Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2018/19

Prof. Dr. Michael Rohs, Tim Dünte, M.Sc.

Assignment 14 – 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 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

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 char* String`. 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.

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 `true` zurückgeben, falls ein `char` im Alphabet vorhanden ist, ansonsten `false`.

b) Implementieren Sie die Funktion `int is_palindrome(char* s)`. Diese Funktion soll `true` zurückgeben, falls ein String ein Palindrom ist, ansonsten `false`. Dabei sollen Zeichen ignoriert werden, welche nicht im Alphabet zu finden sind.

c) Implementieren Sie die Funktion `int contains_palindrome(char* s, int minimum_palindrome_size)`. Diese Funktion soll genau dann `true` zurückgeben, falls ein String mindestens ein Palindrom der Größe `minimum_palindrome_size` 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: nodes_equal_to_parent.c

Die Template-Datei für diese Aufgabe ist `nodes_equal_to_parent.c`. Implementieren Sie die Methode `int number_of_nodes_that_are_equal_to_their_parent(Tree* tree)`. 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 zu implementieren.

Aufgabe 6: is_search_tree.c

Die Template-Datei für diese Aufgabe ist `is_search_tree.c`. Implementieren Sie die Methode `bool is_search_tree(Tree* tree)`. 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 zu implementieren.

Hinweis: Es kann hilfreich sein das Maximum bzw. das Minimum von Teilbäumen zu bestimmen um obige Bedingung zu testen.

Weiteres Training

Im Internet finden sich z. B. unter dem Suchbegriff „programming challenges" viele Webseiten, die weiterführende Übungen in vielen Programmiersprachen anbieten. Teilweise enthalten diese Seiten Web-basierte 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/