Programmentwurf mit Struktogrammen

Aus verschiedenen Gründen ist es sinnvoll, den Quelltext eines Programms vorauszuplanen:

- um komplizierte Algorithmen zu begreifen,
- um ein umfangreiches Projekt in logische, abgegrenzte Teile zu zerlegen,
- um eine von einer konkreten Programmiersprache unabhängige Darstellung zu erreichen.

Seit den 1960er Jahren wurden verschiedene Methoden zur visuellen Beschreibung von Programmen entwickelt. Im folgenden werden Struktogramme, nach den Entwicklern auch Nassi-Shneiderman-Diagramme genannt, erläutert. Mit ihrer Hilfe kann die logische Struktur eines Programms klar dargestellt werden. Im Jahre 1985 wurde die äußere Form in DIN 66261 festgelegt.

1.1 Grundlagen

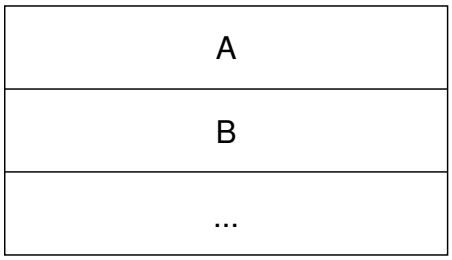
Die Grundbausteine von Struktogrammen sind die in den folgenden Abschnitten gezeigten *Strukturblöcke*. Sie werden jeweils durch ein Rechteck graphisch dargestellt und können durch *Schachtelung* und *Aufeinanderstapeln* kombiniert werden.

Der Programmablauf wird stets von oben nach unten gelesen.

1.1.1 Verarbeitung

Die Verarbeitung ist der einfachste Strukturblock. Er enthält eine oder mehrere Anweisungen in Text- oder Pseudocode-Form.

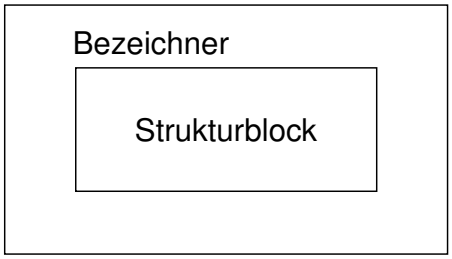
1.1.2 Folge



A;
B;
⋮

Die Folge symbolisiert die Hintereinanderausführung der in ihr auftretenden Strukturblöcke.

1.1.3 Block



{ Anweisung(en); }

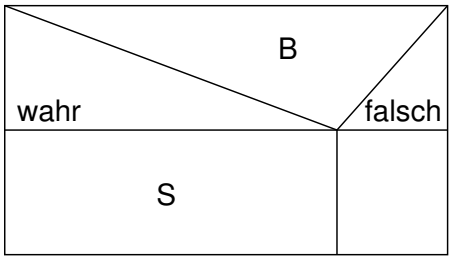
Ein Block stellt eine Hilfskonstruktion dar, mit der Strukturblöcke zusammengefasst werden können, um ihnen einen gemeinsamen Namen (Bezeichner) zu geben.

Man kann einen Block als Funktion im Sinne von C++ auffassen.

1.2 Verzweigung

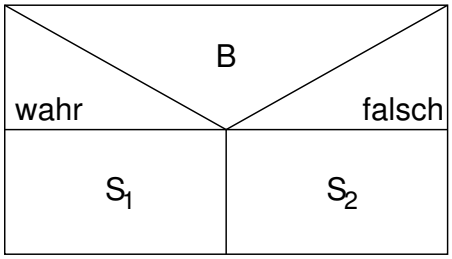
Mit B wird eine Bedingung (logischer Ausdruck) bezeichnet, S, S_1, S_2, ... sind Strukturblöcke.

