Human-Computer Interaction Group

Leibniz Universität Hannover

Übungsblatt 14

Programmieren 1 - WiSe 22/23

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

Dieses Übungsblatt kann auf freiwilliger Basis bearbeitet werden. Das Übungsblatt kann nicht über das Abgabesystem abgegeben werden. Es erfolgt keine Korrektur durch Ihren Tutor.

Dieses Übungsblatt soll bei der Vorbereitung auf den Abschlusstest helfen. 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.

Hinweis: prog1lib

Die Programmieren I C Bibliothek muss, wie bei den vorhergehenden C-Übungen, über den Pfad erreichbar sein. Im Abschlusstest werden alle notwendigen Bestandteile 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 prog1lib finden Sie unter der Adresse:
https://postfix.hci.uni-hannover.de/files/prog1lib/

Aufgabe 1: palindrome.c

Ein Palindrom ist ein Wort, welches von hinten nach vorne genauso gelesen werden kann wie von vorne nach hinten.

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

a) Implementieren Sie `int is_in_alphabet(char c)`. Diese Funktion soll `true` zurückgeben, falls ein `char` im Alphabet vorhanden ist, ansonsten `false`.

b) Implementieren Sie `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 `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.

Hinweis: 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(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 2: 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.

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

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.

Aufgabe 3: balance_donkey.c

Die beiden Satteltaschen eines Esels (auf der linken und rechten Seite des Eselrückens) sollen so mit Gegenständen (items) beladen werden, dass die Gesamtgewichte der Gegenstände in den Satteltaschen möglichst gut ausbalanciert sind.

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

Implementieren Sie die Funktion `bool *balance_donkey(int *items, int n)`, die eine möglichst gute Ausbalancierung berechnet. Nehmen Sie an, dass die Gewichte der Gegenstände positiv sind. Hinweis: Probieren Sie einfach alle Möglichkeiten aus. Verwenden Sie dazu den bereitgestellten Iterator.

Aufgabe 4: 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 5: 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 also ein char-Array mit terminierendem 0-Byte. Daher können Indizes verwendet werden, um auf die einzelnen Buchstaben zuzugreifen: `s[i]`.

Die folgenden Aufgaben bieten noch weitere Möglichkeiten Ihre Programmierkenntnisse zu testen und zu erweitern. Die Aufgaben sind deutlich aufwändiger als die Aufgaben, die Sie in der Klausur erwarten können. Sie erhalten hier aber noch einmal einen breitgefächerten Einblick in mögliche Problemstellungen, die sie in kleinerer Form auch in der Klausur erwarten können.

Zusatzaufgabe: Listen

Implementieren Sie selber eine Datenstruktur vom Typ Liste. Folgen Sie dafür den einzelnen Schritten dieser Aufgabe. Es gibt kein Template und auch keine Musterlösung.

a) Erstellen Sie eine Struktur, die ein Listenelement repräsentiert. Ein Listenelement soll einen Pointer auf eine Zeichenkette speichern (`char *`) und mit einem Pointer auf ein nachfolgendes Listenelement verweisen. Erstellen Sie zusätzlich eine Struktur `ListHead`, die auf das erste Element der Liste verweist und auf das letzte Element. Zusätzlich soll die Struktur `ListHead` einen ganzzahligen Wert beinhalten, der die Länge der Liste speichert.

b) Erstellen Sie eine Konstruktorfunktion, die ein Listenelement initialisiert. Die Funktion soll wie bisher den Wert sowie ein Pointer auf das nachfolgende Element übergeben bekommen. Die übergebene Zeichenkette soll kopiert werden und der dafür nötige Speicher soll dynamisch allokiert werden. Erstellen Sie auch eine Konstruktorfunktion, die Ihnen einen dynamisch allokierten initialisierten Listenkopf erstellt.

c) Schreiben Sie eine Funktion, die den allokierten Speicher der Liste sowie abhängig von einem übergebenen Wahrheitswert auch den Speicher der gespeicherten Zeichenketten freigibt.

