Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2018/19

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

Übungsblatt 13

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

Aufgabe 1: Fußballstatistik

In dieser Aufgabe sollen Ergebnisse von Fußballspielen erfasst werden und für jedes Team eine Statistik geführt werden. Die Template-Datei für diese Aufgabe ist soccer.c. Für jedes Team werden in `struct Result` die Anzahl der bisher gespielten Spiele, die Anzahl selbst geschossener Tore, die Anzahl Gegentore und der Punktestand erfasst. Die Daten liegen in der Textdatei soccer_results.txt mit folgendem Format:

```
Bayern München          // Heim-Team
Bayer Leverkusen        // Auswärts-Team
3                       // Tore Heim-Team
1                       // Tore Auswärts-Team
```

Jeweils vier Zeilen gehören also zu einer Begegnung.

a) Lesen Sie die Datei zeilenweise ein und aktualisieren Sie jeweils die Map m. Diese bildet für jedes Team den Teamnamen (`String`) auf dessen Ergebnis (`Result*`) ab. Das Einlesen einer Zeile kann über `String s = s_input(100);` erfolgen. Die Konvertierung von `String` nach `int` kann über `int i = i_of_s(s);` erfolgen. Wenn für ein Team noch kein Resultat in der Map vorliegt, muss dieses erstellt werden. Ein verlorenes Spiel zählt 0, ein unentschiedenes 1 und ein gewonnenes Spiel zählt 3 Punkte.

b) Implementieren Sie in map.c die Funktion `int size_map(Map* m)`, um die Anzahl der Einträge in einer Map zu ermitteln.

c) Implementieren Sie in map.c die Funktion `Entry* entries_map(Map* m)`, um ein dynamisch allokiertes Array der Einträge der Map zu erzeugen. Die keys und values selbst sollen dabei nicht dupliziert werden.

d) Optional: Sortieren Sie die Teams nach Punktzahl und Tordifferenz, um eine Tabelle zu erhalten und geben Sie diese aus.

Aufgabe 2: Iteratoren für Baumstrukturen

Zu dieser Aufgabe gehören folgende Dateien: iterator.{c|h}, btree.{c|h}, tree.{c|h}, dyn_list.{c|h}, dyn_queue.{c|h}und dyn_stack.{c|h}. In dieser Aufgabe sollen Sie verschiedene Iteratoren für zwei verschiedene Datenstrukturen schreiben. Die Datenstrukturen sind 1. ein Binärbaum (vgl. btree.h) und 2. ein Baum mit beliebig vielen Nachfolgern (vgl. Assignment 12 & tree.h). Die Iteratoren sollen den Baum in „level order" (ebenenweise), sowie in „pre order" durchlaufen können (vgl. https://en.wikipedia.org/wiki/Tree_traversal). Die Methoden und Strukturen in dyn_queue.{c|h} und dyn_stack.{c|h} können für die Aufgabe hilfreich sein.

a) Machen Sie sich mit der Struktur Iterator in iterator.h vertraut. Diese speichert einen Zustand in `state` und bietet 3 Funktionen an: `has_next` gibt true zurück, wenn noch ein weiteres Element vorhanden ist. `next_element` gibt das nächste Element zurück und free gibt den Inhalt von `state` frei. Diese 3 Funktionen werden von den Funktionen `has_next(Iterator* iterator)`, `next_element(Iterator* iterator)` und `free_iterator(Iterator* iterator)` aufgerufen, die am Anfang der iterator.c Datei implementiert sind. Diese 3 Funktionen werden dann genutzt um mit dem Iterator zu interagieren vgl. `main` in iterator.c.

b) Implementieren Sie die Funktion `get_iterator_level_order_BT`, die einen Iterator zurückgibt, der den Binärbaum ebenenweise durchläuft. Schauen Sie sich die Definition von `get_iterator_pre_order_BT` an und wie die einzelnen Funktionen gesetzt werden und wie der Zustand `state` des Iterator definiert ist. Schreiben Sie entsprechende Funktionen um diesen Iterator zu erstellen.

c) Implementieren Sie die Funktion `get_iterator_pre_order_T`, die einen Iterator zurückgibt, der den Baum ebenenweise durchläuft. Implementieren Sie entsprechende Methoden und setzten Sie diese im Iterator.

