Gegeben sei folgendes Java-Programm P:

$\langle\varphi\rangle$ (Vorbedingung)

```
i = 0;
res = 0;
while (i < n){
  res = res + a[i];
  i = i + 1;
}
```

$\langle\psi\rangle$ (Nachbedingung)

a) Als Vorbedingung φ für das oben aufgeführte Programm P gelte $\mathtt{n} \geq 0$ und als Nachbedingung ψ gelte $\mathtt{res} = \sum_{k=0}^{\mathtt{n}-1} \mathtt{a}[k]$. Vervollständigen Sie die folgende Verifikation des Algorithmus, indem Sie die leeren Zeilen durch korrekte Zusicherungen entsprechend des Hoare-Kalküls ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Hinweise:

- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x + 1 = y + 1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen.

$\langle \mathtt{n} \geq 0 \rangle$

$\langle \_\_\_\_\_ \rangle$

```
i = 0;
```

$\langle \_\_\_\_\_ \rangle$

```
res = 0;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
while (i < n){
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
   res = res + a[i];
```

$\langle \_\_\_\_\_ \rangle$

```
   i = i + 1;
```

$\langle \_\_\_\_\_ \rangle$

```
}
```

$\langle \_\_\_\_\_ \rangle$

$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{n}-1} \mathtt{a}[k] \rangle$

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung müssen folgende Schritte durchgeführt werden:

- Angabe einer Variante V
- Beweis, dass es sich für die Bedingung $B = \texttt{i} < \texttt{n}$ um eine gültige Variante handelt ($B \implies V \geq 0$)
- Beweis der Reduzierung der Variante mit Hilfe des Hoare-Kalküls

Lösungsvorschlag

a)
$\langle \mathtt{n} \geq 0 \rangle$
$\langle \mathtt{n} \geq 0 \wedge 0 = 0 \wedge 0 = 0 \rangle$

```
i = 0;
```

$\langle \mathtt{n} \geq 0 \wedge \mathtt{i} = 0 \wedge 0 = 0 \rangle$

```
res = 0;
```

$\langle \mathtt{n} \geq 0 \wedge \mathtt{i} = 0 \wedge \mathtt{res} = 0 \rangle$
$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{a}[k] \wedge \mathtt{i} \leq \mathtt{n} \rangle$

```
while (i < n){
```

$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{a}[k] \wedge \mathtt{i} \leq \mathtt{n} \wedge \mathtt{i} < \mathtt{n} \rangle$
$\langle \mathtt{res} + \mathtt{a}[\mathtt{i}] = \sum_{k=0}^{\mathtt{i}+1-1} \mathtt{a}[k] \wedge \mathtt{i} + 1 \leq \mathtt{n} \rangle$

```
    res = res + a[i];
```

$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{i}+1-1} \mathtt{a}[k] \wedge \mathtt{i} + 1 \leq \mathtt{n} \rangle$

```
    i = i + 1;
```

$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{a}[k] \wedge \mathtt{i} \leq \mathtt{n} \rangle$

```
}
```

$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{a}[k] \wedge \mathtt{i} \leq \mathtt{n} \wedge \neg(\mathtt{i} < \mathtt{n}) \rangle$
$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{n}-1} \mathtt{a}[k] \rangle$

Rezept:

1. Über der Schleife (eine Zeile freilassen) die Vorbedingung und die Zuweisungen vor der Schleife eintragen (Vorsicht bei mehreren Zuweisungen zur selben Variable!)
2. Von dieser Zusicherung aus die Zuweisungsregel rückwärts bis zum Anfang des Programms anwenden:
   Zuweisungsregel rückwärts anwenden: Ersetze alle Vorkommen der linken Seite in der unteren Zusicherung durch die rechte Seite in der oberen Zusicherung!
3. Prüfe, ob zweite Zusicherung aus erster folgt (falls nicht, wurde ein Fehler in Schritt 1 gemacht und man sollte nochmal von dort anfangen)
4. Finde Schleifeninvariante (**Einzige Stelle, wo man wirklich nachdenken muss!**)
   - Allgemeines Vorgehen: Führe Schleife ein paar Mal aus und betrachte, wie sich Variablenwerte verändern; versuche Muster zu erkennen.