1.2.1 Bedingte Verarbeitung (if)



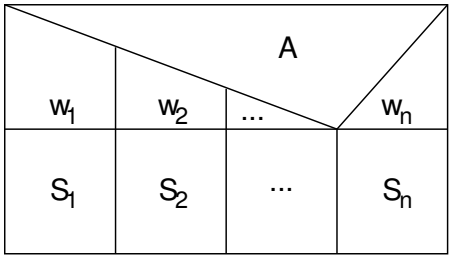

```
if (B) {
  // Anweisungen aus S
}
```

1.2.2 Einfache Alternative (if...else)




```
if (B) {
  // Anweisungen aus S1
}
else {
  // Anweisungen aus S2
}
```

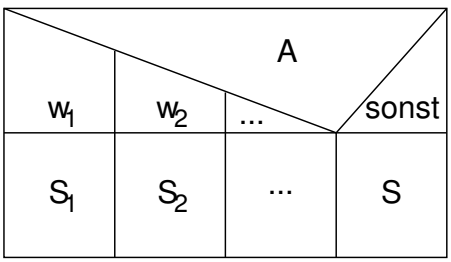
1.2.3 Mehrfache Alternative (switch–case)




```
switch (A) {
  case w1: S1; break;
  case w2: S2; break;
  ...
  case wn: Sn; break;
}
```

In den beiden Diagrammen steht A für einen Ausdruck, der stets einen der (ganzzahligen) Werte w_1, w_2, ... ergibt. Der unter dem entsprechenden w_i stehende Strukturblock S_i wird ausgeführt.

Es gibt folgende Variante, bei der der Strukturblock S unter „sonst“ aufgeführt wird, falls A nicht einen der w_i ergibt:

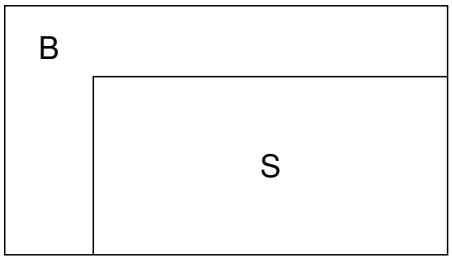



```
switch (A) {
  case w1: S1; break;
  case w2: S2; break;
  ...
  default: S; break;
}
```

1.3 Iteration (Schleife)

Wie im vorigen Abschnitt bezeichnet B einen logischen Ausdruck und S einen Strukturblock.

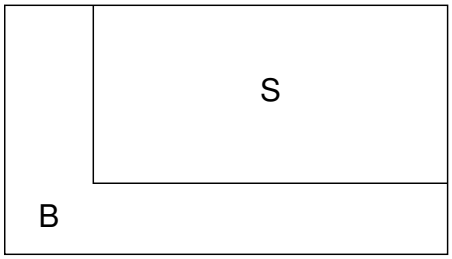
1.3.1 Schleife mit vorausgehender Bedingungsprüfung (while)




```
while (B) {
  // Anweisungen aus S
}
```

Die Abarbeitung beginnt mit einem Test der Bedingung. Ergibt dieser Test $false$, so wird der nächste Strukturblock verarbeitet. Liefert der Test $true$, so wird der Strukturblock S ausgeführt. Danach wird der vorliegende Strukturblock erneut ausgeführt. (Das heißt, B wird getestet, ...)

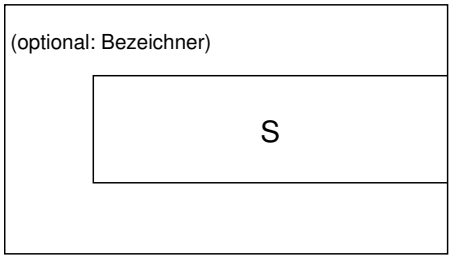
1.3.2 Schleife mit nachfolgender Bedingungsprüfung (do... while)




```
do {
  // Anweisungen aus S
} while (B);
```

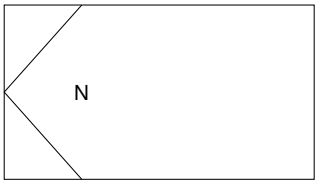
Zuerst wird S ausgeführt, danach wird B getestet. Ergibt dieser Test $false$, so wird der nächste Strukturblock verarbeitet. Liefert der Test $true$, so wird der vorliegende Strukturblock erneut ausgeführt. (Das heißt, S wird ausgeführt, ...)

