Erfundene Beispielangaben für den EP1 Programmiertest

ACHTUNG: Da sich manche Unteraufgaben dieser Beispielaufgaben ähneln, empfiehlt sich nur ein Beispiel am Tag zu programmieren, sodass die Probleme neu überdacht werden müssen.

Die Beispiele sind ähnlich aufgebaut, wie die Beispieltests, die im TUWEL bereitgestellt werden und im Vowi sind.

Jedes Beispiel enthält 3 Methoden:

- **Die erste Methode bezieht sich auf die Generierung neuer Arrays basierend auf Informationen der Parameter.**
- **Die zweite Methode bezieht sich meist direkt auf die Inhalte der Arrays und deren Ordnung. Die Struktur der Arrays wird hierbei meistens behalten.**
- **Die dritte Methode ist häufig eine Interaktion zwischen verschiedenen Datentypen in Verbindung mit Arrays. Hier kann es manchmal helfen, die Datentypen zu ändern. Diese Methode ist IMMER rekursiv zu implementieren.**

Diese Aufgaben sind in drei Schwierigkeitsklassen unterteilt und können beliebig kombiniert werden; Die einzelnen Methoden sind immer unabhängig voneinander.

Bitte nicht davon ausgehen, die Tests zu 100% dieses Format haben werden. Dies sind nur Schätzungen für ähnliche Aufgabenstellungen, die zum Test kommen könnten.

Dauer der Prüfung: 120 Minuten

Stoppt die Zeit, die ihr für die Implementierung dieser Beispiele braucht, um einen etwaigen Richtwert zu bekommen, wie lang ihr für jede Methode brauchen werdet.

Evaluierung

Was zu beachten ist:

- **Deklarationen in der main Methode müssen stimmen**
- **Immer alle Testfälle implementieren und überprüfen**
- **Rückgabewerte müssen stimmen**
- **Schleifen/Rekursionen richtig implementiert**
- **Struktur und Inhalt von Arrays müssen stimmen**
- **Korrekte Handhabung von Randfällen (Fälle die NICHT von den Vorbedingungen abgedeckt werden!)**

Es gilt die Empfehlung alle beschriebenen Methoden ausführlich zu testen.

Alle Randfälle der Methode, sofern sie nicht durch die Vorbedingungen ausgeschlossen sind, müssen abgedeckt werden. Das „Wie“ ist jedem selbst überlassen.

Erlaubte Hilfsmittel

Eigene Methoden und Variablen dürfen hier immer benutzt werden. Jede Klasse (z.B. String oder Integer) darf immer verwendet werden, solange dies nicht ausdrücklich verboten ist.

Es ist jedoch ausdrücklich verboten, die Methodenköpfe zu ändern oder zu erweitern.

Empfehlung für die Beispielzusammensetzung:

1. Beispiel 3 generate() + Beispiel 2 sinusSort() + Beispiel 1 searchSequence()
2. Beispiel 2 generate() + Beispiel 1 sort() + Beispiel 3 getBinary()
3. Beispiel 1 generate() + Beispiel 3 reorder() + Beispiel 2 findClosest()

Beispiel 1 *(einfach)*:

Tipps für den Test:

Angabe genau durchlesen! Achtet auf die Vorbedingungen: Vorbedingungen selbst brauchen nicht kontrolliert werden. Fälle, die von den Vorbedingungen aber nicht abgedeckt werden (z.B. negative Zahlen als Indizes; Indizes, die über die Länge des Arrays hinaus gehen), müssen kontrolliert werden. Sollten Schwierigkeiten beim Verstehen/Implementieren der Aufgabe entstehen, wird empfohlen den Code zuerst als Pseudocode niederzuschreiben und diesen dann in Java umzuwandeln.

Deklarieren und initialisieren Sie in **main** die folgende(n) Variable(n):

4. **int[][] test1 = { {5, 2, 9, 1, 4}, {3}, {8, 1, 1, 0, -4}, {-2, -4, 5} }**
5. **int[][] test2 = { {3, 12, 8, 1, -5}, {-5, 0, -1}, {81, -10, 2, 15, -16} }**
6. **char[] test3 = { 'a', ':', ' ', 'R', 'b', 'h', 'K', 'z', '6', 't' }**
7. **char[] test4 = { 'a', ':', ' ', 'R', 'b', 'z', 'h', 'K', '6', 't' }**

Implementieren Sie folgende Methoden:

