Prof. Dr. J. Giesl
J.-C. Kassing, M. Leven Rosarius, N. Lommen, E. Meyer

Übungsaufgabe 1 (Überblickswissen):

a) Was sagt ein sogenanntes Hoare-Tripel $\langle\varphi\rangle\,\mathtt{P}\,\langle\psi\rangle$ aus?

b) Was ist der Unterschied zwischen partieller und totaler Korrektheit?

c) Wie unterscheiden sich Referenzvariablen von Wertvariablen? Geben Sie ein kleines Java-Beispiel an, in dem dieser Unterschied deutlich wird.

Übungsaufgabe 2 (Verifikation):

Gegeben sei folgendes Java-Programm über den Integer-Variablen `x`, `y`, `z` und `r`. Hierbei steht div für die Integer-Division, d.h. 5 div 3 ergibt 1:

$\langle 0 \leq x \wedge 0 < y \rangle$ (Vorbedingung)

```
z = 0;
r = x;
while (r >= y) {
   r = r - y;
   z = z + 1;
}
```

$\langle z = x \text{ div } y \rangle$ (Nachbedingung)

Vervollständigen Sie die folgende Verifikation der partiellen Korrektheit des Algorithmus im Hoare-Kalkül, indem Sie die unterstrichenen Teile 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:

- Gehen Sie bei allen Aufgaben zum Hoare-Kalkül davon aus, dass keine Integer-Überläufe stattfinden, d.h., behandeln Sie Integers als die unendliche Menge $\mathbb{Z}$.
- 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.
- Es empfiehlt sich oft, bei der Erstellung der Zusicherungen in der Schleife von unten (d. h. von der Nachbedingung aus) vorzugehen.
- Geben Sie jeweils eine kurze Begründung an, warum die Konsequenzregeln korrekt angewandt wurden. D.h. beweisen Sie, dass aus der oberen Zusicherung die untere folgt, wenn diese direkt untereinander stehen.

$\langle 0 \leq \mathtt{x} \wedge 0 < \mathtt{y} \rangle$

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

```
z = 0;
```

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

```
r = x;
```

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

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

```
while (r >= y) {
```

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

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

```
    r = r - y;
```

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

```
    z = z + 1;
```

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

```
}
```

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

$\langle \mathtt{z} = \mathtt{x} \text{ div } \mathtt{y} \rangle$

Übungsaufgabe 4 (Verifikation):

Gegeben sei folgendes Java-Programm P über den Integer-Variablen `i`, `n` und `res`. Hierbei ist $\lceil x \rceil$ die kleinste Zahl $n \in \mathbb{Z}$, sodass $n \geq x$ gilt und $\lfloor x \rfloor$ ist die größte Zahl $n \in \mathbb{Z}$, sodass $n \leq x$ gilt.

$\langle 0 \leq \texttt{n} \rangle$ (Vorbedingung)

```
i = 0;
res = 0;
while (i < n) {
   if (i % 2 == 0) {
      res = res + n;
   } else {
      res = res - 1;
   }
   i = i + 1;
}
```

$\langle \texttt{res} = \lceil \frac{\texttt{n}}{2} \rceil \cdot \texttt{n} - \lfloor \frac{\texttt{n}}{2} \rfloor \rangle$ (Nachbedingung)

a) Vervollständigen Sie die folgende Verifikation der partiellen Korrektheit des Algorithmus im Hoare-Kalkül, indem Sie die unterstrichenen Teile 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.
- Es empfiehlt sich oft, bei der Erstellung der Zusicherungen in der Schleife von unten (d. h. von der Nachbedingung aus) vorzugehen.
- Geben Sie jeweils eine kurze Begründung an, warum die Konsequenzregeln korrekt angewandt wurden. D.h. beweisen Sie, dass aus der oberen Zusicherung die untere folgt, wenn diese direkt untereinander stehen.

$\langle 0 \leq \texttt{n} \rangle$

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

```
i = 0;
```

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

```
res = 0;
```

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

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

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

```
    if (i % 2 == 0) {
```

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

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

```
        res = res + n;
```

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

```
    } else {
```

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

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

```
        res = res - 1;
```

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

```
    }
```

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

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

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

```
}
```

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

$\langle \texttt{res} = \lceil \frac{\texttt{n}}{2} \rceil \cdot \texttt{n} - \lfloor \frac{\texttt{n}}{2} \rfloor \rangle$

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und mit Hilfe des Hoare-Kalküls die Terminierung unter der Voraussetzung $0 \leq \texttt{n}$ bewiesen werden.

Übungsaufgabe 6 (Programmierung):

a) In dieser Aufgabe geht es um Datenstrukturen in Java. Ziel ist es, eine Queue[1] mittels Arrays und einer einfachen Benutzerschnittstelle zu implementieren. Eine Queue ist eine Datenstruktur, welche eine Warteschlange von Objekten in einer festgelegten Reihenfolge darstellt. Mit `ENQUEUE` kann ein Element "hinten" zur Schlange hinzugefügt werden, mit `DEQUEUE` hingegen wird das "vorderste" Element entfernt.

Schreiben Sie ein Java-Programm, welches eine solche Queue darstellt. Ihre Queue soll ausschließlich `String`s als Elemente beinhalten. Initial soll hierbei die (nicht negative) Größe der Queue abgefragt werden und ein Array `queue` mit dieser Größe erstellt werden. Wurde eine negative Eingabe getätigt, dann soll die Größe erneut eingelesen werden. Nun soll immer wieder eine der folgenden Operationen eingelesen werden und die entsprechende Aktion ausgeführt werden, bis das Programm durch `STOP` beendet wird. Es empfiehlt sich, eine Variable `currentSize` zu verwenden, welche die aktuelle Anzahl der Elemente in der Queue speichert. Die Elemente sollen in einem Array so liegen, dass das Element "hinten" in der Schlange (also das Element, welches zuletzt hinzugefügt wurde) bei Position 0 liegt. Das "vorderste" Element der Schlange (also das Element, welches zuerst eingefügt wurde) liegt hingegen immer an Position (`currentSize` -1) im Array.