d) Schreiben Sie eine Funktion, die ein neues Listenelement vorne in der Liste einfügt. Schreiben Sie eine Funktion, die ein neues Listenelement hinten in der Liste einfügt. Die Listenlänge im Listenkopf soll entsprechend inkrementiert werden und die Pointer sollen aktualisiert werden.

e) Schreiben Sie eine Funktion, die Ihnen die Länge der Liste zurückgibt. Warum ist die Implementierung dieser Funktion trivial?

f) Schreiben Sie eine Funktion, die ein neues Listenelement an einer bestimmten Stelle der Liste einfügt.

g) Schreiben Sie eine Funktion, die ein Listenelement an einer bestimmten Stelle der Liste entfernt.

h) Schreiben Sie eine Funktion, die ein neues Listenelement sortiert in eine Liste einfügt. Gehen Sie bei dem Aufruf dieser Funktion davon aus, dass die Liste sortiert ist.

i) Schreiben Sie eine Funktion, die zwei Listen auf Gleichheit überprüft.

j) Implementieren Sie den Mergesort Algorithmus um die Liste zu sortieren.

k) Schreiben Sie eine Funktion, die prüft ob eine gegebene Zeichenkette in der Liste vorhanden ist und den Index zurückgibt.

l) Implementieren Sie einen Iterator für die Liste.

Zusatzaufgabe: Zeichenketten

Implementieren Sie verschiedene Funktionen, die Zeichenketten verarbeiten. Es gibt kein Template und auch keine Musterlösung zu dieser Aufgabe. Beachten Sie für alle Aufgaben, dass Sie nur genau so viel Speicher allokieren, wie Sie auch benötigen. Beachten Sie auch, dass Zeichenketten mit `\0` terminiert werden und dass Sie für dieses Zeichen auch Speicher reservieren müssen. Lösen Sie die Aufgaben ohne die Bibliotheksfunktionen der Programmieren 1 Bibliothek oder die C-String Funktionen (bspw. `strcpy`) zu nutzen.

a) Implementieren Sie eine Funktion, die eine Zeichenkette auf einen bestimmten Inhalt prüft. Die Funktion soll dann `true` zurückgeben, wenn nur zusammenhängende Ziffernfolgen, deren Länge ein Vielfaches von 3 beträgt, in der Zeichenkette auftauchen. Ansonsten gibt die Funktion `false` zurück.

   Beispiele sind: `"123"`, `"ab c de 123456"` oder `"123 asdbderb 890"`.

b) Schreiben Sie eine Funktion `find_and_replace(char* text, char* replace, char* replacement)`, die eine Zeichenkette übergeben bekommt und eine Kopie von `text` zurückgibt in der die Zeichenfolge `replace` durch `replacement` ersetzt wurde.

c) Schreiben Sie eine eigene Funktion, die zwei Zeichenketten hintereinander fügt und einen Pointer auf diese Zeichenkette zurückgibt.

d) Schreiben Sie eine Funktion `split_string(char* to_split, char split)`, die einen Pointer auf ein dynamisch allokiertes Array zurückgibt in dem sich Pointer auf die Teilzeichenketten befinden. Beispielsweise würden `to_split = "abc.bdfg"` und `split = "."` ein Array zurückgeben in dem zwei Pointer vom Typ `char *` sind. Der erste würde auf `"abc"` zeigen und der zweite auf `"bdfg"`. Die Eingabezeichenkette soll dabei nicht verändert werden.

e) Schreiben Sie eine Funktion, die die Länge für eine gegebene Zeichenkette zurückgibt.

f) Schreiben Sie eine Funktion, die die größten gemeinsame zusammenhängende Zeichenkette zurückgibt, die in zwei Zeichenketten enthalten ist. Zum Beispiel würde `"Programmieren 1 macht Spass"` und `"Programmieren 1 macht keinen Spass"` als Ergebnis `"Programmieren 1 macht "` zurückgeben.

g) Schreiben Sie eine Funktion, die bestimmte Wörter aus einer Zeichenkette entfernt und durch Anfangsbuchstabe*** ersetzt. Die ursprüngliche Zeichenkette darf nicht verändert werden. Die Funktion soll eine Zeichenkette übergeben bekommen sowie einen Zeiger auf ein Array von Zeigern auf Zeichenketten. Beispiel: Text: `"Werbung ist doof"`, Wörter die ersetzt werden sollen: `"doof"`, `"bescheuert"`, `"idiotisch"` -> Ergebnis: `"Werbung ist d***"`.

