ARRAYS: BINÄRE SUCHE

Wir wissen bereits, wie wir in einem Array nach einem bestimmten Element suchen. Hier haben wir die *Lineare Suche* kennen gelernt, bei der wir ganz stur das Array durchlaufen und immer prüfen, ob wir den gesuchten Wert gefunden haben. Diese Art der Suche ist aber sehr ineffizient und kann sehr lange dauern, wenn das Array groß ist.

Zum Glück gibt es Such-Algorithmen, die wesentlich *effizienter* und damit *schneller* zum Ziel kommen. Das entscheidende dabei: Das Array muss **sortiert** sein! Nicht nur Menschen, sondern auch digitale Geräte finden in einer sortierten Liste das Gesuchte wesentlich schneller, wenn man einen guten Algorithmus hat.

Algorithmus: Binäre Suche

Die binäre Suche funktioniert im Wesentlichen so: Wir teilen das Array in der Mitte und prüfen dann, ob wir in der linken oder in der rechten Hälfte weitersuchen müssen. Das machen wir dann mit der ausgewählten Hälfte genauso, bis wir den gesuchten Wert gefunden haben oder feststellen, dass der Wert nicht vorhanden ist. Wie wir das Teilen und Suchen genau realisieren, wird im Folgenden beschrieben:

Der Algorithmus

- Setze die Variable *indexAnfang* auf das erste und die Variable *indexEnde* auf das letzte Element des Arrays
- Setze die Variable *indexMitte* auf das mittlere Element des Arrays (hier müssen wir möglicherweise runden!)
- Falls array[*indexMitte*] das gesuchte Element ist, können wir die Suche erfolgreich beenden.
- Falls array[*indexMitte*] > gesuchtes Element => Wir müssen in der linken Hälfte weitersuchen
- Falls array[*indexMitte*] < gesuchtes Element => Wir müssen in der rechten Hälfte weitersuchen
- Falls wir das Element noch nicht gefunden haben: Wiederhole ab dem zweiten Schritt im noch zu durchsuchenden Teil des Arrays.

Beispiel: Wir suchen die Zahl **9**!

Wir sehen natürlich sofort, dass das die **Mitte** des Arrays ist. Der Computer sieht das leider nicht, er kann es aber leicht so ausrechnen:
indexMitte = (indexAnfang + indexEnde) / 2

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33

indexAnfang ***indexMitte*** *indexEnde*

10 > 9 => Wir müssen links suchen!

Falls das gesuchte Element vorhanden ist, muss es sich in diesem Teil des Arrays befinden!

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33
indexAnfang			*indexMitte*			*indexEnde*

Wir müssen also links suchen! Wie sagen wir jetzt dem Computer, dass er links suchen soll? Ganz einfach: Wir verändern den Wert für *indexEnde*:

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33
indexAnfang		***indexEnde***	*indexMitte*			

„Links suchen“ bedeutet: Wir müssen *indexEnde* nach links verschieben!
Also: indexEnde = indexMitte - 1

Jetzt berechnen wir die Mitte neu:

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33
indexAnfang	***indexMitte***	*indexEnde*				

indexMitte = (indexAnfang + indexEnde) / 2

Jetzt geht es wieder von Anfang los: Wir überprüfen, ob wir die Zahl schon gefunden haben oder weiter links bzw. weiter rechts suchen müssen!

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33
indexAnfang	***indexMitte***	*indexEnde*				

5 < 9 => Wir müssen rechts suchen!

Um weiter rechts zu suchen, verändern wir indexAnfang:

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33
	indexMitte	***indexAnfang*** *indexEnde*				

„Rechts suchen“ bedeutet: Wir müssen *indexAnfang* nach rechts verschieben!
Also: *indexAnfang* = indexMitte + 1

Jetzt berechnen wir die Mitte neu:

