

Korrekte Software: Grundlagen und Methoden
Vorlesung 8 vom 07.06.2022
Strukturierte Datentypen: Strukturen und Felder

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2022

Fahrplan

- ▶ Einführung
- ▶ Operationale Semantik
- ▶ Denotationale Semantik
- ▶ Äquivalenz der Operationalen und Denotationalen Semantik
- ▶ Der Floyd-Hoare-Kalkül I
- ▶ Der Floyd-Hoare-Kalkül II: Invarianten
- ▶ Korrektheit des Floyd-Hoare-Kalküls
- ▶ **Strukturierte Datentypen**
- ▶ Verifikationsbedingungen
- ▶ Vorwärts mit Floyd und Hoare
- ▶ Funktionen und Prozeduren I
- ▶ Funktionen und Prozeduren II
- ▶ Referenzen und Speichermodelle
- ▶ Ausblick und Rückblick

Motivation

- ▶ Immer nur ganze Zahlen ist doch etwas langweilig.
- ▶ Weitere Basisdatentypen von C (Felder, Zeichenketten, Strukturen)
- ▶ Noch rein funktional, keine Referenzen
- ▶ Nicht behandelt, aber nur syntaktischer Zucker: **enum**
- ▶ Prinzipiell: keine **union**

Arrays

► Beispiele:

```
int six[6] = {1,2,3,4,5,6};  
int a[3][2];  
int b[][] = { {1, 0},  
              {3, 7},  
              {5, 8} }; /* Ergibt Array [3][2] */
```

► `b[2][1]` liefert 8, `b[1][0]` liefert 3

► Index startet mit 0, *row-major order*

► In C0: Felder als echte Objekte (in C: Felder $\cong$ Zeiger)

► Allgemeine Form:

```
typ name[groesse1][groesse2]...[groesseN] =  
    { ... }
```

► Alle Felder haben **fixe Größe**.

Zeichenketten

- ▶ Zeichenketten sind in C (und C0) Felder von **char**, die mit einer Null abgeschlossen werden.

- ▶ Beispiel:

```
char hallo[6] = { 'h', 'a', 'l', 'l', 'o', '\0' }
```

- ▶ Nützlicher syntaktischer Zucker:

```
char hallo[] = "hallo";
```

- ▶ Auswertung: `hallo[4]` liefert `o`

Strukturen

- ▶ Strukturen haben einen *structure tag* (optional) und Felder:

```
struct Vorlesung {  
    char dozenten[2][30];  
    char titel[30];  
    int cp;  
} ksgm;
```

```
struct Vorlesung pi3;
```

- ▶ Zugriff auf Felder über Selektoren:

```
int i = 0;  
char name1[] = "Serge Autexier";  
while (i < strlen(name1)) {  
    ksgm.dozenten[0][i] = name1[i];  
    i = i + 1;  
}
```

- ▶ Rekursive Strukturen nur über Zeiger erlaubt (kommt noch)

C0: Erweiterte Ausdrücke

- ▶ **Lexp** beschreibt L-Werte (l-values), abstrakte Speicheradressen
- ▶ Neuer Basisdatentyp **C** für Zeichen
- ▶ Erweiterte Grammatik:

Lexp $/ ::= \text{Idt} \mid /[a] \mid /. \text{Idt}$

Aexp $a ::= \mathbf{Z} \mid \mathbf{C} \mid \text{Lexp} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 / a_2$

Bexp $b ::= \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 < a_2 \mid ! b \mid b_1 \&\& b_2 \mid b_1 || b_2$

Exp $e ::= \text{Aexp} \mid \text{Bexp}$

Werte und Zustände

- ▶ Zustände bilden **strukturierte** Adressen auf Werte (wie vorher) ab.

Systemzustände

- ▶ **Locations:** $\text{Loc} ::= \text{Idt} \mid \text{Loc}[\mathbb{Z}] \mid \text{Loc}.\text{Idt}$
 - ▶ Werte: $\mathbf{V} = \mathbb{Z} \uplus \mathbf{C}$
 - ▶ Zustände: $\Sigma \stackrel{\text{def}}{=} \mathbf{Loc} \rightarrow \mathbf{V}$
-
- ▶ Wir betrachten nur Zugriffe vom Typ $\mathbf{Z}$ oder $\mathbf{C}$ (**elementare Typen**)
 - ▶ Nützliche Abstraktion des tatsächlichen C-Speichermodells

Beispiel

Programm

```
struct A {  
    int c[2];  
    struct B {  
        char name[20];  
    } b;  
};  
  
struct A x[] = {  
    {{1,2},  
     {{ 'n', 'a', 'm', 'e', '1', '\0' }}  
    },  
    {{3,4},  
     {{ 'n', 'a', 'm', 'e', '2', '\0' }}  
    }  
};
```

Zustand

$x[0].c[0] \mapsto 1$	$x[1].c[0] \mapsto 3$
$x[0].c[1] \mapsto 2$	$x[1].c[1] \mapsto 4$
$x[0].b.name[0] \mapsto 'n'$	$x[1].b.name[0] \mapsto 'n'$
$x[0].b.name[1] \mapsto 'a'$	$x[1].b.name[1] \mapsto 'a'$
$x[0].b.name[2] \mapsto 'm'$	$x[1].b.name[2] \mapsto 'm'$
$x[0].b.name[3] \mapsto 'e'$	$x[1].b.name[3] \mapsto 'e'$
$x[0].b.name[4] \mapsto '1'$	$x[1].b.name[4] \mapsto '2'$
$x[0].b.name[5] \mapsto '\0'$	$x[1].b.name[5] \mapsto '\0'$

