Human-Computer Interaction Group

Leibniz Universität Hannover

Übungsblatt 13

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.

Hinweis: prog1lib

Die Dokumentation der prog1lib finden Sie unter der Adresse:
https://postfix.hci.uni-hannover.de/files/prog1lib/

Aufgabe 1: Binärbäume

In dieser Aufgabe werden Operationen für einen Binärbaum implementiert, in dem ganze Zahlen gespeichert werden. Die für diese Aufgabe relevanten Dateien sind `integer_tree.{h|c}`, `integer_list.{h|c}` und `integer_tree_test.c`.

a) In der `main`-Funktion von `integer_tree_test.c` findet sich das `int`-Array `values`. Fügen Sie die Elemente des Arrays geordnet in einen Binärbaum ein. Geben Sie die Elemente des Baumes aus, so dass sie in aufsteigend sortierter Reihenfolge erscheinen.

b) Implementieren Sie die Funktion `free_tree` in `integer_tree.c`, die den Speicher des Baumes freigibt. Beachten Sie, dass der Baum nur ganze Zahlen speichert.

c) Die `print_tree`-Funktion in `integer_tree.c` gibt Bäume so aus, dass der Wert eines Knotens in der Mitte und die linken und rechten Unterbäume links und rechts davon erscheinen. Um die Struktur sichtbar zu machen wird jeder Knoten geklammert. Die Form ist also: (<left>, <value>, <right>). Der leere Baum wird als Unterstrich dargestellt.

   Beispiel: `(((_, 11, _), 1, (_, 12, _)), 0, (_, 2, (_, 7, _)))`

   Erweitern Sie die `print_tree`-Funktion so, dass Blätter in abgekürzter Form dargestellt werden. Für das Beispiel wäre die Darstellung also: `((11, 1, 12), 0, (_, 2, 7))`

d) Implementieren Sie die Funktion `get_odd_even_rec` die einen Baum und zwei Zeiger auf Ergebnislisten übergeben bekommt. Die Ergebnisliste `odd` soll später alle ungeraden Elemente enthalten. Die Ergebnisliste `even` nur die geraden Elemente.

e) Beantworten Sie die folgende Frage als Kommentar im Quellcode: Warum wird als Typ für die Ergebnisliste ein Zeiger auf einen Zeiger auf eine Node (bspw.: `Node ** odd`) verwendet und nicht ein Zeiger auf eine Node (`Node * odd`)?

f) Implementieren Sie nun das Filtern von geraden oder ungeraden Zahlen als Filterfunktion. Verwenden Sie die Funktion `filter_tree` und implementieren Sie eine Filterfunktionen `odd_or_even`. Die Funktion soll entweder gerade oder ungerade Elemente filtern. Nutzen Sie den Parameter `x` um dies zu ermöglichen.

g) Implementieren Sie die Funktion `is_sorted`. Diese gibt genau dann `true` zurück, wenn der Baum sortiert ist. Ein Baum ist sortiert, wenn er leer ist, nur aus dem Wurzelknoten besteht oder wenn für alle Nachfolgeknoten links eines Elternknoten gilt, dass Sie kleiner sind als der Elternknoten und für alle Nachfolgeknoten rechts eines Elternknoten gilt, dass Sie größer sind als der Elternknoten. Schauen Sie sich die Testfälle an. Sie können für Ihre Implementation beliebige Hilfsfunktionen erstellen und auch den Option-Type `Option_int` nutzen. Dieser ist entweder nicht initialisiert, wenn `valid` den Wert `false` enthält oder initialisiert, dann enthält `valid` den Wert `true` und in `value` ist ein gültiger Wert gespeichert.

   Tipp: Es kann hilfreich sein, wenn Sie sich einen Baum aufmalen und prüfen, welche Grenzen für jeden Knoten gelten. Bedenken Sie, dass die Grenzen für Wurzelknoten nicht initialisiert sind, da er kein Nachfolgerknoten ist.

Aufgabe 2: Table of Contents

In dieser Aufgabe erstellen Sie eine Datenstruktur, die ein Dokument wie eine Bachelorarbeit speichern kann. Zur Vereinfachung speichert die Struktur nur die Kapitelnamen, sowie die Anzahl der Seiten für jedes Kapitel. Schauen Sie sich `TOC.c` und `TOC.h` an. Nutzen Sie für das Allokieren von Speicher ausschließlich `xmalloc` bzw. `xcalloc`.

a) Implementieren Sie die Konstruktorfunktion `new_TNode`, die den Titel eines Kapitels, die Anzahl der Seiten dieses Kapitels sowie einen Zeiger auf eine Liste von Unterkapiteln übergeben bekommt. Als Rückgabe liefert sie einen Pointer auf eine neue `TNode`. Implementieren Sie zusätzlich die Funktion `new_node`, die eine neue Node erstellt. `new_node` bekommt einen Verweis auf eine `TNode` übergeben, sowie einen Pointer auf eine nachfolgende Node.

b) Implementieren Sie die Funktionen `free_TNode` und `free_Nodes`. Die erste Funktion gibt den von einem Kapitel allokierten Speicher frei (einschließlich der Unterkapitel). Die zweite Funktion gibt den Speicher der übergebenen Node sowie aller nachfolgenden Nodes und deren Kapitel frei. Am Ende soll der komplette vom Baum allokierte Speicher freigegeben werden.

