INF1
Wiederholungen
while, do...while, for

- Schleife mit Goto
- Schleife mit Anfangsprüfung
- Schleife mit Endprüfung
- Schleifen mit Zähler: *for*
- Mehr zum Thema Schleifen

Wiederholungen

- Oft wollen wir gleiche oder ähnliche Anweisungen mehrmals hintereinander ausführen
- Bis jetzt:
  Code für solche Anweisungen mehrfach kopieren
- Beispiel:

```
printf("Zeile %d\n", 1);
printf("Zeile %d\n", 2);
printf("Zeile %d\n", 3);
printf("Zeile %d\n", 4);
printf("Zeile %d\n", 5);
...
```

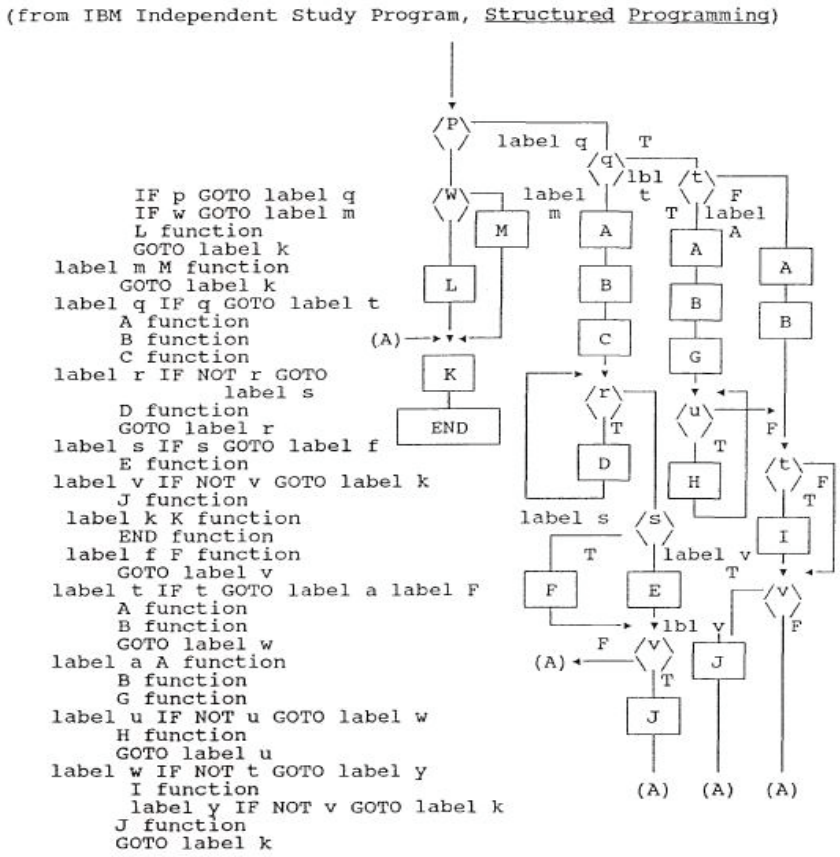
So nicht: Schleife mit goto...

- Im Prinzip würde eine Sprunganweisung genügen:

```
int i = 1;
start:
if ( i < 10 ) {
   printf("Zeile %d\n", i);
   i ++;
   goto start;
}
```

…Goto: Spagetti-Code

- Beispiel: unstrukturiertes Programm mit GOTOs (Spagetti-Code)
  - unübersichtlich
  - schlecht wartbar

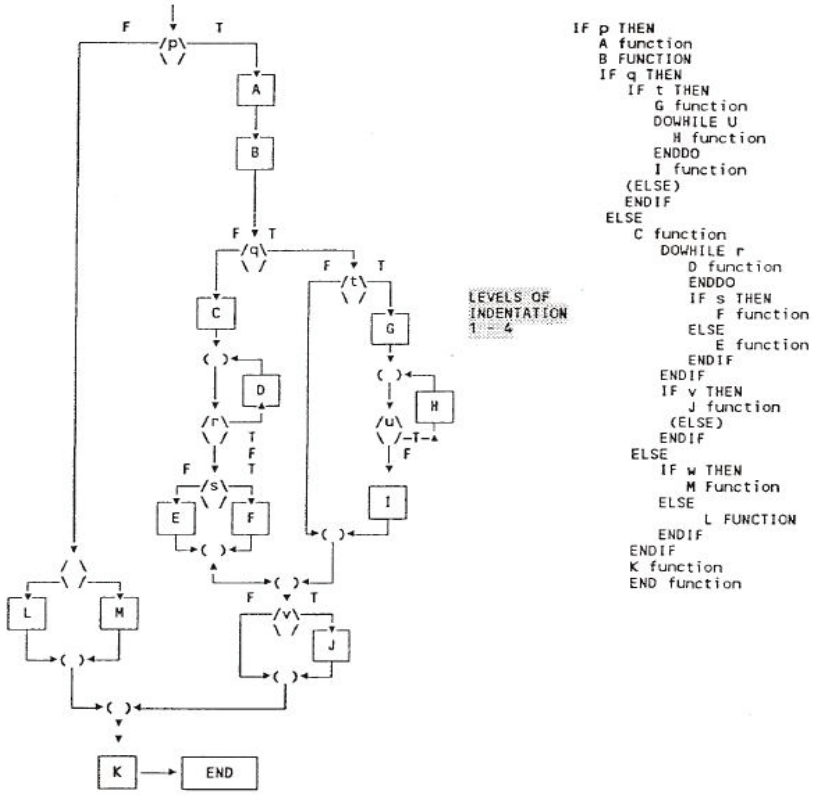


strukturierte Programmierung ohne GOTOs !