1.3.3 Endlosschleife mit Abbruchanweisung (break, continue)




```
while (true) {
  // Anweisungen aus S
}
```

Der Strukturblock S wird stetig wiederholt. Damit diese Konstruktion Sinn ergibt, d. h. die Schleife irgendwann terminiert, muss in S eine *Abbruchanweisung* stehen, die in Struktogrammen so dargestellt wird:



Abbruchanweisung – N ist der Bezeichner des zu verlassenden, umgebenden Strukturblocks. Als nächstes wird der auf N folgende Strukturblock verarbeitet.

Achtung: In C++ kann (ohne größeren Aufwand) nur die gerade ausgeführte Schleife abgebrochen werden. Dies geschieht, wenn im Rumpf der Schleife das Schlüsselwort **break** verarbeitet wird.[1]

[1] C++ bietet außerdem die Möglichkeit, aus jeder Stelle eines Schleifenrumpfes direkt zur zugehörigen Iterationsbedingung zu springen. Dazu verwendet man das Schlüsselwort **continue**.

Sonstiges

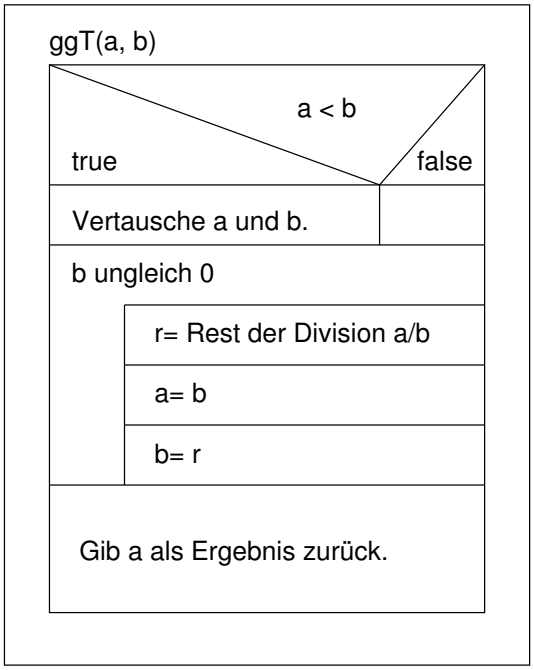
Prinzipiell ist es erlaubt, jedem Strukturblock einen Bezeichner zuzuweisen. Dieser muss zuoberst stehen. Wegen der schrägen Linien bei den Verzweigungs-Strukturblöcken empfiehlt es sich, diese mit einem Block (siehe Abschnitt 1.1.3) zu umgeben, um ihnen einen Namen zu geben.

Reicht die Fläche im Inneren eines Strukturblockes nicht aus, so kann durch entsprechende Beschriftung (Bezeichner...) auf ein weiteres Struktogramm verwiesen werden, welches an dieser Stelle auszuführen ist.

Beispiel

Der Euklidische Algorithmus dient zur Berechnung des größten gemeinsamen Teilers zweier positiver, ganzer Zahlen a, b. Er basiert darauf, die größere durch die kleinere der beiden Zahlen mit Rest zu teilen. Danach wiederholt man dies mit der kleineren Zahl und dem Divisionsrest, bis der Rest Null ist.

Der dann vorliegende Dividend ist der ggT von a und b.[2]



[2] Übung: Überlegen Sie sich, warum dieser Algorithmus für beliebige $a, b \in \mathbb{N}$ nach endlich vielen Schritten terminiert. Warum kommt als Ergebnis ein gemeinsamer Teiler von a und b heraus? Warum ist jeder andere gemeinsame Teiler von a und b auch ein Teiler dieses Ergebnisses?