- **int[][] generate(int[][] input, int len, int increment)** erzeugt ein neues Array, welches am Ende zurückgegeben wird. Das erzeugte Element besitzt gleich viele Zeilen wie **input**. Die Länge der ersten Zeile beträgt **len**, jede weitere Zeile wird um den Wert **increment** vergrößert. Elemente, die nicht mehr in die Zeile passen, werden ausgelassen. Sind mehr Spalten im Rückgabearray als in der jeweiligen Zeile, so werden die restlichen Felder mit -1 befüllt.
  Vorbedingung(en): **input** != null, **len** >= 0, **increment** > 0, input[i] != null für alle gültigen i.
  Wird die Methode z.B. mit **input**=test1, **len**=3, **increment**=1 aufgerufen, entsteht folgendes Array:

  { {5, 2, 9}, {3, -1, -1, -1}, {8, 1, 1, 0, -4}, {-2, -4, 5, -1, -1, -1} }

- **void sort(int[][] input)** überschreibt die Werte im Array **input** und sortiert alle Werte jeder Zeile absteigend. Werte von unterschiedlichen Zeilen werden nicht verglichen. Die Sortiermethode bleibt Ihnen überlassen.
  Vorbedingung(en): **input** != null, **input**.length > 0
  Wird die Methode z.B. mit test2 aufgerufen, wird dieses wie folgt verändert:

  { {12, 8, 3, 1, -5}, {0, -1, -5}, {81, 15, 2, -10, -16} }

- **boolean searchSequence(char[] input, char[] sequence, int index**) gibt zurück, ob sich **sequence** ohne Unterbrechungen im Array **input** befindet. Die Variable **index** kann als Hilfe verwendet werden, falls ein Teil von **sequence** schon gefunden wurde. In dem Fall ist die Methode mit **index**=0 aufzurufen.
  Diese Methode muss rekursiv implementiert werden **(keine Schleifen)!** Es dürfen **keine** Methoden der Klasse String verwendet werden, Operatoren sind aber erlaubt.
  Vorbedingung(en): **input** != null, sequence != null, index >= 0.
  Wird die Methode mit **input**=test3, **sequence**={'b', 'h', 'K'}, **index**=0 aufgerufen, so ist die Rückgabe true.
  Wird die Methode mit **input**=test4, **sequence**={'b', 'h', 'K'}, **index**=0 aufgerufen, so ist die Rückgabe false.

Beispiel 2 *(mittel)*:

Tipps für den Test:

Angabe genau durchlesen! Achtet auf die Vorbedingungen: Vorbedingungen selbst brauchen nicht kontrolliert werden. Fälle, die von den Vorbedingungen aber nicht abgedeckt werden (z.B. negative Zahlen als Indizes; Indizes, die über die Länge des Arrays hinaus gehen), müssen kontrolliert werden. Sollten Schwierigkeiten beim Verstehen/Implementieren der Aufgabe entstehen, wird empfohlen den Code zuerst als Pseudocode niederzuschreiben und diesen dann in Java umzuwandeln.

Deklarieren und initialisieren Sie in **main** die folgende(n) Variable(n):

8. **int[][] test1 = { {1, 2, 3, -4, -9}, {-20, 13}, {30, 19, 12, -12} }**
9. **int[][] test2 = { {}, {1, 2, 3, 4, 5}, {-1, -2, -3, -4}, {(int) Math.PI} }**
10. **char[] test3 = { 'a', '.', 'S', 'r', '4', ' ', '=', 'K' }**

Implementieren Sie folgende Methoden:

- **int[][] generate(int[][] input)** erzeugt ein neues Array mit derselben Struktur wie input. Die Inhalte jeder Zeile werden in die nächste Zeile geschrieben. Die Inhalte der letzten Zeile kommen in die erste Zeile. Ist die Länge einer Zeile zu kurz für alle verschobenen Zahlen, so werden die übergebliebenen Zahlen auf jedes Element dieser Zeile dazu addiert. Ist die Länge der Zeile zu lang für alle verschobenen Zahlen, so werden die restlichen Felder mit dem Durchschnitt der Zahlen dieser Zeile aufgefüllt.
  Vorbedingung(en): **input** != null, input[i] != null für alle gültigen i.
  Wird die Methode z.B. mit test1 aufgerufen, entsteht folgendes Array:

  { {30, 19, 12, -12, 12}, {-9, -8}, {-20, 13, -3, -3} }

- void sinusSort(int[][] input) sortiert das übergebene Array aufsteigend nach den jeweiligen sinuswerten. Beispielsweise werden die Zahlen 1, 2 nicht vertauscht, da sin(1) < sin(2). Die Zahlen 3, 4 werden schon vertauscht, da sin(3) > sin(4). Die Sortiermethode bleibt Ihnen überlassen.
  Vorbedingung(en): **input** != null.
  Wird die Methode z.B. mit test2 aufgerufen, wird dieses wie folgt verändert:

  { {}, {5, 4, 3, 1, 2}, {-2, -1, -3, -4}, {3} }

- **int findClosest(char[] sequence, int goal, int index)** gibt den Index des Elements in **sequence** zurück, das basierend auf den ASCII Werten der Charaktere am nächsten zu **goal** ist. Anders gesagt: Jeder Wert des Arrays wird mit **goal** verglichen. Der Index des Elements mit der kleinsten absoluten Differenz zu **goal** wird zurückgegeben. Dabei muss nicht gewährleistet sein, dass das Element **goal** im Array selbst ist.
  Diese Methode muss rekursiv implementiert werden **(keine Schleifen)!** Es dürfen **keine** Methoden der Klasse String verwendet werden, Operatoren sind aber erlaubt.
  Vorbedingung(en): **sequence** != null
  Wird die Methode mit **sequence**=test3, **goal**=81 aufgerufen, so ist die Rückgabe **2**.

Beispiel 3 *(schwer)*:

Tipps für den Test:

Angabe genau durchlesen! Achtet auf die Vorbedingungen: Vorbedingungen selbst brauchen nicht kontrolliert werden. Fälle, die von den Vorbedingungen aber nicht abgedeckt werden (z.B. negative Zahlen als Indizes; Indizes, die über die Länge des Arrays hinaus gehen), müssen kontrolliert werden. Sollten Schwierigkeiten beim Verstehen/Implementieren der Aufgabe entstehen, wird empfohlen den Code zuerst als Pseudocode niederzuschreiben und diesen dann in Java umzuwandeln.

Deklarieren und initialisieren Sie in **main** die folgende(n) Variable(n):

11. int[][] test1 = { {5, 12, -5}, {-10, 18}, {7}, {1} }
12. int[][] test2 = { {0, 1, -1, 2, -2, 3, -3}, {8, -2, -5, -13, -7, 9}, {-1, 10, 6} }
13. int[] test3 = {15, 32, 2, 108}

Implementieren Sie folgende Methoden:

- **int[][] generate(int[][] input)** erstellt ein neues Array mit derselben Struktur. Jeder Wert wird jedoch mit seinem übernächsten addiert, dabei wird keine Rücksicht auf Zeilen oder Spalten genommen. Die zwei letzten Elemente (der letzten – oder vorletzten – Zeile) werden mit dem respektiv ersten bzw. zweiten Elements (der ersten Zeile) aufsummiert. Ist eine Zeile Beispielsweise nur ein Element lang, muss zwei Zeilen im Voraus geschaut werden.
  Vorbedingung(en): **input** != null, **input**.length > 2, input[i].length > 0, input[i].length != null für alle gültigen i.
  Wird die Methode z.B. mit test1 aufgerufen, entsteht folgendes Array:

  { {0, 2, 13}, {-3, 19}, {12}, {13} }

- **void reorder(int[][] input)** ändert die Positionen des Inhalts jeder Zeile so um, dass negative Zahlen in der Mitte dieser Zeile stehen. Die Reihenfolgen unter den negativen bzw. positiven Zahlen bleibt jedoch erhalten. Um dies zu bewerkstelligen muss die Länge der Zeile und die Anzahl an negativen Zahlen ungerade sein und die Anzahl an positiven Zahlen (0 wird als positive Zahl gezählt) gerade sein. Ist dies nicht der Fall wird dies Zeile auf null gesetzt.
  Vorbedingung(en): **input** != null**.**
  Wird die Methode z.B. mit aufgerufen, wird dieses wie folgt verändert:
  { {0, 1, -1, -2, -3, 2, 3}, null, {10, -1, 6} }

- **int[][] getBinary(int[] input)** gibt ein neues zweidimensionales Array zurück. Dabei wird jede Zahl aus **input** in binär umgewandelt. Das Array **input** wird dabei nicht verändert. Jede binäre Zahl wird anschließend in eine Zeile des zurückgegebenen Arrays sein, wobei jede Ziffer dieser binären Zahl ein eigenes Element der jeweiligen Zeile ist. Die Länge jeder Zeile wird in jedem Fall möglichst kurzgehalten.
  Diese Methode muss rekursiv implementiert werden. Eine Schleife darf nur für die einzelnen Spalten verwendet werden, nicht aber für die Zeilen.
  Vorbedingung(en): **keine**.
  Hinweis: Eventuell benötigen Sie Umwandlungen von Integer zu String und zurück.
  Wird die Methode mit test3 aufgerufen, so ist die Rückgabe wie folgt:

  { {1, 1, 1, 1}, {1, 0, 0, 0, 0, 0}, {1, 0}, {1, 1, 0, 1, 1, 0, 0} }