Prof. Dr. M. Müller C. Aschermann, J. Hensel, J. Protze, P. Reble

Tutoraufgabe 1 (Verifikation):

Gegeben sei folgendes Java-Programm P über den Integer-Variablen `x`, `y`, `c` und `res`:

$\langle\, \texttt{y} \geq 0 \,\rangle$ (Vorbedingung)

```
res = 1;
c = y;
while (c > 0) {
    res = res * x;
    c = c - 1;
}
```

$\langle\, \texttt{res} = \texttt{x}^{\texttt{y}} \,\rangle$ (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:

- 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.

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 $\texttt{y} \geq 0$ bewiesen werden.

Lösung:

a)

$\langle \texttt{y} \geq 0 \rangle$
$\langle \texttt{y} \geq 0 \wedge 1 = 1 \wedge \texttt{y} = \texttt{y} \rangle$

```
res = 1;
```

$\langle \texttt{y} \geq 0 \wedge \texttt{res} = 1 \wedge \texttt{y} = \texttt{y} \rangle$

```
c = y;
```

$\langle \texttt{y} \geq 0 \wedge \texttt{res} = 1 \wedge \texttt{c} = \texttt{y} \rangle$
$\langle \texttt{x}^{\texttt{y}} = \texttt{res} * \texttt{x}^{\texttt{c}} \wedge \texttt{c} \geq 0 \rangle$

```
while (c > 0) {
```

$\langle \texttt{x}^{\texttt{y}} = \texttt{res} * \texttt{x}^{\texttt{c}} \wedge \texttt{c} \geq 0 \wedge \texttt{c} > 0 \rangle$
$\langle \texttt{x}^{\texttt{y}} = \texttt{res} * \texttt{x} * \texttt{x}^{\texttt{c}-1} \wedge \texttt{c} - 1 \geq 0 \rangle$

```
    res = res * x;
```

$\langle \texttt{x}^{\texttt{y}} = \texttt{res} * \texttt{x}^{\texttt{c}-1} \wedge \texttt{c} - 1 \geq 0 \rangle$

```
    c = c - 1;
```

$\langle \texttt{x}^{\texttt{y}} = \texttt{res} * \texttt{x}^{\texttt{c}} \wedge \texttt{c} \geq 0 \rangle$

```
}
```

$\langle \texttt{x}^{\texttt{y}} = \texttt{res} * \texttt{x}^{\texttt{c}} \wedge \texttt{c} \geq 0 \wedge \neg\texttt{c} > 0 \rangle$
$\langle \texttt{res} = \texttt{x}^{\texttt{y}} \rangle$

Alernative Invariante: $\texttt{res} = \texttt{x}^{(\texttt{y}-\texttt{c})}$.

b) Eine gültige Variante für die Terminierung ist $V = \texttt{c}$, denn die Schleifenbedingung $B = \texttt{c} > 0$ impliziert $\texttt{c} \geq 0$ und es gilt:

$\langle \texttt{c} = m \wedge \texttt{c} > 0 \rangle$
$\langle \texttt{c} - 1 < m \rangle$
```
res = res * x;
```
$\langle \texttt{c} - 1 < m \rangle$
```
c = c - 1;
```
$\langle \texttt{c} < m \rangle$

Damit ist die Terminierung der einzigen Schleife in P gezeigt.

Aufgabe 2 (Verifikation): (7 + 2 = 9 Punkte)

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

$\langle \varphi \rangle$ (Vorbedingung)
```
res = 0;
i = 1;
while (i <= n) {
   res = res + 2*i;
   i = i + 1;
}
```
$\langle \psi \rangle$ (Nachbedingung)

a) Als Vorbedingung φ für das oben aufgeführte Programm P gelte $\texttt{n} > 0$ und als Nachbedingung ψ gelte $\texttt{res} = \texttt{n}^2 + \texttt{n}$. 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:

- 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.