     Im Beispiel ergibt sich:

Iteration	i	res
0	0	0
1	1	a[0]
2	2	a[0] + a[1]
3	3	a[0] + a[1] + a[2]
...	...	...

     $\Longrightarrow$ $\mathtt{res} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{a}[k]$ ist Teil der Schleifeninvariante. Auf den zweiten Teil $\mathtt{i} \leq \mathtt{n}$ stößt man, wenn man sich fragt, was noch fehlt, um aus dem ersten Teil und der negierten Schleifenbedingung die Nachbedingung folgern zu können.

- Meistens funktioniert folgende Heuristik: Versuche zunächst, die gewünschte Nachbedingung als Schleifeninvariante zu nehmen. Ändere sie dann so ab, dass sie die folgenden drei Bedingungen erfüllt:
  i. Sie ist eine Schleifeninvariante (d. h. wenn sie zu Beginn des Schleifenrumpfes gilt und dort auch die Schleifenbedingung zutrifft, dann gilt sie am Ende des Schleifenrumpfes ebenfalls).
  ii. Sie folgt aus der Zusicherung vor der Schleife.
  iii. Aus ihr folgt die gewünschte Nachbedingung, wenn die Schleifenbedingung nicht mehr gilt.

  Im Beispiel:
  Ersetze obere Grenze `n` der Schleifenbedingung in Nachbedingung durch Laufvariable `i` und füge Bedingung $\texttt{i} \leq \texttt{n}$ hinzu, die mit negierter Schleifenbedingung die Gleichheit von Laufvariablen und oberer Grenze impliziert
- Falls man keine geeignete Schleifeninvariante findet, kann man das weitere Rezept zuerst mit abstrakten Zusicherungen durchführen. Dabei muss man aber darauf achten, dass durch Zuweisungsregeln Substitutionen auf diese abstrakten Zusicherungen anzuwenden sind. Findet man später doch noch die Invariante, kann man die abstrakten Zusicherungen einfach entsprechend definieren. Eine 2-Schritte-Lösung dieser Aufgabe unter Verwendung abstrakter Zusicherungen ist unter diesem Rezept aufgeführt.

5. Setze Schleifeninvariante unmittelbar über der Schleife und unmittelbar vor Ende der Schleife ein
6. Erste Zusicherung in der Schleife ist Schleifeninvariante und Schleifenbedingung
7. Erste Zusicherung nach der Schleife ist Schleifeninvariante und negierte Schleifenbedingung
8. Prüfe, ob Schleifeninvariante aus der bisherigen Zusicherung vor der Schleife folgt (falls nicht, ist die „Schleifeninvariante“ keine Invariante und man muss eine andere Schleifeninvariante finden, also zurück zu Schritt 4)
9. Prüfe, ob Nachbedingung aus vorletzter Zusicherung folgt (falls nicht, ist die Schleifeninvariante zu schwach und man muss eine stärkere Schleifeninvariante finden, also zurück zu Schritt 4)
10. Von der letzten Zusicherung innerhalb der Schleife aus (darin steht genau die Schleifeninvariante) die Zuweisungsregel rückwärts bis zum Anfang der Schleife anwenden
11. Prüfe, ob die zweite Zusicherung innerhalb der Schleife aus der ersten Zusicherung innerhalb der Schleife folgt (falls nicht, ist die „Schleifeninvariante“ keine Invariante und man muss eine andere Schleifeninvariante finden, also zurück zu Schritt 4)

Lösung mit abstrakten Zusicherungen

1) Zunächst Schema ohne Kenntnis der Invariante durchführen:

$\langle \mathtt{n} \geq 0 \rangle$
$\langle \mathtt{n} \geq 0 \wedge 0 = 0 \wedge 0 = 0 \rangle$

```
i = 0;
```

$\langle \mathtt{n} \geq 0 \wedge \mathtt{i} = 0 \wedge 0 = 0 \rangle$

```
res = 0;
```

$\langle \mathtt{n} \geq 0 \wedge \mathtt{i} = 0 \wedge \mathtt{res} = 0 \rangle$
$\langle I \rangle$

```
while (i < n){
```

