Beweisregel (U1): $\{B\}S\{B\}$, falls keine in B vorkommende Variable von S verändert wird.

Beweisregel (U2): Falls S keine in B vorkommende Variable verändert, und $\{V'\}S\{P'\}$, dann gilt
$\{V' \text{ und } B\}S\{P' \text{ und } B\}$.

Beweisregel (U3): Falls ① $V \Rightarrow V'$ ② $\{V'\}S\{P'\}$
③ $P' \Rightarrow P$ ④ $\{B\}S\{B\}$
dann gilt, daß $\{V \text{ und } B\}S\{P \text{ und } B\}$.

Beweise und Konstruktionsaufgaben

Beweis / Aufgabe	$(\{V\}S\{P\})?$	$\{V?\}S\{P\}$	$\{V\}S?\{P\}$
x:=A	(Z2) $V^? \Rightarrow P_A^x$	(Z1) $V = P_A^x$	x?, A? derart, daß $V \Rightarrow P_A^x$
if B then S_1 else S_2 endif	(IF1) $\{V \text{ und } B\}S_1\{P\}$? $\{V \text{ und } \overline{B}\}S_2\{P\}$?	(IF2) $\{V_1^?\}S_1\{P\}$ $\{V_2^?\}S_2\{P\}$ $V = V_1 \wedge B \vee V_2 \wedge \overline{B}$	(IF1) B? S_1? S_2? derart, daß $\{V \text{ und } B\}S_1\{P\}$ und $\{V \text{ und } \overline{B}\}S_2\{P\}$
init; while B do S endwhile	(W2) I ? $\{V\}$init$\{I\}$? $\{I \text{ und } B\}S\{I\}$? I und $\overline{B} \Rightarrow P$? (evtl. Terminierung?)	(W2) (I ist bekannt) $\{V?\}$init$\{I\}$	(W2) I ? $\{V\}$init?$\{I\}$ I und $\overline{B}^? \Rightarrow P$ $\{I \text{ und } B\}S?\{I\}$
S_1; S_2	(F1) $\{V_2?\}S_2\{P\}$ $\{V\}S_1\{V_2\}$?	(F1) $\{V_2?\}S_2\{P\}$ $\{V_1?\}S_1\{V_2\}$ $V := V_1$	(F1) V_2? $\{V\}S_1?\{V_2\}$ $\{V_2\}S_2?\{P\}$
$S_1\{V_2\}S_2$ (wie eben m. gegebener Zwischenbedingung)	(F1) $\{V_2\}S_2\{P\}$? $\{V\}S_1\{V_2\}$?	(F1) $\{V_1?\}S_1\{V_2\}$ $\{V_2\}S_2\{P\}$? $V := V_1$	(F1) $\{V\}S_1?\{V_2\}$ $\{V_2\}S_2?\{P\}$

Prozeduraufrufe

Bei einem Prozeduraufruf der folgenden Form

```
(call) proc (a);
..
..
proc (f: integer)
   S;
end proc;
```

stellt sich immer die Frage der Parameterübergabe. Wird der formale Parameter by-value oder by-reference übergeben? Dies kann nur durch das Handbuch des vorliegenden Zielsystems geklärt werden, wie z.B. das Pascal User Manual (N. Wirth), oder ein "Report of the Algorithmic language Algol 60" (1962).

Übersetzung der verschiedenen Schemata für obige Programmzeilen in "unsere" Sprache:

	Pascal	"unsere" Sprache
call-by-value	(call) proc (a);	declare (f, integer, a); S; release f;
call-by-reference (call-by-name)	(call) proc (a);	$(S')_a^f$ wobei S' eine namenskonflikt-freie Version von S ist

Dabei sollte das Wort "namenskonfliktfrei" vielleicht noch ein wenig erläutert werden. Bei call-by-name wird in "unserer" Sprache anstatt des Funktionsaufrufes der gesamte Programmtext der Unterfunktion eingesetzt. Bei blinder Einsetzung würden sich jedoch die im Unterprogramm deklarierten lokalen Variablen und die Variablen der aufrufenden Prozedur überlagern. "Namenskonfliktfrei" bedeutet hier, daß die Variablen von S' so gewählt werden, daß keine Konflikte auftreten. Vergleiche hierzu auch den Algol60 Report 4.7.3.2.

