2 Kontrollstrukturen, strukturierte Programmierung

2.1 Algorithmen

Für die Entwicklung von Programmen, die komplexere Aufgaben lösen sollen, bewährt sich ein zweistufiges Verfahren. In der ersten Phase bemüht man sich um die grundsätzliche Lösung, das heißt, man formuliert die einzelnen Schritte zur Lösung der Aufgabe. Eine Anleitung zur Lösung einer Aufgabe nennt man einen Algorithmus. Nach der Erfassung der Aufgabenstellung (Problemanalyse) ist die Formulierung einer solchen Anleitung die im Grunde wichtigste und oft auch schwierigste Phase der Programmentwicklung. Der Anfänger muss sich zunächst daran gewöhnen, die Lösung einer Aufgabe in relativ primitiven Einzelschritten anzugeben. Im Gegensatz zum Computer verfügen wir über erstaunliche Fähigkeiten und können Aufgaben nach völlig anderen Mustern lösen als dieser. Um für einen Computer Algorithmen zu entwerfen, müssen wir wieder lernen, mit einfachen Anweisungen Aufgaben lösen. Die Stärke des Computers ist die Geschwindigkeit und Zuverlässigkeit, mit der er diese einfachen Anweisungen ausführt, die große Speicherkapazität und die Tatsache, dass Maschinen beliebig oft und beliebig lange ihre eintönige Arbeit verrichten.

Ein Algorithmus ist also eine Folge von Anweisungen, deren korrekte Ausführung eine bestimmte Aufgabe durchführt. Die Ausführung selber wird als Prozess bezeichnet. Prozessor nennt man allgemein jene Einheit, die Prozesse ausführt. Beispiele für Algorithmen aus dem täglichen Leben sind Kochrezepte, Bastel- oder Bauanleitungen. Aber auch Eingelernte und im Laufe der Zeit automatisierte Rechentechniken wie das händische Multiplizieren, Dividieren oder Wurzelziehen sind Beispiele für Algorithmen.

Die Anweisungen in Algorithmen formuliert man zunächst mit einfachen, knappen Sätzen. Man nennt diese Formulierung Pseudocode. Pseudocode deshalb, weil dieser Code nicht für einen bestimmten Compiler und schon gar nicht für einen bestimmten Prozessor bestimmt ist. Alternativen zu Pseudocode sind Flussdiagramme oder Struktogramme.

2.2 Strukturierte Programmierung

Bald hatte man auch in der Geschichte der Softwareentwicklung erkannt, dass ein unsystematisches drauflos programmieren nicht möglich ist. Deshalb wurden neue Methoden des Entwurfs, mit Einschränkungen in den Kontrollstrukturen, neuen Darstellungen der Kontrollstruktur und neue Programmiersprachen entwickelt. Einige Meilensteine dieser Entwicklung sind verknüpft mit Namen wie Dijkstra, Hoare, Wirth, Nassi und Shneiderman. Es entstand das Schlagwort der strukturierten Programmierung. Ein wesentliches Element der strukturierten Programmierung ist die Einschränkung des Kontrollflusses auf einige wenige Grundelemente. Für die Notation oder die graphische Darstellung des Kontrollflusses gibt es mehrere Möglichkeiten. Für Anfänger sind die Verwendung Pseudocode und Struktogrammen (Nassi-Shneiderman-Diagramme) immer noch eine gute Entscheidung.

Der Vorteil von Pseudocode ist, daß man ihn mit einem normalen Editor schreiben kann. Für Pseudocode gibt es keine einheitliche Norm. Ich verwende in diesem Buch eine leicht lesbare Form mit deutschsprachigen Schlüsselwörtern.

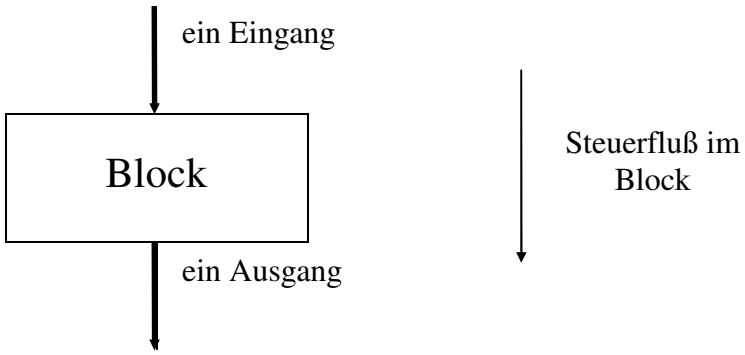
Die Struktogramme in diesem Buch entsprechen aus didaktischen Gründen nicht exakt der DIN-Norm. Dadurch ist es möglich, bei der Darstellung von Schleifen zwischen Ausführungsbedingung und Abbruchbedingung zu unterscheiden.