$\langle I \wedge \mathtt{i} < \mathtt{n} \rangle$
$\langle Z_1 \rangle$

```
   res = res + a[i];
```

$\langle Z_2 \rangle$

```
   i = i + 1;
```

$\langle I \rangle$

```
}
```

$\langle I \wedge \neg(\mathtt{i} < \mathtt{n}) \rangle$
$\langle \mathtt{res} = \sum_{k=0}^{\mathtt{n}-1} \mathtt{a}[k] \rangle$

$Z_1 = Z_2[\mathtt{res}/\mathtt{res} + \mathtt{a}[\mathtt{i}]]$

$Z_2 = I[\mathtt{i}/\mathtt{i} + 1]$

2) Sobald Invariante erkannt wurde, abstrakte Zusicherungen explizit definieren:
~~$Z_1 = Z_2[\mathtt{res}/\mathtt{res} + \mathtt{a}[\mathtt{i}]]$~~

~~$Z_2 = I[\mathtt{i}/\mathtt{i} + 1]$~~

$I = \mathtt{res} = \sum_{k=0}^{\mathtt{i}-1} \mathtt{a}[k] \wedge \mathtt{i} \leq \mathtt{n}$

$Z_1 = \mathtt{res} + \mathtt{a}[\mathtt{i}] = \sum_{k=0}^{\mathtt{i}+1-1} \mathtt{a}[k] \wedge \mathtt{i} + 1 \leq \mathtt{n}$

$Z_2 = \mathtt{res} = \sum_{k=0}^{\mathtt{i}+1-1} \mathtt{a}[k] \wedge \mathtt{i} + 1 \leq \mathtt{n}$

b) Wir wählen als Variante $V = \mathtt{n} - \mathtt{i}$. Hiermit lässt sich die Terminierung von P beweisen, denn für die einzige Schleife im Programm (mit Schleifenbedingung $B = \mathtt{i} < \mathtt{n}$) gilt:

- $B \Rightarrow V \geq 0$, denn $B \Leftrightarrow \mathtt{i} < \mathtt{n} \Leftrightarrow \mathtt{n} - \mathtt{i} > 0 \Leftrightarrow V > 0$ und
- $\langle \mathtt{n} - \mathtt{i} = m \wedge \mathtt{i} < \mathtt{n} \rangle$
  $\langle \mathtt{n} - (\mathtt{i} + 1) < m \rangle$

```
res = res + a[i];
```

$\langle \mathtt{n} - (\mathtt{i} + 1) < m \rangle$

```
i = i + 1;
```

$\langle \mathtt{n} - \mathtt{i} < m \rangle$

Rezept:

1. Falls Laufvariable aufwärts zählt: Variantenkandidat ist obere Grenze minus Laufvariable;
   falls Laufvariable abwärts zählt: Variantenkandidat ist Laufvariable minus untere Grenze
   (falls keiner dieser Fälle zutrifft, nach komplexeren Varianten suchen; „Laufvariablen“ können auch komplexere Ausdrücke wie z. B. der Unterschied $|x - y|$ zweier Variablen x und y sein)
2. Schreibe $B \implies V \geq 0$ explizit (!) hin (also ersetze B und V entsprechend) und prüfe, ob die Implikation wahr ist
3. Erste Zusicherung im Hoare-Kalkül:
   $$\langle V = m \wedge B \rangle$$
4. Letzte Zusicherung im Hoare-Kalkül:
   $$\langle V < m \rangle$$
5. Gesamten (!) Programmcode **innerhalb der Schleife** dazwischen schreiben
6. Von letzter Zusicherung aus Zuweisungsregel rückwärts anwenden bis zum Anfang des Schleifenkörpers
7. Prüfe, ob zweite Zusicherung aus der ersten folgt (falls nicht, ist die Variante falsch und man muss mit einer anderen von vorne beginnen)

Gegeben sei folgendes Java-Programm P:

$\langle\varphi\rangle$ (Vorbedingung)

```
i = n - 1;
res = 1;
while (i >= 0){
  res = res * a[i];
  i = i - 1;
}
```

$\langle\psi\rangle$ (Nachbedingung)

a) Als Vorbedingung φ für das oben aufgeführte Programm P gelte $\mathtt{n} \geq 0$ und als Nachbedingung ψ gelte $\mathtt{res} = \Pi_{k=0}^{\mathtt{n}-1}\mathtt{a}[k]$. Vervollständigen Sie die folgende Verifikation des Algorithmus, indem Sie die leeren Zeilen durch korrekte Zusicherungen entsprechend des Hoare-Kalküls ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Hinweise:

- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x + 1 = y + 1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen.

