

Programmiersprachenunabhängige Darstellung von Algorithmen

Zeichenkettenoperationen und Kontrollstrukturen in verschiedenen Sprachen

Sie haben inzwischen einige hilfreiche Operationen kennengelernt, um Algorithmen zur Verarbeitung von Zeichenketten zu entwerfen. Dabei haben Sie mit einer grafischen Programmiersprache oder mit Processing bzw. Java gearbeitet. Die Operationen, die die verschiedenen Programmiersprachen zur Verfügung stellen sind in ihrer Funktion sehr ähnlich, auch wenn sich die Notation unterscheidet. Dazu werden wir uns in den Aufgaben 1 und 2 noch einmal einen Überblick verschaffen.

Aufgabe 1:

- a) Tragen Sie in der Tabelle 1 ein, mit welchen Operationen und Methoden bzw. welchen Bausteinen Sie die jeweilige Aufgabe in Processing bzw. einer Ihnen bekannten grafischen Sprache umsetzen würden.

Bedeutung	Processing	grafische Sprache
Verbinden von zwei Zeichenketten zu einer		
Die Länge einer Zeichenkette bestimmen.		
Das Zeichen an einer bestimmten Position aus einer Zeichenkette auslesen		
Den Inhalt zweier Zeichenketten vergleichen		
Den ASCII-Wert zu einem Zeichen bestimmen.		
Das Zeichen zu einem ASCII-Wert bestimmen.		
Einen Text vom Anwender erfragen		
Einen Text ausgeben		

Tabelle 1: Operationen zur Verarbeitung von Zeichenketten in verschiedenen Programmiersprachen

Aufgabe 2: Texte, in denen die Vokale fehlen, lassen sich noch erstaunlich gut lesen. Probieren Sie es aus, indem Sie aus einem Text alle Vokale löschen.

Variante A (Einzelarbeit): Lösen Sie dieses Problem sowohl in Processing als auch in einer grafischen Programmiersprache.

Variante B (Partnerarbeit): Teilen Sie sich auf. Partner 1 verwendet Processing und Partner 2 verwendet eine grafische Sprache, um das Problem zu lösen.

Für beide Varianten: Vergleichen Sie anschließend Ihre Lösungen in den verschiedenen Sprachen. Versuchen Sie die algorithmischen Bausteine einander zuzuordnen und sortieren Sie die Unterschiede Ihrer Lösungen in zwei Kategorien ein:

1. Unterschiede, die sich dadurch ergeben, dass sie unterschiedliche Lösungsansätze gewählt haben. In diesem Fall sollte eine entsprechende Umsetzung auch in der jeweils anderen Sprache möglich sein.
2. Unterschiede, die sich aus sprachspezifischen Eigenheiten ergeben. In diesem Fall wäre eine exakte Umsetzung in der jeweils anderen Sprache nicht möglich.

In Aufgabe 1 und 2 haben Sie vermutlich festgestellt, dass Probleme im Bereich Zeichenkettenverarbeitung in den zwei betrachteten Programmiersprachen algorithmisch sehr ähnlich gelöst werden können. Dies gilt auch für Probleme aus anderen Bereichen. So kann z. B. ein Pongspiel sowohl mit Processing als auch mit einer grafischen Programmiersprache implementiert werden. Insbesondere die elementaren Kontrollstrukturen Anweisung, Verzweigung und Schleife finden wir in den meisten Programmiersprache wieder. Auch dazu verschaffen wir uns in der nächsten Aufgabe einen Überblick.

Aufgabe 3:

- a) Vervollständigen Sie die Tabelle 2 mit der Notation der Kontrollstrukturen in den unterschiedlichen Programmiersprachen. Ergänzen Sie ggf. weitere Kontrollstrukturen, die Ihnen wichtig erscheinen.

Bedeutung	Processing	grafische Sprache
Wertzuweisung an eine Variable		
Verzweigung	<pre>if (Bedingung) { Anweisungen; }else{ Anweisungen; }</pre>	
Verzweigung mit vielen Alternativen (Fallunterscheidung)		
Wiederhole-Schleife mit Eingangsbedingung		
Zählschleife		
Endlosschleife		

Tabelle 2: Kontrollstrukturen in unterschiedlichen Programmiersprachen

- b) Die Auswahl an Kontrollstrukturen variiert ein wenig. Diskutieren Sie, ob die Sprachen deshalb unterschiedlich mächtig sind. Das heißt, ob es Probleme gibt, die nur mit einer der beiden Sprachen gelöst werden können.