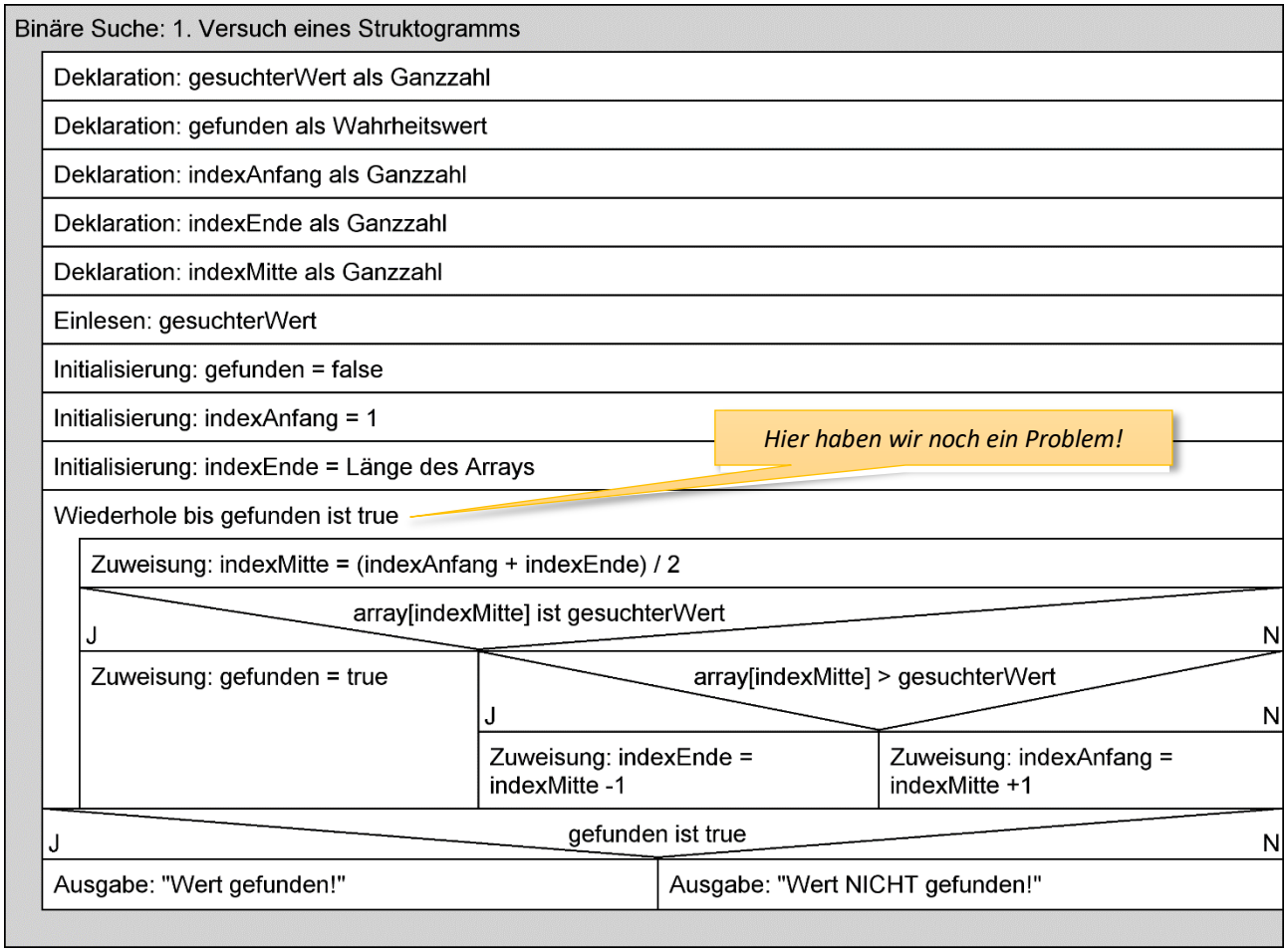
array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33

indexAnfang
indexEnde
indexMitte

indexMitte = (indexAnfang + indexEnde) / 2

array[indexMitte] ist das gesuchte Element! Wir haben es gefunden!

Struktogramm: Ein erster Versuch



In der Schleife stimmt etwas noch nicht! Wann wird die Schleife denn eigentlich abgebrochen? Klar, dann wenn *gefunden* den Wert *true* annimmt, wir das gesuchte Element also gefunden haben.

Was passiert aber, wenn das gesuchte Element gar nicht im Array vorhanden ist? Dann nimmt *gefunden* ja **nie** den Wert *true* an! Wir müssen also eine Möglichkeit finden, es zu erkennen, falls der gesuchte Wert nicht vorhanden ist.