$\langle \mathbf{n} \geq 0 \rangle$

$\langle$ _____ $\rangle$

```
i = n - 1;
```

$\langle$ _____ $\rangle$

```
res = 1;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
while (i >= 0){
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
   res = res * a[i];
```

$\langle$ _____ $\rangle$

```
   i = i - 1;
```

$\langle$ _____ $\rangle$

```
}
```

$\langle$ _____ $\rangle$

$\langle \mathbf{res} = \Pi_{k=0}^{\mathbf{n}-1} \mathbf{a}[k] \rangle$

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung müssen folgende Schritte durchgeführt werden:

- Angabe einer Variante V
- Beweis, dass es sich für die Bedingung $B = \texttt{i} \geq 0$ um eine gültige Variante handelt ($B \implies V \geq 0$)
- Beweis der Reduzierung der Variante mit Hilfe des Hoare-Kalküls

Lösungsvorschlag

a)
$\langle \mathtt{n} \geq 0 \rangle$
$\langle \mathtt{n} \geq 0 \wedge n-1 = n-1 \wedge 1 = 1 \rangle$

```
i = n - 1;
```

$\langle \mathtt{n} \geq 0 \wedge \mathtt{i} = n-1 \wedge 1 = 1 \rangle$

```
res = 1;
```

$\langle \mathtt{n} \geq 0 \wedge \mathtt{i} = n-1 \wedge \mathtt{res} = 1 \rangle$
$\langle \mathtt{res} = \Pi_{k=\mathtt{i}+1}^{\mathtt{n}-1} \mathtt{a}[k] \wedge \mathtt{i} \geq -1 \rangle$

```
while (i >= 0){
```

$\langle \mathtt{res} = \Pi_{k=\mathtt{i}+1}^{\mathtt{n}-1} \mathtt{a}[k] \wedge \mathtt{i} \geq -1 \wedge \mathtt{i} \geq 0 \rangle$
$\langle \mathtt{res} \cdot \mathtt{a}[\mathtt{i}] = \Pi_{k=\mathtt{i}-1+1}^{\mathtt{n}-1} \mathtt{a}[k] \wedge \mathtt{i} - 1 \geq -1 \rangle$

```
   res = res * a[i];
```

$\langle \mathtt{res} = \Pi_{k=\mathtt{i}-1+1}^{\mathtt{n}-1} \mathtt{a}[k] \wedge \mathtt{i} - 1 \geq -1 \rangle$

```
   i = i - 1;
```

$\langle \mathtt{res} = \Pi_{k=\mathtt{i}+1}^{\mathtt{n}-1} \mathtt{a}[k] \wedge \mathtt{i} \geq -1 \rangle$

```
}
```

$\langle \mathtt{res} = \Pi_{k=\mathtt{i}+1}^{\mathtt{n}-1} \mathtt{a}[k] \wedge \mathtt{i} \geq -1 \wedge \neg(\mathtt{i} \geq 0) \rangle$
$\langle \mathtt{res} = \Pi_{k=0}^{\mathtt{n}-1} \mathtt{a}[k] \rangle$

b) Wir wählen als Variante $V = \mathtt{i}$. Hiermit lässt sich die Terminierung von P beweisen, denn für die einzige Schleife im Programm (mit Schleifenbedingung $B = \mathtt{i} \geq 0$) gilt:

- $B \Rightarrow V \geq 0$, denn $B \Leftrightarrow \mathtt{i} \geq 0 \Leftrightarrow V \geq 0$ und
- $\langle \mathtt{i} = m \wedge \mathtt{i} \geq 0 \rangle$
  $\langle \mathtt{i} - 1 < m \rangle$

```
res = res * a[i];
```

  $\langle \mathtt{i} - 1 < m \rangle$

```
i = i - 1;
```

  $\langle \mathtt{i} < m \rangle$