Programmiersprachenunabhängige Darstellung von Algorithmen

Für manche Problemklassen ist eine Sprache besonders geeignet, weil sie hilfreiche Zusatzfunktionen zur Verfügung stellt, mit denen eine Problemlösung kürzer aufgeschrieben werden kann. Auch die persönlichen Vorlieben sind unterschiedlich. So schreiben manche lieber, während andere das Zusammenbauen von Blöcken einfacher finden. Die Grundstruktur eines Algorithmus finden wir jedoch in jeder Programmiersprache wieder. Unabhängig von der gewählten Sprache benötigt man daher zunächst eine Idee, wie man algorithmisch Vorgehen kann, um ein Problem zu lösen. Wenn man diese Idee gefunden hat, kann man sie in einer beliebigen Programmiersprache umsetzen.

Damit sich Programmierer unabhängig von der bevorzugten Sprache über Algorithmen austauschen oder diese gemeinsam entwerfen können, erscheint es daher sinnvoll eine einheitliche Notationsform zu verwenden, die unabhängig von einer bestimmten Programmiersprache ist.

Eine solche Notationsform bieten die Nassi-Shneiderman-Diagramme¹, die nach ihren Erfindern Isaac Nassi und Ben Shneiderman benannt sind. Da sie die Struktur von Algorithmen darstellen, werden sie in der Regel etwas kürzer als *Struktogramme* bezeichnet. In der Tabelle 2 mit der Übersicht über die Kontrollstrukturen werden wir daher eine weitere Spalte für die programmiersprachenunabhängige Darstellung im Struktogramm ergänzen (s. Tabelle 3). Damit die Tabelle nicht zu groß wird, verzichten wir auf die Spalte für die grafische Programmiersprache. Eine entsprechende Zuordnung wäre aber natürlich ebenso möglich. Die Darstellung orientiert sich an den Vorgaben für das Abitur in Niedersachsen². Zum Zeichnen von Struktogrammen eignet sich z. B. das Programm *Structorizer*³.

Elementare Anweisungen werden im Struktogramm als Text in einen rechteckigen Kasten geschrieben. Die Wertzuweisung an eine Variable wird mithilfe eines Pfeils dargestellt. Andere Anweisungen können umgangssprachlich erfolgen. Wichtig ist, dass jede elementare Anweisung einen eigenen Kasten erhält, damit man weiß, welches die einzelnen Schritte sind, die später umgesetzt werden müssen. Wenn Sie sich unsicher sind, ob eine Anweisung elementar ist, gibt es zwei mögliche Vorgehensweisen. Eine Möglichkeit ist, sich an den Programmiersprachen, die Sie kennen, zu orientieren. Wenn sich die Anweisung mithilfe einer Operation bzw. eines Bausteins darstellen lässt, ist sie vermutlich elementar. Eine zweite Möglichkeit ist, zu überlegen, ob sich die Anweisung in mehrere Schritte zerlegen lässt. Dann ist sie noch nicht elementar und sollte weiter aufgeteilt werden. Damit bei komplexeren Problemen, das Struktogramm nicht zu unübersichtlich wird, möchte man für einen ersten groben Entwurf vielleicht bewusst mehrere Schritte in einer Anweisung zusammenfassen. Das sollte dann aber vorher abgesprochen werden. Sollen diese Anweisungen in einen eigenen Block oder eine eigene Methode ausgelagert werden, kann dies auch im Struktogramm kenntlich gemacht werden, indem der Kasten für die Anweisung links und rechts durch eine doppelte Linie begrenzt wird.

