Tutoriumslösung - Übung 2 (Abgabe bis Donnerstag, den 04.11.2021, um 18:00 Uhr)

Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, N. Lommen, D. Meier, F. Meyer

Tutoraufgabe 1 (Überblickswissen):

a) Wie unterscheidet sich `if ... else ...` von der (neuen) `switch`-Anweisung?

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

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

Lösung:

a) Es gibt einige Unterschiede. `Switch` erlaubt gewissermaßen nur sehr spezielle Bedingungen, nämlich die Prüfung von Gleichheit von einer Variable von einem speziellen Datentypen (hauptsächlich `int`, `char`, `String` und (später in der Vorlesung behandelten) Aufzählungstypen) mit ausgewählten Werten, während nach einem `if` jede Art von boolescher Bedingung geprüft werden kann, d.h. auch die Untersuchung ganz unterschiedlicher Variablen ist in jedem `else`-Level möglich. Weiterhin ist die neue `switch`-Anweisung auch als Ausdruck nutzbar. Auch bei der Übersichtlichkeit bei vielen Fällen liegt die `switch`-Anweisung vorne.

Eine `if`-Anweisung ist eine Fallunterscheidung mit nur 2 Fällen (für mehrere Fälle werden geschachtelte `if`-Anweisungen benötigt). Eine `switch`-Anweisung kann hingegen beliebig viele Fälle haben.

b) Wenn vor der Ausführung des Programms `P` die Vorbedingung φ gilt, so gilt die Nachbedingung ψ nach Ausführung von `P`, falls `P` terminiert. Insbesondere gilt $\langle\varphi\rangle\,\mathtt{P}\,\langle\psi\rangle$ für alle Zusicherungen φ, ψ, wenn `P` *nicht* terminiert.

c) Partielle Korrektheit garantiert Korrektheit für jedes Programm, das terminiert. Insbesondere ist also ein Programm, das niemals terminiert, immer partiell korrekt. Totale Korrektheit garantiert zusätzlich Terminierung.

Tutoraufgabe 2 (Programmierung):

In dieser Aufgabe geht es um die Ein- und Ausgabe in Java. Dafür soll die bereitgestellte Klasse `SimpleIO` genutzt werden. Um einen String `str1` in einem Fenster mit dem Titel[1] `str2` auszugeben, nutzen Sie `SimpleIO.output(str1, str2)`. Um einen Wert vom Typ `type` mit der Klasse `SimpleIO` einzulesen, nutzen Sie `SimpleIO.getType(str)`, wobei `str` der Text ist, der der*dem Benutzer*in im Eingabefenster angezeigt wird. Um einen Wert vom Typ `int` einzulesen, benutzen Sie also z.B.
`SimpleIO.getInt("Bitte eine ganze Zahl eingeben")`.

Schreiben Sie ein einfaches Java-Programm, welches den*die Benutzer*in auffordert, eine positive ganze Zahl (d. h. größer als 0) einzugeben. Danach soll das Programm die Zahl einlesen. Diese Eingabeaufforderung mit anschließendem Einlesen soll solange wiederholt werden, bis der*die Benutzer*in wirklich eine positive Zahl eingibt. Wenn die Eingabe keine Zahl ist, darf sich das Programm beliebig verhalten. Anschließend soll der*die Benutzer*in aufgefordert werden, ein Wort einzugeben. Das Wort soll eingelesen und schließlich so oft hintereinander geschrieben ausgeben werden, wie durch die eingegebene positive Zahl festgelegt wurde.
Ein Ablauf des Programms könnte z.B. so aussehen:

```
Bitte geben Sie eine positive Zahl ein
0
Bitte geben Sie eine positive Zahl ein
3
Bitte geben Sie ein Wort ein
Programmierung
ProgrammierungProgrammierungProgrammierung
```

[1] Sie dürfen in dieser Aufgabe immer z.B. “MultiEcho” als Titel verwenden.