In Verbindung mit geeigneten Sprachen ist das prinzipielle Ziel der strukturierten Programmierung, im Quelltext des Programms den Steuerfluß des Programmes gut sichtbar zu machen. Dies erreicht man durch Einschränkung auf einige wenige Grundmuster. Die Theorie der Algorithmen kann zeigen, daß jedes prinzipiell mit einem Computer lösbare Problem durch die Anwendung von drei Grundstrukturen des Kontrollflusses gelöst werden kann:

- Folge, Sequenz
- **Auswahl, Selektion**
- **Wiederholung, Iteration**

Ein weiteres Prinzip der Programmstrukturierung ist die Top-Down-Entwicklung durch schrittweise Verfeinerung. Dabei wird eine zu lösende Teilaufgabe zunächst als Block (Strukturblock) in den Programmentwurf aufgenommen. Für den Aufbau eines solchen Bausteins gelten folgende Regeln:

- Ein Block ist eine abgeschlossene funktionale Einheit, d.h. jeder Baustein steht für eine klar definierte Aufgabe mit einem bekannten Anfangszustand und einem bekannten Endzustand.
- Ein Block hat, als Ganzes betrachtet, genau einen Eingang und genau einen Ausgang, d.h. er kann nur am Beginn betreten und am Ende verlassen werden. Damit sind Sprünge an beliebige Stellen ausgeschlossen. Zugelassen ist nur der Aufruf einer Funktion.
- Der Steuerfluß läuft in einem Block immer von oben nach unten.



Regeln für die Verknüpfung von Strukturblöcken:

- Auf einer Ebene des Entwurfs werden Blöcke aneinandergereiht oder im Falle der Auswahl nebeneinandergestellt.
- Der Steuerfluß läuft durch ein aus Bausteinen zusammengesetztes Programm immer von oben nach unten und entspricht damit der Reihenfolge im Programmcode. Nur bei Wiederholungen treten Rücksprünge auf, die jedoch an der Gesamtlogik nichts ändern.
- Zunehmende Verfeinerung führt zur Auflösung eines Blocks in weitere Blöcke. So entstehen aufeinander folgende, aber auch ineinander geschachtelte Blöcke. Zu einem Block in einer Entwurfsebene gibt es weitere Struktogramme in der nächst feineren Entwurfsebene. Die verfügbare Software für die Entwicklung von Struktogrammen unterstützt diese Methode. Was für den Konstrukteur das CAD Programm ist, ist für den Softwareentwickler ein System von CASE-Tools (CASE = Computer Aided Software Engineering). Am Papier kann die Änderung oder Verfeinerung von Struktogrammen bald mühsam werden. Vielleicht ist dies einer der Gründe, warum

Struktogramme trotz ihrer Vorteile nicht von allen Entwicklern gern verwendet werden.

- Ein untergeordneter Baustein erhält die Steuerung nur von dem ihm übergeordneten Baustein und gibt sie wieder dorthin zurück.

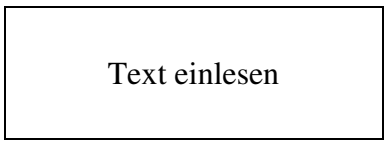
2.3 Arten von Strukturblöcken

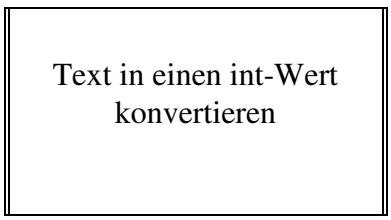
Die Forderungen der strukturierten Programmierung erzwingen die bereits erwähnten Einschränkungen im Kontrollfluß auf Sequenz, Auswahl und Wiederholung. Im Prinzip kann ein nach den Regeln der strukturierten Programmierung erstellter Programmentwurf in jeder Programmiersprache codiert werden. Die Umsetzung eines Struktogramms in Code ist immer möglich. Es kann jedoch nicht der Steuerfluß zu einem beliebigen Code nachträglich mit Struktogrammen dargestellt werden. Für die Beurteilung der Qualität einer Programmiersprache ist entscheidend, wie die strukturierte Programmierung durch geeignete Konstrukte unterstützt wird. Die Gegenüberstellung von Struktogrammen und Codierung in C soll die Arbeitstechnik der Trennung von Entwurf und Codierung fördern. Die allgemeine Darstellung wird durch praktische Beispiele ergänzt.