School of Engineering

- Beispiel: strukturiertes Programm ohne GOTOs
  - I.m. Nur 3 Konstrukte (Blöcke), die aneinander gereiht oder auch verschachtelt werden können:
    - ***Sequenz***
    - ***Entscheidung***
    - ***Wiederholung***
  - übersichtlich
  - gut wartbar

(from IBM Independent Study Program, Structured Programming)



goto

- Mit *goto* (Sprüngen) können ein paar wenige sinnvolle Dinge gemacht, aber auch viele Probleme herbeigeführt werden
- Neuere Sprachen wie Java/C# bieten in der Regel nur noch **eingeschränktes** goto-Statement an
- Auch in C vermeiden

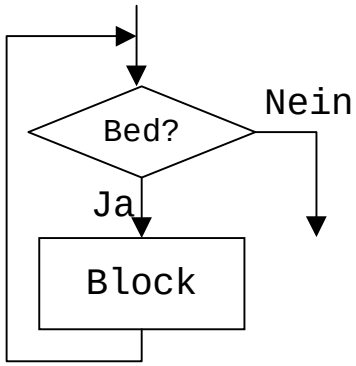
- “Go To Statement Considered Harmful”, von Edsger W. Dijkstra, Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148.

Schleifenarten

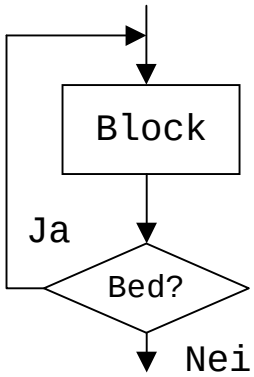
Die 3 Schleifenarten

- Eigentlich würde eine Schleifenart genügen
- Aus Bequemlichkeitsgründen hat man jedoch 3 definiert.
- 1. Prüfung **vor** dem eigentliche Verabeitungsblock
- 2. Prüfung **nach** dem eigentlichen Verabeitungsblock
- 3. **Feste Anzahl** Durchgänge
- Meist ist anhand Situation klar, welche Schleifenart zu wählen ist

