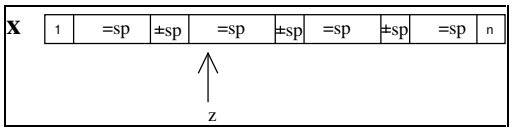
25. Übung zur
Konstruktion fehlerfreier Software

Frank Pilhofer

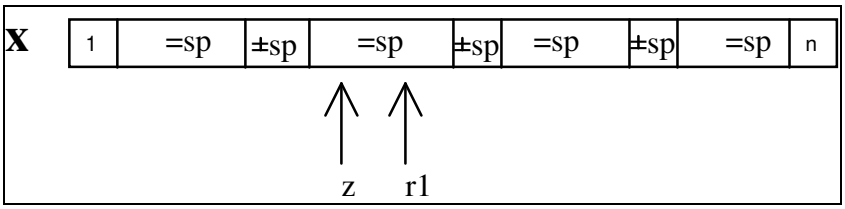
Montag 13-14 Uhr

Konstruktion eines Unterprogramms

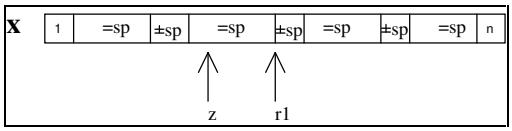
Zunächst wird die Schleifeninvariante festgelegt. Die Bedingungen der Schleifeninvariante überlegt man sich am besten grafisch:



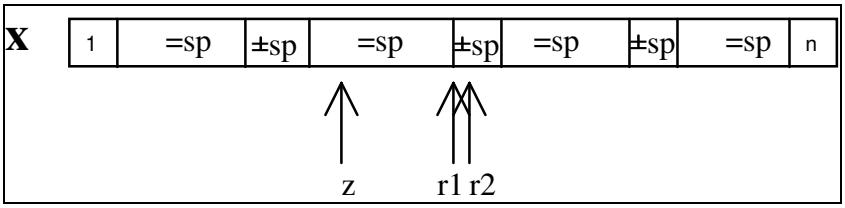
Vor den Schleifen



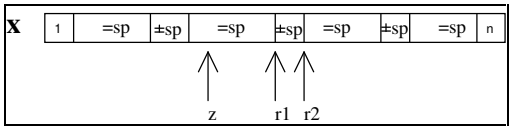
Während der 1. Schleife



Nach der 1. Schleife -- Vor der 2. Schleife



Während der 2. Schleife



Nach der 2. Schleife

Vor der ersten Schleife wird r1 mit z initialisiert und in der Schleife so lange vergrößert, bis r1>n oder x(r1)≠sp ist. Dann wird r2 mit r1 initialisiert und so lange vergrößert, bis r2>n oder x(r2)=sp ist.

Die erste Schleifeninvariante ist daher $I_1 = V \wedge 1 \le z \le r1 \le n+1 \wedge \text{und}_{i=z}^{r1-1}\, x(i) = sp$. Nach Terminierung der ersten Schleife, also wenn r1=n+1 oder x(r1)≠sp ist, muß folgende Postbedingung gelten: $P_1 = V \wedge 1 \le z \le r1 \le n+1 \wedge \text{und}_{i=z}^{r1-1}\, x(i) = sp \wedge \left(r1 = n+1 \vee x(r1) \ne sp\right)$. Dies ist gleichzeitig die Vorbedingung der zweiten Schleife, also $V_2 = P_1$. Die zweite

Schleifeninvariante ist $I_2 = P_1 \wedge 1 \le z \le r1 \le r2 \le n+1 \wedge \text{und}_{i=r1}^{r2-1} x(i) \neq sp$. Nach Ende beider Schleifen gilt die angegebene Postbedingung.

Schleifenbedingung

Die erste Schleife terminiert, wenn entweder r1 über das Feld hinausgeschossen ist (r1>n), oder wenn x(r1)≠sp ist. Daher ist $\overline{B_1} = r1 > n \vee x(r1) \neq sp$ und $B_1 = r1 \le n \wedge x(r1) = sp$. Die zweite Schleife terminiert, wenn entweder r2 über das Feld hinausgelaufen ist (r2>n) oder wenn x(r2)=sp. Daher ist $\overline{B_2} = r2 > n \vee x(r2) = sp$ und $B_2 = r2 \le n \wedge x(r2) \neq sp$.