Operationale Semantik: L-Werte

► **Lexp** m wertet zu **Loc** l aus: $\langle m, \sigma \rangle \rightarrow_{Lexp} l$

$$\frac{x \in \mathbf{Idt}}{\langle x, \sigma \rangle \rightarrow_{Lexp} x}$$

$$\frac{\langle m, \sigma \rangle \rightarrow_{Lexp} l \quad \langle a, \sigma \rangle \rightarrow_{Aexp} i}{\langle m[a], \sigma \rangle \rightarrow_{Lexp} l[i]} \quad \frac{\langle m, \sigma \rangle \rightarrow_{Lexp} l}{\langle m.i, \sigma \rangle \rightarrow_{Lexp} l.i}$$

Operationale Semantik: Ausdrücke

- ▶ Ein L-Wert als Ausdruck wird ausgewertet, indem er ausgelesen wird:

$$\frac{\langle m, \sigma \rangle \rightarrow_{Lexp} l \quad l \in Dom(\sigma)}{\langle m, \sigma \rangle \rightarrow_{Aexp} \sigma(l)}$$

- ▶ Auswertung für **C**:

$$\overline{\langle c :: \mathbf{C}, \sigma \rangle \rightarrow_{Aexp} Ord(c)}$$

wobei $Ord : \mathbf{C} \rightarrow \mathbf{Z}$ eine bijektive Funktion ist, die jedem Character eine Ordinalzahl zuweist (zum Beispiel ASCII Wert).

Operationale Semantik: Zuweisungen

- Zuweisungen sind nur definiert für elementare Typen:

$$\frac{\langle m :: \tau_1, \sigma \rangle \rightarrow_{Lexp} l \quad \langle e :: \tau_2, \sigma \rangle \rightarrow v \quad \tau_1 = \tau_2 \text{ elementarer Typ}}{\langle m = e, \sigma \rangle \rightarrow_{Stmt} \sigma[l \mapsto v]}$$

- Die restlichen Regeln bleiben

Denotationale Semantik

► Denotation für **Lexp**:

$$\llbracket - \rrbracket_{\mathcal{L}} : \mathbf{Lexp} \rightarrow (\Sigma \rightarrow \mathbf{Loc})$$

$$\llbracket x \rrbracket_{\mathcal{L}} = \{(\sigma, x) \mid \sigma \in \Sigma\}$$

$$\llbracket m[a] \rrbracket_{\mathcal{L}} = \{(\sigma, l[i]) \mid (\sigma, l) \in \llbracket m \rrbracket_{\mathcal{L}}, (\sigma, i) \in \llbracket a \rrbracket_{\mathcal{A}}\}$$

$$\llbracket m.i \rrbracket_{\mathcal{L}} = \{(\sigma, l.i) \mid (\sigma, l) \in \llbracket m \rrbracket_{\mathcal{L}}\}$$

► Denotation für **Characters** $c \in \mathbf{C}$:

$$\llbracket c \rrbracket_{\mathcal{A}} = \{(\sigma, \text{Ord}(c)) \mid \sigma \in \Sigma\}$$

► Denotation für **Zuweisungen**:

$$\llbracket m = e \rrbracket_{\mathcal{C}} = \{(\sigma, \sigma[l \mapsto v]) \mid (\sigma, l) \in \llbracket m \rrbracket_{\mathcal{L}}, (\sigma, v) \in \llbracket e \rrbracket_{\mathcal{A}}\}$$

Floyd-Hoare-Kalkül

- ▶ Die Regeln des Floyd-Hoare-Kalküls berechnen geltende Zusicherungen
- ▶ Nötige Änderung: Substitution in Zusicherungen wird zur Ersetzung von **Lexp**-Ausdrücken

$$\overline{\vdash \{P[e/x]\} x = e \{P\}}$$

- ▶ Jetzt werden **Lexp** ersetzt, keine **Idt**

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Anmerkung: I und e enthalten **keine** logischen Variablen.

- ▶ Gleichheit und Ungleichheit von **Lexp** nicht immer entscheidbar
- ▶ Problem: Feldzugriffe

Von der Substitution zur Ersetzung

$$\text{Assn } b ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid p(e_1, \dots, e_n) \quad (\text{Literale})$$
$$\mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid b_1 \longrightarrow b_2 \mid \forall v. b \mid \exists v. b$$

$$\text{true}[e/l] := \text{true}$$

$$\text{false}[e/l] := \text{false}$$

$$(a_1 = a_2)[e/l] := (a_1[e/l] = a_2[e/l])$$

$$(b_1 \wedge b_2)[e/l] := (b_1[t/x] \wedge b_2[e/l])$$

$$(\forall v. b)[e/l] := \forall v. (b[e/l])$$

$$n[e/l] := n$$

$$(n \in \mathbf{Z} \uplus \mathbf{C})$$

$$l'[e/l] := l'[e/l] \begin{cases} e & \text{falls } l = l' \\ l' & \text{sonst} \end{cases} \quad (l' \in \mathbf{Lexp})$$

$$(a_1 + a_2)[e/l] := a_1[e/l] + a_2[e/l]$$

...