Nachschlag

Zum Nachschlagen und tief einprägen:

- Übung 1 bzw. Übung 9
- Übersicht der Beweisregeln
- Übung 6, Aufgabe 0.14 + Lösung
- Korrektheitsbeweise der Unterprogramme

Beispiel: Mischen zweier Listen

Gegeben sind zwei sortierte Listen und eine Zielliste. Nun sollen die Elemente beider Listen so in die Zielliste eingefügt werden, daß die Zielliste hinterher ebenfalls sortiert ist. Wir definieren die Vor- und Nachbedingungen:

V: $na, nb \in Z \wedge na \geq 0 \wedge nb \geq 0 \wedge \left(\bigwedge_{i=1}^{na-1} A(i) \leq A(i+1)\right) \wedge \left(\bigwedge_{i=1}^{nb-1} B(i) \leq B(i+1)\right)$

P: $\&_{i=1}^{na+nb}[c(i)]$ ist Permutation von $\&_{i=1}^{na}[A(i)] \& _{i=1}^{nb}[B(i)]$ und
$\bigwedge_{i=1}^{na+nb-1} c(i) \leq c(i+1)$

```
declare (ia, Z, 1), (ib, Z, 1), (ic, Z, 1)
while ia≤na oder ib≤nb do
  if ib>nb oder (ia≤na und A(ia)≤B(ib)) then
    c(ic) := A(ia);  ia := ia + 1;
  else
    c(ic) := B(ib);  ib := ib + 1;
  endif
  ic := ic + 1;
endwhile
release ia; release ib; release ic;
```

Aussagen: ① $\{V\}\text{Prog}\{P\}$

② $\text{Prog}.d \overset{strukturell}{=} d$ für alle $d \in \text{Prog}^{-1}.D$

③ Prog verändert nur die Werte der Feldvariablen c(i), i=1...na+nb

Konstruktion eines Unterprogramms

Dieser Abschnitt soll an einem Beispiel zeigen, daß und wie es möglich ist, aus einer gegebenen Vor- und Nachbedingung ein Unterprogramm zu konstruieren. Die Aufgabe ist dem Übungsblatt Nr. 24 entnommen. Es soll ein Unterprogramm erstellt werden, das die in zwei sortierten Feldern $\{D(id)|1 \le id \le nd\}$ und $\{E(ie)|0 \le ie \le ne\}$ gespeicherten Werte zusammenfügt und in das Feld $\{F(if)|1 \le if \le nd+ne\}$ sortiert überträgt. Das Feld D ist **ab**steigend, das Feld E **auf**steigend sortiert. Das resultierende Feld F soll **ab**steigend sortiert sein. Das Unterprogramm soll nur die Werte der Feldvariablen $\{F(if)|1 \le if \le nd+ne\}$ verändert, d.h. $S.d \overset{strukt}{=} d$.

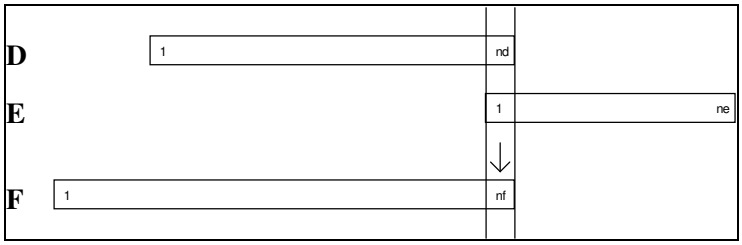
Der Systemkonstrukteur hat die folgenden Vor- und Nachbedingungen vorgegeben:

$$V = \begin{cases} nd, ne \in Z \wedge 0 \le nd, ne \wedge \\ \text{und}_{i=1}^{nd-1} D(i) \ge D(i+1) \wedge \text{und}_{i=1}^{ne-1} E(i) \le E(i+1) \\ \wedge \{D(1)\ldots D(nd), E(1)\ldots E(ne)\} \text{linear geordnet} \\ \wedge \text{und}_{i=1}^{nd+ne} \{D(1)\ldots D(nd), E(1)\ldots E(ne)\} \subseteq \text{Menge."}\, F(i)\text{"} \end{cases}$$

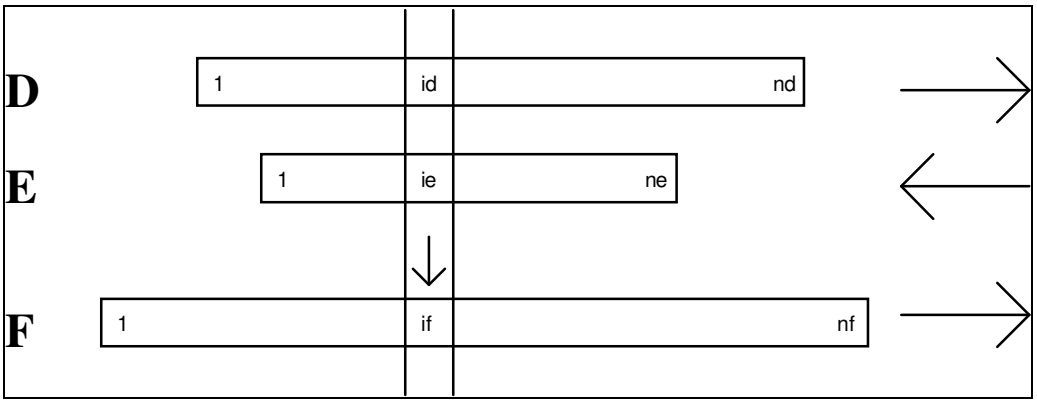
$$P = \begin{cases} [F(1)\ldots F(nd+ne)]\,\text{Perm}[D(1)\ldots D(nd), E(1)\ldots E(ne)] \\ \wedge \text{und}_{i=1}^{nd+ne-1} F(i) \ge F(i+1) \end{cases}$$

Bestimmung der Schleifeninvarianten

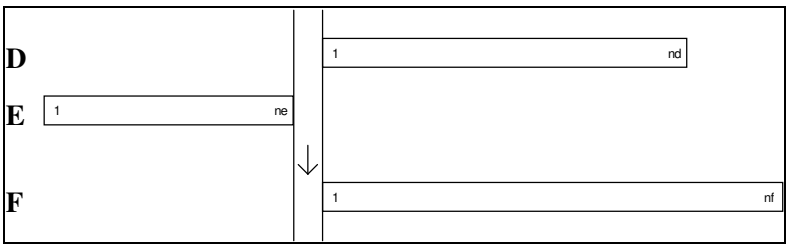
Zunächst wird die Schleifeninvariante festgelegt. Die Bedingungen der Schleifeninvariante überlegt man sich am besten grafisch:



Vor der Schleife



Innerhalb der Schleife



Nach Ende der Schleife

Die drei Felder werden so verschoben, daß ihre Felder am "Kopierpunkt" aufsteigend vorbeilaufen. Die Schleifeninvariante soll nun die Zustände vor Anfang und nach Ende des Schleifendurchlaufs zusammenfassen und all unsere Kenntnis der Zustände zu einem beliebigen Zeitpunkt innerhalb der Schleife beschreiben.