2.3.1 Elementarblock

Ein Elementarblock ist ein Strukturblock, der aus einer Arbeitsanweisung besteht. Diese Anweisung soll in der Entwurfsphase eine kurze Beschreibung der Aufgabe sein, die auf der betrachteten Ebene nicht weiter aufgelöst (verfeinert) wird. Struktogramme sind primär ein Entwurfshilfsmittel, und enthalten deshalb in dieser Phase keinen Programmcode.

Beispiele:





Die Verfeinerung in einer speziellen Funktion, die an dieser Stelle aufgerufen wird, wird durch einen seitlichen Doppelstrich ausgedrückt.

In C# taucht der Elementarblock auf als einzelne Anweisung

```
c = text[i];
```

als Liste von Anweisungen, die in einem Block (Verbundanweisung) zusammengefasst sind.

```
{
   z = z*10;

   z = z + (int) c - (int) '0';

}
```

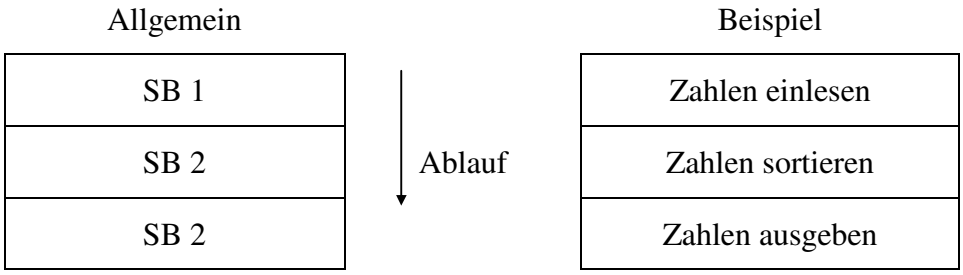
und als Aufruf einer Methode:

```
calculator.Add();
```

Die von der strukturierten Programmierung geforderte Abgeschlossenheit eines Strukturblocks ist in idealer Weise erfüllt, wenn der Strukturblock als Methode implementiert wird. Aber auch ein Block bietet mit einem optionalen Vereinbarungsteil die Möglichkeit z.B. Hilfsvariablen lokal zu definieren:

2.3.2 Sequenz

Die Sequenz ist die Aneinanderreihung von elementaren Strukturblöcken (SB).



Es ist eine Frage der Verfeinerungsstufe, ob man die Zusammenfassung mehrerer Anweisungen zu einer Verbundanweisung als Beispiel für eine Sequenz oder einen Elementarblock betrachtet.

2.3.3 Auswahl

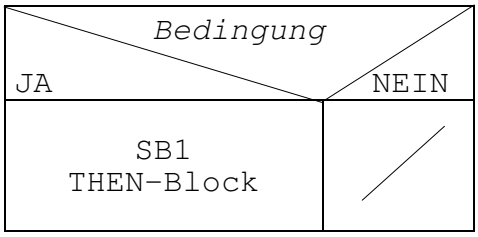
Bei der Auswahl (Selektion, Verzweigung) kann man vier Möglichkeiten unterscheiden.

Einfachauswahl

Pseudocode

```
Falls Bedingung dann
  Strukturblock SB1
```

Struktogramm



C#-Syntax:

```
if-statement-1:
  if ( boolean-expression ) statement
```

boolean-expression ist ein Ausdruck, dessen Resultat entweder `true` oder `false` ist. Die Anweisung wird nur ausgeführt, wenn dieser Ausduckt `true` ist Benötigt man mehr als eine "Anweisung" dann werden die Anweisungen in einem durch { … } zu einem Block zusammengefasst.

Eine andere Bezeichnung für die Einfachauswahl ist "bedingte Anweisung". Ein Strukturblock wird in Abhängigkeit von einer Bedingung ausgeführt oder nicht ausgeführt.

Beispiele:

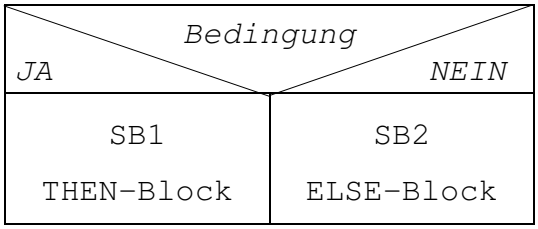
```
if (x > y)
{ // x und y tauschen
  h = x;
  x = y;
  y = h;
}


Console.Write("Der Schüler hat ");
if ( gesamtnote <= 1.5 )
  Console.Write("mit Auszeichnung ");
Console.Write("bestanden\n");

if (wert > maximum)
  Console.Write("Maximalwert ueberschritten !\n");

if (x < 0.0)
{
  x = -x;
  Debug.WriteLine(
    "Nur Werte >= 0 erlaubt. Vorzeichen wurde geändert");
}
```