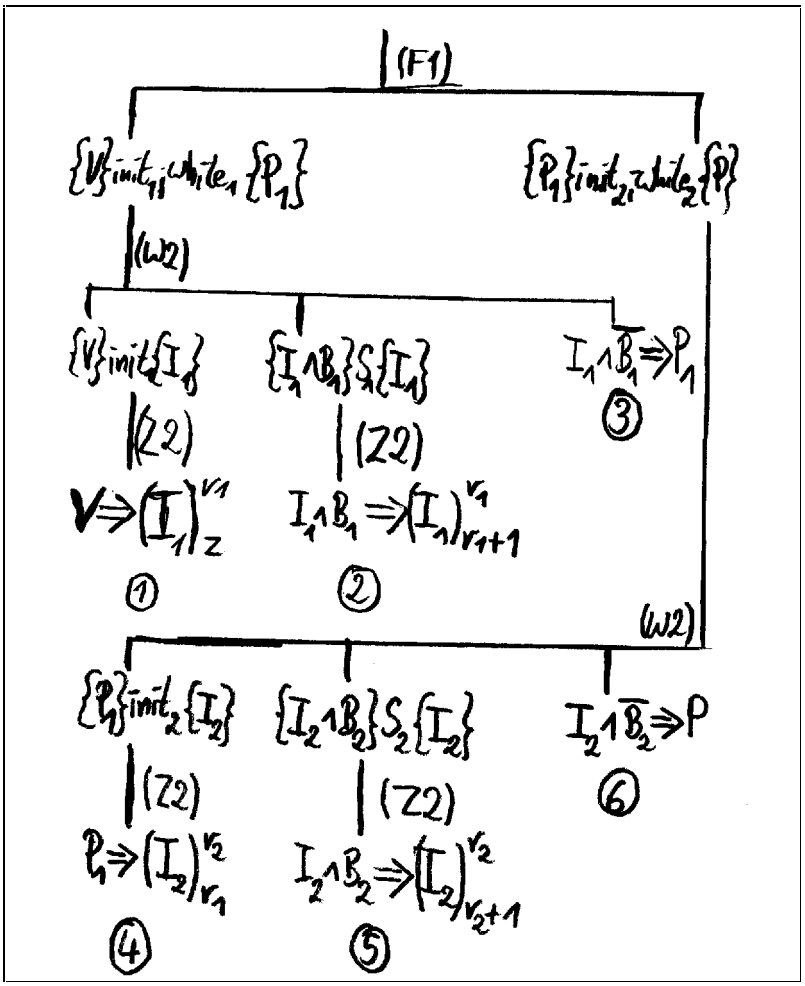
Schleifenkern

Dem Schleifenkern bleibt nichts anderes mehr zu tun, als jeweis die eine Variable hochzuzählen.

Insgesamt

```
declare (r1, Z, z);
while r1≤n und x(r1)=sp do
  r1 := r1 + 1;
endwhile;
declare (r2, Z, r1);
while r2≤n und x(r2)≠sp do
  r2 := r2 + 1;
endwhile;
```

Einzelimplikationen



Korrektheitsbeweis

① $(I_1)_z^{r1}$ = $V \wedge 1 \le z \le z \le n+1 \wedge \text{und}_{i=z}^{z-1}\, x(i) = sp$

= V

② $I_1 B_1$ = $V \wedge 1 \le z \le r1 \le n+1 \wedge \text{und}_{i=z}^{r1-1}\, x(i) = sp \wedge r1 \le n \wedge x(r1) = sp$

= $V \wedge 1 \le z \le r1 \le n \wedge \text{und}_{i=z}^{r1}\, x(i) = sp$

$(I_1)_{r1+1}^{r1}$ = $V \wedge 1 \le z \wedge z \le r1+1 \wedge r1+1 \le n+1 \wedge \text{und}_{i=z}^{r1}\, x(i) = sp$

$\Leftarrow$ $V \wedge 1 \le z \wedge z \le r1 \wedge r1 \le n \wedge \text{und}_{i=z}^{r1}\, x(i) = sp$

= $V \wedge 1 \le z \le r1 \le n \wedge \text{und}_{i=z}^{r1}\, x(i) = sp$ (wie oben)

also

$I_1 B_1$ $\Rightarrow$ $(I_1)_{r1+1}^{r1}$

③ $I_1 \overline{B}_1$ = $V \wedge 1 \le z \le r1 \le n+1 \wedge \text{und}_{i=z}^{r1-1}\, x(i) = sp \wedge (r1 > n \vee x(r1) \ne sp)$

= $V \wedge 1 \le z \le r1 \le n+1 \wedge \text{und}_{i=z}^{r1-1}\, x(i) = sp \wedge (r1 = n+1 \vee x(r1) \ne sp)$

= P_1

④ $(I_2)_{r1}^{r2}$ = $P_1 \wedge 1 \le z \le r1 \le r1 \le n+1 \wedge \text{und}_{i=r1}^{r1-1}\, x(i) \ne sp$

= P_1

⑤ $I_2 B_2 = P_1 \wedge 1 \le z \le r1 \le r2 \le n+1 \wedge \text{und}_{i=r1}^{r2-1}\, x(i) \ne sp \wedge r2 \le n \wedge x(r2) \ne sp$

$= P_1 \wedge 1 \le z \le r1 \le r2 \le n \wedge \text{und}_{i=r1}^{r2}\, x(i) \ne sp$

$(I_2)_{r2+1}^{r2} = P_1 \wedge 1 \le z \wedge z \le r1 \wedge r1 \le r2+1 \wedge r2+1 \le n+1 \wedge \text{und}_{i=r1}^{r2}\, x(i) \ne sp$

$\Leftarrow P_1 \wedge 1 \le z \wedge z \le r1 \wedge r1 \le r2 \wedge r2 \le n \wedge \text{und}_{i=r1}^{r2}\, x(i) \ne sp$

$= P_1 \wedge 1 \le z \le r1 \le r2 \le n \wedge \text{und}_{i=r1}^{r2}\, x(i) \ne sp$ (wie oben)

also

$I_2 B_2 \Rightarrow (I_2)_{r2+1}^{r2}$

⑥ $I_2 \overline{B}_2 = P_1 \wedge 1 \le z \le r1 \le r2 \le n+1 \wedge \text{und}_{i=r1}^{r2-1}\, x(i) \ne sp \wedge (r2 > n \vee x(r2) = sp)$

$= P_1 \wedge 1 \le z \le r1 \le r2 \le n+1 \wedge \text{und}_{i=r1}^{r2-1}\, x(i) \ne sp \wedge (r2 = n+1 \vee x(r2) = sp)$

$= P$

Was macht das Programm

Das Programm sucht Token in einem String, die durch Trennzeichen getrennt sind. Von z ausgehend wird mit r1 der Anfang des nächsten Tokens gesucht, anschließend sucht r2 nach dem Ende und positioniert sich auf das erste Zeichen nach dem Token.