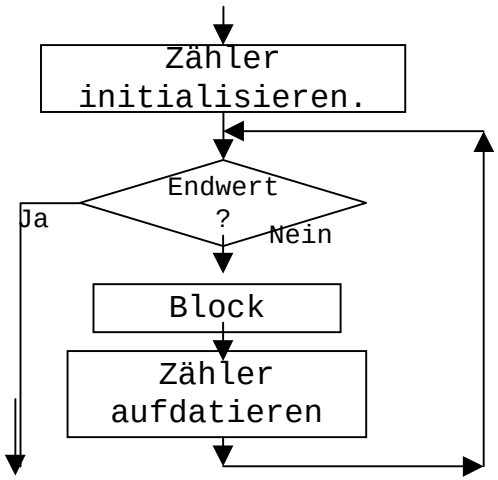
1. Vor



2. Nach



3. Feste Anzahl



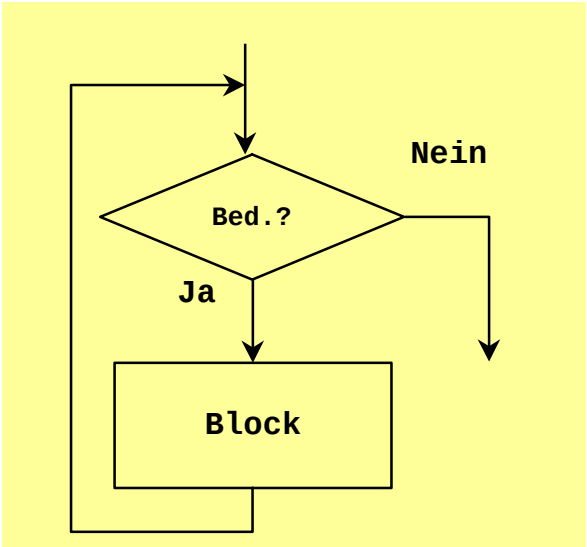
1. Vorabprüfung

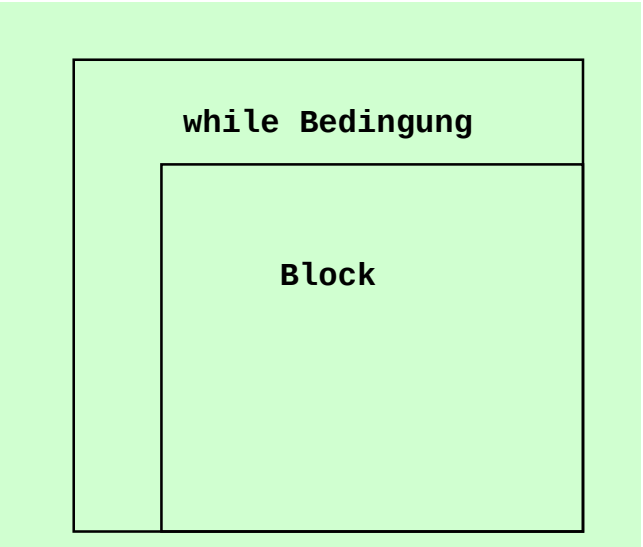
Arten von Schleifen - 1. While mit Vorabprüfung

- Schleife mit Vorabprüfung ("while")
- Der Block wird solange wiederholt, wie die Bedingung gilt
- Ist die Bedingung bereits am Anfang nicht erfüllt, wird der Block nie ausgeführt

```
while Bedingung do
  Block
end while
```

Pseudocode





While mit Vorabprüfung

while (*Ausdruck* **)**

Block

- Ausdruck: logischer Ausdruck, der wahr oder falsch ist
- Block wird solange ausgeführt, wie die Bedingung erfüllt ist (evtl. gar nicht)
- Block kann auch leer sein!

Beispiel: while

- Wir zeichnen eine Reihe „Sternchen“:

```
int counter = 0;

while (counter < 8) {
    printf("*");
    counter = counter + 1;
}
```

Wie viele Sternchen werden gezeichnet? Wieso?

- Kein so gutes Beispiel für eine while-Schleife!

  Für eine feste Anzahl von Wiederholungen sehen wir gleich einen besser geeigneten Schleifentyp

Beispiel: while (Forts.)

- Wie ist es, wenn wir die Bedingung ersetzen durch:
  - counter <= 8
  - counter > 8
  - counter == 8
  - counter != 8

```
int counter = 0;
while ( ... ) {
    printf("*");
    counter = counter + 1;
}
```

while-Schleife: Beispiel 2

- Was macht das Programm?

```
int i = 1;

while (i < 100) {
    printf("%d\n", i);
    i = i * 2;
}
```

Programmierstil

- Zur Erinnerung: Konsequent konsistente Einhaltung der Darstellung und Einrückung von Blocks

```
bool done;
while (! done) {
  printf("Processing...\n");
  done = next_entry();
}
```

Fallstricke

- Sind die im Test und im Schleifenblock verwendeten Variablen richtig initialisiert?
- Ist sichergestellt, dass der Bedingungsteil irgendwann logisch falsch wird? D.h.: Erreicht die Schleife je die Abbruchbedingung?

Endlosschleifen

- Überlegen Sie sich, wie Sie eine Endlosschleife programmieren können
- Wieso würden Sie so etwas programmieren wollen?
- Wie viele Sterne werden ausgegeben?

```
int counter = 0;

while (counter < 8){
    printf("*");
}
```

A

```
int counter = 0;

while (counter < 8);{
    printf("*");
    counter = counter + 1;
}
```

B

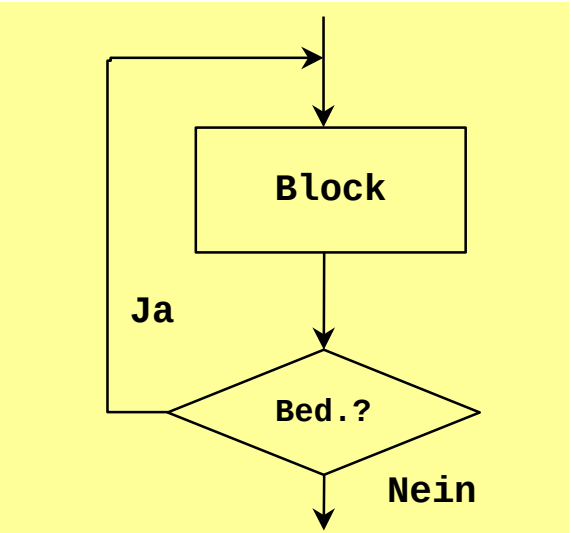
2. Endprüfung

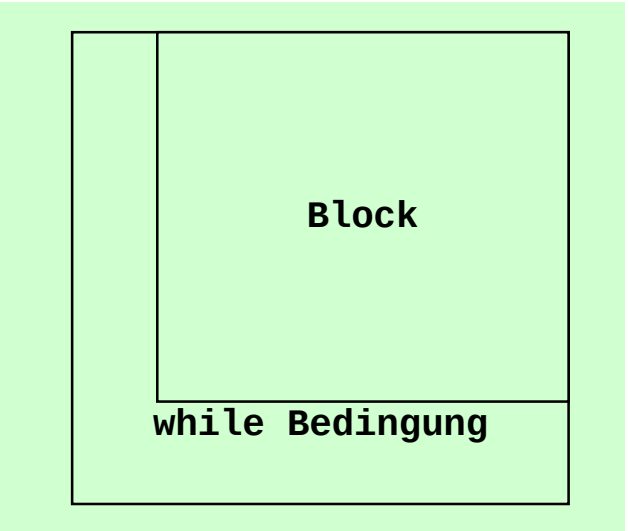
Arten von Schleifen - While mit Endprüfung