Zweifachauswahl

Pseudocode

```
Falls Bedingung dann
  Strukturblock SB1
sonst
  Strukturblock SB2
```

Struktogramm



C#-Syntax :

```
conditional-statement-2 :
  if ( boolean-expression )
    statement
  else
    statement
```

Hat *boolean-expression* das Resultat true, so wird der THEN-Block ausgeführt; ist das Resultat false, so wird der ELSE-Block ausgeführt.

Beispiele:

```
if (Char.IsControl(c))
  editUnicode.Text = "Control";
else
  editUnicode.Text = c.ToString();
```

```
if (x <= y)       // den kleineren von zwei Werten
  min = x;        // auswaehlen
else
  min = y;
```

```
if (c >= 'a' && c <= 'z')  // ist c ein Kleinbuchstabe ?
  lowerCaseLetters++;
else
  otherLetters++;
```

Da das *if-statement* zur Syntaxkategorie *statement* zählt, kann man natürlich schreiben

```
if (a == 1)
  if (b == 2)
    Console.Write("a = 1 und b = 2\n");
```

Genauso kann eine `if-else`-Anweisung als Teil einer anderen `if`-Anweisung verwendet werden:

```
if (a == 1)
  if (b == 2)
    Console.Write("a ist gleich 1 und b ist gleich 2\n");
  else
    Console.Write("a ist gleich 1 aber b ist ungleich 2\n");
```

Hier ergibt sich eine semantische Schwierigkeit. Von der Syntax her ist nicht klar, welchem `if` der `else`-Teil zugeordnet ist. Der folgende Code ist syntaktisch äquivalent:

```
if (a == 1)
  if (b == 2)
    Console.Write("a ist gleich 1 und b ist gleich 2\n");
else
  Console.Write("a ist gleich 1 aber b ist ungleich 2\n");
```

Es gilt die Regel: Ein `ELSE`-Block wird dem nächstliegenden `if` zugeordnet, so daß der erste Code die Situation richtig darstellt.

Zur Zweifachauswahl ist auch der bedingte Ausdruck (*conditional-expression*) zu zählen, es gilt die Syntax:

conditional-expression **::=** *expr1* **?** *expr2* **:** *expr3*

Zuerst wird *expr1* ausgewertet; falls das Resultat ungleich Null ist (TRUE)
dann wird *expr2* ausgewertet, und der Ausdruck als Ganzes hat das Resultat von *expr2*
sonst wird *expr3* ausgewertet, und der Ausdruck als Ganzes hat das Resultat von *expr3*.

Beispiel :

Der Code

```
x = (y < z) ? y : z;
```

ist äquivalent zu

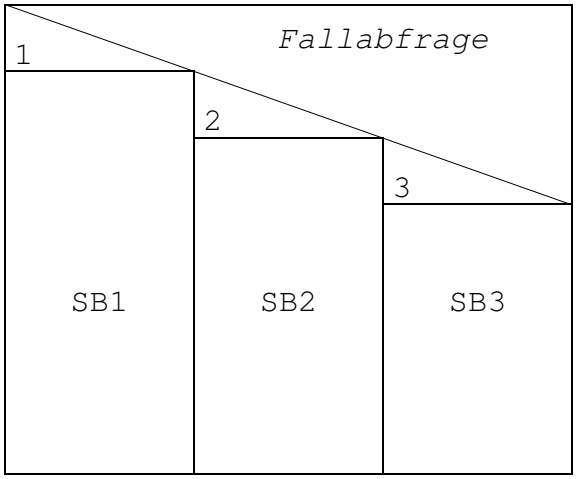
```
if (y < z)
  x = y;
else
  x = z;
```

Mehrfachauswahl oder Fallunterscheidung

Pseudocode

```
Unterscheide zwischen
  Fall 1:
    SB1
  Fall 2:
    SB2
  Fall 3
    SB3
```

Struktogramm



Es sind beliebig viele Fälle möglich. Das Struktogramm nach Nassi-Shneiderman ist nicht sehr günstig, weil wenig Platz für die Beschreibung der einzelnen Fälle ist. Als Auswahlvariable können Ganzzahl-Typen, Enumerationen und Zeichenketten verwendet werden. Die Realisierung in C# erfolgt mit der `switch`-Anweisung:

```
switch ( expression ) {
  case k1:
    statement
    break;
  case k2:
    statement
    break;
```

```
    case k3:
      statement
      break;
}
```