¹ DIN 66261 Informationsverarbeitung; Sinnbilder für Struktogramme nach Nassi-Shneiderman
<https://www.din.de/de/wdc-beuth:din21:1255956>

² Niedersächsisches Kultusministerium (Hrsg.) (2017) Ergänzende Hinweise zum Kerncurriculum Informatik für die gymnasiale Oberstufe am Gymnasium, an der Gesamtschule sowie für das Kolleg.
http://db2.nibis.de/1db/cuvo/datei/INF_Ergaenzende-Hinweise_GO_2018.pdf [Datum des Zugriffs: 18.07.2019]

³ Bob Fisch (2009). Structorizer. <https://structorizer.fisch.lu/index.php?include=news>. [Datum des Zugriffs: 18.07.2019]


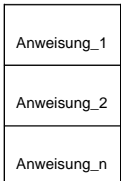
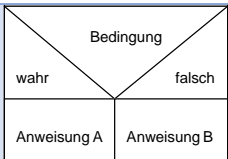
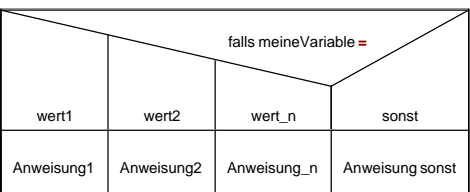
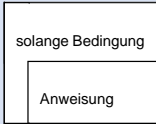
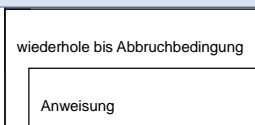
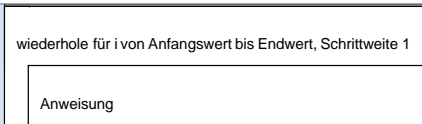
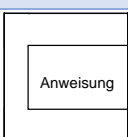
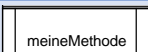
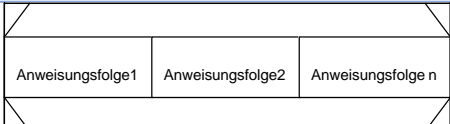
Bedeutung	Processing	Struktogramm
Wertzuweisung an eine Variable	<code>meineVariable = 3;</code>	
Folge von Anweisungen	<code>Anweisung_1; Anweisung_2; Anweisung_n;</code>	
Verzweigung	<code>if (Bedingung) { AnweisungA; } else { AnweisungB; }</code>	
Verzweigung mit vielen Alternativen (Fallunterscheidung)	<code>switch (meineVariable) { case wert1: Anweisung1; break; case wert2: Anweisung2; break; case wert_n: Anweisung_n; break; default: Anweisung sonst; }</code>	
Wiederhole-Schleife mit Eingangsbedingung	<code>while (Bedingung) { Anweisung; }</code>	
Wiederhole-Schleife mit Abbruchbedingung	<code>while (!AbbruchBedingung) { Anweisung; }</code>	
Zählschleife	<code>for(int i = Anfangswert; i <= Endwert; i++){ Anweisung; }</code>	
Endlosschleife	<code>while(true) { Anweisung; }</code>	
Aufruf einer eigenen Methode	<code>meineMethode();</code>	
Parallele Abläufe		

Tabelle 3: Programmiersprachenunabhängige Darstellung der Kontrollstrukturen im Struktogramm

Innerhalb der Verzweigungen und Schleifen ist in Tabelle 3 immer nur eine exemplarische Anweisung enthalten. Hier kann auch eine Folge von Anweisungen stehen. Bei der Verzweigung sieht man statt

der Beschriftung der beiden Zweige mit *wahr* und *falsch* auch häufig *ja* und *nein* oder nur *w* und *f*. Wenn der Zweig, für den Fall, dass die Bedingung nicht erfüllt ist, nicht benötigt wird, bleibt der Kasten für die Anweisung in diesem Zweig einfach leer.

Manchmal fehlt bei der Zählschleife die Angabe der Schrittweite, dann wird als Standardwert die Schrittweite 1 verwendet. Alle anderen Schrittweiten müssen aber somit unbedingt angegeben werden.

In Processing programmieren wir normalerweise keine Algorithmen, die parallel ablaufen. In den grafischen Programmiersprachen ist das hingegen häufiger der Fall. Tabelle 3 enthält daher auch das Symbol für parallele Abläufe.

Betrachten wir als Beispiel einen Algorithmus, der einen Text, den der Anwender eingibt, rückwärts wieder ausgibt. Schauen wir uns zu diesem Problem zunächst ein Struktogramm an.