- `ENQUEUE`: Wenn im Array, welches die Queue repräsentiert, noch Platz ist (also `currentSize` $<$ `queue.length`), dann soll ein String eingelesen werden, alle Einträge im Array an Position i zur Position $i+1$ verschoben werden und der String an Position 0 im Array gespeichert werden. Ist das Array jedoch voll, dann soll eine geeignete Meldung ausgegeben werden. Das Array wird in diesem Fall nicht verändert.
- `DEQUEUE`: Wenn sich in der Queue mindestens ein Element befindet (also `currentSize` > 0), dann soll das Element entfernt werden, welches am längsten in der Queue lag. Die Variable `currentSize` soll also um 1 verringert werden. Befindet sich kein Element in der Queue, dann soll nichts passieren.
- `CLEAR`: Alle Elemente sollen gelöscht werden (es soll also danach `currentSize` $= 0$ gelten).
- `SETSIZE`: Zuerst soll erneut eine nicht negative neue Größe `size` eingelesen werden. Es soll dann ein neues Array der Größe `size` erstellt werden, die zuerst eingefügten $\min($`currentSize`$,$ `size`$)$-Elemente in das neue Array kopiert werden und schließlich das ursprüngliche Array `queue` durch das neue Array ersetzt werden.
- `PRINT`: Hier sollen die Elemente der Queue durch Kommata getrennt ausgegeben werden. Hierbei sollen die zuerst eingefügten Elemente (also die Elemente, welche am längsten in der Queue liegen) auch zuerst ausgegeben werden. Wenn die Queue leer ist, dann soll hingegen "Queue ist leer" ausgegeben werden.
- `STOP`: Hier sollen ebenfalls alle Elemente der Queue ausgegeben werden. Jedoch soll danach das Programm beendet werden.

Bei fehlerhaften Eingaben soll eine geeignete Fehlermeldung ausgegeben werden und danach eine neue Operation eingelesen werden. Für Ausgaben und Eingaben können Sie die aus der Vorlesung bekannte Klasse `IO` verwenden.

Ein Ablauf des Programms könnte z.B. so aussehen:

```
Bitte geben Sie die initiale (nicht negative) Groesse ein:
1
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
ENQUEUE
Geben Sie ein zu speicherndes Element ein:
EINS
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
ENQUEUE
Queue ist voll.
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
SETSIZE
Bitte geben Sie die (nicht negative) Groesse ein:
2
```

[1] https://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur)

```
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
ENQUEUE
Geben Sie ein zu speicherndes Element ein:
ZWEI
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
PRINT
Queue: EINS,ZWEI
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
DEQUEUE
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
PRINT
Queue: ZWEI
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
ENQUEUE
Geben Sie ein zu speicherndes Element ein:
DREI
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
SETSIZE
Bitte geben Sie die (nicht negative) Groesse ein:
1
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,SETSIZE,PRINT,STOP) ein:
STOP
Queue: ZWEI
```

Hinweise:

- Sie können `Math.min(x,y)` und `Math.max(x,y)` verwenden, um das Minimum bzw. Maximum von `x` und `y` zu berechnen.
- Im Moodle-Lernraum steht die Java-Datei `OurQueue.java` zum Download zur Verfügung. Vervollständigen Sie die Implementierung in dieser Datei!

b) In Java existieren für die meisten gängigen Datenstrukturen vorgefertigte Implementierungen. In dieser Teilaufgabe soll nun Ihre Lösung aus der vorherigen Teilaufgabe vereinfacht werden, indem Sie die bereits existierende Datenstruktur `Queue`[2] verwenden und Queues nicht selber mithilfe von Arrays implementieren. Insbesondere müssen Sie die Größe nicht mehr verändern oder speichern. Dies geschieht bereits intern in der Implementation der Java-Queues. Sie können eine `Queue` erstellen mit

`Queue<String> queue = new LinkedList<String>();`.

Benutzen Sie `queue.add(new_element);`, um den `String` `new_element` zur Queue `queue` hinzuzufügen. Mit `queue.poll();` entfernen Sie das älteste Element der Queue. Sie können `queue.clear();` verwenden, um die Queue zu leeren und `queue.toString();`, um eine geeignete `String`-Repräsentation der Queue zu erhalten. Dies ist auch bei leeren Queues möglich. Ein Ablauf des Programms könnte z.B. so aussehen:

```
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,PRINT,STOP) ein:
ENQUEUE
Geben Sie ein zu speicherndes Element ein:
EINS
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,PRINT,STOP) ein:
ENQUEUE
Geben Sie ein zu speicherndes Element ein:
ZWEI
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,PRINT,STOP) ein:
PRINT
Queue: [EINS,ZWEI]
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,PRINT,STOP) ein:
DEQUEUE
```

[2] `https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/Queue.html`

```
Bitte geben Sie eine Operation (ENQUEUE,DEQUEUE,CLEAR,PRINT,STOP) ein:
STOP
Queue: [ZWEI]
```

Hinweise:

- In Java werden `Queue`s mittels sogenannter `LinkedList`en implementiert. Diese ist wesentlich effizienter als unsere obige Implementierung mittels Arrays.
- Im Moodle-Lernraum steht die Java-Datei `JavaQueue.java` zum Download zur Verfügung. Vervollständigen Sie die Implementierung in dieser Datei!
- In dieser Teilaufgabe muss keine Operation `SETSIZE` implementiert werden.