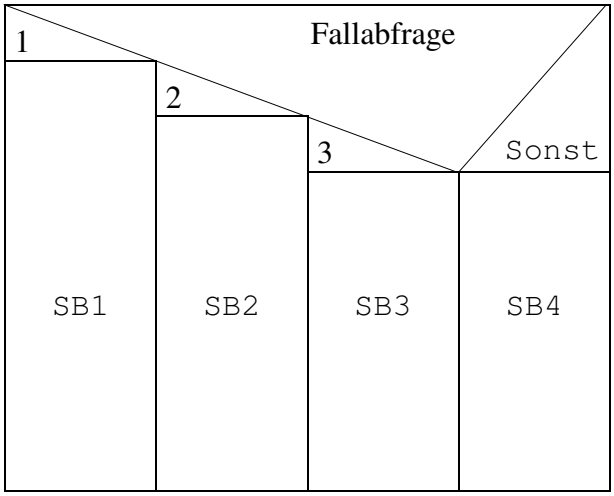
Für die Fallunterscheidung wird der Kontrollausdruck *expression* ausgewertet. Das Resultat dieses Ausdrucks muss ein Ganzzahltyp oder eine Zeichenkette sein. Anschließend wird der Programmablauf nach jenem `case` fortgesetzt, dessen Konstante mit dem Wert des Kontrollausdrucks übereinstimmt. Bei diesem Konstrukt sollte gewährleistet sein, daß wirklich einer der aufgezählten Fälle eintritt. Es gibt aber auch eine Variante der Fallunterscheidung, welche einen Strukturblock enthält, der immer dann ausgeführt wird, wenn keiner der explizit aufgezählten Fälle eintritt.

Mehrfachauswahl mit SONST-Fall:

Pseudocode

```
Unterscheide zwischen
  Fall 1:
    SB1
  Fall 2:
    SB2
  Fall 3
    SB3
  Sonst:
    SB4
```

Struktogramm



Der Sonstfall ist sehr oft als Fehlerfall zu bezeichnen. In C wird dieser Block mit dem Schlüsselwort `default` bezeichnet:

```
switch ( integral-expression ) {
  case k1:
    statement
    break;
  case k2:
    statement
    break;
  case k3:
    statement
    break;
  default:
    statement
    break;
}
```

Das *switch-statement* hat die allgemeine Form:

```
switch ( expression ) {
  case constant-expression :
    statement-list
    break;
  case constant-expression :
    statement-list
    break;
  .....
  default :
    statement-list
    break;
}
```

In den meisten Fällen wird man die Möglichkeit des `default`-Zweiges nutzen. Beachten Sie unbedingt, daß die Liste der Anweisungen mit einer `break`-Anweisung oder einer anderen Anweisung, welche die Sequenz beendet, abgeschlossen werden muss. Im Gegensatz zu C und C++ erzwingt das auch die Grammatik von C#.

Die Liste der Anweisungen kann auch leer sein. Diese Schreibweise benötigt man, wenn Anweisungen für mehrere Fälle gelten sollen:

```
Console.Write("J[a] oder N[ein] :");
input = Console.ReadLine;

switch (input[0])
{
  case 'j':
  case 'J':
    Console.Write("Die Antwort war Ja\n");
    break;
  case 'n': case 'N':
    Console.Write("Die Antwort war Nein\n");
    break;

}
```

Eine Aufzählung in der Form

```
  case 'j','J':
```

ist (leider) nicht möglich.

Zusammenfassung zur Ausführung einer `switch`-Anweisung:

1) Auswertung des `switch`-Ausdruckes (Ganzzahl-Typ oder Zeichenkette)
2) Sprung zu jenem `case`-Label, dessen konstanter Wert mit dem in Schritt 1 berechneten Wert übereinstimmt. Falls kein übereinstimmender Wert gefunden wird, wird zum `default`-Label verzweigt. Falls kein `default`-Label existiert, wird die `switch`-Anweisung beendet.

Die Fallanweisung wird oft verwendet, um die über ein Menü angebotenen Wahlmöglichkeiten zu implementieren oder auf die den verschiedenen Sondertasten zugeordneten Funktionen (z.B. innerhalb eines Editors) zu reagieren. Der ausgewählte Strukturblock ist dann meist der Aufruf einer Funktion.