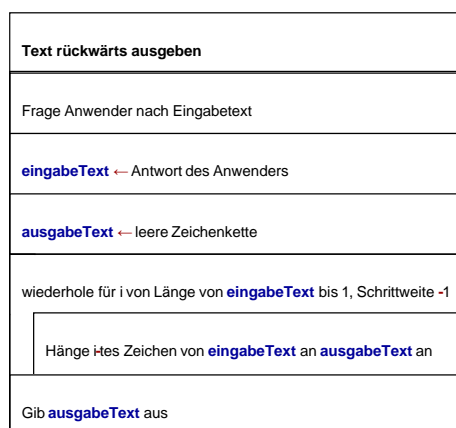


Abbildung 1: Struktogramm zu "Text rückwärts ausgeben"

Dieses Struktogramm lässt sich nun sowohl in einer grafischen Sprache wie *Snap!*⁴ als auch in einer textbasierten Sprache wie Processing implementieren.



Abbildung 2: Implementierung von "Text rückwärts ausgeben" in Snap!

Im Struktogramm wurde von einer natürlichen Nummerierung der Zeichen, die bei 1 anfängt, ausgegangen. Bei einer Implementierung in Processing müssen der Start- und der Endwert daher entsprechend angepasst werden.

⁴ Snap! wird von der University of California, Berkeley zur Verfügung gestellt: <https://snap.berkeley.edu>

```
1 String eingabeText = showInputDialog("Bitte gib einen Text ein");
2 String ausgabeText = "";
3 for(int i = eingabeText.length()-1; i >= 0; i--){
4     ausgabeText = ausgabeText + eingabeText.charAt(i);
5 }
6 System.out.println(ausgabeText);
```

Beispiel 1: Implementierung von "Text rückwärts ausgeben" in Processing

Aufgabe 4: Erstellen Sie ein Struktogramm für die Verschlüsselung mithilfe des Caesar-Verfahrens.

Aufgabe 5: Algorithmen begegnen uns auch im Alltag. Bei vielen Tätigkeiten verwenden wir Strategien, die sich in Form eines Algorithmus formulieren lassen.

- Stellen Sie sich z. B. einen Korb mit frisch gewaschener Wäsche vor, der Waschlappen, Handtücher und Badetücher enthält. Die Waschlappen sollen alle auf einen Stapel gelegt werden. Die Handtücher werden einmal längs in der Mitte gefaltet und dann zusammengerollt. Die Badetücher werden einmal längs in der Mitte gefaltet und dann zweimal in der Mitte parallel zur kürzeren Seite. Erstellen Sie ein Struktogramm, nach dessen Ausführung die Wäsche vollständig zusammengelegt ist.
- Besonders Spaßig wird es, wenn der Wäschekorb auch Socken enthält. Zu einem Socken muss systematisch der zweite Socken gesucht werden. Erst dann kann das Sockenpaar zusammengelegt werden. Erstellen sie ein geeignetes Struktogramm. Sie dürfen auch einen zweiten Wäschekorb zur Hilfe nehmen.
- Erstellen Sie ein Struktogramm zu einem selbst gewählten Vorgang, der Ihnen aus dem Alltag bekannt ist.

Aufgabe 6:

- Dem Anwender werden im Programmfenster eine Frage und zwei Antwortmöglichkeiten präsentiert. Je nachdem, ob der Anwender die richtige oder die falsche Antwort anklickt, soll *richtig* oder *falsch* angezeigt werden. Erstellen sie ein entsprechendes Struktogramm.
- Implementieren Sie Ihr Struktogramm in einer Programmiersprache, die Ihnen geeignet erscheint.

Aufgabe 7:

- Dem Anwender werden insgesamt zehn Rechenaufgaben mithilfe von Zufallszahlen gestellt. Wenn der Anwender die Aufgabe richtig löst, wird der Punktezähler um 1 hochgezählt. Am Ende erfährt der Anwender, wie viele Aufgaben er richtig gelöst hat. Erstellen Sie ein entsprechendes Struktogramm.
- Verändern Sie Ihr Struktogramm so, dass dem Anwender solange zufällige Rechenaufgaben gestellt werden, bis er fünf Aufgaben richtig gelöst hat.
- Implementieren Sie Ihre Struktogramme in einer Programmiersprache, die Ihnen geeignet erscheint.