- Schleife mit Endprüfung ("do..while")
- Der Block wird mindestens einmal ausgeführt
- Er wird dann wiederholt, solange die Bedingung erfüllt ist

```
do
  Block;
while Bedingung
```

Pseudocode





While mit Endprüfung

- Ausdruck: logischer Ausdruck, der wahr oder falsch ist
- Anweisung wird mindestens einmal ausgeführt
- Anweisung wird danach solange ausgeführt, wie die Bedingung erfüllt ist (evtl. kein weiteres mal)

- EBNF: 

```
do-anweisung ::=
        "do" anweisung "while" "(" ausdruck ")"
```

Beispiel: do..while

- Wir würfeln:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
int main(){
   int wurf = 0;
   int versuch = 0;
   /* Intializes random number generator */
   srand(time(0));
   do {
       wurf = rand()%6 + 1;
       versuch += 1;
       printf("Versuch %d : %d\n",versuch, wurf);
   } while (wurf != 6);
}
```

andernfalls wird immer dieselbe Folge von Pseudozufallszahlen erzeugt ev. millis() verwenden

```
long millis() {
    return (long)(clock()*1000
         /CLOCKS_PER_SEC);
}
```

```
void sleep_ms(int millis) {
#ifdef WIN32
    Sleep(millis);
#else
    usleep(millis * 1000);
#endif
}
```

http://www.tutorialspoint.com/cprogramming/

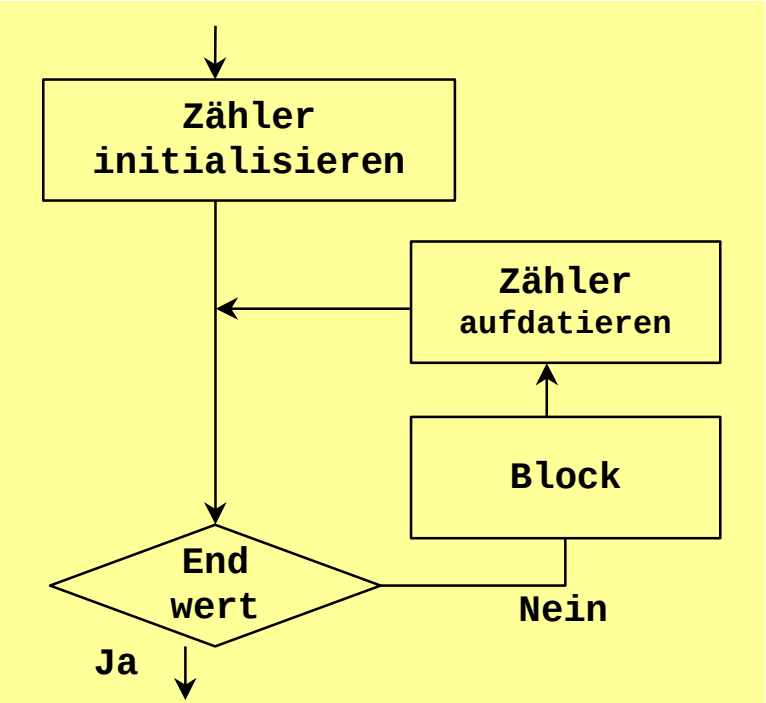
3. Fixe Anzahl Wiederholungen

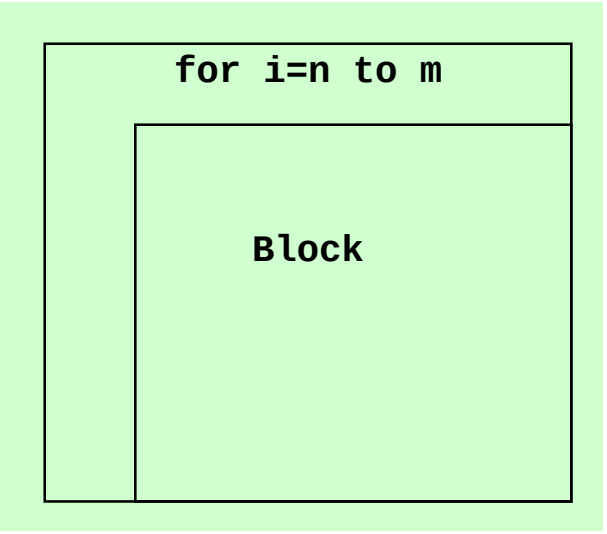
Arten von Schleifen - 3. Schleife mit Zähler

- Schleife mit fixer Anzahl Wiederholungen ("for"-Schleife)
- Der Block soll eine vorbestimmte Anzahl mal ausgeführt werden