Die Mehrfachauswahl kann auch mit `else if` - Folgen realisiert werden. Dann fällt auch die Einschränkung des Kontrollausdruckes auf einen Ganzzahltyp weg.

```
if ( expression )
   statement
else if ( expression )
   statement
else if ( expression )
   statement
else if ( expression )
   statement
else
   statement
```

Die einzelnen Ausdrücke werden in der angegebenen Reihenfolge bewertet. Sobald eine Bedingung erfüllt ist, wird die abhängige Anweisung ausgeführt; damit ist die Ausführung der gesamten Kette beendet.

Als Beispiel soll der Code für das binäre Suchen dienen. Dies ist ein sehr effizientes Verfahren, um in einer sortierten Liste ein bestimmtes Element zu suchen.

```
Solange die (Teil-)Liste noch ein Element hat
  und das gesuchte Element nicht gefunden ist wiederhole {
  Bestimme das mittlere Element der Liste
  Falls das gesuchte Element kleiner ist als das mittlere Element
    dann suche in der ersten Hälfte weiter
  sonst falls das gesuchte Element größer ist als das mittlere Element
    dann suche in der zweiten Hälfte weiter
  sonst ist das mittlere Element das gesuchte Element
}
```

Beispiel: in der folgenden Liste soll die Zahl 9 gesucht werden. Nach der Teilung der Liste in zwei Teile wird festgestellt, daß die Zahl 9 in der zweiten Hälfte der Liste zu suchen ist (Zahlen 6, 8, 9, 12). Diese Teilliste wird fortlaufend weiter geteilt, bis die Zahl gefunden wird oder festgestellt wird, daß die Zahl nicht enthalten ist.

0:	1	1	1	1
1:	2	2	2	2
2:	3	3	3	3
3:	**5**	5	5	5
4:	6	6	6	6
5:	8	**8**	8	8
6:	9	9	**9**	**9**
7:	12	12	12	12

```
public int BinarySearch(int item)
{
  // only for sorted lists
  int first, last, middle;   // item
  first = 0; last = n-1;
  while (first <= last)
  {
    middle = (first+last) / 2;
    if (item < list[middle])
      last = middle - 1;
    else if (item > list[middle])
      first = middle + 1;
    else  // found
      return middle;
  }
  return -1;
}
```

2.3.4 Wiederholung, Iteration

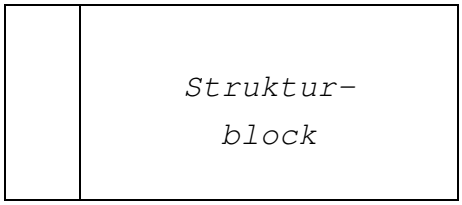
Wiederholung ist einer der wichtigsten Gründe, warum wir Computer verwenden. Computer arbeiten schnell und ohne Ermüdung. Aus didaktischen Gründen möchte ich mit der Endlosschleife beginnen.

Endlosschleife

Pseudocode

```
Wiederhole endlos
  Strukturblock
```

Struktogramm



Mit dem Balken links wird die Wiederholung des Strukturblocks SB zum Ausdruck gebracht. Die Realisierung in C erfolgt in der Form

```
while (true)
  statement
```

oder mit einer `for`-Schleife

```
for(;;)
  statement
```

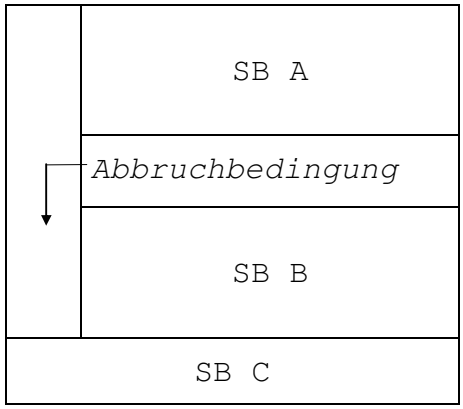
Absichtliche Endlosschleifen sind in Programmen eher selten. Im Normalfall will man die Schleife beim Eintreffen einer bestimmten Bedingung abbrechen. Dann ergibt sich folgende Situation:

Abbruchbedingungen an beliebiger Stelle

Pseudocode

```
Wiederhole
  SB A
 Abbrechen falls Bedingung
  SB B

Nächster Strukturblock
```

Struktogramm



C# unterstützt diese Form mit der `break`-Anweisung. Den äußeren Rahmen bildet eine Endlosschleife. Eine `break`-Anweisung wird an eine Abbruchbedingung gekoppelt und beendet die Wiederholung.

```
while (true) {
  statement  /* SB A */
 if ( expr ) break;
```