Zusatzaufgabe: Arrays

Implementieren Sie verschiedene Funktionen, die Arrays verarbeiten. Es gibt kein Template und auch keine Musterlösung zu dieser Aufgabe.

a) Schreiben Sie eine allgemeine Funktion `copy`, die einen Pointer vom Typ `void *` und eine Anzahl an Bytes übergeben bekommt und einen `void *` Pointer zurückgibt, der auf eine Kopie des Inhalts vom übergebenen Pointer zeigt.

b) Schreiben Sie eine Funktion, die für ein Array von Doubles den Mittelwert berechnet.

c) Schreiben Sie eine Funktion, die die Werte in einem Integer Array so vertauscht, dass das 1. Element mit dem letzten Element getauscht wird, das 2. mit dem vorletzten Element getauscht wird, usw.

d) Schreiben Sie je eine Funktion, die das Maximum, das Minimum und das mittlere Element (Median) in einem Integer Array findet und zurückgibt.

e) Schreiben Sie eine Funktion, die eine $N \times N$ Matrix mit Double Werten transponieren kann. Die zurückgegebene transponierte Matrix soll dynamisch allokiert sein.

f) Schreiben Sie eine Funktion, die zwei gleichlange Vektoren (Double Arrays) sowie deren Länge übergeben bekommt und das Skalarprodukt berechnet.

g) Schreiben Sie einen Iterator für Arrays.

h) Schreiben Sie eine Funktion, die zwei Double Arrays ($N \times M$) miteinander vergleicht und prüft ob beide die gleichen Werte enthalten.

i) Schreiben Sie eine `map` Funktion für 2-dimensionale Double Arrays ($N \times M$). `map` soll eine Mapping-Funktion übergeben bekommen, die auf jedes Element angewendet wird. Die Mapping Funktion soll eine dynamisch allokierte veränderte Kopie der Elemente zurückgeben. Implementieren Sie als Mapping-Funktion eine binäre Funktion, die 0 zurückgibt, wenn der Array Wert kleiner ist als ein übergebener Parameter oder 1 zurückgibt, wenn der Wert größer oder gleich dem Parameter ist.

j) Schreiben Sie eine einfache Funktion die Double Arrays sortiert. Dabei sollen solange Elemente getauscht werden bis das Arrays sortiert ist. Beginnen Sie mit dem ersten Element und vergleichen Sie es mit dem zweiten Element. Falls das zweite Element kleiner als das erste ist tauschen Sie beide. Vergleichen Sie dann das erste Element mit dem dritten Element usw. Vergleichen Sie danach das zweite Element mit jedem anderen Element usw.

k) Schreiben Sie eine Funktion `split_array`, die ein Array von Integern, ein zweites Arrays von Integern mit Indices bekommt und ein Array mit Pointern auf Teil-Arrays zurückgibt. Das Array soll an jedem gegebenen Index aufgeteilt werden. Die Teilarrays werden dann von der Funktion zurückgegeben. Beispiel: Array: `[1, 2, 3, 4]`, Indices Array: `[1, 2]` und Rückgabe: `[[1], [2], [3, 4]]`. Allokieren Sie den Speicher dynamisch und verändern Sie die übergebenen Arrays nicht.

Zusatzaufgabe: Bäume

Implementieren Sie verschiedene Funktionen, die Bäume verarbeiten. Es gibt kein Template und auch keine Musterlösung zu dieser Aufgabe.

a) Erstellen Sie eine Struktur `TreeNode`, die einen Knoten eines Baumes repräsentieren soll. Der Knoten soll einen Pointer enthalten, der auf Zeichenketten zeigen kann. Zudem soll es einen linken und einen rechten Nachfolger geben.

b) Erstellen Sie eine weitere Struktur, die `Tree` heißt. Diese Struktur soll einen Zeiger `root` haben, der auf die Wurzel des Baumes zeigt, sowie einen `int count`, der die Anzahl an Knoten im Baum enthält, sowie einen `int depth`, dass die maximale Baumtiefe speichert.