```
for i=n to m do
  Block;
end for
```

Pseudocode





Schleifen mit Zähler: for

- Die *for*-Schleife ist primär dazu gedacht, einen Block eine fixe Anzahl mal zu wiederholen

- EBNF: 

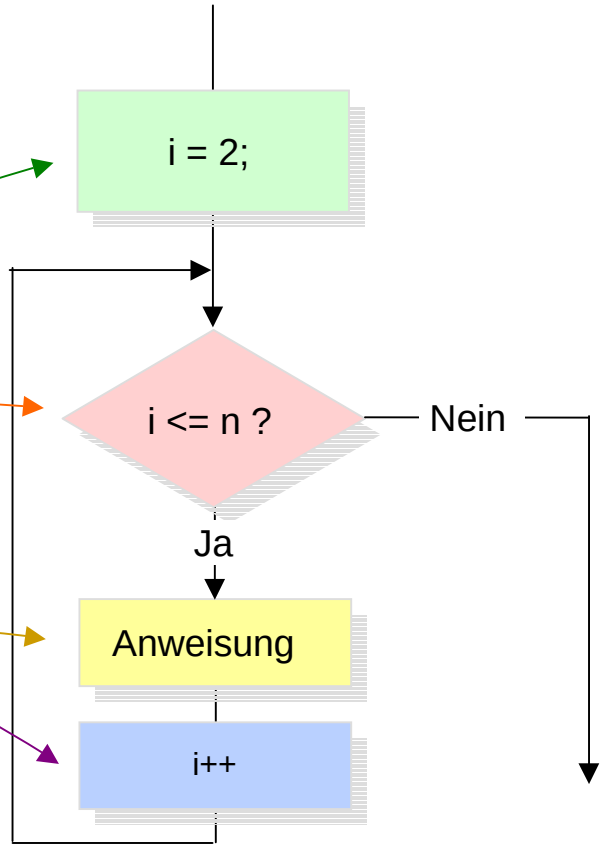
siehe Anhang

```
for-anweisung ::=
            "for"
            "(" initialisierung ";" bedingung ";" aufdatierung ")"
    anweisung
```

3. Schleife mit Zähler

Was macht der Compiler aus einer for-Anweisung ?

```
int i;
for  (i = 2 ;   i <= n ;   i++)
  Anweisung;
```



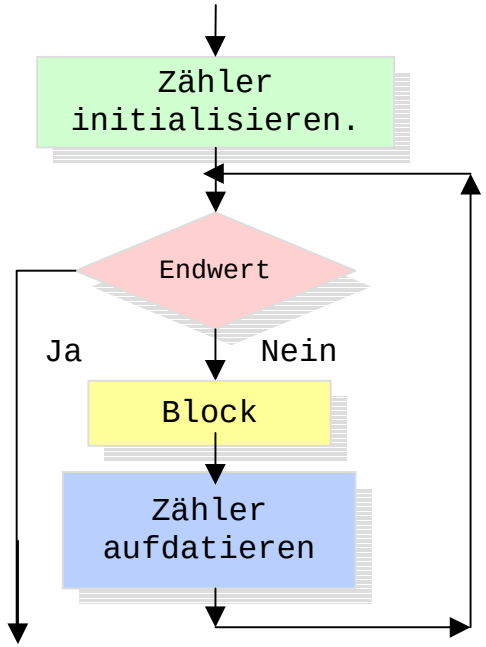
... Schleife mit Zähler

- einen Block eine fixe Anzahl mal zu wiederholen ist die for-Scheife primär dazu gedacht,
- EBNF:

  for (*[Initialisierung]* **;** *[Bedingung]* **;** *[Aufdatierung]* **)**

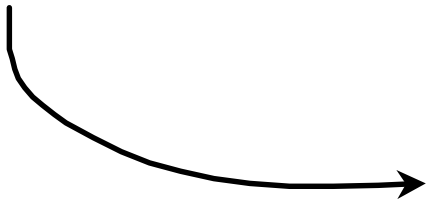
  Block

- Initialisierung: wird **GENAU** ein Mal, vor dem ersten Durchlauf der Schleife ausgeführt.
  - Kann auch Deklarationen enthalten (nur innerhalb der Schleife sichtbar)
  - Laufvariable dort zu deklarieren ist guter Stil
- Bedingung: muss logischer Ausdruck sein
  - Bedingung wird VOR jedem Schleifendurchlauf getestet
- Aufdatierung: kann Zuweisungen enthalten, keine Deklarationen
  - Wird am ENDE des Schleifendurchlaufs ausgeführt



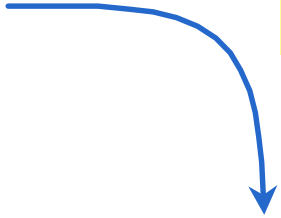
… Schleifen mit Zähler: for

Altes Beispiel mit *while*