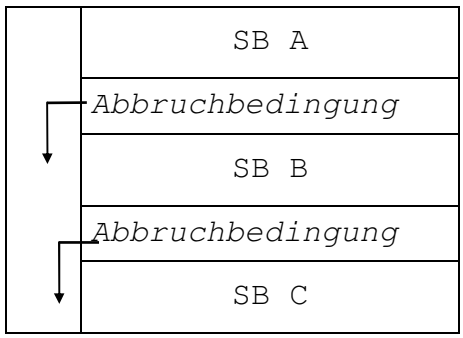
```
    statement   /* SB B */
  }
  statement     /* SB C */
```

Auch Wiederholungen mit mehreren Abbruchbedingungen sind möglich:

Pseudocode

```
Wiederhole
  SB A
 Abbrechen falls Bedingung
  SB B
 Abbrechen falls Bedingung
  SB C
```

Struktogramm



Die Codierung in C# ist:

```
while (true) {
    statement   /* SB A */
   if ( exp ) break;
    statement   /* SB B */
   if ( exp ) break;
    statement   /* SB C */
}
```

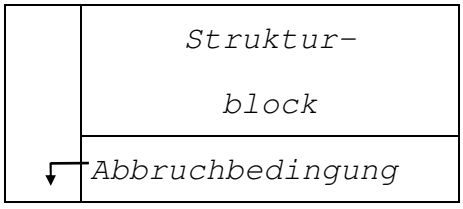
Die Abbruchbedingung kann aber auch erst am Ende der Schleife auftreten, womit wir wieder bei der bereits bekannten fußgesteuerten Schleife sind.

Fußgesteuerte Schleife mit Abbruchbedingung

Pseudocode

```
Wiederhole
  Strukturblock
bis Bedingung
```

Struktogramm



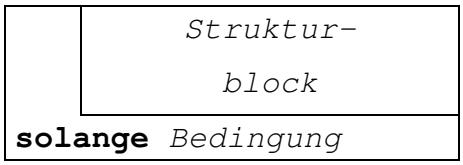
Diese Form kann in C# entweder wieder mit einer um ein bedingtes `break` erweiterten "Endlosschleife" realisiert werden, oder man ersetzt die Abbruchbedingung durch eine Ausführungsbedingung und erhält damit die

Fußgesteuerte Schleife mit Ausführungsbedingung

Pseudocode

```
Wiederhole
  Strukturblock
solange Bedingung
```

Struktogramm



Für diesen Typ gibt es in C eine genau passende Anweisung, das *do-statement*:

```
do-statement:
  do
    statement
  while ( boolean-expression );
```

Am Ende des Strukturblocks (des Schleifenrumpfs) wird geprüft, ob der Strukturblock wiederholt wird. Der Schleifenrumpf wird auf jeden Fall einmal ausgeführt, man spricht auch von einer nicht abweisenden Schleife.

Beispiel:

Häufig will man eine Eingabe solange einfordern, bis sie korrekt ist:

```
string errormsg = "Error: Zahl muss >= 0 sein!"
bool error;
do {
  Console.WriteLine("Positive ganze Zahl eingeben: ");
  n = int.Parse(Console.ReadLine());
  if (error = (n < 0))
     Console.WriteLine(errormsg);
} while (error);
```

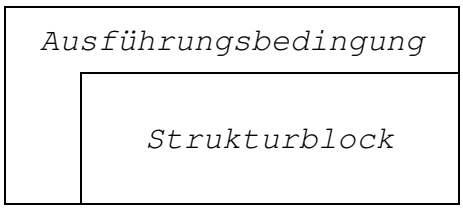
Verschiebt man die Kontrolle der Schleife an den Beginn der Schleife und formuliert dazu eine Ausführungsbedingung, so entsteht die schlußendlich wichtigste Form einer Schleife, die kopfgesteuerte Schleife mit Ausführungsbedingung.

Kopfgesteuerte Schleife

Pseudocode

```
Solange Bedingung wiederhole
Strukturblock
```

Struktogramm



Die Umsetzung in C# lautet:

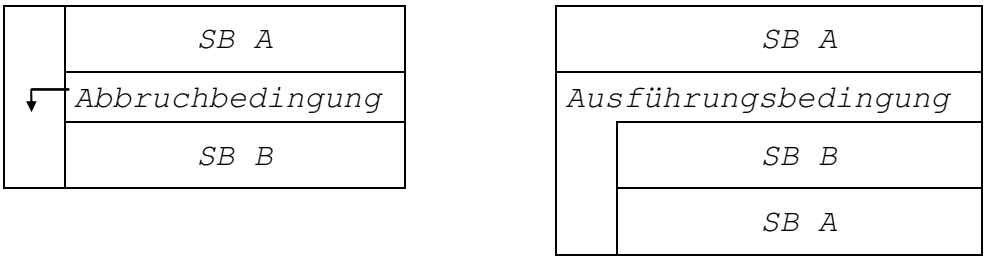
```
while-statement:
  while ( boolean-expression )
    statement
```

Die Steuerung einer Wiederholung zu Beginn des Blocks bietet die Möglichkeit, den Block unter Umständen auch nie auszuführen. Man nennt diese Form der Schleife deshalb auch die abweisende Schleife.