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)

$\langle \texttt{n} \geq 0 \rangle$
$\langle 0 = 0 \wedge 1 = 1 \wedge \texttt{n} \geq 0 \rangle$
```
res = 0;
```
$\langle \texttt{res} = 0 \wedge 1 = 1 \wedge \texttt{n} \geq 0 \rangle$
```
i = 1;
```
$\langle \texttt{res} = 0 \wedge \texttt{i} = 1 \wedge \texttt{n} \geq 0 \rangle$
$\langle \texttt{res} = (\texttt{i} - 1)^2 + (\texttt{i} - 1) \wedge \texttt{i} - 1 \leq \texttt{n} \wedge \texttt{n} \geq 0 \rangle$
```
while (i <= n) {
```

$\langle \texttt{res} = (\texttt{i}-1)^2 + (\texttt{i}-1) \wedge \texttt{i}-1 \leq \texttt{n} \wedge \texttt{n} \geq 0 \wedge \texttt{i}-1 \leq \texttt{n} \wedge \texttt{i} \leq \texttt{n} \rangle$
$\langle \texttt{res} + 2*i = (\texttt{i}+1-1)^2 + (\texttt{i}+1-1) \wedge \texttt{i}+1-1 \leq \texttt{n} \wedge \texttt{n} \geq 0 \wedge \texttt{i}+1-1 \leq \texttt{n} \rangle$

```
    res = res + 2*i;
```

$\langle \texttt{res} = (\texttt{i}+1-1)^2 + (\texttt{i}+1-1) \wedge \texttt{i}+1-1 \leq \texttt{n} \wedge \texttt{n} \geq 0 \rangle$

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

$\langle \texttt{res} = (\texttt{i}-1)^2 + (\texttt{i}-1) \wedge \texttt{i}-1 \leq \texttt{n} \wedge \texttt{n} \geq 0 \rangle$

```
}
```

$\langle \texttt{res} = (\texttt{i}-1)^2 + (\texttt{i}-1) \wedge \texttt{i}-1 \leq \texttt{n} \wedge \texttt{n} \geq 0 \wedge \texttt{i} \not\leq \texttt{n} \rangle$
$\langle \texttt{res} = \texttt{n}^2 + \texttt{n} \rangle$

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

- $B \Rightarrow V \geq 0$, denn $B \Leftrightarrow \texttt{i} \leq \texttt{n} \Leftrightarrow \texttt{n} - \texttt{i} \geq 0 \Leftrightarrow V \geq 0$ und
- die folgende Ableitung ist korrekt:

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

```
    res = res + 2*i;
```

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

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

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

Aufgabe 3 (Verifikation): (7 Punkte)

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

$\langle \varphi \rangle$ (Vorbedingung)

```
  i = 0;
  res = 0;
  while (i < n) {
     j = 0
     while (j < n) {
         res = res + 1;
         j = j + 1;
     }
     i = i + 1;
  }
```

$\langle \psi \rangle$ (Nachbedingung)

a) Als Vorbedingung φ für das oben aufgeführte Programm P gelte $\texttt{n} \geq 0$ und als Nachbedingung ψ gelte $\texttt{res} = \texttt{n}^2$. 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:

- 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.

```
i = 0;
res = 0;
while (i < n) {
    j = 0
    while (j < n) {
        res = res+1
        j = j+1
    }
    i = i + 1
}
```
$\langle \texttt{res} = \texttt{n}^2 \rangle$

Lösung:

a)
$\langle \texttt{n} \geq 0 \wedge 0 = 0 \wedge 0 = 0 \rangle$
```
i = 0;
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} = 0 \wedge 0 = 0 \rangle$
```
res = 0;
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} = 0 \wedge \texttt{res} = 0 \rangle$
$\langle \texttt{n} \geq 0 \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} \rangle$
```
while (i < n) {
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} \wedge \texttt{i} < \texttt{n} \rangle$
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} \wedge \texttt{i} \wedge 0 = 0 \rangle$
```
    j = 0
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} \wedge \texttt{j} = 0 \rangle$
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} + \texttt{j} \wedge \texttt{j} \leq \texttt{n} \rangle$
```
    while (j < n) {
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} + \texttt{j} \wedge \texttt{j} \leq \texttt{n} \wedge \texttt{j} < \texttt{n} \rangle$
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} + 1 = \texttt{n} * \texttt{i} + \texttt{j} + 1 \wedge \texttt{j} + 1 \leq \texttt{n} \rangle$
```
        res = res+1
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} + \texttt{j} + 1 \wedge \texttt{j} + 1 \leq \texttt{n} \rangle$
```
        j = j+1
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} + \texttt{j} \wedge \texttt{j} \leq \texttt{n} \rangle$
```
    }
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} < \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} + \texttt{j} \wedge \texttt{j} \leq \texttt{n} \wedge \neg \texttt{j} < \texttt{n} \rangle$
$\langle \texttt{n} \geq 0 \wedge \texttt{i} + 1 \leq \texttt{n} \wedge \texttt{res} = \texttt{n} * (\texttt{i} + 1) \rangle$
```
    i = i + 1
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} \rangle$
```
}
```
$\langle \texttt{n} \geq 0 \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{res} = \texttt{n} * \texttt{i} \wedge \neg \texttt{i} < \texttt{n} \rangle$
$\langle \texttt{res} = \texttt{n}^2 \rangle$

Tutoraufgabe 4 (Zählen):

In dieser Aufgabe soll eine Methode implementiert werden, die zählt, wie oft das Zeichen `'a'` in einem String enthalten ist. Verwenden Sie dafür nur eine `for`-Schleife und eine `if`-Abfrage.

Listing 1: CountA.java