Aufgabe 8: Im Leitfaden zur Zeichenkettenverarbeitung sollte in Aufgabe 12 das Problem gelöst werden, dass in einem Text die Zahlen 1 bis 12 durch das entsprechende Zahlwort ersetzt werden, alle anderen Zahlen aber erhalten bleiben. Eine konkrete Implementierung zur Lösung dieses Problems ist relativ anspruchsvoll, so dass Sie diese Aufgabe vielleicht noch nicht vollständig gelöst haben. Hier bietet es sich daher an, zunächst ein grobes Struktogramm zu entwerfen und dieses Schrittweise zu verfeinern. Abbildung 3 zeigt einen ersten Entwurf eines groben Struktogramms für

dieses Problem. Vervollständigen und verfeinern Sie das Struktogramm, indem Sie z. B. die umgangssprachlich formulierten Bedingungen in elementare Schritte zerlegen.

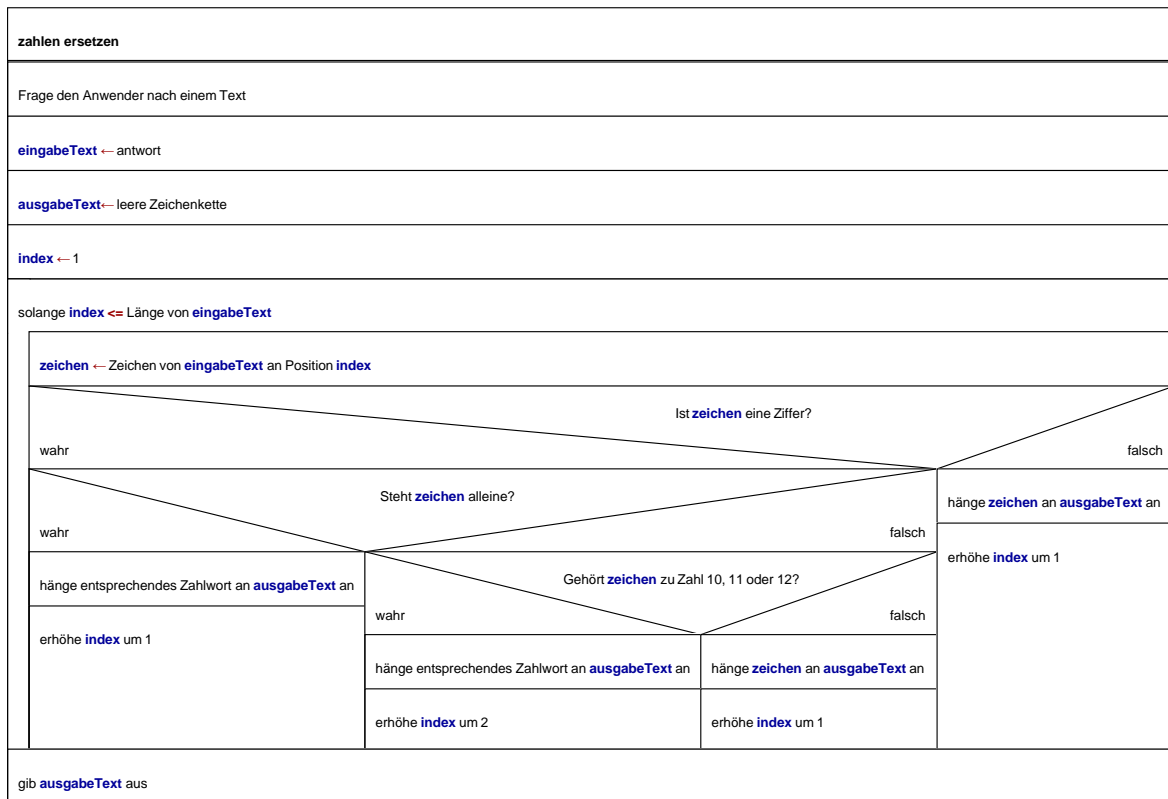


Abbildung 3: grobes Struktogramm zum Ersetzen der Zahlen 1 bis 12 in einem Text

Lizenz

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](#). Sie erlaubt Bearbeitungen und Weiterverteilung des Werks unter Nennung meines Namens und unter gleichen Bedingungen, jedoch keinerlei kommerzielle Nutzung.

Für die korrekte Ausführbarkeit der Quelltexte in diesem Leitfaden wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.