Wir wissen (zusätzlich zu den Bedingungen der Vorbedingung)

- Die Indizes id, ie und if zeigen auf Felder innerhalb des Arrays; am Ende der Schleife zeigen sie genau auf das erste Feld außerhalb.
- Das zuletzt nach F kopierte Feldelement ist kleiner als das nicht kopierte Feldelement von D bzw. E
- Der Feldindex in F läßt sich über die Indizes in E und D genau bestimmen
- Die beschriebenen Teile von F sind absteigend sortiert.
- Die beschriebenen Teile von F sind eine Permutation der gelesenen Teile der Felder D und E

Insgesamt folgt für die Schleifeninvariante:

$$I = V \wedge \begin{cases} 0 \le id \le nd \wedge 1 \le ie \le ne+1 \wedge 0 \le if \le nd+ne \wedge \\ \big((ie = ne+1 \vee F(if+1) \le E(ie)) \vee (nd = 0 \vee F(if+1) \le D(id))\big) \\ \wedge if = ne + id - ie + 1 \wedge \big(\wedge_{i=1}^{nd+ne-if-1} F(if+i) \ge F(if+i+1)\big) \wedge \\ \big[F(if+1),\ldots F(nd+ne)\big]\mathrm{Perm}\big[D(id+1),\ldots D(nd), E(1),\ldots E(ie-1)\big] \end{cases}$$

Die Initialisierung der Schleife wird so gewählt, daß der vorhin grafisch dargestellte Zustand erreicht wird:

```
declare (id, Z, nd), (ie, Z, 1), (if, Z, nd+ne);
```

Schleifenbedingung

Die Schleifenbedingung wird so gewählt, daß die Schleife erst terminiert, wenn der ebenfalls vorhin grafisch dargestellte Zustand erreicht ist, d.h.

$\overline{B} = id = 0 \wedge ie = ne + 1$, d.h. $B = id > 0 \vee ie \leq ne$

Bei der Negation der Bedingung wurde bereits die Schleifeninvariante berücksichtigt, obwohl das streng genommen nicht nötig wär.

Schleifenkern

Der Schleifenkern soll uns nun dem Ergebnis, dem sortierten Feld *F*, näher bringen. Deshalb muß im Schleifenkern die Variable *if* um 1 erniedrigt werden. Das zieht folgendes nach sich:

① Damit die Bedingung $if = ne + id - ie + 1$ immer noch gilt, muß entweder id erniedrigt oder ie erhöht werden.

② Der beschriebene Teil von F muß eine Permutation der gelesenen Teile von D und E bleiben, d.h. wenn id erniedrigt wurde, muß ein Element von D nach F kopiert werden; wenn ie erhöht wurden, muß ein Element von E nach F kopiert werden.

③ Falls ein Element von E kopiert wurde, d.h. $F(if+1) = E(ie-1)$, so muß $ie = ne + 1 \vee F(if+1) \leq E(ie)$ nicht unbedingt erfüllt sein. Analog gilt, falls ein Element von D kopiert wurde, d.h. $F(if+1) = D(id+1)$, so muß $id = 0 \vee F(if+1) \leq D(id)$ nicht unbedingt erfüllt sein. Wenn wir jeweils das kleinere Element von $D(id+1)$ und $E(ie-1)$ kopieren, so wird eine dieser Bedingungen auf jeden Fall wahr. Eine Ausnahme ist, wenn nur noch Elemente in einem Feld zur Verfügung stehen, d.h. $id = 0 \vee ie = ne + 1$. Wir schließen, daß, wenn beide Felder noch nicht "am Ende" sind, das kleinere Element kopiert werden muß. Das reicht aus, um Zeile 2 der Schleifeninvariante zu erfüllen.

④ Der vorige Punkt stellt sicher, daß das Feld F absteigend sortiert sein wird.

Zusammenfassung:

- Wenn die Felder noch nicht "am Ende" sind, so wird das kleinere Element von D bzw. E nach F kopiert.
- Der entsprechende Feldindex in D bzw. E wird erniedrigt bzw. erhöht.
- Anschließend wird der Feldindex in F erniedrigt.

