Übungsaufgaben Woche 5
Array-Wiederholungen und Zweidimensionale Arrays

Hinweis: Aufgabe 2 bis 5 behandelt eindimensionale, 6 bis 10 zweidimensionale Arrays.

Aufgabe 1: Fizz Buzz?

Schreibt eine Funktion, welche das bereits bekannte Fizz-Buzz Pattern von 1 bis 100 (alles inklusive) ausgibt.

Versucht euch dabei eine möglichst „dumme“ und komplizierte Lösung einfallen zu lassen, die in der Praxis wahrscheinlich nie eine Anwendung finden wird.

Ladet eure Lösungen für Aufgabe 1 - wenn ihr wollt - auf Moodle hoch *(oder stellt ein Pull-Request auf GitHub)*.

Aufgabe 2: Binary Addition

Schreibt eine Funktion, welche zwei eindimensionale Arrays gleicher Länge und einen Parameter `output` (Datentyp: bool) übergeben bekommt. Diese Arrays sind ausschließlich mit den Zahlen 0 und 1 gefüllt, welche eine Binärzahl darstellen sollen.

Beispiel Parameter:

```
> int[] a = { 0, 1, 1, 0, 0, 1 };
> int[] b = { 1, 0, 0, 0, 1, 1 }; // gleiche Länge
```

Die Funktion addiert nun intern die beiden Binärzahlen (oder auch: die beiden Arrays) und gibt das Ergebnis als neues Array zurück.

Der Parameter `output` soll den Default-Wert `false` besitzen.
Falls dieser Parameter allerdings bewusst auf `true` gesetzt wurde, soll die Funktion das Ergebnis noch auf der Konsole ausgeben, bevor das Ergebnis zurückgegeben wird.

Anmerkung: den letzten Übertrag - falls einer vorhanden ist - soll ignoriert werden!

Der Methodenkopf kann folgendermaßen aussehen:

```
static int[] AddBinaryNums(int[] a, int[] b, bool output = false)
```

Bei einem „Input“ von

```
> int[] a = { 1, 1, 1, 1, 0, 1 };
> int[] b = { 1, 0, 1, 0, 0, 1 };
```

ist der Ergebnis (da der letzte Übertrag nach links ignoriert wird):

```
> int[] r = { 1, 0, 0, 1, 1, 0 };
```

Aufgabe 3: Berechnen von Mittelwerten

Schreibt ein Programm (Funktion nicht nötig), welche auf 10 Eingaben des Users wartet und diese in ein eindimensionales Integer-Array speichert.

Es kann davon ausgegangen werden, dass der User **immer** Zahlen von 0 bis 9 (inklusive) eingibt. Es muss also keine Fehlerbehandlung implementiert werden!

Sobald der User eine Zahl eingegeben hat, soll diese Zahl sofort gelesen werden, ohne dass noch zusätzlich die „Enter“-Taste gedrückt werden muss.
Tipp: benutzt hierfür die Funktion `Console.ReadKey();`!

Sobald alle Zahlen eingegeben sind, sollen folgende Werte berechnet und die Ergebnisse auf der Konsole ausgegeben werden.

Folgende Werte dieser Zahlen sollen berechnet werden:

- Mittelwert (https://de.wikipedia.org/wiki/Mittelwert) und
- Median (https://de.wikipedia.org/wiki/Median)

Aufgabe 4: Array-Multiplikation

Vorgegeben ist folgendes eindimensionales Integer-Array:

```
> int[] mult = { 1, 0, 5, -9, 54, 2, 68, -17, 46, 3};
```

Schreibt eine Funktion, welche dieses Array als Parameter bekommt und jedes benachbarte Element miteinander multipliziert und folgende Ausgabe erzeugt:

```
> Multiplikation dieses Arrays:
> 1 x 0 = 0
> 0 x 5 = 0
> 5 x -9 = -45
> -9 x 54 = -486
> . . .
> 46 x 3 = 138
>
> Anzahl Rechenschritte: 9
```

Frage:

- ist die Anzahl der insgesamt durchgegangenen Rechenschritte irgendwie von der Länge des Arrays ableitbar?
- gibt es einen Schleifentypen, welcher für die Lösung der Aufgabe zu präferieren ist?

Aufgabe 5: Bestimmen einer Summe in einem Array

a) Simples Addieren

Für diese Aufgabe ist ein **geordnetes** eindimensionales Array vorgegeben.
Welche Werte dieses Array genau besitzt, ist für diese Aufgabe nicht von Relevanz, solange es geordnet ist.

Beispiel:

```
> int[] sortedArray = { 1, 2, 4, 4, 5, 7 };
```

Zudem sind die Zahlen in diesem Array immer positiv, und niemals 0.

Schreibt eine Funktion, welche dieses Array und einen weitere Parameter (`int sum`) übergeben bekommt, und einen Wahrheitswert basierend auf folgenden Kriterien zurückgibt:

- falls zwei Zahlen (an unterschiedlichen Indizes) zusammenaddiert die gewollte Summe (`sum`) ergeben, gibt die Funktion `true` zurück
- falls keine zwei Zahlen diese Summe ergibt, gibt die Funktion `false` zurück.

Beispiel mit dem Beispiels-Array:

```
> Summe ist 7
> Funktion gibt true zurück, da die Zahlen 2 + 5 = 7
```

b) Ohne geschachtelte Schleifen **(fortgeschritten)**

Gibt es eine Möglichkeit dieses gewollte Verhalten in der Funktion ohne eine geschachtelte Schleife zu implementieren?