```
int counter = 0;
while( counter < 8 ) {
        printf("*");
        counter += 1;
}
```

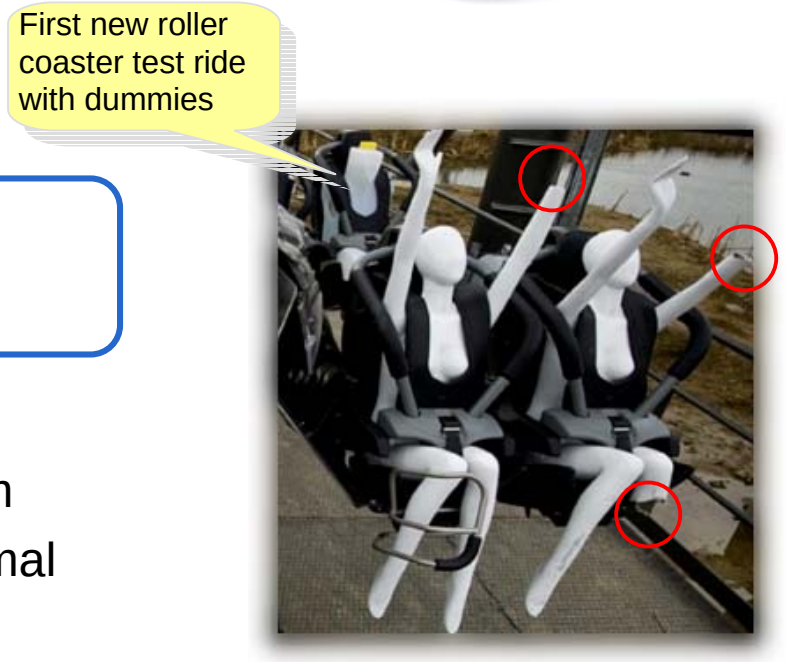
Jetzt mit *for*



```
int counter;
for (counter = 0; counter < 8; counter++ ) {
    printf("*");
}
```

Fallstricke

- Ändern Sie die Laufvariable in der Schleife nicht (es geht!)
- Verwechseln Sie nicht Laufvariablen
- Nicht einmal zu viel oder zu wenig durchlaufen
  (0 ... n-1 oder 1 ...n ?)



Die Schleifenvariable sollte vom Typ int sein

double/float können zu Ungenauigkeiten beim Aufaddieren führen, so dass die Schleife einmal mehr als erwartet durchlaufen wird.

Programmierstil

```
// Summe von 1 bis 10
const int n = 10;
int sum = 0, i;
for (i = 1; i <= n; i++) {
      sum += i;
}
```

Don't try be clever

```
// kürzere Version – aber unleserlich
const int n = 10; int i, sum;
for (i = 1, sum = 0; i <= n; sum += i++);
```

Wie oft wird der Schleifenkörper durchlaufen

- Quizfrage

```
double k;
double step = 1;
for (k = 0; k < 10.0; k += step){
    printf("%.1f\n",k);
}
```

A

```
double k;
double step = 0.1;
for (k = 0; k < 10.0; k += step){
    printf("%.1f\n",k);
}
```

B

Mehr zum Thema Schleifen

Verwendung der Arten

- Verwenden Sie je nach Anforderung die passende Schleifenart
- Meist ergibt sich die passende Art anhand folgender Bedingungen

wenn die Anzahl der **Wiederholungen bekannt** -> ***for*** Schleife

wenn die Anzahl nicht bekannt ist (durch eine Bedingung geprüft wird)

- ***do-while*** wenn der Schleifenrumpf mindestens einmal ausgeführt werden soll
- ***while*** in allen andern Fällen

Welche Schleifenart?

- Es sollen alle Monate des Jahres ausgegeben werden
- Es soll so lange gewürfelt werden, bis ein 6 kommt
- Es sollen so lange eine Zahl eingelesen werden, bis sie eine Primzahl ist
- Es soll ein Intervall halbiert werden, bis die verbleibende Strecke < EPS ist
- Alle Grossbuchstaben sollen zu Kleinbuchstaben umgewandelt werden

Vorzeitiger Abbruch

break

- Mit *break* kann eine Schleife vorzeitig beendet werden:
  Aus einer Schleife *heraus springen*
- Bitte verwenden Sie dieses Konstrukt sparsam!
- Beispiel:

```
int found = 0;
for( i = 0; i < length; i++ ) {
    if( liste[i] == meineVar ) {
        found = 1;
        break;
    }
}
```



```
int found = 0;
for( i = 0; i < length && ! found; i++ ) {
    if( liste[i] == meineVar ) {
        found = 1;
   }
}
```

continue

- Mit *continue* kann der Rest eines Schleifenkörpers übersprungen werden
- Bitte verwenden Sie dieses Konstrukt sparsam!
- Beispiel:

```
for( int i = 0; i < length; i++ ) {
    if( i != meineVar ) continue;

    // sonst machen wir etwas Komplexes,
    // was viele Zeilen Code lang ist
}
```

Geschachtelte Schleifen

Zeilen voller Sterne

- Neue Aufgabe: Es sollen mehrere Zeilen voller Sterne ausgegeben werden
- Für eine Zeile haben wir das Problem bereits gelöst:

```
int counter;
for(counter = 0; counter < 8; counter++ ) {
    printf("*");
}
printf("\n");
```

Zeilen voller Sterne

School of Engineering

- Wie zeichnen wir nun *drei* Zeilen voller Sterne?
- Eine Möglichkeit:

```
int counter;
for(counter = 0; counter < 8; counter++ ) {
    printf("*");
}
printf("\n");
for(counter = 0; counter < 8; counter++ ) {
    printf("*");
}
printf("\n");
for(counter = 0; counter < 8; counter++ ) {
    printf("*");
}
printf("\n");
```

Noch eine Variante

- Gleiches Ergebnis
- Unübersichtlich !