Beispiele zur `while`-Anweisung:

```
i = fact = 1 ;      // Berechnet n! (Fakultaet)
while (i++ < n)
   fact *= i;

int i = 1;
while (i <= n)
{
  Console.Write("{0} ", i);
  i++;
}
Console.WriteLine();
```

Die Kontrolle einer Wiederholung mit einer Abbruchbedingung an einer beliebigen Stelle kann auch durch eine kopfgesteuerte Schleife erreicht werden. Dazu ist folgende Umformung erforderlich:



Die Abbruchbedingung muß in eine Ausführungsbedingung umgewandelt werden. Der Strukturblock A taucht jetzt zweimal auf.

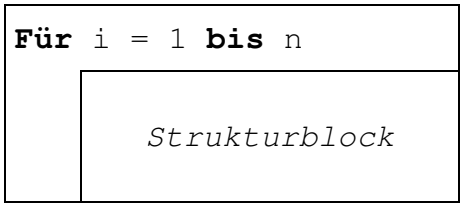
Zählschleife

Eine Sonderform der kopfgesteuerten Schleife ist die Zählschleife. Diese Schleife wird verwendet, um die Kontrollvariable, das ist jene Variable, mit welcher die Schleife kontrolliert wird, fortlaufende ganzzahlige Werte annehmen zu lassen.

Pseudocode

```
Für i = 1 bis n führe aus
  Strukturblock
```

Struktogramm



In C#:

```
for (i = 1; i <= n; i++)
  statement
```

`i = 1`	ist die Initialisierung der Kontrollvariablen i,
`i <= n`	ist die Ausführungsbedingung der Schleife,
`i++`	erhöht die Kontrollvariable am Ende des Schleifenrumpfs jeweils um 1

Der Anfangswert und Endwert kann beliebig gewählt werden, die Kontrollvariable kann mit `i--` auch dekrementiert werden. Aber auch eine beliebige Schrittweite kann z.B. mit dem Ausdruck `i += step` gewählt werden. In C ist die Verwendung der `for`-Schleife als klassische Zählschleife mit einem Ganzzahlwert als Kontrollvariable nur eine Möglichkeit dieser Anweisung. Da die Kontrollvariable ein beliebiger Datentyp sein kann, ist die `for`-Anweisung die in C wohl häufigste Anweisung zur Implementierung von Wiederholungen.

Der Programmteil:

```
for (expr1; expr2; expr3)
  statement
next-statement
```

ist (wenn der Rumpf keine `continue`-Anweisung enthält) äquivalent zu

```
expr1;
while ( expr2 ) {
  statement
  expr3;
}
statement
```

Syntax der `for`-Anweisung:

for-statement ::=
  `for` (*for-initializer*$_{opt}$; *for-condition*$_{opt}$; *for-iterator*$_{opt}$)
    statement

Die `for`-Anweisung hat alle zur Steuerung der Schleife notwendigen Anweisungen in der Kopfzeile zusammengefaßt und ist deshalb besonders gut lesbar. Der erste Ausdruck initialisiert die Kontrollvariable. Sie kann an dieser Stelle auch deklariert werden. Das passiert nur einmal, bevor die Schleife begonnen wird. Der dritte Ausdruck wird nach jeder Wiederholung berechnet, er verändert die Kontrollvariable. Der Typ des ersten und dritten Ausdrucks unterliegt keiner Einschränkung. Der zweite Ausdruck formuliert die Ausführungsbedingung der Schleife. Er wird vor jeder Wiederholung berechnet und muss entweder true oder false sein. Die einzelnen Ausdrücke sind optional. Fehlt die Ausführungsbedingung, so ist der Ersatzwert true.

Einige gleichwertige Möglichkeiten, die Summe aller Zahlen von 1 bis 10 ausrechnen:

```
i = 1;
sum = 0;
for (; i <= 10; i++)
  sum += i;
```

```
i = 1;
sum = 0;
for (; i <= 10; )
  sum += i++;
```

```
sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
```