c) Schreiben Sie eine Funktion, die Ihnen einen neuen Baumknoten zurückgibt, der mit `xmalloc` allokiert ist. Schreiben Sie auch eine Konstruktorfunktion, die Ihnen einen neuen `Tree` zurückgibt. Beide Funktionen sollen Speicher für die Strukturen dynamisch allokieren.

d) Schreiben Sie eine Funktion zum Einfügen einer neuen Zeichenkette, bestehend aus den Zeichen a-z und A-Z. Die Funktion soll eine Zeichenkette, sowie einen Pointer auf einen Tree übergeben bekommen und einen neuen Knoten sortiert einfügen. Die Zeichenkette soll kopiert werden. Groß- und Kleinschreibung sollen bei der Sortierung keine Rolle spielen. Alphabetisch früher auftretende Worte sollen in den linken Teilbaum eingefügt werden. Alphabetisch größere Werte sollen in den rechten Teilbaum eingefügt werden. Aktualisieren Sie nach dem Einfügen die beiden Attribute in Tree.

e) Ein Baum soll als degeneriert bezeichnet werden, wenn `log2(count) + 1 < max_depth`. Schreiben Sie eine Funktion `degenerated`, die dies prüft und true oder false zurückgibt. Fügen Sie mehrere Zeichenketten in einen Baum ein und prüfen Sie, ob dieser degeneriert ist.

f) Schreiben Sie eine Funktion, die Ihnen alle Elemente auf einer bestimmten Baumtiefe ausgibt. Sollte die Baumtiefe nicht existieren, soll eine Fehlermeldung ausgegeben werden. Die Ausgabe soll entgegengesetzt alphabetisch sortiert sein. D.h. Wörter die mit dem Buchstaben Z anfangen sollen vor Wörtern ausgegeben werden, die mit A anfangen.

g) Schreiben Sie eine Funktion, die über den Baum läuft und Ihnen die Anzahl an Elementen zurückgibt, die mit einer bestimmten als Parameter übergebenen Zeichenkette anfangen. Zum Beispiel enthält der Baum die Zeichenketten `"einsammeln"`, `"einer"`, `"Einzigartig"` und `"einmalig"`. Bei Übergabe von `"ein"` soll in dem Fall 4 zurückgegeben werden.

h) Schreiben Sie eine Funktion, die Ihnen nun ein Array von Pointern auf Zeichenketten zurückgibt, die mit einer bestimmten als Parameter übergebenen Zeichenkette anfangen. Geben Sie nach dem Funktionsaufruf die Zeichenketten auf der Konsole aus. Warum ist es notwendig, die vorher in g) implementierte Funktion, sowohl innerhalb der Funktion als auch nach Ausführung der Funktion aufzurufen?

i) Schreiben Sie eine neue Funktion, die das Problem in der vorherigen Teilaufgabe löst, indem sie statt eines Arrays einen Zeiger auf eine Liste zurückgibt, die die Werte enthält.

j) Schreiben Sie eine neue Funktion, die das Problem in h) löst, indem Sie als Rückgabewert eine Struktur zurückgeben, die nicht nur einen Verweis auf das Array liefert, sondern auch dessen Länge enthält.

k) Schreiben Sie eine neue Funktion, die das Problem in h) löst, indem Sie einen weiteren Übergabeparameter nutzen, der nach Aufruf der Funktion die Länge des zurückgegebenen Arrays enthält.

l) Schreiben Sie eine Funktion die zwei Bäume auf Gleichheit prüft. Die Funktion soll auf Strukturelle und inhaltliche Gleichheit prüfen.

m) Schreiben Sie eine Funktion, die zwei Bäume auf inhaltliche Gleichheit prüft. Die Strukturen der Bäume dürfen unterschiedlich sein, wie im folgendem Beispiel zu sehen ist:

Weitere Übungen

Im Internet finden sich z. B. unter dem Suchbegriff „programming challenges" viele Webseiten, die 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.

Beispiele hierfür sind unter anderem:

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