Beispiel Problemsituationen:

$$(c[i].x[0])[5/c[1].x[0]] = ?$$

$$(c[1].x[0])[8/c[1].x[j]] = ?$$

$$(c[i].x[0])[8/c[1].x[j]] = ?$$

Beispiel

```
int a[3];  
// {true}  
//  
a[2] = 3;  
//  
//  
a[1] = 4;  
//  
//  
a[0] = 5;  
// {a[0] · a[1] · a[2] = 60}
```

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel

```
int a[3];  
// {true}  
//  
a[2] = 3;  
//  
//  
a[1] = 4;  
//  
// {5 · a[1] · a[2] = 60}  
a[0] = 5;  
// {a[0] · a[1] · a[2] = 60}
```

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel

```
int a[3];  
// {true}  
//  
a[2] = 3;  
//  
//  
a[1] = 4;  
// {a[1] · a[2] = 12}  
// {5 · a[1] · a[2] = 60}  
a[0] = 5;  
// {a[0] · a[1] · a[2] = 60}
```

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel

```
int a[3];  
// {true}  
//  
a[2] = 3;  
//  
// {4 · a[2] = 12}  
a[1] = 4;  
// {a[1] · a[2] = 12}  
// {5 · a[1] · a[2] = 60}  
a[0] = 5;  
// {a[0] · a[1] · a[2] = 60}
```

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel

```
int a[3];  
// {true}  
//  
a[2] = 3;  
// {a[2] = 3}  
// {4 · a[2] = 12}  
a[1] = 4;  
// {a[1] · a[2] = 12}  
// {5 · a[1] · a[2] = 60}  
a[0] = 5;  
// {a[0] · a[1] · a[2] = 60}
```

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel

```
int a[3];  
// {true}  
// {3 = 3}  
a[2] = 3;  
// {a[2] = 3}  
// {4 · a[2] = 12}  
a[1] = 4;  
// {a[1] · a[2] = 12}  
// {5 · a[1] · a[2] = 60}  
a[0] = 5;  
// {a[0] · a[1] · a[2] = 60}
```

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];  
int i;  
// {0 ≤ i < 2}  
//  
//  
a[0] = 3;  
//  
//  
a[1] = 7;  
//  
a[2] = 9;  
//  
//  
a[i] = -1;  
// {a[1] = 7}
```

$$\overline{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];
int i;
// {0 ≤ i < 2}
//
//
a[0] = 3;
//
//
a[1] = 7;
//
a[2] = 9;
//
// {a[1] = 7}
a[i] = -1;
// {a[1] = 7}
```

$$\frac{}{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];
int i;
// {0 ≤ i < 2}
//
//
a[0] = 3;
//
//
a[1] = 7;
//
a[2] = 9;
//
// {(i = 1 ∧ -1 = 7) ∨ (i ≠ 1 ∧ a[1] = 7)}
a[i] = -1;
// {a[1] = 7}
```

$$\frac{}{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];
int i;
// {0 ≤ i < 2}
//
//
a[0] = 3;
//
//
a[1] = 7;
//
a[2] = 9;
// {i ≠ 1 ∧ a[1] = 7}
// {(i = 1 ∧ -1 = 7) ∨ (i ≠ 1 ∧ a[1] = 7)}
a[i] = -1;
// {a[1] = 7}
```

$$\frac{}{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];
int i;
// {0 ≤ i < 2}
//
//
a[0] = 3;
//
//
a[1] = 7;
// {i ≠ 1 ∧ a[1] = 7}
a[2] = 9;
// {i ≠ 1 ∧ a[1] = 7}
// {(i = 1 ∧ -1 = 7) ∨ (i ≠ 1 ∧ a[1] = 7)}
a[i] = -1;
// {a[1] = 7}
```

$$\frac{}{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];
int i;
// {0 ≤ i < 2}
//
//
a[0] = 3;
//
// {i ≠ 1 ∧ 7 = 7}
a[1] = 7;
// {i ≠ 1 ∧ a[1] = 7}
a[2] = 9;
// {i ≠ 1 ∧ a[1] = 7}
// {(i = 1 ∧ -1 = 7) ∨ (i ≠ 1 ∧ a[1] = 7)}
a[i] = -1;
// {a[1] = 7}
```

$$\frac{}{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];
int i;
// {0 ≤ i < 2}
//
// {i ≠ 1}
a[0] = 3;
// {i ≠ 1}
// {i ≠ 1 ∧ 7 = 7)}
a[1] = 7;
// {i ≠ 1 ∧ a[1] = 7}
a[2] = 9;
// {i ≠ 1 ∧ a[1] = 7}
// {(i = 1 ∧ -1 = 7) ∨ (i ≠ 1 ∧ a[1] = 7)}
a[i] = -1;
// {a[1] = 7}
```

$$\frac{}{\vdash \{P[e/I]\} I = e \{P\}}$$

Beispiel: Problem

```
int a[3];
int i;
// {0 ≤ i < 2}
// 1
// {i ≠ 1}
a[0] = 3;
// {i ≠ 1}
// {i ≠ 1 ∧ 7 = 7)}
a[1] = 7;
// {i ≠ 1 ∧ a[1] = 7}
a[2] = 9;
// {i ≠ 1 ∧ a[1] = 7}
// {(i = 1 ∧ -1 = 7) ∨ (i ≠ 1 ∧ a[1] = 7)}
a[i] = -1;
// {a[1] = 7}
```

$$\frac{}{\vdash \{P[e/I]\} I = e \{P\}}$$

Arbeitsblatt 8.1: Jetzt seid ihr dran

Annotiert die beiden folgenden Programme:

```
int a[2];
int b[2];
// { $0 \leq n \wedge 0 \leq m \wedge n \leq m$ }
a[0] = m;
//
b[0] = a[0] - n;
//
b[1] = a[0] + n
//
a[1] = b[0] * b[1];
// { $a[1] = m^2 - n^2$ }
```

```
int a[3];
int i;
// { $0 \leq n$ }
i = 2;
//
a[i] = 3;
//
a[0] = n;
//
a[2] = a[2] * a[0];
// { $a[2] = 3 * n$ }
```

Erstes Beispiel: Ein Feld initialisieren

```
1  // {0 ≤ n}
2  //
3  i= 0;
4  //
5  while (i < n) {
6    //
7    //
8    //
9    //
10   //
11   //
12   a[ i]= i;
13   //
14   i= i+1;
15   //
16   }
17  //
18  // {∀j. 0 ≤ j < n → a[j] = j}
```

Erstes Beispiel: Ein Feld initialisieren

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 //
5 while (i < n) {
6     // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n ∧ i < n}
7     //
8     //
9     //
10    //
11    //
12    a[i] = i;
13    //
14    i = i + 1;
15    //
16 }
17 // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n ∧ i ≥ n}
18 // {(∀j. 0 ≤ j < n → a[j] = j)}
```

Erstes Beispiel: Ein Feld initialisieren

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n}
5 while (i < n) {
6   // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n ∧ i < n}
7   //
8   //
9   //
10  //
11  //
12  a[i] = i;
13  //
14  i = i + 1;
15  // { (∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n }
16 }
17 // { (∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n ∧ i ≥ n }
18 // { ∀j. 0 ≤ j < n → a[j] = j }
```

Erstes Beispiel: Ein Feld initialisieren

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n}
5 while (i < n) {
6   // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n ∧ i < n}
7   //
8   //
9   //
10  //
11  //
12  a[i] = i;
13  // {(∀j. 0 ≤ j < i + 1 → a[j] = j) ∧ i + 1 ≤ n}
14  i = i + 1;
15  // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n}
16 }
17 // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n ∧ i ≥ n}
18 // {(∀j. 0 ≤ j < n → a[j] = j}
```

Erstes Beispiel: Ein Feld initialisieren

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n}
5 while (i < n) {
6   // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n ∧ i < n}
7   //
8   //
9   //
10  // {∀j. 0 ≤ j < i + 1 → ((i = j ∧ i = j) ∨ (j ≠ i ∧ a[j] = j))
11   //   ∧ i + 1 ≤ n}
12  a[i] = i;
13  // {(∀j. 0 ≤ j < i + 1 → a[j] = j) ∧ i + 1 ≤ n}
14  i = i + 1;
15  // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n}
16 }
17 // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n ∧ i ≥ n}
18 // {(∀j. 0 ≤ j < n → a[j] = j}
```

Erstes Beispiel: Ein Feld initialisieren

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n}
5 while (i < n) {
6   // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n ∧ i < n}
7   //
8   // {∀j. 0 ≤ j < i → ((i = j ∧ i = j) ∨ (j ≠ i ∧ a[j] = j))
9   //   ∧ ((i = i ∧ i = i) ∨ (i ≠ i ∧ a[i] = i)) ∧ i + 1 ≤ n}
10  // {∀j. 0 ≤ j < i + 1 → ((i = j ∧ i = j) ∨ (j ≠ i ∧ a[j] = j))
11  //   ∧ i + 1 ≤ n}
12  a[i] = i;
13  // {(∀j. 0 ≤ j < i + 1 → a[j] = j) ∧ i + 1 ≤ n}
14  i = i + 1;
15  // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n}
16 }
17 // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n ∧ i ≥ n}
18 // {∀j. 0 ≤ j < n → a[j] = j}
```

► Wichtiges Theorem:

$$(\forall j. 0 \leq j < n \longrightarrow P[j]) \wedge P[n] \implies \forall j. 0 \leq j < n + 1 \longrightarrow P[j]$$

Erstes Beispiel: Ein Feld initialisieren

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n}
5 while (i < n) {
6   // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n ∧ i < n}
7   // {∀j. 0 ≤ j < i → a[j] = j ∧ i + 1 ≤ n}
8   // {∀j. 0 ≤ j < i → ((i = j ∧ i = j) ∨ (j ≠ i ∧ a[j] = j))
9   //   ∧ ((i = i ∧ i = i) ∨ (i ≠ i ∧ a[i] = i)) ∧ i + 1 ≤ n}
10  // {∀j. 0 ≤ j < i + 1 → ((i = j ∧ i = j) ∨ (j ≠ i ∧ a[j] = j))
11  //   ∧ i + 1 ≤ n}
12  a[i] = i;
13  // {(∀j. 0 ≤ j < i + 1 → a[j] = j) ∧ i + 1 ≤ n}
14  i = i + 1;
15  // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n}
16 }
17 // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n ∧ i ≥ n}
18 // {∀j. 0 ≤ j < n → a[j] = j}
```

► Wichtiges Theorem:

$$(\forall j. 0 \leq j < n \rightarrow P[j]) \wedge P[n] \implies \forall j. 0 \leq j < n + 1 \rightarrow P[j]$$

Erstes Beispiel: Ein Feld initialisieren