d) Implementieren Sie die Funktion `get_iterator_level_order_T`, die einen Iterator zurückgibt, der den Baum in „level order" durchläuft. Implementieren Sie entsprechende Methoden und setzten Sie diese im Iterator.

e) Mit dem Iterator, der ebenenweise den Baum durchläuft, können Sie nun die Breitensuche (vgl. https://en.wikipedia.org/wiki/Breadth-first_search) implementieren. Implementieren Sie die Funktion `contains_Btree` und `contains_tree`, die `true` zurückliefern, wenn der key in der übergebenen Datenstruktur enthalten ist.

f) Optional: Schreiben Sie weitere `contains` Funktion, die mit Tiefensuche (vgl. https://en.wikipedia.org/wiki/Depth-first_search) die Bäume durchsuchen.

Aufgabe 3: Multimengen

In dieser Aufgabe geht es um Multimengen, das sind Mengen, in denen Elemente beliebig oft vorkommen können. Wie bei einfachen Mengen sind die Elemente nicht geordnet. Die Template-Dateien für diese Aufgabe sind multiset.{c|h}. Machen Sie sich mit dem Template-Code vertraut. Insbesondere ist ein Eintrag in der Multimenge repräsentiert als `MultisetNode`-Struktur, die einen Zeiger auf das eigentliche Element (`void* element`) und die Anzahl des Auftretens des Elements in der Menge enthält (`int count`). Die Menge selbst ist durch die Struktur `Multiset` repräsentiert. Diese umfasst neben dem bucket-Array die Anzahl der verschiedenen Elemente (`int n_elements`) und die Anzahl der Instanzen (`int n_instances`). Beispielsweise gilt für die Menge {30, 4, 4, 7, 7 }: n_elements = 3 und n_instances = 5. `MultisetNode` enthält außerdem Zeiger auf eine Hash- und eine Vergleichsfunktion für Elemente. Kommentieren Sie die Beispielaufrufe in der `main`-Funktion geeignet ein bzw. aus.

a) Implementieren Sie die interne Hilfsfunktion `find_multiset`, um den Knoten zu ermitteln, in dem ein bestimmtes Element gespeichert wird (Vergleich mit `s->equal`).

b) Implementieren Sie die Funktion add_multiset, um ein Element in einer bestimmten Anzahl zu einer Multiset hinzuzufügen. Der Zustand der Multiset wird dadurch verändert.

c) Implementieren Sie die Funktion `copy_multiset`, um eine `Multiset` zu kopieren. Wenn `copy` ungleich `NULL` ist, sollen die Elemente selbst damit kopiert werden.

d) Implementieren Sie die Funktion `free_multiset`, um eine `Multiset` freizugeben. Wenn `free_element` ungleich `NULL` ist, sollen die Elemente selbst damit freigegeben werden.

e) Implementieren Sie die Funktion `intersection_multiset`, um die Schnittmenge zweier Multisets zu berechnen. Die Funktion soll die Eingabemengen nicht verändern und die Elemente selbst nicht kopieren. Es wird eine neue Ergebnismenge erzeugt. Die Schnittmenge ist wie folgt definiert. Sei c:= intersect(a, b). Dann gilt für alle x: count(c, x) = min(count(a, x), count(b, x)). Beispiel: a = { 1, 1, 2 }, b = { 1, 1, 1, 2, 2, 3 } ➔ c = a ∩ b = { 1, 1, 2 }.

f) Implementieren Sie die Funktion `union_multiset`, um die Vereinigungsmenge zweier Multisets zu berechnen. Die Funktion soll die Eingabemengen nicht verändern und die Elemente selbst nicht kopieren. Es wird eine neue Ergebnismenge erzeugt. Die Vereinigungsmenge ist wie folgt definiert. Sei c:= union(a, b). Dann gilt für alle x: count(c, x) = max(count(a, x), count(b, x)). Beispiel: a = { 1, 1, 2 }, b = { 1, 1, 1, 2, 2, 3 } ➔ c = a ∪ b = { 1, 1, 1, 2, 2, 3 }.

g) Implementieren Sie die Funktion `sum_multiset`, um die Summe zweier Multisets zu berechnen. Die Funktion soll die Eingabemengen nicht verändern und die Elemente selbst nicht kopieren. Es wird eine neue Ergebnismenge erzeugt. Die Summe ist wie folgt definiert. Sei c:= sum(a, b). Dann gilt für alle x: count(c, x) = count(a, x) + count(b, x). Beispiel: a = { 1, 1, 2 }, b = { 1, 1, 1, 2, 2, 3 } ➔ c = a + b = { 1, 1, 1, 1, 1, 2, 2, 2, 3 }.

h) Implementieren Sie die Funktion `remove_multiset`, um die gegebene Anzahl Instanzen eines bestimmten Elements aus der `Multiset` zu entfernen. Wenn `free_element` ungleich `NULL` ist, soll das Element damit freigegeben werden. Diese Teilaufgabe ist schwieriger als die anderen Teilaufgaben.

