185.A91 Einführung in die Programmierung 1 (VU 4,0) 2022W

Zur LVA in TISS

[redacted]

n	Freitag, 13. Januar 2023, 21:19
s	Beendet
n	Freitag, 13. Januar 2023, 21:59
t	40 Minuten
g	**16,00** von 20,00 (**80**%)

Sie haben mehrere Code-Fragmente gegeben. Jedes Code-Fragment beinhaltet eine Deklaration mit einer Zuweisung und einen Kommentar, der beschreibt, was direkt nach der gegebenen Code-Zeile ohne weitere Initialisierungsschritte möglich ist. Beurteilen Sie die Richtigkeit der Kommentare!

Richtig	Falsch	
○ ✗	● ✓	`long[][][] result = new long[3][3][1];` `// Im Array result können 30 ganze Zahlen gespeichert werden.`
● ✓	○ ✗	`long[][][][] result = new long[2][1][1][1];` `// Im Array result können 2 ganze Zahlen gespeichert werden.`
○ ✗	● ✓	`double[][][] result = new double[2][1][2];` `// Im Array result können 5 Gleitkommazahlen gespeichert werden.`
● ✓	○ ✗	`char[][][] result = new char[10][1][1];` `// Im Array result können 10 Zeichen gespeichert werden.`

```
long[][][] result = new long[3][3][1];
// Im Array result können 30 ganze Zahlen gespeichert werden.
```

: Falsch

```
long[][][][] result = new long[2][1][1][1];
// Im Array result können 2 ganze Zahlen gespeichert werden.
```

: Richtig

```
double[][][] result = new double[2][1][2];
// Im Array result können 5 Gleitkommazahlen gespeichert werden.
```

: Falsch

```
char[][][] result = new char[10][1][1];
// Im Array result können 10 Zeichen gespeichert werden.
```

Richtig

Sie haben mehrere Code-Fragmente gegeben. Jedes Code-Fragment beinhaltet einen Kommentar, der den Inhalt des Arrays `data` nach der Ausführung des Code-Fragments beschreiben soll. Beurteilen Sie die Richtigkeit der Kommentare für beliebige Werte von 1 < n < 1000.

Hinweis: Mit [a, b] wird das Intervall von a (inklusive) bis b (inklusive) beschrieben.

Richtig	Falsch	
● ☑	○ ✗	✓
○ ✗	● ☑	✓
● ☑	○ ✗	✓
● ☑	○ ✗	✓

```
int[] data = new int[n];
int i = n / 2;
while (data[i] == 0) {
    data[i] = i % n;
    i = (i + 1) % n;
}
//  data enthält nur Werte aus dem Intervall [0, n - 1]
```

```
int[] data = new int[n];
for (int i = 0; i < data.length - 1; i++) {
    data[i] = i + 1;
}
// data enthält nur Werte aus dem Intervall [1, n - 1]
```

```
int[] data = new int[n];
for (int i = data.length - 1; i > 0; i--) {
    data[i] = i - 1;
}
// data enthält keine negativen Werte
```

```
int[] data = new int[n];
int i = n / 2;
while (i != 0) {
    data[i] = i;
    i = (i + 1) % n;
}
// data enthält nur Werte >= 0
```

```
int[] data = new int[n];
```

```
int i = n / 2;
while (data[i] == 0) {
    data[i] = i % n;
    i = (i + 1) % n;
}
//  data enthält nur Werte aus dem Intervall [0, n - 1]
```

:

Richtig

```
int[] data = new int[n];
for (int i = 0; i < data.length - 1; i++) {
    data[i] = i + 1;
}
// data enthält nur Werte aus dem Intervall [1, n - 1]
```

: Falsch

```
int[] data = new int[n];
for (int i = data.length - 1; i > 0; i--) {
    data[i] = i - 1;
}
// data enthält keine negativen Werte
```

: Richtig

```
int[] data = new int[n];
int i = n / 2;
while (i != 0) {
    data[i] = i;
    i = (i + 1) % n;
}
// data enthält nur Werte >= 0
```

