vorsicht mit *break: vergessen*

```
case 1: printf("Montag");
case 2: printf("Dienstag");
```

- schreibt im Fall 1 sowohl Montag als auch Dienstag!

- Aber:

```
case 1: case 2: case 3: case 4: case 5:
  printf("Wochentag"); break;
case 6: case 7:
  printf("Wochenende"); break;
```

Sinnvolle Anwendung des Weglassens von „break“ sog. fall through

Grenzen der Mehrfachauswahl

- Die Mehrfachauswahl ist nicht beliebig flexibel

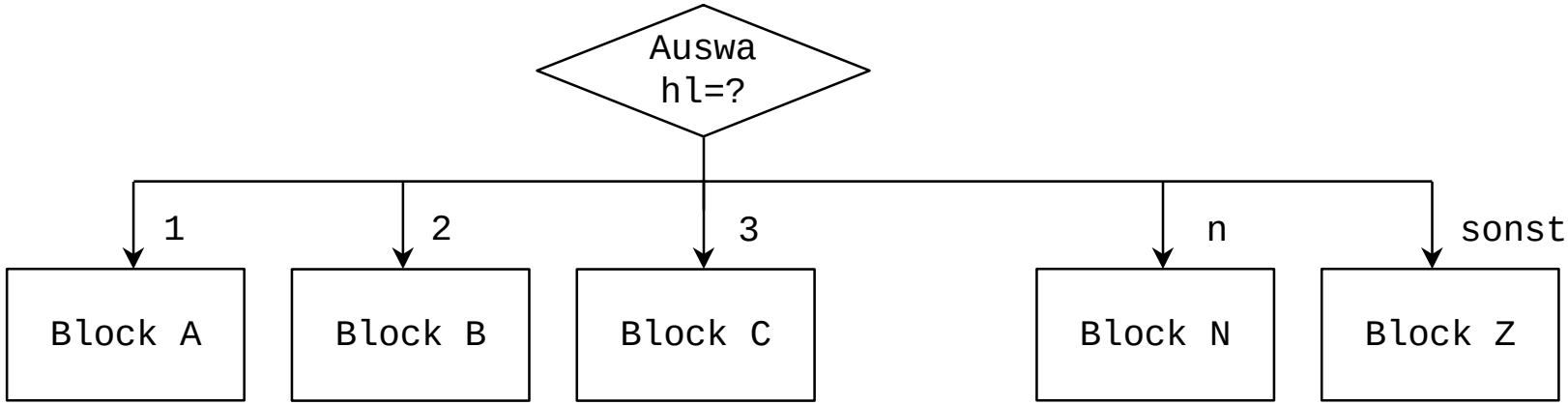
```
switch (a, b) {
    case a>b:  printf("grösser");
               break;
    case b>a:  printf("kleiner");
               break;
    case a==b: printf("gleich");
}
```

- Dies ist *kein korrektes C!*
  Geben Sie eine korrekte Alternative an!

Darstellung der Mehrfachauswahl

Auswahl = ?					
1	2	3	..	n	sonst
Block A	Block B	Block C	Block D	Block E	Block F

```
case Auswahl of
   1: Block A;
   2: Block B;
   3: Block C;
   …
   n: Block N;
else
   Block Z;
endcase;
```



Weiteres Beispiel

```
if (punkte == 15) {
    note = 6.0
} else if (punkte == 14) {
    note = 5.5
} else if (punkte == 13) {
    note = 5.0
} else if ...
```

```
switch ( punkte ) {
     case 15: note = 6.0; break;
     case 14: note = 5.5; break;
     case 13: note = 5.0; break;
     ...
     default: note = 3.0;
 }
```

Programmierstil Switch Anweisung

- Eine Case Auswahl sollte immer mit einer break Anweisung beendet werden; sonst denn Kommentar "fall through"
- Auch der letzte Case sollte ein break haben (wenn Switch Anweisung später erweitert wird)
- Immer einen Default-Case definieren - auch wenn nur eine Fehlermeldung ausgegeben wird.

Konstanten und Aufzählltypen

Konstanten (ab ANSI-C)

- Werden wie Variablen deklariert, allerdings mit vorgestelltem *const*
- Müssen bei der Deklaration auch definiert werden

```
const double EULERZAHL =  2.718281828459;
const unsigned long MAXIMUM = 87000L;
```

Aufzähltypen

- Beispiel

```
enum frucht { Apfel,
              Birne,
              Zitrone };
```

- Dem ersten Element wird der Wert 0 zugewiesen, dann 1, 2, …

```
enum frucht { Apfel = 1,
              Birne,
              Zitrone };
```

- Dem ersten Element wird der Wert 1 zugewiesen, dann 2, 3, …

Aufzählltypen

- Variable eines Aufzählltyps

```
// Aufzählltyp definieren
enum frucht { Apfel, Birne, Zitrone };

// Variable von diesem Typ
enum frucht einkauf;
einkauf = Zitrone;
printf("Ergebnis: %d\n", einkauf);    // 2
```

Aufzählltypen

- Der Typbezeichner kann auch weggelassen werden:

```
enum { Apfel,
       Birne,
      Zitrone };
```

- Ähnlich wie:

```
const Apfel = 0;
const Birne = 1;
const Zitrone = 2;
```

Aufzählltypen

- Der Wert eines Elements wird in Ausdrücken automatisch ermittelt
- Die Ein-/Ausgabe von Aufzählltypen ist nicht vordefiniert
- Bei der Definition können auch beliebige Werte zugewiesen werden

```
enum frucht { Apfel = 17, Birne = 15, Kirsche = 10};
```

Noch Fragen?

Lösung Mehrfachauswahl

```
if ( a > b ) {
    printf("grösser");
} else if ( b > a ) {
    printf("kleiner");
} else {
    printf("gleich");
}
```