Schauen wir uns dieses an einem Beispiel an:

Beispiel: Wir suchen die Zahl **8**!

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33

indexAnfang ***indexMitte*** *indexEnde*

10 > 8 => Wir müssen links suchen!

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33

indexAnfang ***indexMitte*** *indexEnde*

5 < 8 => Wir müssen rechts suchen!

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33

indexAnfang
indexEnde
indexMitte

9 > 8 => Wir müssen links suchen!

„Links suchen“ bedeutet: Wir müssen *indexEnde* verschieben! Also:
indexEnde = indexMitte - 1

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]
1	5	9	10	12	22	33

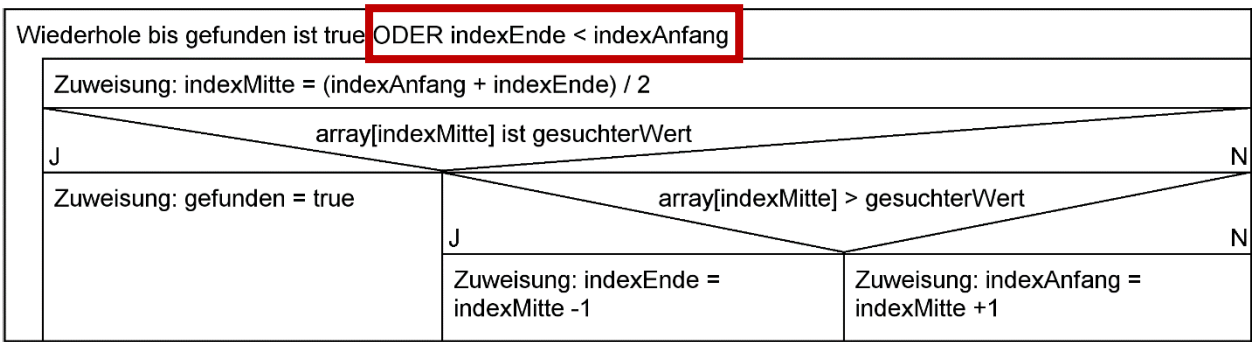
indexEnde ***indexAnfang***
indexMitte

indexEnde < indexAnfang!?!

Ist es Ihnen aufgefallen? Das Ende (*indexEnde*) steht weiter links als der Anfang (*indexAnfang*) des zu durchsuchenden Bereichs!
Daran können wir erkennen, dass der gesuchte Wert nicht im Array vorhanden ist!

Struktogramm: Zweiter Versuch

Wir ändern die Abbruchbedingung in unserem Struktogramm, so dass die Schleife verlassen wird, sobald wir erkennen, dass wir den gesuchten Wert nicht finden können:



Ein letztes Problem: Runden!

Ein Problem ist uns noch gar nicht aufgefallen: Bei der Berechnung von *indexMitte* kann es sein, dass wir eine Kommazahl erhalten!

Beispiel:

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]
1	5	9	10	12	22

indexAnfang *indeEnde*

⇨ **indexMitte = (indexAnfang + indexEnde) / 2 = (1 + 6) / 2 = 3,5**

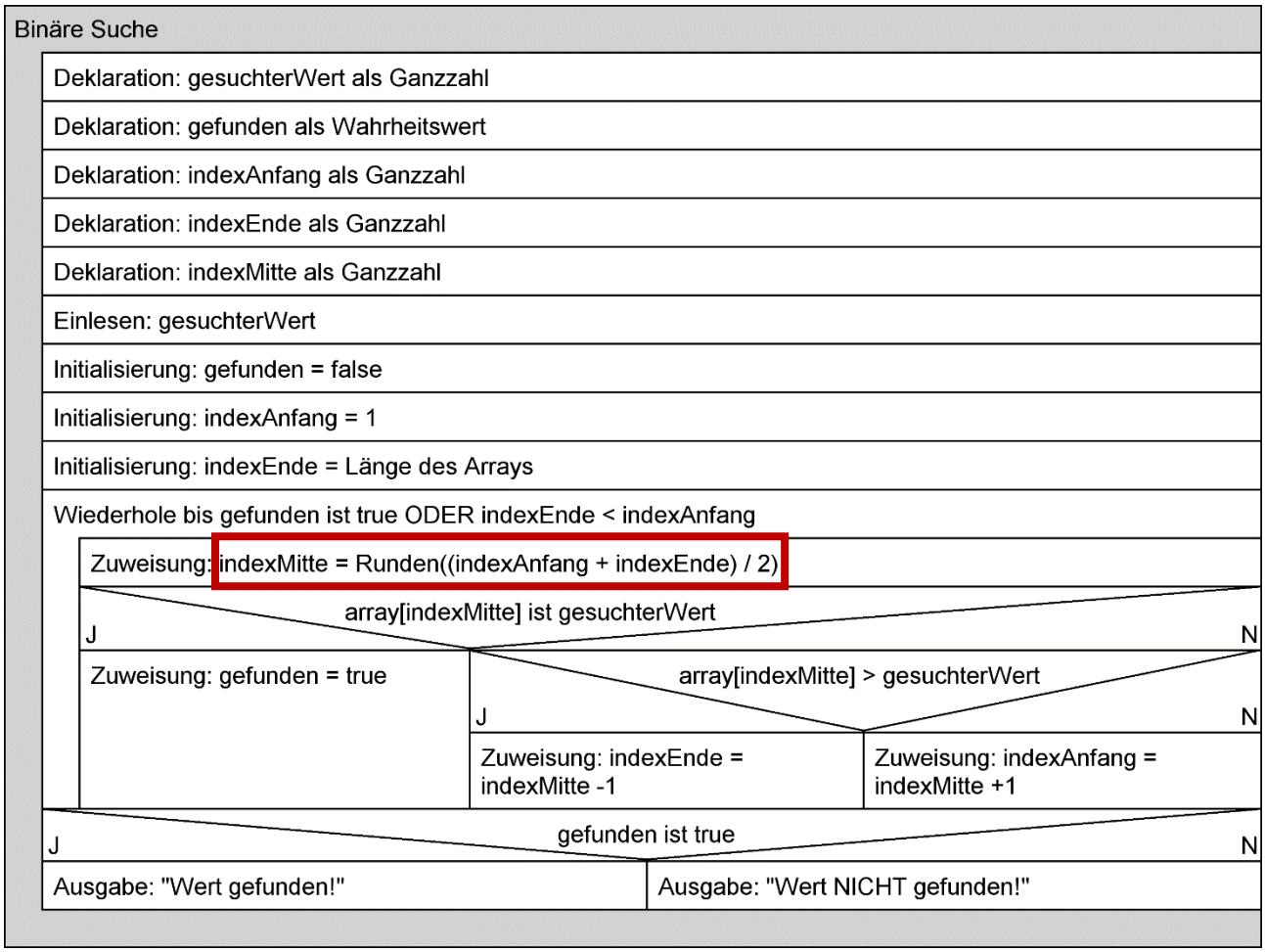
Ein Index muss ganzzahlig sein!

Die Elemente eines Arrays benötigen einen ganzzahligen Index!
Das heißt: array[**3,5**] gibt es nicht!

Das Problem lösen wir ganz einfach: Wenn wir *indexMitte* mit unserer Formel berechnen, dann **runden** wir das Ergebnis ganzzahlig!

Struktogramm: Binäre Suche

Wir nehmen die letzten Änderungen an unserem Struktogramm vor:



Aufgabe:

1. Suchen Sie im folgenden Array nach der Zahl **330**. Wenden Sie dabei den Algorithmus *Binäre Suche* an! Zeigen Sie dabei auch in jeder Zeile, welchen Wert die Variablen indexAnfang, indexEnde und indexMitte annehmen.

array[1]	array[2]	array[3]	array[4]	array[5]	array[6]	array[7]	array[8]
5	9	33	110	170	220	330	550

Verwenden Sie dafür die Vorlage *7500_BinaereSuche_Aufgabe1.docx*

Sie müssen also nichts programmieren, sondern zeigen, wie gesucht wird!

2. Suchen Sie im Array aus Aufgabe 1 nach der Zahl **7**, indem Sie zeigen, wie der Algorithmus *Binäre Suche* funktioniert. Verwenden Sie hierzu die Vorlage aus Aufgabe 1.

3. Öffnen Sie in Scratch das Projekt 7500-BinaereSuche-AUFGABE. Entwerfen Sie hier den Programmcode, mit dem Sie im Array nach einer PLZ suchen können!