```
int zaehler = 0;
while ( zaehler < 24 ) {
    printf("*");
    zaehler++;
    if ( zaehler % 8 == 0 ) {
        printf("\n");
    }
}
```

Verschachtelte Schleifen

- Eigentlich wollen wir in 2 Dimensionen wiederholen:
  - Wir wollen x Sternchen auf einer Zeile darstellen
  - Wir wollen y solche Zeilen untereinander darstellen

- Wie könnte also eine elegantere Lösung aussehen?

- Rückblick: for-Schleife

  for (*[Initialisierung]* **;** *[Bedingung]* **;** *[Aufdatierung]* **)**

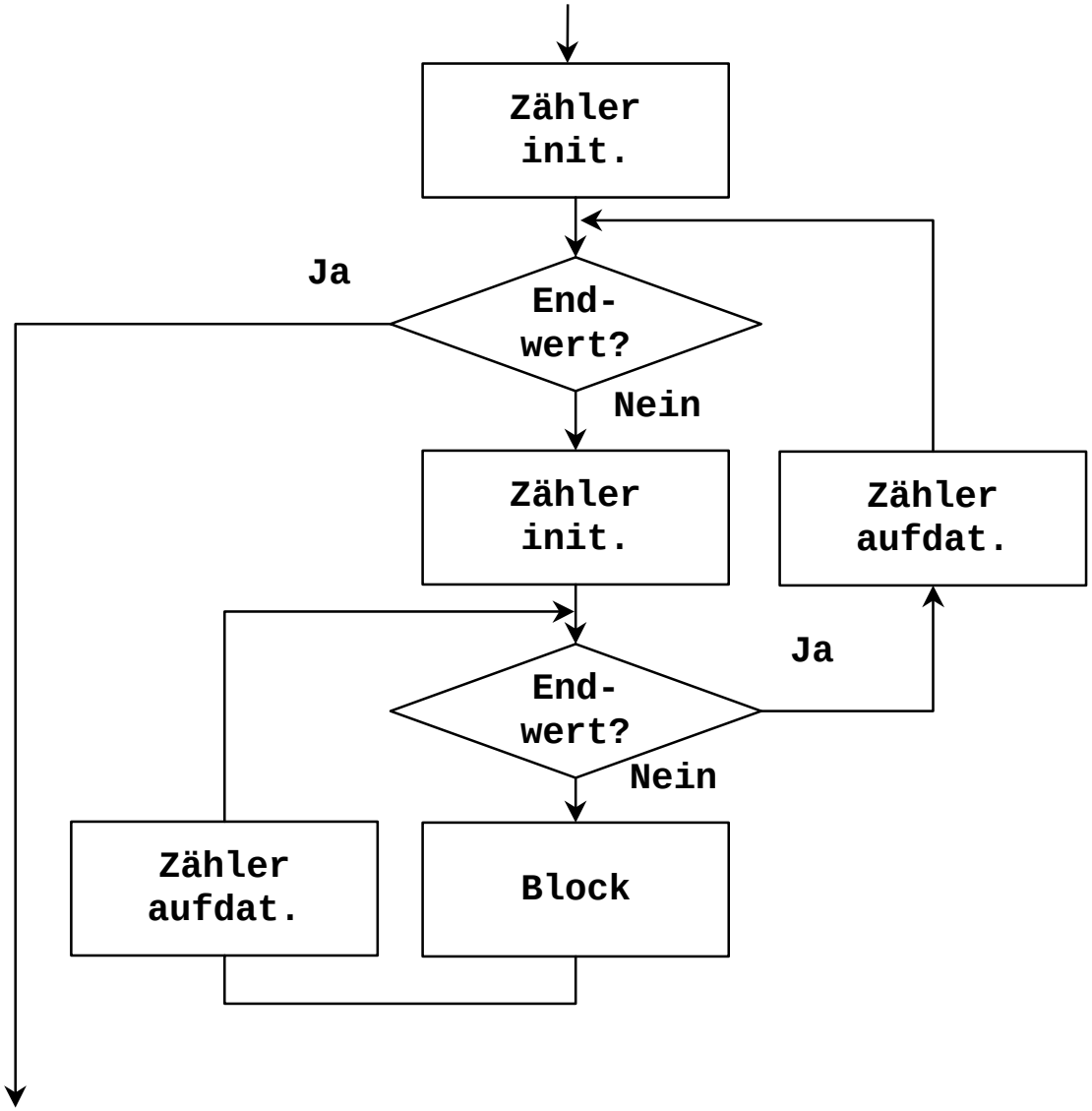
  Block

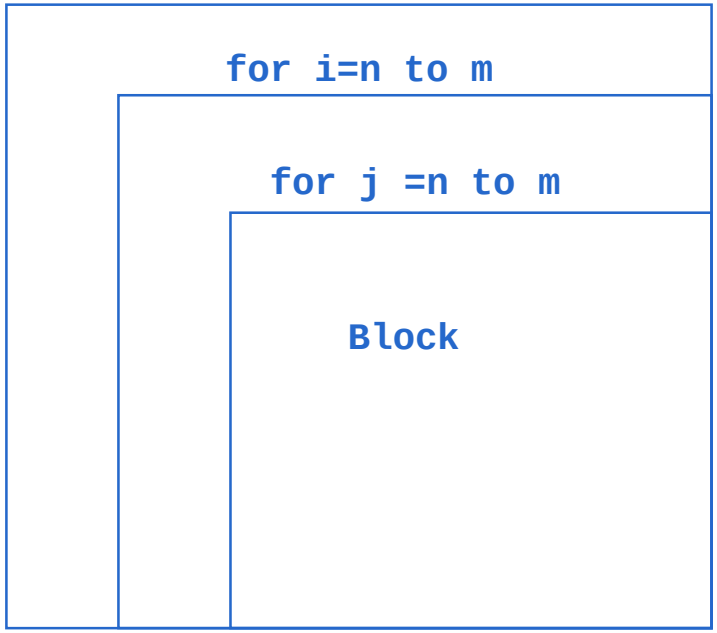
Verschachtelte Schleifen

