Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, D. Meier

Aufgabe 3 (Verifikation): (10 Punkte)

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

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

```
res = -x;
c = x;
while (c > 0) {
   res = res + 2 * c;
   c = c - 1;
}
```

$\langle \texttt{res} = \texttt{x} \cdot \texttt{x} \rangle$ (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:

- Sie können die Gauß'sche Summenformel benutzen:
  Für eine natürliche Zahl $\texttt{x}$ gilt $\texttt{0} + \texttt{1} + \cdots + \texttt{x} = \sum_{\texttt{k}=\texttt{0}}^{\texttt{x}} \texttt{k} = \frac{\texttt{x} \cdot (\texttt{x}+\texttt{1})}{2}$. Somit ist also $\texttt{x}^2 = \texttt{2} \cdot \sum_{\texttt{k}=\texttt{0}}^{\texttt{x}} \texttt{k} - \texttt{x}$.
- Die leere Summe (bei der der Startindex größer als der Zielindex ist) wird als $\texttt{0}$ interpretiert. Beispielsweise gilt also $\sum_{\texttt{k}=\texttt{x}+\texttt{1}}^{\texttt{x}} \texttt{k} = \texttt{0}$.
- 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.

$\langle x \geq 0 \rangle$

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

```
res = -x;
```

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

```
c = x;
```

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

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

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

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

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

```
    res = res + 2 * c;
```

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

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

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

```
}
```

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

$\langle res = x \cdot x \rangle$

Aufgabe 5 (Verifikation): (9 + 3 = 12 Punkte)

Gegeben sei folgendes Java-Programm P über den Integer-Variablen `p`, `q`, `y`, `x` und `n`:

$\langle \texttt{q} \geq 1 \rangle$ (Vorbedingung)

```
n = q;
x = p;
y = 1;
while (n > 1) {
   if (n % 2 == 0) {
      x = x * x;
      n = n / 2;
   } else {
      y = x * y;
      x = x * x;
      n = (n-1) / 2;
   }
}
x = x * y;
```

$\langle \texttt{p}^{\texttt{q}} = \texttt{x} \rangle$ (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.
- Es empfiehlt sich oft, bei der Erstellung der Zusicherungen in der Schleife von unten (d. h. von der Nachbedingung aus) vorzugehen.

$\langle \texttt{q} \geq 1 \rangle$

$\langle$ _____ $\rangle$

```
n = q;
```

$\langle$ _____ $\rangle$

```
x = p;
```

$\langle$ _____ $\rangle$

```
y = 1;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
while (n > 1) {
```

$\langle$ _____ $\rangle$

```
if (n % 2 == 0) {
```
⟨_____⟩
⟨_____⟩
```
  x = x * x;
```
⟨_____⟩
```
  n = n / 2;
```
⟨_____⟩
```
} else {
```
⟨_____⟩
⟨_____⟩
```
  y = x * y;
```
⟨_____⟩
```
  x = x * x;
```
⟨_____⟩
```
  n = (n-1) / 2;
```
⟨_____⟩
```
}
```
⟨_____⟩
```
}
```
⟨_____⟩
⟨_____⟩
```
x = x * y;
```
⟨$p^q = x$⟩

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 $q \geq 1$ bewiesen werden.

Aufgabe 7 (Programmierung): (14 Punkte)

Bubblesort ist ein Algorithmus zum Sortieren von Arrays, der wie folgt vorgeht, um ein Array `a` zu sortieren: Das Array wird wiederholt von links nach rechts durchlaufen. Am Ende des n-ten Durchlauf gilt, dass die letzten n Array-Elemente an ihrer endgültigen Position stehen. Folglich müssen im $(n+1)$-ten Durchlauf nur noch die ersten `a.length`$-n$ Elemente betrachtet werden. In jedem Durchlauf wird in jedem Schritt das aktuelle Element mit seinem rechten Nachbarn verglichen. Falls das aktuelle Element größer ist als sein rechter Nachbar, werden sie getauscht.
Als Beispiel betrachten wir das Array `{3,2,1}`. Im ersten Durchlauf wird erst `3` mit `2` getauscht (dies ergibt `{2,3,1}`) und dann `3` mit `1`, was `{2,1,3}` ergibt. Im zweiten Durchlauf wird `2` mit `1` getauscht, was zu `{1,2,3}` führt.
Implementieren Sie eine Klasse `BubbleSort` mit einer Methode `public static void sort(int[] a)`, die das Array `a` mithilfe des Algorithmus *Bubblesort* aufsteigend sortiert.

Hinweise:

- Im Moodle stehen die Java-Dateien `Sort.java` und `Bubble.java` zum Download zur Verfügung. Speichern Sie beide Dateien in einem neuen Ordner. Die Klasse `BubbleSort` enthält eine Methode `sort` mit leerem Rumpf. Wenn Sie die Implementierung vervollständigen und anschließend mit `javac Sort.java` compilieren, dann können Sie Ihre Implementierung mit `java Sort` testen.

Aufgabe 9 (Verifikation mit Arrays): (10 + 4 = 14 Punkte)

Gegeben sei folgendes Java-Programm P:

$\langle$ `true` $\rangle$ (Vorbedingung)

```
i = 0;
res = false;
while(i < a.length) {
  if(x == a[i]) {
    res = true;
  }
  i = i + 1;
}
```

$\langle$ `res` $=$ `x` $\in \{$`a`$[j] \mid 0 \leq j \leq$ `a.length`$-1\}$ $\rangle$ (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.

Beachten Sie bei der Anwendung der "Bedingungsregel 1" mit Vorbedingung φ und Nachbedingung ψ, dass $\varphi \wedge \neg B \implies \psi$ gelten muss. D. h. die Nachbedingung ψ der `if`-Anweisung muss aus der Vorbedingung φ der `if`-Anweisung und der negierten Bedingung $\neg B$ folgen. Geben Sie beim Verwenden der Regel einen entsprechenden Beweis an.

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.
- Der Ausdruck `x` $\in$ `M` hat den Wert `true`, wenn `x` in der Menge `M` enthalten ist, sonst hat der Ausdruck den Wert `false`.

$\langle$`true`$\rangle$

$\langle$_____$\rangle$

```
i = 0;
```

$\langle$_____$\rangle$

```
res = false;
```

$\langle$_____$\rangle$

$\langle$_____$\rangle$

```
while (i < a.length) {
```

$\langle$_____$\rangle$

```
    if (x == a[i]) {
```

$\langle$_____$\rangle$

$\langle$_____$\rangle$

```
        res = true;
```

$\langle$_____$\rangle$

```
    }
```

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

```
}
```

$\langle$_____$\rangle$

$\langle$`res` $=$ `x` $\in \{$`a`$[j] \mid 0 \leq j \leq$ `a.length` $-1\}\rangle$

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und unter Verwendung des Hoare-Kalküls die Terminierung bewiesen werden.

Geben Sie auch bei dieser Teilaufgabe einen Beweis für die Aussage $\varphi \wedge \neg B \implies \psi$ bei der Anwendung der "Bedingungsregel 1" an.

Aufgabe 10 (Deck 2): (Codescape)

Lösen Sie die Missionen von Deck 2 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn sie Ihre Lösung vor der einheitlichen Codescape Deadline am Samstag, den 16.01.2021, um 23:59 Uhr abschicken.