: Richtig

Sie haben folgende Methode gegeben:

```
private static int[][] generate(int n) {
    int[][] result = new int[n][];
    for (int i = 0; i <= n / 2; i++) {
        int[] row = new int[i + 1];
        for (int j = 0; j < row.length; j++) {
            row[j] = i + j;
        }
        result[i] = row;
    }
    for (int i = n / 2 + 1; i < n; i++) {
        int[] row = new int[n - i];
        for (int j = 0; j < row.length; j++) {
            row[j] = i - j;
        }
        result[i] = row;
    }
    return result;
}
```

Vorbedingung: 0 < n < 100

Beurteilen Sie die Richtigkeit der folgenden Aussagen!

Richtig **Falsch**

		Der größte Wert im Array entspricht immer n - 1.	✔
		Die Werte innerhalb jeder Zeile werden aufsteigend sortiert.	✖
		Im erzeugten Array kommen keine negativen Werte vor.	✔
		Das erzeugte Array hat immer n Spalten.	✔

Der größte Wert im Array entspricht immer n - 1.: Falsch
Die Werte innerhalb jeder Zeile werden aufsteigend sortiert.: Falsch
Im erzeugten Array kommen keine negativen Werte vor.: Richtig
Das erzeugte Array hat immer n Spalten.: Falsch

Die Methode `mirror(String s)` liefert einen String zurück, der sich aus `s` gefolgt von der Umkehrung von `s` zusammensetzt.

Beispiele für korrektes Verhalten:

```
mirror("") liefert "" zurück
mirror("gut") liefert "guttug" zurück
mirror("Hund") liefert "HunddnuH" zurück
```

Die Vorbedingung lautet `s != null`.

Sie haben unterschiedliche rekursive Implementierungen gegeben: Beurteilen Sie die Richtigkeit dieser Implementierungen bezüglich Rückgabe und Termination.

Richtig	Falsch	
		`private static String mirror(String s) { if (s.isEmpty()) { return ""; } char last = s.charAt(s.length() - 1); String out = mirror(s.substring(0, s.length() - 1)); return out.substring(0, out.length() / 2) + last + last + out.s } `
		`private static String mirror(String s) { if (s.length() <= 1) { return s + s; } return s.charAt(0) + mirror(s.substring(1)) + mirror(s.substrir } `

```java
private static String mirror(String s) {
    if (s.isEmpty()) {
        return s;
    }
    return s.charAt(0) + mirror(s.substring(1)) + s.charAt(0);
}
```

```java
private static String mirror(String s) {
    if (s.isEmpty()) {
        return "";
    }
    char last = s.charAt(s.length() - 1);
    String out = mirror(s.substring(0, s.length() - 1));
    return out + last + last + out;
}
```

```java
private static String mirror(String s) {
    if (s.isEmpty()) {
        return "";
    }
    char last = s.charAt(s.length() - 1);
    String out = mirror(s.substring(0, s.length() - 1));
    return out.substring(0, out.length() / 2) + last + last + out.substring(out.le
}
```

: Richtig

```java
private static String mirror(String s) {
    if (s.length() <= 1) {
        return s + s;
    }
    return s.charAt(0) + mirror(s.substring(1)) + mirror(s.substring(1)) + s.charA
}
```

: Falsch

```java
private static String mirror(String s) {
    if (s.isEmpty()) {
        return s;
    }
    return s.charAt(0) + mirror(s.substring(1)) + s.charAt(0);
}
```

: Richtig

```java
private static String mirror(String s) {
    if (s.isEmpty()) {
        return "";
    }
    char last = s.charAt(s.length() - 1);
    String out = mirror(s.substring(0, s.length() - 1));
    return out + last + last + out;
}
```

Falsch

Sie haben folgende Methode gegeben.

```
private static boolean match(String s, char c, int n) {
    if (s.isEmpty()) {
        return true;
    }
    if (s.charAt(0) != c) {
        return false;
    }
    return s.length() <= n || match(s.substring(n), c, n);      // Zeile A
}
```

Die Vorbedingung lautet: `s != null`, `n > 0`.

