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

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Montag, den 16.11.2015 um 15:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- In einigen Aufgaben müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** schicken Sie sie per **E-Mail** vor Montag, dem 16.11.2015 um 15:00 Uhr an Ihre Tutorin/Ihren Tutor.
  Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird, ansonsten werden keine Punkte vergeben.

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.

$\langle \texttt{y} \geq 0 \rangle$

$\langle$ _____ $\rangle$

```
res = 1;
```

$\langle$ _____ $\rangle$

```
c = y;
```

$\langle$_____$\rangle$

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

```
}
```

$\langle$_____$\rangle$

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

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

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.

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

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

```
res = 0;
```

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

```
i = 1;
```

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

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

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

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

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

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

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

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

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

```
}
```

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

$\langle \texttt{res} = \texttt{n}^2 + \texttt{n} \rangle$

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 bewiesen werden.

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.

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

$\langle$ _____ $\rangle$

```
i = 0;
```

$\langle$ _____ $\rangle$

```
res = 0;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

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

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
    j = 0
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
    while (j < n) {
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
        res = res+1
```

$\langle$ _____ $\rangle$

```
        j = j+1
```

```
                    ⟨_____________________________________________⟩
        }

                    ⟨_____________________________________________⟩

                    ⟨_____________________________________________⟩
        i = i + 1

                    ⟨_____________________________________________⟩
    }

                    ⟨_____________________________________________⟩
                    ⟨res = n²⟩
```

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.

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 2: 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.