c) Berechnen Sie die Anzahl der Seiten, die ein Kapitel benötigt. Implementieren Sie dafür die Funktion `calculate_pages`, die eine `TNode` übergeben bekommt und die Summe der Seiten zurückgibt. Gehen Sie davon aus, dass jedes Kapitel (auch Unterkapitel) auf einer neuen Seite beginnt. Die Anzahl der Seiten für ein Kapitel ist die Summe aus den Seiten des Kapitels plus die Anzahl der Seiten der Unterkapitel. Bsp. Das Kapitel Introduction hat eine eigene Seite plus die Seiten aus Motivation, Research Questions und Goals. Geben Sie die Anzahl der Seiten der Bachelorarbeit auf der Konsole aus.

d) Implementieren Sie die Funktion `print_TOC`, die das Inhaltsverzeichnis gut formatiert auf der Konsole ausgibt. Dazu gehören, die Hierarchie, der Titel, sowie die Seitenzahl.

   Die Formatierte Ausgabe könnte beispielsweise wie folgt aussehen:

```
The thesis has: 131 pages.
My Bachelor thesis
1 Introduction ..........................   0
```

Aufgabe 3: Berechnungsbaum

In dieser Aufgabe sollen Sie sich mit sogenannten Berechnungsbäumen auseinandersetzen. Eine `TreeNode` enthält entweder eine Konstante, einen Parameter, dessen Wert erst zur Evaluation bekannt ist, oder eine Funktion, die ausgeführt werden soll auf ihren nachfolgenden Knoten `left` und `right` in der Form: `<left> <function> <right>` (bspw.: `2 + 3`). Die Knoten `left` und `right` können dabei wieder Teilberechnungsbäume sein.

Ein Berechnungsbaum kann wie folgt ausehen:

```
                    ( * )
          ( + )                   ( / )
    ( + )      ( * )        ( z )      ( - )
(0.0)(3.0) ( x )(0.0)             ( z )(0.0)
```

Der gegebene Baum lässt sich zu dem Baum, der nur aus der Node `(3.0)` besteht, reduzieren.

Die für diese Aufgabe relevanten Dateien sind `computational_tree.{h|c}` und `comp_tree_print.{h|c}`.

a) Implementieren Sie die Funktion `ComputationalResult evaluate(TreeNode* comp_tree, PNode* parameter_list)`. Die Funktion bekommt einen Berechnungsbaum übergeben, sowie eine Liste von Parametern, die die Werte für Parameter enthalten. Nach Ausführung der Funktion soll in der zurückgegebenen Struktur vom Typ `ComputationalResult` entweder der Wert der Berechnung stehen und das enthaltene `bool valid` den Wert `true` enthalten oder, falls eine Berechnung nicht möglich ist, soll `valid` den Wert `false` enthalten. Es sind Testfälle vorgegeben. Zum besseren Verständnis werden die Testbäume auch auf der Konsole vor Ausführung des Testfalls ausgegeben. Der übergebene Baum soll nicht verändert werden.

b) Wenn die gleichen Berechnungen auf größeren Datenmengen durchgeführt werden, dann ist es sinnvoll, vorher die Berechnungen zu optimieren. Implementieren Sie hierzu die Funktion `TreeNode* precompute_tree(TreeNode* tree)`, die einen vorberechneten Berechnungsbaum zurückgibt, der eine äquivalente Berechnung ausführt wie der übergebene Baum. Der übergebene Baum soll dabei nicht verändert werden. Folgende Vereinfachungen sollen implementiert werden:

- Ein Teilbaum der nur aus Konstanten und Funktionen besteht kann schon berechnet werden.
- Ein Baum, der aus zwei gleichen Parametern in `left` und `right` besteht und dessen Funktion – oder / ist, kann durch eine konstante Node ersetzt werden.
- Ein Baum, der einen Parameter und eine Konstante hat, kann unter folgender Bedingung vereinfacht werden:
  - Für * gilt: `a * 0 = 0`, `a * 1 = a` (das gilt auch, wenn Parameter und Konstante getauscht werden)
  - Für + gilt: `a + 0 = a` (das gilt auch, wenn Parameter und Konstante getauscht werden)
  - Für – gilt: `a - 0 = a` (gilt nur in dieser Reihenfolge)
  - Für / gilt `a / 1 = a`, `0 / a = 0` (gilt nur in dieser Reihenfolge)

c) Was ist die maximal mögliche Baumtiefe, die die Funktion `print_tree` ausgeben kann? Hinweis: Es geht hier nicht um die Limitierung der Konsolenfensterbreite.

d) Implementieren Sie eine Funktion, die einen Baum in einen String umwandelt. Die Funktion soll einen Pointer auf die Zeichenkette zurückgeben und nur den Baum als Parameter erhalten. Die Ausgabe in der dynamisch allokierten Zeichenkette soll die folgende Form aufweisen: (`left function right`), wobei `left` und `right` wieder geklammerte Ausdrücke sind, wenn Sie nicht ein Parameter oder eine Konstante sind. Berechnen Sie als erstes, wie viele Zeichen Sie benötigen, allokieren Sie dann den Speicher. Konstanten sollen immer nur mit einer Nachkommastelle ausgeben werden. Schreiben Sie dann die String Repräsentation des Baumes in den Speicher.

Beispiel:

```
        ( + )
  ( * )       ( * )
(3.0)( y )  ( x )(4.0)
```

Wird zu:

```
((3.0 * y) + (x * 4.0))
```