```
1 class CountA {
2   static int count_a(String str){
```

```
3       // Implement Me
4     }
5
6     public static void main(String[] args){
7       assert(count_a("") == 0);
8       assert(count_a("a") == 1);
9       assert(count_a("aa") == 2);
10      assert(count_a("abb") == 1);
11      assert(count_a("abba") == 2);
12      assert(count_a("bb") == 0);
13      System.out.println("Everything good!");
14    }
15  }
```

Hinweise:

- Sie dürfen die Methoden `length()` und `charAt(int i)` der Klasse `String` verwenden.
- Benutzen Sie das gegebene Gerüst, um ihre Methode zu testen.
- Die Methode muss auf allen Eingaben das richtige Ergebnis liefern, nicht nur auf den gegebenen Beispielen.

Lösung:

Listing 2: CountA.java

```
1   class CountA {
2     static int count_a(String str){
3       int count = 0;
4       for(int i=0; i < str.length(); i++){
5         if(str.charAt(i) == 'a'){
6           count += 1;
7         }
8       }
9       return count;
10    }
11
12    public static void main(String[] args){
13      assert(count_a("") == 0);
14      assert(count_a("a") == 1);
15      assert(count_a("aa") == 2);
16      assert(count_a("abb") == 1);
17      assert(count_a("abba") == 2);
18      assert(count_a("bb") == 0);
19      System.out.println("Everything good!");
20    }
21  }
```

Aufgabe 5 (Palindrom): (3 Punkte)

Implementieren Sie eine Methode, die überprüft, ob ein String ein Palindrom ist. Ein String ist ein Palindrom, falls er vorwärts und rückwärts gelesen das gleiche ergibt. Verwenden Sie dafür nur eine `for`-Schleife und eine `if`-Abfrage.

Listing 3: Palindrom.java

```
1  class Palindrom {
2    static boolean is_palindrom(String str){
3      // Implement Me
4    }
5
6    public static void main(String[] args){
7      assert(is_palindrom(""));
8      assert(is_palindrom("a"));
9      assert(is_palindrom("aa"));
10     assert(is_palindrom("aba"));
11     assert(!is_palindrom("abab"));
12     assert(!is_palindrom("abb"));
13     System.out.println("Everything good!");
14   }
15 }
```

Hinweise:

- Sie dürfen die Methoden `length()` und `charAt(int i)` der Klasse `String` verwenden.
- Benutzen Sie das gegebene Gerüst, um ihre Methode zu testen.
- Die Methode muss auf allen Eingaben das richtige Ergebnis liefern, nicht nur auf den gegebenen Beispielen.

Lösung:

Listing 4: Palindrom.java

```
1  class Palindrom {
2    static boolean is_palindrom(String str){
3      for(int i=0; i < str.length(); i++){
4        if(str.charAt(i) != str.charAt(str.length()-i-1)){
5          return false;
6        }
7      }
8      return true;
9    }
10
11   public static void main(String[] args){
12     assert(is_palindrom(""));
13     assert(is_palindrom("a"));
14     assert(is_palindrom("aa"));
15     assert(is_palindrom("aba"));
16     assert(!is_palindrom("abab"));
17     assert(!is_palindrom("abb"));
18     System.out.println("Everything good!");
19   }
20 }
```