```
1 // {0 ≤ n}
2 // {∀j. 0 ≤ j < 0 → a[j] = j ∧ 0 ≤ n}
3 i = 0;
4 // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n}
5 while (i < n) {
6   // {∀j. 0 ≤ j < i → a[j] = j ∧ i ≤ n ∧ i < n}
7   // {∀j. 0 ≤ j < i → a[j] = j ∧ i + 1 ≤ n}
8   // {∀j. 0 ≤ j < i → ((i = j ∧ i = j) ∨ (j ≠ i ∧ a[j] = j))
9   //   ∧ ((i = i ∧ i = i) ∨ (i ≠ i ∧ a[i] = i)) ∧ i + 1 ≤ n}
10  // {∀j. 0 ≤ j < i + 1 → ((i = j ∧ i = j) ∨ (j ≠ i ∧ a[j] = j))
11  //   ∧ i + 1 ≤ n}
12  a[i] = i;
13  // {(∀j. 0 ≤ j < i + 1 → a[j] = j) ∧ i + 1 ≤ n}
14  i = i + 1;
15  // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n}
16 }
17 // {(∀j. 0 ≤ j < i → a[j] = j) ∧ i ≤ n ∧ i ≥ n}
18 // {∀j. 0 ≤ j < n → a[j] = j}
```

► Wichtiges Theorem:

$$(\forall j. 0 \leq j < n \longrightarrow P[j]) \wedge P[n] \implies \forall j. 0 \leq j < n + 1 \longrightarrow P[j]$$

Beispiel: Suche nach dem maximalen Element

```
1  // {0 < n}
2  //
3  i= 0;
4  //
5  r= 0;
6  //
7  while (i < n) {
8      //
9      //
10     if (a[r] < a[i]) {
11         //
12         //
13         //
14         r= i;
15         //
16     }
17     else {
18         //
19         //
20     }
21     //
22     i= i+1;
23     //
24 }
25 //
26 // { (∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n }
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   //
9   //
10  if (a[r] < a[i]) {
11    //
12    //
13    //
14    r= i;
15    //
16  }
17  else {
18    //
19    //
20  }
21  //
22  i= i+1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 //
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   //
9   //
10  if (a[r] < a[i]) {
11    //
12    //
13    //
14    r= i;
15    //
16  }
17  else {
18    //
19    //
20  }
21  //
22  i= i+1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   //
10  if (a[r] < a[i]) {
11    //
12    //
13    //
14    r= i;
15    //
16  }
17  else {
18    //
19    //
20  }
21  //
22  i= i+1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   //
10  if (a[r] < a[i]) {
11    //
12    //
13    //
14    r= i;
15    //
16  }
17  else {
18    //
19    //
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i= i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   //
10  if (a[r] < a[i]) {
11    //
12    //
13    //
14    r= i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    //
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i= i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   //
10  if (a[r] < a[i]) {
11    //
12    //
13    //
14    r= i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])}
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i= i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
10  if (a[r] < a[i]) {
11    //
12    //
13    //
14    r= i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])}
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i= i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
10  if (a[r] < a[i]) {
11    //
12    //
13    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[i]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
14    r= i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])}
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i= i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
10  if (a[r] < a[i]) {
11    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ a[r] < a[i]}
12    //
13    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[i]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
14    r= i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])}
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i= i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i= 0;
4 //
5 r= 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
10  if (a[r] < a[i]) {
11    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ a[r] < a[i]}
12    // {(∀j. 0 ≤ j < i → a[j] ≤ a[i]) ∧ a[i] ≤ a[r] ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
13    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[i]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
14    r= i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])}
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i= i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 //
3 i = 0;
4 // {(∀j. 0 ≤ j < i → a[j] ≤ a[0]) ∧ 0 ≤ i ∧ 0 ≤ 0 < n}
5 r = 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
10  if (a[r] < a[i]) {
11    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ a[r] < a[i]}
12    // {(∀j. 0 ≤ j < i → a[j] ≤ a[i]) ∧ a[i] ≤ a[r] ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
13    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[i]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
14    r = i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])}
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i = i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Beispiel: Suche nach dem maximalen Element

```
1 // {0 < n}
2 // {(∀j. 0 ≤ j < 0 → a[j] ≤ a[0]) ∧ 0 ≤ 0 ∧ 0 ≤ 0 < n}
3 i = 0;
4 // {(∀j. 0 ≤ j < i → a[j] ≤ a[0]) ∧ 0 ≤ i ∧ 0 ≤ 0 < n}
5 r = 0;
6 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ∧ 0 ≤ r < n}
7 while (i < n) {
8   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i < n ∧ 0 ≤ r < n}
9   // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
10  if (a[r] < a[i]) {
11    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ a[r] < a[i]}
12    // {(∀j. 0 ≤ j < i → a[j] ≤ a[i]) ∧ a[i] ≤ a[r] ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
13    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[i]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ i < n}
14    r = i;
15    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
16  }
17  else {
18    // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n ∧ ¬(a[r] < a[i])}
19    // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
20  }
21  // {(∀j. 0 ≤ j < i + 1 → a[j] ≤ a[r]) ∧ 0 ≤ i + 1 ≤ n ∧ 0 ≤ r < n}
22  i = i + 1;
23  // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n}
24 }
25 // {(∀j. 0 ≤ j < i → a[j] ≤ a[r]) ∧ 0 ≤ i ≤ n ∧ 0 ≤ r < n ∧ n ≤ i}
26 // {(∀j. 0 ≤ j < n → a[j] ≤ a[r]) ∧ 0 ≤ r < n}
```

