Vorkurs zur *Einführung in die Programmierung* (HS 2023)

Aufgaben Kapitel 5 – 8

Für diese Programme können Sie rekursive Methoden entwickeln oder Schleifen verwenden, die dann in eine Methode eingebettet sind.

In den meisten Fällen geben die Aufgabenstellungen den Namen der Methode, die Parameter (sowohl Typen als auch Reihenfolge) sowie den Rückgabewert vor. Sie sollten diese nicht ändern. (Wenn Sie die Aufgaben auf Ihrem Computer lösen spielen die Namen der Methoden keine Rolle, aber wenn Sie später in der Vorlesung Aufgaben lösen, dann sind diese Vorgaben sehr wichtig. Gewöhnen Sie sich daran, diese Vorgaben zu erfüllen.) Die Namen der Parameter können Sie natürlich immer frei wählen. In vielen Fällen können Sie bessere Namen finden als wir vorschlagen.

Jede Ihrer Lösungen sollten Sie mit verschiedenen Beispielwerten (für die geforderten Parameter) ausprobieren.

Wenn Sie `int` Variable verwenden, so können Sie davon ausgehen, dass der Wertebereich des Basistyps `int` ausreicht, um alle Ergebnisse etc. zu speichern.

5 Wiederholungen: Schleifen und Methoden

1. Schreiben Sie eine Methode die zählt, wie oft des Zeichen 7 in dem String enthalten ist. Also für `"124"` ist des Ergebnis 0, genauso für `"abc"` und den leeren String. Für "707" ist die Antwort 2, für "007" ist die Antwort 1.

   ```
   int count7(String s)
   ```

2. Schreiben Sie eine Methode die einen String entgegennimmt und einen neuen String konstruiert, in dem hinter jedes Zeichen des Ursprungsstrings ein "+" eingefügt ist.

   ```
   String insertPlus(String s)
   ```

   Also `insertPlus("a")` ergibt `"a+"`, `insertPlus("abc")` ergibt `"a+b+c+"`, `insertPlus("a b c")` ergibt `"a+ +b+ +c+"`.

   // insert star – nur wenn Zeichen nicht Zwischenraum

3. Schreiben Sie eine Methode die einen String entgegennimmt und einen neuen String konstruiert, in dem zwei identische aufeinanderfolgende Zeichen des Ursprungsstrings in Grossbuchstaben erscheinen.

   ```
   String doubleCap(String s)
   ```

Also `doubleCap("aa")` ergibt `"AA"`, `doubleCap("Aac")` ergibt `"Aac"`, `doubleCap("11ac")` ergibt `"11ac"`, `doubleCap("bAAac")` ergibt `"bAAac"`, `doubleCap("aaaBc")` ergibt `"AAABc"`.

6 und 7 Arrays

Für diese Programme können Sie erst die Anweisungen in der JShell entwickeln und dann in eine Methode umwandeln, oder sofort als Methode entwicklen. Ob Sie weitere (Hilfs)Methoden und/oder rekursive Methoden verwenden, ist Ihnen frei gestellt.

Bei allen Methoden, die Arrays als Parameter erwarten, sollten die Methode prüfen, ob der Array existiert (also der Parameter nicht `null` ist). Wenn einer der Parameter nicht exisistiert ist das Verhalten nicht festgelegt (Sie können einen Wert Ihrer Wahl zurückgeben).

1. Wenn Sie Schleifen üben wollen, so können Sie auch die Aufgaben früherer Tage nocheinmal mit Schleifen lösen.

2. Stellen Sie fest, ob für zwei `int` Arrays das erste und das letzte Element idenisch sind.

   Also für {1, 2, 3, 4} und {1, 4} ist des Ergebnis `true`, für {0, 2, 3, 4} und {1, 4} ist das Ergebnis false,

   ```
   boolean similar1(int[] a, int[] b)
   ```

3. Summieren Sie die Elemente eines `int` Arrays.

   Also für {1, 2, 3, 4} ist das Ergebnis 10.

   ```
   int arraySum(int[] a)
   ```

4. Zählen Sie wie oft die Zahl 7 in einem `int` Array auftritt.

   Also für {1, 2, 3, 4} ist das Ergebnis 0, für {7, 2, 7, 4} ist das Ergebnis 2.

   ```
   int  count7(int[] x)
   ```

5. Berechnen Sie das Element-Produkt zweier `int` Arrays.

   Also für {1, 2, 3, 4} und {1, 2, 3, 4} ist das Ergebnis {1, 4, 9, 16}. Die Arrays müssen dieselbe Länge haben. Ist das nicht der Fall, so geben Sie `null` zurück.

   ```
   int[]  innerProduct(int[] x, int[] y)
   ```

6. Summieren Sie die Elemente eines `int` Arrays, wobei Elemente zwischen zwei Elementen mit Wert 0 nicht addiert werden. Elemente mit Wert 0 tretten immer paarweise auf und jedes Element mit Wert 0 ist entweder der Anfang einer Folge von Elementen, die nicht addiert werden, oder das Ende.