Beurteilen Sie die Richtigkeit der folgenden Aussagen.

Richtig	Falsch		
● ☑	○ ✗	Die Methode gibt `true` zurück, wenn der String `s` das Zeichen `c` an den Stellen enthält, deren Indizes durch `n` teilbar sind.	✔
● ☑	○ ✗	Ist `n` gleich 1, gibt die Methode `true` zurück, falls alle Zeichen in `s` gleich `c` sind.	✔
● ☑	○ ✗	Der Aufruf `match("AAAAAAAAAAAAAAAAAAAA", 'A', n)` gibt für jeden gültigen Wert von `n` den Wert `true` zurück.	✔
○ ✗	● ☑	Eine äquivalente Methode kann erstellt werden, indem die beiden If-Verzweigungen durch die Anweisung `if (s.length() <= 1) return s.charAt(0) == n;` ersetzt werden.	✔

Die Methode gibt `true` zurück, wenn der String `s` das Zeichen `c` an den Stellen enthält, deren Indizes durch `n` teilbar sind.: Richtig
Ist `n` gleich 1, gibt die Methode `true` zurück, falls alle Zeichen in `s` gleich `c` sind.: Richtig
Der Aufruf `match("AAAAAAAAAAAAAAAAAAAA", 'A', n)` gibt für jeden gültigen Wert von `n` den Wert `true` zurück.: Richtig
Eine äquivalente Methode kann erstellt werden, indem die beiden If-Verzweigungen durch die Anweisung `if (s.length() <= 1) return s.charAt(0) == n;` ersetzt werden.: Falsch

Sie haben folgende Methode gegeben.

```
private static boolean check(int[] a, int start, int end) {
    if (end <= start) {      // Zeile A
        return true;
    }
    int mid = start + (end - start) / 2;
    boolean b1 = check(a, start, mid);
    boolean b2 = check(a, mid + 1, end);
    return b1 && a[mid] <= a[mid + 1] && b2;     // Zeile B
}
```

Die Vorbedingung lautet: `a != null`, `a.length > 0`, `start <= end`, `start` und `end` sind gültige Indizes von `a`.

Beurteilen Sie die Richtigkeit der folgenden Aussagen.

Richtig	Falsch		
● ☑	○ ✗	Der Rückgabewert wird am Return-Pfad erzeugt.	✔
○ ✗	● ☑	Der Aufruf `check(new int[]{1, 3, 6, 8, 10}, start, end)` kann für bestimmte Werte von `start` und `end` den Wert `false` zurückgeben.	✔
● ✗	○ ☑	Die Methode liefert nur dann `true` zurück, falls das Array `a` vollständig aufsteigend sortiert ist, ansonsten `false`.	✗
○ ✗	● ☑	Die Methode funktioniert gleich, falls die Bedingung in Zeile A durch `end < start` ersetzt wird.	✔

Der Rückgabewert wird am Return-Pfad erzeugt.: Richtig
Der Aufruf `check(new int[]{1, 3, 6, 8, 10}, start, end)` kann für bestimmte Werte von `start` und `end` den Wert `false` zurückgeben.: Falsch
Die Methode liefert nur dann `true` zurück, falls das Array `a` vollständig aufsteigend sortiert ist, ansonsten `false`.: Falsch
Die Methode funktioniert gleich, falls die Bedingung in Zeile A durch `end < start` ersetzt wird.: Falsch

Sie haben mehrere Code-Fragmente mit dazugehörigen Kommentaren gegeben. Jeder Kommentar beschreibt die kleinste obere Schranke für die Laufzeit in Abhängigkeit von n. Dabei steht "n^x" für n^x. Beurteilen Sie die Richtigkeit der Kommentare!