Vorgehensweise

```
1 // {}  
2 while (b) {  
3     // {}  
4     c  
5     // {}  
6 }  
7 // {}  
8 // { $\Phi$ }
```

Vorgehensweise

```
1 // {}  
2 while (b) {  
3     // {I ∧ b}  
4     c  
5     // {}  
6 }  
7 // {}  
8 // {Φ}
```

① Finde/rate/formuliere Invariante I

Vorgehensweise

```
1 // {}  
2 while (b) {  
3     // {I ∧ b}  
4     c  
5     // {}  
6 }  
7 // {I ∧ ¬b}  
8 // {Φ}
```

- ① Finde/rate/formuliere Invariante I
- ② Beweise $(I \wedge \neg b) \longrightarrow \Phi$

Vorgehensweise

```
1 // {}
2 while (b) {
3     // {I ∧ b}
4     c
5     // {I}
6 }
7 // {I ∧ ¬b}
8 // {Φ}
```

- ① Finde/rate/formuliere Invariante I
- ② Beweise $(I \wedge \neg b) \longrightarrow \Phi$
- ③ Zeige mittels Floyd-Hoare-Regeln, dass Invariante durch Schleifenrumpf c erhalten bleibt

Vorgehensweise

```
1 // {I}
2 while (b) {
3     // {I ∧ b}
4     c
5     // {I}
6 }
7 // {I ∧ ¬b}
8 // {Φ}
```

- ① Finde/rate/formuliere Invariante I
- ② Beweise $(I \wedge \neg b) \longrightarrow \Phi$
- ③ Zeige mittels Floyd-Hoare-Regeln, dass Invariante durch Schleifenrumpf c erhalten bleibt
- ④ Setze Beweis mit Floyd-Hoare Regeln vor der Schleife fort

Längeres Beispiel: Suche nach einem Null-Element

```
1  i= 0;
2  r= -1;
3  while (i < n) {
4      if (a[i] == 0) {
5          r= i;
6      }
7      else {
8      }
9      i= i+1;
10 }
11 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1  i= 0;
2  r= -1;
3  while (i < n) {
4      if (a[i] == 0) {
5          r= i;
6      }
7      else {
8      }
9      i= i+1;
10 }
11 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Merkt euch folgende korrekten logischen Umformungen:

- ▶ $(F \wedge H) \vee (G \wedge H)$ ist äquivalent zu $(F \vee G) \wedge H$
- ▶ $\neg F \vee G$ ist äquivalent zu $F \longrightarrow G$

Längeres Beispiel: Suche nach einem Null-Element

```
1  // {0 ≤ n}
2  //
3  i= 0;
4  //
5  r= -1;
6  //
7  while (i < n) {
8      //
9      //
10     if (a[i] == 0) {
11         //
12         //
13         //
14         //
15         //
16         r= i;
17         //
18     }
19     else {
20         //
21         //
22     }
23     //
24     i= i+1;
25     //
26 }
27 //
28 //
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1 // {0 ≤ n}
2 //
3 i= 0;
4 //
5 r= -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   //
10  if (a[i] == 0) {
11    //
12    //
13    //
14    //
15    //
16    r= i;
17    //
18  }
19  else {
20    //
21    //
22  }
23  //
24  i= i+1;
25  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 //
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1 // {0 ≤ n}
2 //
3 i= 0;
4 //
5 r= -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   //
10  if (a[i] == 0) {
11    //
12    //
13    //
14    //
15    //
16    r= i;
17    //
18  }
19  else {
20    //
21    //
22  }
23  //
24  i= i+1;
25  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ i = n}
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1 // {0 ≤ n}
2 //
3 i= 0;
4 //
5 r= -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   //
10  if (a[i] == 0) {
11    //
12    //
13    //
14    //
15    //
16    r= i;
17    //
18  }
19  else {
20    //
21    //
22  }
23  // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24  i= i+1;
25  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ i = n}
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1 // {0 ≤ n}
2 //
3 i= 0;
4 //
5 r= -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   //
10  if (a[i] == 0) {
11    //
12    //
13    //
14    //
15    //
16    r= i;
17    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18  }
19  else {
20    //
21    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22  }
23  // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24  i= i+1;
25  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ i = n}
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 //
5 r = -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   //
10  if (a[i] == 0) {
11    //
12    //
13    //
14    //
15    //
16    r = i;
17    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18  }
19  else {
20    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22  }
23  // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24  i = i + 1;
25  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ i = n}
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 //
5 r = -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10  if (a[i] == 0) {
11    //
12    //
13    //
14    //
15    //
16    r = i;
17    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18  }
19  else {
20    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22  }
23  // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24  i = i + 1;
25  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ i = n}
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```
1 // {0 ≤ n}
2 //
3 i = 0;
4 //
5 r = -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10  if (a[i] == 0) {
11    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}
12    //
13    //
14    //
15    //
16    r = i;
17    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18  }
19  else {
20    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21    // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22  }
23  // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24  i = i + 1;
25  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ i = n}
29 // {r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0}
```

Längeres Beispiel: Suche nach einem Null-Element