Zusatzaufgabe 1 – 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 Listenkopf, die auf das erste Element der Liste verweist und auf das letzte Element. Zusätzlich soll die Struktur Listenkopf 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. Versuchen Sie dies doch einmal ohne die Bibliotheksfunktionen. 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 2 – 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 von einer gegebenen Zeichenkette eine Kopie ohne Vokale erstellt. Die Kopie soll dynamisch allokiert sein und darf nicht mehr Speicher verwenden als benötigt wird.

b) Implementieren Sie eine Funktion, die eine Zeichenkette auf einen bestimmten Inhalt prüft. Die Funktion soll true zurückgeben, wenn nur zusammenhängende Ziffernfolgen, die ein Vielfaches von 3 lang sind, in der Zeichenkette auftauchen, bspw. „123", „ab c de 123456" oder „123 asdbderb 890". Ansonten gibt die Funktion false zurück.
c) 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.
d) Schreiben Sie eine eigene Funktion, die zwei Zeichenketten hintereinander fügt und einen Pointer auf diese Zeichenkette zurückgibt.
e) 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. Bspw: „abc.bdfg" und split = „." würde ein Array zurückgeben in dem zwei Pointer vom Type char* sind. Der erste würde auf „abc" zeigen und der zweite auf „bdfg". Die Eingabezeichenkette soll dabei nicht verändert werden.
f) Schreiben Sie eine Funktion, die die Länge für eine gegebene Zeichenkette zurückgibt.
g) Schreiben Sie eine Funktion, die die größten gemeinsame zusammenhängende Zeichenkette zurückgibt, die in zwei Zeichenketten enthalten ist. Bsp.: „Programmieren 1 macht Spass" und „Programmieren 1 macht keinen Spass" würde als Ergebnis „Programmieren 1 macht „ zurückgeben.
h) 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. Bsp.: Text: „Werbung ist doof", Wörter die ersetzt werden sollen: „doof", „bescheuert", „idiotisch" -> Ergebnis: „Werbung ist d***".

Zusatzaufgabe 3 – 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 nxn Array mit Double Werten transponieren kann. Das zurückgegebene transponierte Array soll dynamisch allokiert sein.
f) Schreiben Sie eine Funktion, die zwei gleich lange Vektoren (Double Arrays) sowie die Länge übergeben bekommt und das Skalarprodukt berechnet.
g) Schreiben Sie einen Iterator für Arrays.
h) Schreiben Sie eine Funktion, die zwei Double Arrays (nxm) miteinander vergleicht und prüft ob beide die gleichen Werte enthalten.
i) Schreiben Sie eine map Funktion für 2-dimensionale Double Arrays (nxm). map soll eine Map-Funktion übergeben bekommen, die auf jedes Element angewendet werden soll. Die

Funktion map soll eine dynamisch allokierte veränderte Kopie zurückgeben. Implementieren Sie als Map-Funktion eine binäre Funktion, die 0 in zurückgibt, wenn ein Wert kleiner ist als ein übergebener Parameter oder 1 zurückgibt, wenn der Wert größer als 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 Indicies bekommt und ein Array mit Pointern auf Teil-Arrays zurückgibt. Bspw. Array: [1 2 3 4], Indicies Array: [1 2] und Rückgabe [[1], [2], [3 4]]. Allokieren Sie den Speicher dynamisch und verändern Sie die übergebenen Arrays nicht.

Hinweise zum Editieren, Compilieren und Ausführen:

- mit Texteditor `file.c` editieren und speichern
- `make file` ← ausführbares Programm erstellen
- `./file` ← Programm starten (evtl. ohne `./`)
- Die letzten beiden Schritte lassen sich auf der Kommandozeile kombinieren zu: `make file && ./file`