Lösung:

```
/**
 * Programm zum Einlesen einer positiven Zahl und eines Worts, welches das Wort
 * anschliessend so oft ausgibt, wie durch die Zahl festgelegt wurde.
 */
public class Multiecho {
  public static void main(String[] args) {
    // Einlesen der Zahl mit Ueberpruefung, dass die Zahl positiv ist:
    int zahl = 0;
    while (zahl < 1) {
      zahl = SimpleIO.getInt("Bitte geben Sie eine positive Zahl ein");
    }
    // Einlesen des Wortes:
    String wort = SimpleIO.getString("Bitte geben Sie ein Wort ein");
    // Ausgabe des Wortes so oft wie durch die Zahl festgelegt wurde:
    int i = 0;
    String multi = "";
    while (i < zahl) {
      multi += wort;
      i++;
    }
    SimpleIO.output(multi, "MultiEcho");
  }
}
```

Tutoraufgabe 4 (Verifikation):

Gegeben sei folgendes Java-Programm über den Integer-Variablen `x`, `y`, `z` und `r`:

$\langle 0 \leq x \wedge 0 < y \rangle$ (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 davon aus, dass keine Integer-Überlaufe stattfinden, d.h., behandeln Sie Integers als die unendliche Menge $\mathbb{Z}$.
- div steht für die Integer-Division, d.h. 5 div 3 ergibt z.B. 1.
- 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.

Lösung:

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

```
z = 0;
```

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

```
r = x;
```

$\langle \mathtt{z} = 0 \wedge \mathtt{r} = \mathtt{x} \wedge 0 \leq \mathtt{x} \wedge 0 < \mathtt{y} \rangle$
$\langle \mathtt{x} \operatorname{div} \mathtt{y} = \mathtt{z} + \mathtt{r} \operatorname{div} \mathtt{y} \wedge 0 \leq \mathtt{r} \wedge 0 < \mathtt{y} \rangle$

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

$\langle \mathtt{x} \operatorname{div} \mathtt{y} = \mathtt{z} + \mathtt{r} \operatorname{div} \mathtt{y} \wedge 0 \leq \mathtt{r} \wedge 0 < \mathtt{y} \wedge \mathtt{r} \geq \mathtt{y} \rangle$
$\langle \mathtt{x} \operatorname{div} \mathtt{y} = \mathtt{z} + 1 + (\mathtt{r} - \mathtt{y}) \operatorname{div} \mathtt{y} \wedge 0 \leq \mathtt{r} - \mathtt{y} \wedge 0 < \mathtt{y} \rangle$

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

$\langle \mathtt{x} \operatorname{div} \mathtt{y} = \mathtt{z} + 1 + \mathtt{r} \operatorname{div} \mathtt{y} \wedge 0 \leq \mathtt{r} \wedge 0 < \mathtt{y} \rangle$

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

$\langle \mathtt{x} \operatorname{div} \mathtt{y} = \mathtt{z} + \mathtt{r} \operatorname{div} \mathtt{y} \wedge 0 \leq \mathtt{r} \wedge 0 < \mathtt{y} \rangle$

```
}
```

$\langle \mathtt{x} \operatorname{div} \mathtt{y} = \mathtt{z} + \mathtt{r} \operatorname{div} \mathtt{y} \wedge 0 \leq \mathtt{r} \wedge 0 < \mathtt{y} \wedge \mathtt{r} \not\geq \mathtt{y} \rangle$
$\langle \mathtt{z} = \mathtt{x} \operatorname{div} \mathtt{y} \rangle$

r < y => r div y = 0

x = 5 y = 2

=> x div y = 2

r	z	r div y
5	0	2
3	1	1
1	2	0

Tutoraufgabe 6 (Verifikation):

Gegeben sei folgendes Java-Programm P über den Integer-Variablen `n`, `i` und `res`:

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

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

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

a) Vervollständigen Sie die folgende Verifikation 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 davon aus, dass keine Integer-Überlaufe 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.
- $\lceil x \rceil$ ist die kleinste Zahl $n \in \mathbb{Z}$, sodass $n \geq x$ gilt. $\lfloor x \rfloor$ ist die größte Zahl $n \in \mathbb{Z}$, sodass $n \leq x$ gilt.

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 bewiesen werden.