Richtig	Falsch		
○ ✗	● ☑		✓

```
// Die Laufzeit liegt in O(n)
int result = 0;
int i = 0, j = 1;
while (i < n) {
    i++;
    while (j < n - i) {
        j++;
        result += i + j;
    }
    j = 1;
}
```

○ ☑	● ✗		✗

```
// Die Laufzeit liegt in O(n)
int result = 0;
int i = 0, j = 0;
while (i < n) {
    i++;
    while (j < n) {
        j++;
        result += i + j;
    }
}
```

● ☑	○ ✗		✓

```
// Die Laufzeit liegt in O(n^2)
int result = 0;
int i = 0, j = 0;
while (i < n) {
    i++;
    while (j < n) {
        j++;
        result += i + j;
    }
    j = 0;
}
```

○ ✗	● ☑		✓

```
// Die Laufzeit liegt in O(n^2)
int result = 0;
int i = 0, j = 0, k = 0;
while (i < n / 2) {
    i++;
    while (j < n / 4) {
        j++;
        while (k < n / 8) {
            k++;
            result += i + j + k;
        }
        k = j;
    }
    j = i;
}
```

```
// Die Laufzeit liegt in O(n)
int result = 0;
int i = 0, j = 1;
while (i < n) {
    i++;
    while (j < n - i) {
        j++;
        result += i + j;
    }
    j = 1;
}
```

: Falsch

```
// Die Laufzeit liegt in O(n)
int result = 0;
int i = 0, j = 0;
while (i < n) {
    i++;
    while (j < n) {
        j++;
        result += i + j;
    }
}
```

: Richtig

```
// Die Laufzeit liegt in O(n^2)
int result = 0;
int i = 0, j = 0;
while (i < n) {
    i++;
    while (j < n) {
        j++;
        result += i + j;
    }
    j = 0;
}
```

: Richtig

```
// Die Laufzeit liegt in O(n^2)
int result = 0;
int i = 0, j = 0, k = 0;
while (i < n / 2) {
    i++;
    while (j < n / 4) {
        j++;
        while (k < n / 8) {
            k++;
            result += i + j + k;
        }
        k = j;
    }
    j = i;
}
```

: Falsch

Gegeben sei die folgende Variante des linearen Suchalgorithmus.

```
private static final int NOT_FOUND = -1;
```

```java
private static int search(int[] data, int key) {
    int i = 0, m = data.length;
    while (i < m) {
        if (data[i] == key) {
            m = i;                          // Zeile A
        } else {
            i++;
        }
    }
    if (i < data.length) {
        return i;
    }
    return NOT_FOUND;
}
```

Vorbedingung: data != null

Beurteilen Sie die Richtigkeit der folgenden Aussagen zu dieser Variante!

Richtig	Falsch		
○ ✗	● ✓	Negative Werte werden im Array `data` nicht gefunden.	✓
○ ✗	● ✓	Kommt der gesuchte Wert `key` in `data` mehrmals vor, dann kann auch der größte Index, an dem der Wert vorkommt, zurückgeliefert werden.	✓
● ✓	○ ✗	Es wird vom kleinsten Index des Arrays `data` weg nach dem Element `key` gesucht.	✓
● ✗	○ ✓	Werte, die größer als n (Länge von `data`) sind, werden nicht gefunden.	✗

Negative Werte werden im Array `data` nicht gefunden.: Falsch
Kommt der gesuchte Wert `key` in `data` mehrmals vor, dann kann auch der größte Index, an dem der Wert vorkommt, zurückgeliefert werden.: Falsch
Es wird vom kleinsten Index des Arrays `data` weg nach dem Element `key` gesucht.: Richtig
Werte, die größer als n (Länge von `data`) sind, werden nicht gefunden.
: Falsch