   Also für {1, 3, 0, 4, 6, 0, 2, 0, 3, 2, 0, 1} ist das Ergebnis 7.

```
int arrayZSum(int[] a)
```

7. Stellen Sie fest, ob ein `int` Array `second` die selben Werte hat wie das Ende des `int` Arrays `first`. Also für `second` {1, 2, 0, 3, 0, 4} und `first` {9, 7, 12, 4, 3, 1, 2, 0, 3, 0, 4} ist die Antwort `true` Für `second` {1, 2} und `first` {9, 7, 12, 8} ist die Antwort `false`.

   ```
   boolean endSameI(int[] f, int[] s)
   ```

   Stellen Sie fest, ob ein `String` Array `second` die selben Werte hat wie das Ende des `String` Arrays `first`. Also für `second` {"one", "two", "three"} und `first` {"hello", "haus", "one", "two", "three"} ist die Antwort `true`, für `second` {"one", "two", "three""one", "two", "three"} und `first` {"three", "two", "one"} ist die Antwort `false`.

   ```
   boolean endSameS(String[] f, String[] s)
   ```

8 Sonstige Aufgaben

Bei allen Methoden, die Arrays als Parameter erwarten, sollten die Methode prüfen, ob der Array existiert (also der Parameter nicht null ist). Wenn einer der Parameter nicht exisistiert ist das Verhalten nicht festgelegt (Sie können einen Wert Ihrer Wahl zurück geben).

1. Zählen Sie Anzahl der geraden Zahlen in einem `int` Array. 0 zählt als gerade Zahl.

   Also für {1, 2, 3, 4} und {-2, 4} ist des Ergebnis 2, für {0, 2, 10, 4} ist das Ergebnis 4, und für {1, 3} ist das Ergebnis 0.

   ```
   boolean anzahlGerade(int[] a)
   ```

2. Ihre Methode soll für den Input `int` Array einen `int` Array zurück geben, in dem jede 0 im Eingabearray durch das grösste ungerade Element, welches rechts von der 0 im Eingabearray erscheint, ersetzt wurde. Wenn es kein ungerades Element (rechts von der 0) gibt, dann soll die 0 stehen bleiben. (Das Element mit Index 0 steht am weitesten links; das Element mit Index 1 steht rechts daneben.)

   Also für {0, 7, 0, 3} ist das Ergebnis {7, 7, 3, 3}, für {0, 4, 0, 3} ist das Ergebnis {3, 4, 3, 3}, für {0, 3, 0, 4} ist das Ergebnis {3, 3, 0, 4}, und für {0, 1, 0} ist das Ergebnis {1, 1, 0}.

   ```
   int[] oddZero(int[] a)
   ```

3. Für einen Input `int` Array sollen Sie einen Array zurück geben, der die selben Zahlen wir die Input Array enthält, aber die Zahlen so umsortiert hat, dass alle geraden Zahlen vor allen ungeraden Zahlen erscheinen. Sonst gibt es keine Einschränkungen, insbesondere können die Zahlen in beliebiger Reihenfolge erscheinen so lange alle geraden Zahlen vor den ungeraden Zahlen erscheinen. Sie können den Input Array modifizieren oder einen neuen Array erstellen.

   Also für {0, 1, 0, 1, 0, 1, 1, 0} ist das Ergebnis {0, 0, 0, 0, 1, 1, 1, 1}, für {10, 10, 10} ist das Ergebnis {10, 10, 10}, und für {11, 9, 10} {10, 11, 9} oder {10, 9, 11}.

```
int[]  simpleOrder(int[] x)
```

4. Zählen Sie wie oft in einem String ein Zeichen dreimal hintereinander auftritt. Achtung: Ein Zeichen kann zu mehr als einer Dreier-Gruppe gehören.

   Also für `"abcXXXabc"` ist das Ergebnis 1, für `"xxxabyyyycd"` ist das Ergebnis 3, für `"a"` ist das Ergebnis 0.

   ```
   int dreierFolge(String s)
   ```

5. Gegeben sei ein `int` Array. Sie sollen feststellen ob in diesem Array die Zahl 3 genau dreimal auftritt **und** die Zahl 3 nie direkt neben einer anderen Zahl mit Wert 3 erscheint.

   Für {3, 1, 3, 1, 3} ist das Ergebnis true. Für {3, 1, 3, 3} ist das Ergebnis false. Für {3, 4, 3, 3, 4} ist das Ergebnis false. Für {3, 4, 3, 5, 3, 8, 3, 5, 4} ist das Ergebnis false. Für {3, 4, 3, 5, 3, 8, 3, 3, 4} ist das Ergebnis false.

   ```
   int hat3(int[] a)
   ```