```

1  // {0 ≤ n}
2  //
3  i = 0;
4  //
5  r = -1;
6  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7  while (i < n) {
8    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10   if (a[i] == 0) {
11     // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}
12     //
13     //
14     //
15     // {(i ≠ -1 → 0 ≤ i < i + 1 ∧ a[i] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$A(i)$

$B(i)$

C

```

16   r = i;
17   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18   }
19   else {
20     // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21     // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22   }
23   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24   i = i + 1;
25   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }
27 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ ¬(i < n)}
28 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ i = n}
29 // {(r ≠ -1 → 0 ≤ r < n ∧ a[r] = 0)}

```

Längeres Beispiel: Suche nach einem Null-Element

```

1 // {0 ≤ n}
2 //
3 i = 0;
4 //
5 r = -1;
6 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10  if (a[i] == 0) {
11    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$B(i) \wedge C$

```

12 // {0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}
13 //
14 //
15 // {(i ≠ -1 → 0 ≤ i < i + 1 ∧ a[i] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$\underbrace{\hspace{10em}}_{A(i)} \quad \underbrace{\hspace{10em}}_{B(i)} \quad \underbrace{\hspace{10em}}_C$

```

16   r = i;
17   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18 }
19 else {
20   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22 }
23 // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24 i = i + 1;
25 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
26 }

```

Längeres Beispiel: Suche nach einem Null-Element

```

1  // {0 ≤ n}
2  //
3  i = 0;
4  //
5  r = -1;
6  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7  while (i < n) {
8    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10   if (a[i] == 0) {
11     // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$B(i) \wedge C$

```

12 // {0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$\neg A(i)$ C $B(i)$ C

```

13 // {(i = -1 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0) ∨ (0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0)}
14 //
15 // {(i ≠ -1 → 0 ≤ i < i + 1 ∧ a[i] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$A(i)$ $B(i)$ C

```

16 r = i;
17 // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18 }
19 else {
20   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22 }
23 // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}

```

```

24 i = i + 1;
25 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i < n}

```

Längeres Beispiel: Suche nach einem Null-Element

```

1  // {0 ≤ n}
2  //
3  i = 0;
4  //
5  r = -1;
6  // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7  while (i < n) {
8    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9    // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10   if (a[i] == 0) {
11     // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$B(i) \wedge C$

```

12 // {0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$\neg A(i)$

C

$B(i)$

C

```

13 // {(i = -1 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0) ∨ (0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0)}

```

```

14 // {(i = -1 ∨ (0 ≤ i < i + 1 ∧ a[i] = 0)) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

```

15 // {(i ≠ -1 → 0 ≤ i < i + 1 ∧ a[i] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$A(i)$

$B(i)$

C

```

16 r = i;
17 // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18 }
19 else {
20   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21   // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22 }
23 // {(r ≠ -1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24 i = i + 1;
25 // {(r ≠ -1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i < n}

```

Längeres Beispiel: Suche nach einem Null-Element

```

1  // {0 ≤ n}
2  //
3  i = 0;
4  // {(−1 ≠ −1 → 0 ≤ −1 < i ∧ a[−1] = 0) ∧ 0 ≤ i ≤ n}
5  r = −1;
6  // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7  while (i < n) {
8    // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9    // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10   if (a[i] == 0) {
11     // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$B(i) \wedge C$

```

12 // {0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$\neg A(i)$ C $B(i)$ C

```

13 // {(i = −1 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0) ∨ (0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0)}
14 //
15 // {(i ≠ −1 → 0 ≤ i < i + 1 ∧ a[i] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$A(i)$ $B(i)$ C

```

16 r = i;
17 // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18 }
19 else {
20 // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21 // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22 }
23 // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24 i = i + 1;
25 // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i < n}

```

Längeres Beispiel: Suche nach einem Null-Element

```

1 // {0 ≤ n}
2 // {(−1 ≠ −1 → 0 ≤ −1 < 0 ∧ a[−1] = 0) ∧ 0 ≤ 0 ≤ n}
3 i = 0;
4 // {(−1 ≠ −1 → 0 ≤ −1 < i ∧ a[−1] = 0) ∧ 0 ≤ i ≤ n}
5 r = −1;
6 // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n}
7 while (i < n) {
8   // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i ≤ n ∧ i < n}
9   // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
10  if (a[i] == 0) {
11    // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$B(i) \wedge C$

```

12 // {0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$\neg A(i)$

C

$B(i)$

C

```

13 // {(i = −1 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0) ∨ (0 ≤ i < i + 1 ∧ a[i] = 0 ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0)}

```

```

14 //
15 // {(i ≠ −1 → 0 ≤ i < i + 1 ∧ a[i] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] = 0}

```

$A(i)$

$B(i)$

C

```

16 r = i;
17 // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
18 }
19 else {
20   // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n ∧ a[i] ≠ 0}
21   // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
22 }
23 // {(r ≠ −1 → 0 ≤ r < i + 1 ∧ a[r] = 0) ∧ 0 ≤ i + 1 ≤ n}
24 i = i + 1;
25 // {(r ≠ −1 → 0 ≤ r < i ∧ a[r] = 0) ∧ 0 ≤ i < n}