Lösung:

a)

$\langle 0 \leq \mathtt{n} \rangle$
$\langle 0 = \lceil \frac{0}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{0}{2} \rfloor \wedge 0 \leq \mathtt{n} \rangle$

```
i = 0;
```

hochrunden runterrunden

$\langle 0 = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}}{2} \rfloor \wedge \mathtt{i} \leq \mathtt{n} \rangle$

```
res = 0;
```

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}}{2} \rfloor \wedge \mathtt{i} \leq \mathtt{n} \rangle$

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

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}}{2} \rfloor \wedge \mathtt{i} \leq \mathtt{n} \wedge \mathtt{i} < \mathtt{n} \rangle$

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

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}}{2} \rfloor \wedge \mathtt{i} \leq \mathtt{n} \wedge \mathtt{i} < \mathtt{n} \wedge \mathtt{i}\%2 = 0 \rangle$ i ist gerade
$\langle \mathtt{res} + \mathtt{n} = \lceil \frac{\mathtt{i}+1}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}+1}{2} \rfloor \wedge \mathtt{i} + 1 \leq \mathtt{n} \rangle$ (i + 1) / 2 hat Dezimalzahlen

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

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}+1}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}+1}{2} \rfloor \wedge \mathtt{i} + 1 \leq \mathtt{n} \rangle$ hochrunden: I (i + 1) / 2 I = I i / 2 I + 1
beim runterrunden passiert nichts

```
    } else {
```

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}}{2} \rfloor \wedge \mathtt{i} \leq \mathtt{n} \wedge \mathtt{i} < \mathtt{n} \wedge \neg(\mathtt{i}\%2 = 0) \rangle$
$\langle \mathtt{res} - 1 = \lceil \frac{\mathtt{i}+1}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}+1}{2} \rfloor \wedge \mathtt{i} + 1 \leq \mathtt{n} \rangle$

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

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}+1}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}+1}{2} \rfloor \wedge \mathtt{i} + 1 \leq \mathtt{n} \rangle$

```
    }
```

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}+1}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}+1}{2} \rfloor \wedge \mathtt{i} + 1 \leq \mathtt{n} \rangle$

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

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}}{2} \rfloor \wedge \mathtt{i} \leq \mathtt{n} \rangle$

```
}
```

$\langle \mathtt{res} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{i}}{2} \rfloor \wedge \mathtt{i} \leq \mathtt{n} \wedge \neg(\mathtt{i} < \mathtt{n}) \rangle$
$\langle \mathtt{res} = \lceil \frac{\mathtt{n}}{2} \rceil \cdot \mathtt{n} - \lfloor \frac{\mathtt{n}}{2} \rfloor \rangle$

Im "if-Fall" gilt $\mathtt{i}\%2 = 0 \implies \lceil \frac{\mathtt{i}+1}{2} \rceil \cdot \mathtt{n} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n} + \mathtt{n}$ sowie $\mathtt{i}\%2 = 0 \implies \lfloor \frac{\mathtt{i}+1}{2} \rfloor = \lfloor \frac{\mathtt{i}}{2} \rfloor$.
Im "else-Fall" gilt $\mathtt{i}\%2 \neq 0 \implies \lceil \frac{\mathtt{i}+1}{2} \rceil \cdot \mathtt{n} = \lceil \frac{\mathtt{i}}{2} \rceil \cdot \mathtt{n}$ sowie $\mathtt{i}\%2 \neq 0 \implies \lfloor \frac{\mathtt{i}+1}{2} \rfloor = \lfloor \frac{\mathtt{i}}{2} \rfloor + 1$.

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 $B = \mathtt{i} < \mathtt{n} \Rightarrow \mathtt{n} - \mathtt{i} \geq 0$ und
- die folgende Ableitung ist korrekt:

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

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

$\langle \mathtt{n} - \mathtt{i} = m \wedge \mathtt{i} < \mathtt{n} \wedge \mathtt{i}\%2 = 0 \rangle$
$\langle \mathtt{n} - \mathtt{i} = m \rangle$

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

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

```
} else {
```
$\langle \texttt{n} - \texttt{i} = m \wedge \texttt{i} < \texttt{n} \wedge \neg(\texttt{i}\%2 = 0) \rangle$
$\langle \texttt{n} - \texttt{i} = m \rangle$
```
  res = res - 1;
```
$\langle \texttt{n} - \texttt{i} = m \rangle$
```
}
```
$\langle \texttt{n} - \texttt{i} = m \rangle$
$\langle \texttt{n} - (\texttt{i} + 1) + 1 = m \rangle$
```
i = i + 1;
```
$\langle \texttt{n} - \texttt{i} + 1 = m \rangle$
$\langle \texttt{n} - \texttt{i} < m \rangle$