Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2016/17

Prof. Dr. Michael Rohs, Oliver Beren Kaul, M.Sc.

Aufgabensammlung Programmieren I

Dieses Übungsblatt soll bei der Vorbereitung auf den Abschlusstest helfen. **Ihre Lösung soll nicht dem Tutor vorgestellt und nicht in das Abgabesystem hochgeladen werden.** Schauen Sie sich zur Vorbereitung auf den Abschlusstest auf jeden Fall auch erneut die vorhergehenden Übungen sowie die Vorlesungsfolien an.

Diese Aufgabensammlung dient nur zur Orientierung. **Im Abschlusstest können auch andere als die hier vorgestellten Aufgabentypen vorkommen**.

Die Musterlösung dieser Aufgabensammlung wird Mitte Februar über Stud.IP zur Verfügung gestellt. Wir empfehlen, die Musterlösung nur im Notfall als Hilfsmittel zu benutzen, da der Lerneffekt stark eingeschränkt wird.

Allgemeine Hinweise zur Kompilierung

Die Programming I Java Library ist als jar-Datei in Prog1TaskCollection.zip enthalten. Die Kompilierung erfolgt auf der Kommandozeile. Die Programming I C Library muss, wie bei den vorhergehenden C-Übungen, über den Pfad erreichbar sein. Im Abschlusstest werden alle notwendigen Bestandteile in kompilierter Form vorhanden sein. Die Dokumentation wird ebenfalls zur Verfügung stehen.

Anweisungen zum Kompilieren und Ausführen finden Sie in der jeweiligen Template-Datei jeder Aufgabe.

Die Dokumentation der Programming I C Library finden sie unter:

http://hci.uni-hannover.de/files/prog1lib/files.html

Die Dokumentation der Programming I Java Library finden sie unter:

http://hci.uni-hannover.de/files/prog1javalib/index.html

Aufgabe 1: four_sorted_digits.c

Die Template-Datei für diese Aufgabe ist `four_sorted_digits.c`. Implementieren Sie die Funktion `bool four_sorted_digits(String s)`. Diese Funktion soll `true` zurückgeben, wenn `s` mindestens 4 hintereinander stehende und aufsteigend sortierte Dezimalziffern enthält. Sonst gibt die Funktion `false` zurück.

Hinweis: Zur Erinnerung: String ist definiert als typedef String char*. Ein String ist ein char-Array mit terminierendem 0-Zeichen. Daher können Indizes verwendet werden, um auf die einzelnen Buchstaben zuzugreifen: `s[i]`.

Aufgabe 2: sequence_count.c

Die Template-Datei für diese Aufgabe ist `sequence_count.c`.

a) Implementieren Sie die Funktion `int sequence_count(String s, String t)`. Diese Funktion soll die Anzahl an Positionen zurückgeben, an denen t in s vorkommt.
b) Korrigieren Sie die Funktion `bool parentheses_correct(String s)`. Diese soll genau dann wahr zurückgeben, wenn in s zu jeder öffnenden Klammer eine korrespondierende schließende Klammer existiert (und umgekehrt). Die Funktion darf andere Zeichen als '(' und ')' ignorieren.
c)

Aufgabe 3: center_or_zero.c

Die Template-Datei für diese Aufgabe ist `center_or_zero.c`. Implementieren Sie die Funktion `double center_or_zero(DoubleList *list)`. Diese Funktion soll das Element an der mittleren Position der Liste zurückgeben. Wenn die Liste eine gerade Anzahl an Elementen enthält, soll die Funktion den Wert 0 zurückgeben. In dieser Aufgabe dürfen die Listen der prog1lib nicht verwendet werden.

Aufgabe 4: palindrome.c

Die Template-Datei für diese Aufgabe ist `palindrome.c`. Ein Palindrom ist ein Wort, welches von hinten nach vorne genauso gelesen werden kann wie von vorne nach hinten.

a) Implementieren Sie die Funktion `int is_in_alphabet(char c)`. Diese Funktion soll eine 1 zurückgeben, falls ein `char` im Alphabet vorhanden ist, ansonsten eine 0.
b) Implementieren Sie die Funktion `int is_palindrome(char* s)`. Diese Funktion soll eine 1 zurückgeben, falls ein String ein Palindrom ist, ansonsten eine 0. Dabei sollen Zeichen ignoriert werden, welche nicht im Alphabet zu finden sind.
c) Implementieren Sie die Funktion `int contains_palindrome(char* s, int minimumPalindromeSize)`. Diese Funktion soll 1 zurückgeben, falls ein String mindestens ein Palindrom der Größe `minimumPalindromeSize` oder größer enthält.

Hinweise: Aufgabenteil c ist etwas komplexer. Der String muss aufgeteilt werden. Initialisieren Sie für jeden Teilstring ein neues char Array auf dem Stack: `char test[count];`. Nutzen Sie dann die Funktion `memcpy` mit der Signatur `memcpy(void* destination, const void* source, size_t num)`, um einen Teilstring zu kopieren und diesen dann mit der vorher implementierten Funktion `is_palindrome` zu untersuchen.

Aufgabe 5: NodesEqualToParent.java

Die Template-Datei für diese Aufgabe ist `NodesEqualToParent.java`. Implementieren Sie die Methode `int numberOfNodesThatAreEqualToTheirParent()`. Diese Methode soll die Anzahl der Knoten des Binärbaums zurückgeben, die den gleichen Wert haben, wie ihre Elternknoten.

Hinweis: Es kann zweckmäßig sein, hierfür eine (rekursive) Hilfsmethode in der Klasse `Tree` oder der Klasse `Node` zu implementieren.