Gegeben sei der aus den Vorlesungsfolien bekannte Sortieralgorithmus Bubblesort.

```java
private static void bubbleSort(int[] data) {
```

```java
    for (int i = 0; i < data.length - 1; i++)          // Zeile A
    {
        for (int j = 0; j < data.length - i - 1; j++) // Zeile B
        {
            if (data[j] > data[j + 1])                 // Zeile C
            {
                exchange(data, j, j + 1);              // Zeile D
            }
        }
    }
}

private static void exchange(int[] a, int i, int j) {
    int swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

Entscheiden Sie für jede der unten angeführten Veränderungen, ob folgende Aussage richtig oder falsch ist:

Der Algorithmus

- terminiert nach der Veränderung noch immer korrekt,
- sortiert alle möglichen `data`-Arrays mit einer Länge größer 0 noch immer aufsteigend und
- ist auch für beliebige Belegungen von `data` stabil.

Richtig	Falsch		
● ☑	○ ✗	Zeile A: `for (int i = 0; i < data.length; i++)`	✔
○ ✗	● ☑	Zeile B: `for (int j = i; j < data.length - i - 1; j++)`	✔
● ☑	○ ✗	Zeile B: `for (int j = 0; i + j + 1 < data.length; j++)`	✔
● ☑	○ ✗	Zeile D: `exchange(data, j + 1, j);`	✔

Zeile A: `for (int i = 0; i < data.length; i++)` : Richtig
Zeile B: `for (int j = i; j < data.length - i - 1; j++)` : Falsch
Zeile B: `for (int j = 0; i + j + 1 < data.length; j++)` : Richtig
Zeile D: `exchange(data, j + 1, j);` : Richtig

Sie haben ein Array `data` mit n Werten gegeben.

Es soll ein Algorithmus verwendet werden, der den Inhalt des Arrays umdreht.

Beispiele:

- Wird das Array [1, 2, 3, 4, 5] bearbeitet, dann ist das Ergebnis [5, 4, 3, 2, 1].

- Wird das Array [1, 2, 3, 4, 5, 6] bearbeitet, dann ist das Ergebnis [6, 5, 4, 3, 2, 1].

Ihnen steht folgende Implementierung (Methode `reverse1`) zur Verfügung:

```
private static void reverse1(int[] data) {
    for (int i = 0; i < data.length / 2; i++) {
        int temp = data[i];
        data[i] = data[data.length - 1 - i];
        data[data.length - 1 - i] = temp;
    }
}
```

Daneben gibt es noch eine weitere Implementierung (Methode `reverse2`):

```
private static void reverse2(int[] data) {
    for (int start = 0, end = data.length - 1; start <= end; start++, end--) {
        int help = data[start];
        data[start] = data[end];
        data[end] = help;
    }
}
```

Die Vorbedingung für beide Methoden lautet: data != null.

Beurteilen Sie die Richtigkeit der folgenden Aussagen zu den beiden Methoden!

Richtig	Falsch		
○ ✗	● ☑	Die Laufzeit von `reverse2` liegt im Worst-Case in $O(\log n)$.	✔
● ☑	○ ✗	Der zusätzliche Speicherbedarf von `reverse1` liegt im Worst-Case in $O(1)$.	✔
● ☑	○ ✗	Wird die Methode `reverse2` zwei Mal direkt hintereinander auf ein Array aufgerufen, dann entspricht der Inhalt dieses Arrays dem Inhalt vor den zwei Aufrufen.	✔
○ ✗	● ☑	Die Methode `reverse1` dreht nur eine Hälfte des Arrays um.	✔

Die Laufzeit von `reverse2` liegt im Worst-Case in $O(\log n)$.
: Falsch
Der zusätzliche Speicherbedarf von `reverse1` liegt im Worst-Case in $O(1)$.
: Richtig
Wird die Methode `reverse2` zwei Mal direkt hintereinander auf ein Array aufgerufen, dann entspricht der Inhalt dieses Arrays dem Inhalt vor den zwei Aufrufen.: Richtig
Die Methode `reverse1` dreht nur eine Hälfte des Arrays um.: Falsch

Überprüfung beenden

(20.12.) Direkt zu: Literaturliste

Test-Navigation

1 2 3 4 5 6 7 8 9 10

Seiten einzeln anzeigen

Überprüfung beenden

© Technische Universität Wien - support@tuwel.tuwien.ac.at - Datenschutzerklärung - Data Protection Information

AMC - Academic Moodle Cooperation

Meine Kurse