```

Benutzte Logische Umformungen

- ▶ Zeilen 11-12:
 - ▶ $[D \wedge C] \Rightarrow [C]$ und
 - ▶ Erweiterung von C auf $B(i) \wedge C$, weil $C \vdash B(i)$ gilt.
- ▶ $[\varphi] \Rightarrow [\psi \vee \varphi]$ in der Form

$$[(B(i) \wedge C)] \Rightarrow [(\neg A(i) \wedge C) \vee (B(i) \wedge C)]$$

- ▶ DeMorgan:

$$[(\neg A(i) \wedge C) \vee (B(i) \wedge C)] \Rightarrow [(\neg A(i) \vee B(i)) \wedge C]$$

- ▶ Klassische Implikation:

$$[\neg U \vee V] \Leftrightarrow [U \Rightarrow V]$$

Längeres Beispiel: Suche nach einem Null-Element

```
10 /** {  $0 \leq n$  } */
11 /** {  $0 \leq 0 \leq n$  } */
12 i = 0;
13 /** {  $0 \leq i \leq n$  } */
14 /** {  $(-1 \neq -1 \rightarrow 0 \leq -1 < i \wedge a[-1] = 0) \wedge 0 \leq i \leq n$  } */
15 r = -1;
16 /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i \leq n$  } */
17 while (i < n) {
18   /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i \leq n \wedge i < n$  } */
19   /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i+1 \leq n$  } */
20   if (a[i] == 0) {
21     /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i+1 \leq n \wedge a[i] = 0$  } */
22     /** {  $0 \leq i+1 \leq n \wedge a[i] = 0$  } */
23     /** {  $(i \neq -1 \rightarrow 0 \leq i < i+1 \wedge a[i] = 0) \wedge 0 \leq i+1 \leq n$  } */
24     r = i;
25     /** {  $(r \neq -1 \rightarrow 0 \leq r < i+1 \wedge a[r] = 0) \wedge 0 \leq i+1 \leq n$  } */
26   }
27   else {
28     /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i+1 \leq n \wedge a[i] \neq 0$  } */
29     /** {  $(r \neq -1 \rightarrow 0 \leq r < i+1 \wedge a[r] = 0) \wedge 0 \leq i+1 \leq n$  } */
30   }
31   /** {  $(r \neq -1 \rightarrow 0 \leq r < i+1 \wedge a[r] = 0) \wedge 0 \leq i+1 \leq n$  } */
32   i = i+1;
33   /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i \leq n$  } */
34 }
35 /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i \leq n \wedge \neg(i < n)$  } */
36 /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge 0 \leq i \leq n \wedge i \geq n$  } */
37 /** {  $(r \neq -1 \rightarrow 0 \leq r < i \wedge a[r] = 0) \wedge i = n$  } */
38 /** {  $r \neq -1 \rightarrow 0 \leq r < n \wedge a[r] = 0$  } */
```

Allgemeine Regel bei Ersetzungen?

Wie sieht nun die allgemeine Regel aus für

$$\frac{}{\vdash \{P[e/I]\} \quad I = e \{P\}}$$

```
int a[3];
int i;
a[0] = 3;
a[1] = 7;
a[2] = 9;
a[a[2]-a[1]] = -1;
// {a[2] = -1}
```

```
int a[3];
int i;
i = 8;
a[0] = 3;
a[1] = i;
a[2] = 9;
a[a[2]-a[1]] = -1;
// {a[1] = -1}
```

Allgemeine Regel bei Ersetzungen (Nur Arrays)

Wie sieht nun die allgemeine Regel aus für

$$\overline{\vdash \{P[e/I]\} \mid I = e \{P\}}$$

① Wenn I Programmvariable ist, wie gewohnt substituieren

② Wenn $I = a[s]$:

🌀 Vorkommen der Form $a[t]$ in **Literalen** $L(a[t])$ und s und t beide in $\mathbb{Z}$ oder **Idt**,

▶ dann ersetze $L(a[t])$ durch $L(e)$, falls $s = t$

🌀 Vorkommen der Form $a[t]$ in **Literalen** $L(a[t])$ und s oder t sind nicht aus $\mathbb{Z}$,

▶ dann ersetze $L(a[t])$ durch $(t = s \wedge L(e)) \vee (t \neq s \wedge L(a[t]))$

2.2 könnt ihr immer machen, 2.1 ist eine Optimierung

▶ Das ist jetzt immer noch nicht die ganz allgemeine Form, aber für unsere Belange reicht das.

Zusammenfassung

- ▶ Strukturierte Datentypen (Felder und Structs) erfordern strukturierte Adressen
- ▶ Abstraktion über „echtem“ Speichermodell
- ▶ Änderungen in der Semantik und im Floyd-Hoare-Kalkül überschaubar
- ▶ ... aber mit erheblichen Konsequenzen:
 - ▶ Substitution wird zur Ersetzung
 - ▶ Anwendung der Zuweisungsregel führt i.A. zu großen Formeln

Fahrplan

- ▶ Einführung
- ▶ Operationale Semantik
- ▶ Denotationale Semantik
- ▶ Äquivalenz der Operationalen und Denotationalen Semantik
- ▶ Der Floyd-Hoare-Kalkül I
- ▶ Der Floyd-Hoare-Kalkül II: Invarianten
- ▶ Korrektheit des Floyd-Hoare-Kalküls
- ▶ **Strukturierte Datentypen**
- ▶ Verifikationsbedingungen
- ▶ Vorwärts mit Floyd und Hoare
- ▶ Funktionen und Prozeduren I
- ▶ Funktionen und Prozeduren II
- ▶ Referenzen und Speichermodelle
- ▶ Ausblick und Rückblick