Aufgabe 6: IsSearchTree.java

Die Template-Datei für diese Aufgabe ist `IsSearchTree.java`. Implementieren Sie die Methode `boolean isSearchTree()`. Diese Methode soll `true` zurückgeben, wenn es sich um einen Suchbaum handelt. Andernfalls soll sie `false` zurückgeben.

Hinweis: Ein binärer Baum ist dann ein Suchbaum, wenn für jeden Knoten gilt, dass die Werte im linken Unterbaum alle kleiner sind als der Wert des Knotens und die Werte im rechten Unterbaum alle größer sind als der Wert des Knotens.

Hinweis: Es kann zweckmäßig sein, hier eine (rekursive) Hilfsmethode in der Klasse `SearchTree` oder der Klasse `SearchNode` zu implementieren.

Aufgabe 7: TableOfContentsTree.java

Die Template-Datei für diese Aufgabe ist `TableOfContentsTree.java`. Die Klassen Tree und Node repräsentieren das Inhaltsverzeichnis einer Bachelorarbeit.

a) Implementieren Sie die Methode `String Tree.tableOfContents()`. Diese soll für den leeren Baum einen leeren String zurückgeben und ansonsten die Methode `Node.tableOfContents("")` aufrufen.
b) Implementieren Sie die Methode String `Node.tableOfContents(String id)`. Diese soll das Inhaltsverzeichnis als String ausgeben. Der Testaufruf zeigt das erwartete Format.

Hinweis: Der Parameter `id` repräsentiert die Nummerierung eines Kapitels oder Unterkapitels, z.B. 2.3.

Aufgabe 8: DirectoryTree.java

Die Template-Datei für diese Aufgabe ist `DirectoryTree.java`. Die Klassen Tree und Node repräsentieren ein hierarchisches Dateisystem.

a) Implementieren Sie die Methode String `Tree.directory()`. Diese soll für den leeren Baum einen leeren String zurückgeben und ansonsten die Methode `Node.directory("")` aufrufen, um eine String-Repräsentation des Verzeichnisses auszugeben.
b) Implementieren Sie die Methode String `Node.directory(String path)`. Diese soll den Verzeichnisbaum als String ausgeben. Der Testaufruf zeigt das erwartete Format.

Aufgabe 9: Set.java

Die Template-Datei für diese Aufgabe ist `Set.java`. Die Klasse Item repräsentiert eine LinkedList im Set.

a) Implementieren Sie die Methode `String toString()`, welche einen String gemäß der dafür vorhandenen Testfälle generiert.
b) Implementieren Sie die Methode Set `endsWithFilter(String endingString)`, welche ein neues Set zurückgibt, in dem nur Elemente liegen, welche mit dem übergebenen `endingString` enden.
c) Implementieren Sie die Methode `String maximumAverageCharacterValue()`, welche den String im Set zurückgibt, welcher den höchsten durchschnittlichen Wert einzelner Characters hat.

Hinweis: Benutzen Sie die Methode `String.charAt(index)`, um einen `char` aus einem String zu extrahieren.

Aufgabe 10: HappyNumbers.java

Die Template-Datei für diese Aufgabe ist `HappyNumbers.java`. „Glückliche Zahlen" sind natürliche Zahlen, die nach einem Siebprinzip erzeugt werden[1].

Gestartet wird mit allen natürlichen Zahlen.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 …
```

Die Zahl 1 ist als „glücklich" definiert. Die Folge wird schrittweise verändert, indem zuerst jede zweite Zahl gestrichen wird.

```
1 * 3 * 5 * 7 * 9 ** 11 ** 13 ** 15 ** 17 ** 19 ** …
```

Danach befindet sich die Zahl 3 an Stelle zwei, was bedeutet, dass im nächsten Schritt jede dritte, **noch existierende** Zahl gestrichen wird.

```
1 - 3 - * - 7 - 9 -- ** -- 13 -- 15 -- ** -- 19 -- …
```

Die nächste Zahl wäre 7 (An Stelle drei), also muss danach jede siebte **noch existierende** Zahl gestrichen werden, danach jede 9., 13., 15., etc.

Am Ende kommt die Folge der „Glücklichen Zahlen" dabei heraus:

1, 3, 7, 9, 13, 15, 21, 25, 31, 33, 37, 43, 49, 51, 63, 67, 69, 73, 75, 79, 87, 93, 99, …

Implementieren Sie die Methode `String calculateList(int upperLimit)`, welche eine Liste mit „Glücklichen Zahlen" gemäß der oben genannten Regeln generiert. Eine Ausgabe der Liste und das erwartete Resultat sind bereits im Template implementiert.

Hinweise:

- Benutzen Sie unbedingt Listen. Diese Aufgabe könnte man auch mit Arrays implementieren, allerdings ist dies erheblich komplexer.
- Die Komplexität dieser Aufgabe ist höher als die durchschnittliche Komplexität der Aufgaben im Abschlusstest

[1] https://de.wikipedia.org/wiki/Gl%C3%BCckliche_Zahl

Weiteres Training

Im Internet finden sich z. B. unter dem Suchbegriff „programming challenges" viele Webseiten, welche weiterführende Übungen in vielen Programmiersprachen anbieten. Teilweise enthalten diese Seiten webbasierte Editoren, welche den übermittelten Code auf einem Server laufen lassen und über Testfälle direktes Feedback zur Korrektheit geben können.

Beispielhaft seien hier genannt:

- Project Euler - https://projecteuler.net/
- Coder byte - https://www.coderbyte.com/
- CodingBat - http://codingbat.com/java
- LeetCode - https://leetcode.com/problemset/algorithms/