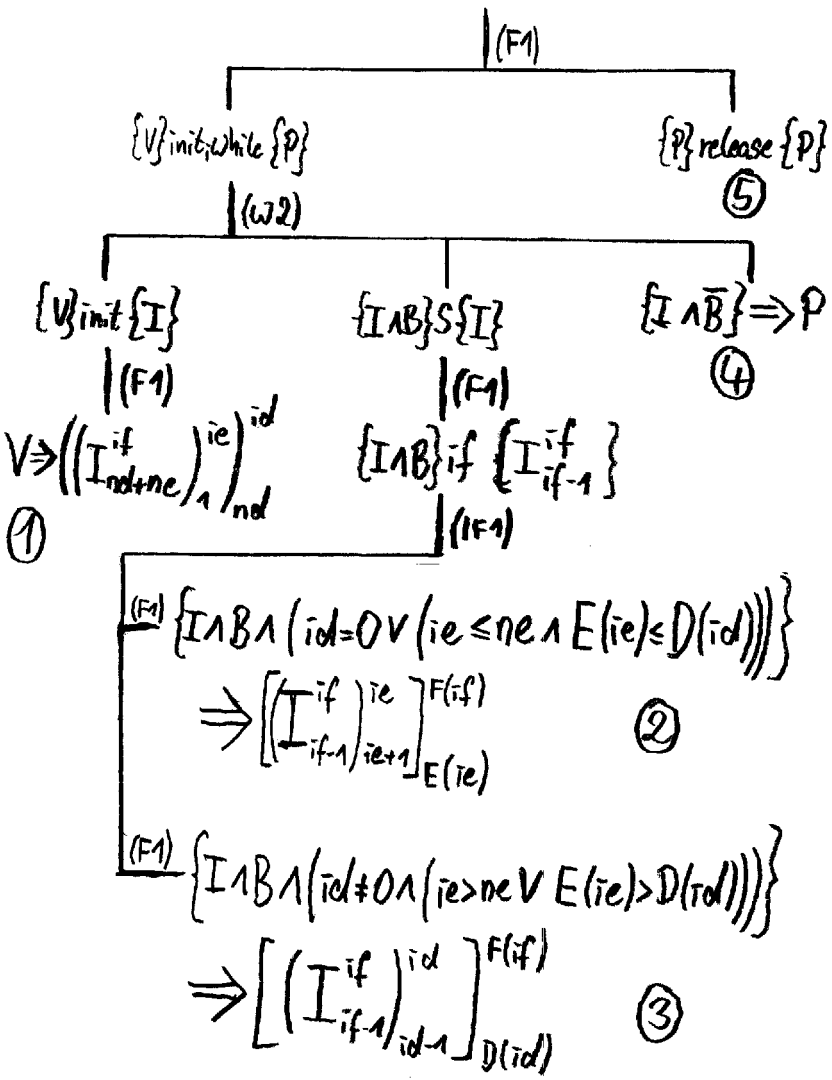
Beendigung der Schleife

Nach Beendigung der Schleife können die Einzelindizes verworfen werden.

Insgesamt

Insgesamt ergibt sich das folgende Unterprogramm:

```
declare (id, Z, nd), (ie, Z, 1), (if, nd+ne, 1);
while id>0 oder ie≤ne do
  if id=0 oder (ie≤ne und E(ie)≤D(id)) then
        F(if) := E(ie);  ie := ie+1;
  else
        F(if) := D(id);  id := id-1;
  end if;
  if := if - 1;
end while;
release id, ie, if;
```

Zwischenbedingungen

In der folgenden Grafik sind alle *zu beweisenden* Zwischenbedingungen aufgeführt. Triviale Implikationen, die durch Zuweisungen entstehen, sind der Übersichtlichkeit halber nicht einzeln aufgeführt.