```
for (rows = 0; rows < 3; rows++ ) {
    // Eine Reihe Sterne ausgeben
        for(counter = 0; counter < 8; counter++ ) {
            printf("*");
        }
    printf("\n");
}
```

- Zeichnet 3 Reihen à 8 Sternchen
- Die Zweidimensionalität drückt sich in zwei verschachtelten Schleifen aus

Darstellung






```
for i=n to m do
    for j=n to m do
        Block;
    end for
end for
```

Verschachtelte Schleifen

```
for (i = 0; i < 20; i++) {
  ...
  // do something nice..
  ...
  for (j = 0; j < 20; j++) {
    // do even more
  }
  for (j = 0; j < 5; j ++) {
    // another loop
  }
  ...
  //more code
}
```

Verschachtelte Schleifen

- Nicht nur *for*-Schleifen können verschachtelt werden
- Auch *while* und *do..while* enthalten einen Block, der weitere Schleifen enthalten kann

```
int i = 0;
int step = 1;
while (i < 10) {
    printf("i:%d\n", i);
    int j = 0;
    while (j < 10) {
        printf("j:%d\n", j);
        j += step;
    }
    i += step;
}
```

Rückblick

School of Engineering

- Wir haben in diesem Kapitel behandelt:
  - Das Konzept einer Wiederholschleife
  - Drei Arten von Schleifen
  - Die Verschachtelung von Schleifen und anderen Ablaufstrukturen
  - Der situationsgerechte Einsatz von Schleifen
  - Die grafische Darstellung von Schleifen

Noch Fragen?

Syntaxbeschreibung: EBNF, Syntaxdiagramme

EBNF

- Problem: Wie kann man möglichst kompakt und vollständig beschreiben, was eine gültiges Programm-Codestück ist?

- Möglichkeit, die Grammatik einer Programmiersprache exakt und kompakt zu beschreiben bietet EBNF (Erweiterte Backus-Naur-Form)

- Eine EBNF-Grammatikregel hat folgende allgemeine Form:

```
Name ::= Grammatikregel in EBNF
```

EBNF

Metasymbole *[ ], { }, | ,* und *( )* mit folgender Bedeutung:

- `[a]` a optional
- `{a}` a kann 0, 1, oder mehrmals vorkommen
- `a|b` a oder b kann vorkommen, aber nicht beide
- `(a|b) c` a c oder a c kann vorkommen

- Weitere Konventionen
  - Wlemente, die genau so im Programmcode vorkommen, werden in Anführungszeichen "..." oder '...' eingeschlossen (= Terminalsymbole)

EBNF while-Anweisung (teilw.)

```
while-anweisung ::=
     "while" "(" ausdruck ")" anweisung

anweisung ::=
     block-anweisung
     | while-anweisung
     | return-anweisung
     | ...

block-anweisung::=
     "{" vereinbarungsliste anweisungsliste "}"
```

if-Anweisung

```
if-anweisung ::=
        "if" "(" ausdruck ")" anweisung
        [ "else" anweisung ]
```

switch-Anweisung

```
switch-anweisung ::=
      "switch" "(" ausdruck ")" "{" case { case } "}"

case ::=
      "case" konstanter-ausdruck ":" { ausdruck }
      | "default" ":" { ausdruck }
```