Falls du auf eine mögliche Lösung kommst, implementiere sie! (Falls nicht, überspringe diese und die nächste Teilaufgabe und mache - wenn du willst - bei Aufgabe 5d weiter. Diese Teilaufgaben sind schon (sehr) fortgeschritten, also keine Sorge, falls dir hierzu nichts einfallen sollte!)

c) Ohne geschachtelte Schleifen *Part 2* **(sehr fortgeschritten)**

Jetzt kann nicht mehr länger garantiert werden, dass dieses Array sortiert ist (alle anderen Kriterien sind noch gegeben [keine negativen Nummern und keine 0]).

Basierend auf Aufgabe 5b), gibt es jetzt hier noch eine Möglichkeit das Verhalten ohne eine geschachtelte Schleife zu implementieren?

Falls dir auch hier eine Lösung einfallen sollte, implementiere sie!

Zusatzfrage zu c): weißt du auch, welche Big O Notation deine Lösung hat? Gibt es einen optimalen Weg, die Laufzeit dieses Programms zu optimieren?

d) Keinen Wahrheitswert mehr

Wir gehen jetzt wieder davon aus, dass das Array in sortierter Form uns vorliegt.

Nun soll die Funktion keinen Wahrheitswert mehr, sondern ein Integer-Array der Länge 2 mit den Indizes der ersten beiden Zahlen, welche die Summe bilden, zurückgeben.

In unserem Beispiel:

```
> Summe ist 8
> Rückgabe der Funktion: {0, 9}
```

Die Summe von 1 und 7 ist 8, die Summe von 4 und 4 allerdings auch. Da die 1 vor der 4 steht, ist 1 und 7 das erste Zahlenpaar, wessen Indizes dann zurückgegeben werden.

Aufgabe 6: Unterschied zwischen zweidimensionaler Arrays

Für diese Aufgabe muss nichts in Code umgesetzt werden.

Was ist der Unterschied zwischen einem zweidimensionalen Arrays, welches nicht ausgefranst ist, und einem zweidimensionalen Array, welches ausgefranst ist?

Welche Einsatzgebiete könnten ausgefranste Arrays besitzen?

Aufgabe 7: Zweidimensionales Array

Schreibt ein Programm, welches als Basis ein zweidimensionales Array der Längen 5 x 7 hat (kein ausgefranstes Array).

a) Ausgabe eines zweidimensionalen Arrays

Schreibt eine Funktion, welche das zweidimensionale Array auf der Konsole ausgibt.

b) Interaktion mit diesem Array

Dem Benutzer soll die Möglichkeit geboten werden, ein Feld in diesem Array anzusprechen, indem er zunächst einen Integer für die erste Dimension, danach einen Integer für die zweite Dimension eingibt.

Es kann auch hier davon ausgegangen werden, dass der Benutzer sich richtig verhält (keine Fehlererkennung nötig).

Danach soll der Benutzer einen Wert zwischen 1 und 50 (inklusive) in der angesprochenen Zelle speichern können.

Bevor der Benutzer eine Zahl eingibt, und danach, soll das Array auf der Konsole ausgegeben werden. Benutze hierfür die Funktion aus der ersten Teilaufgabe.

Aufgabe 8: Ausgabe einer Reihe aus einem zweidimensionalen Arrays

Schreibt eine Funktion, welche ein **ausgefranstes** zweidimensionales Array und eine weitere Integer-Variable (`int row`) als Parameter übergeben bekommt und nichts zurückgibt.

Die Funktion soll in diesem Array die Zeile `row` auf der Konsole ausgeben.
Achtet hierbei auf das ausgefranste Array. Die Zeilen können hierbei unterschiedliche Längen aufweisen!

Der Methodenkopf dieser Funktion könnte folgendermaßen aussehen:

```
static void PrintRowOf2DArray(int[][] arrayToPrint, int row)
```

Aufgabe 9: Sehr simples Tic Tac Toe

Benutzt für diese Aufgabe den Code aus Aufgabe 7.

Schreibt ein Programm, welches den Benutzer nacheinander nach zwei Integer-Zahlen fragt (Indizes in das zweidimensionale Array) und setzt danach an dieser Stelle bei jedem zweiten Zug ein „O“, sonst ein „X“.

Bereits gesetzte Werte sollen nicht überschrieben werden können. Wie implementiert man hier die Erkennungslogik am besten?

Eine Logik zur Gewinnerkennung soll **nicht** implementiert werden.

Aufgabe 10: Addieren zwei zweidimensionaler Arrays

Schreibt eine Funktion, welche zwei zweidimensionale Arrays der Längen 3 x 3 übergeben bekommt und die Arrays an jeden Index zusammen rechnet und in einem neuen Array speichert.

Dieses neue Array soll am Ende der Funktion zurückgegeben werden.

Beispiel:

```
> int[,] a = {{1, 0, 2}, {4, 2, 8}, {5, 0, 2}}; // new int[3,3];
> int[,] b = {{4, 2, 2}, {0, 0, 3}, {7, 0, 99}};
>
> // Aufruf der Funktion
> var result = Add2DArrays(a, b);
> Console.WriteLine(result);
>
> // Ausgabe (Benutzen der Funktion aus Aufgabe 6a) möglich)
> {{5, 2, 4}, {4, 2, 11}, {12, 0, 101}} // schönere Formatierung?
```