Human-Computer Interaction Group

Leibniz Universität Hannover

Übungsblatt 10 (Optional)

Programmieren 1 – WiSe 21/22

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

Alle Übungen (bis auf die erste) müssen in Zweiergruppen bearbeitet werden. Beide Gruppenmitglieder müssen die Lösung der Zweiergruppe einzeln abgeben. Die Namen beider Gruppenmitglieder müssen sowohl in der PDF Abgabe, als auch als Kommentar in jeglichen Quelltextabgaben genannt werden. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Sonntag den 09.01.um 23:59 Uhr über *https://assignments.hci.uni-hannover.de/WiSe2021/Programmieren1*. Die Abgabe muss aus einer einzelnen Zip-Datei bestehen, die den Quellcode, ein PDF für Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen auf.

Die Bearbeitung dieses Assignments ist optional. Ist mindestens eine Aufgabe korrekt gelöst, erhalten Sie einen Bonuspunkt. Sind alle Aufgaben korrekt gelöst, erhalten Sie zwei Bonuspunkte.

Hinweis: prog1lib

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

Aufgabe 1: Der Wolf, die Ziege und der Kohlkopf

In dieser Aufgabe soll ein Spiel implementiert werden, in dem ein Bauer einen Wolf, eine Ziege und einen Kohlkopf in einem Boot über einen Fluss transportieren muss. Zunächst sind Bauer, Wolf, Ziege, Kohlkopf und Boot am linken Ufer des Flusses. Leider hat das Boot (wenn der Bauer im Boot ist) nur einen freien Platz. Der Bauer darf aber den Wolf und die Ziege nicht alleine lassen, weil sonst der Wolf die Ziege frisst. Er darf auch die Ziege mit dem Kohlkopf nicht alleine lassen, weil sonst die Ziege den Kohlkopf frisst. Ist der Bauer am gleichen Ufer, besteht keine Gefahr. Das Spiel ist dann erfolgreich gelöst, wenn Wolf, Ziege und Kohlkopf sicher am rechten Ufer angelangt sind.

Die Template-Datei für diese Aufgabe ist `wolf_goat_cabbage.c`. Nutzen Sie die String-Listen aus der *prog1lib* um den Spielzustand darzustellen.

a) Implementieren Sie die Funktion `evaluate_situation`, die aktuelle Situation analysiert und ausgibt, ob die Aufgabe gelöst wurde bzw. kritisch ist. Die Funktion darf das Programm auch abbrechen (`finish`), falls das Spiel verloren wurde.

b) Implementieren Sie die Funktion `play_wolf_goat_cabbage`, die den Anfangszustand des Spiels setzt, Eingaben des Spielers entgegen nimmt und die Listen entsprechend manipuliert. Ist beispielsweise das Boot auf der linken Seite und die Ziege im Boot, dann soll nach Eingabe von `goat` bzw. `g` die Ziege aus dem Boot genommen und auf die linke Seite gesetzt werden. Die Eingabe `l` bewegt das Boot nach links, die Eingabe `r` nach rechts. Die Eingabe von `q` soll das Spiel beenden. Nach jeder Eingabe soll `evaluate_situation` aufgerufen werden.

c) Erweitern Sie die Funktion `finish`, um allen dynamisch allokierten Speicher freizugeben.

Ein möglicher Spielablauf könnte wie folgt aussehen:

```
[wolf, goat, cabbage][]      []
> g
[wolf, cabbage][goat]      []
> r
[wolf, cabbage]      [goat][]
> goat
[wolf, cabbage]      [][goat]
> l
[wolf, cabbage][]      [goat]
> wo
[cabbage][wolf]      [goat]
> r
[cabbage]      [wolf][goat]
> w
[cabbage]      [][goat, wolf]
> l
[cabbage][]      [goat, wolf]
wolf eats goat
```

Aufgabe 2: Bleep Censor

In dieser Aufgabe sollen Sie unschöne Wörter aus einer Zeichenkette durch entsprechend zensierte Versionen ersetzen. Beispiel: "`Du Idiot `" soll zu "`Du I****`" verändert werden. Die Wörter, die ersetzt werden sollen, sind in einer unsortierten Zeichenkette vorgegeben und durch Leerzeichen voneinander getrennt. Sie können selber Wörter hinzufügen. Diese Wörter sollen in einen binären Baum sortiert eingefügt werden um eine schnelle Ersetzung zu gewährleisten. Dort wird nicht neuer Speicher allokiert, sondern es wird eine Tokenstruktur erstellt mit einem Zeiger auf den Start des Wortes und einen Zeiger auf das Ende des Wortes

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

a) Implementieren Sie die Funktion `print_token`, die ein Token ausgibt. Wenn `censored == true` ist, soll nur der erste Buchstabe des Tokens ausgegeben werden und danach entsprechend der Länge des Wortes Sterne '`*`' folgen.

b) Implementieren Sie die Funktion `compare_token`, die zwei Token vergleicht und prüft ob ein Token lexikographisch vor das andere gehört. Schauen Sie sich auch die Testfälle an. Wenn `t1` vor `t2` gehört, soll `-1` zurückgegeben werden. Wenn `t1` gleich `t2` ist, soll `0` zurückgegeben werden. Wenn `t1` nach `t2` gehört, soll `1` zurückgegeben werden.

c) Implementieren Sie die Funktion `insert_in_tree`, die einen Token in den Baum sortiert einfügt. Nutzen Sie für das sortierte Einfügen die Funktion `compare_token`. Ist das Ergebnis negativ muss es im linken Teilbaum eingefügt werden, ist es 0 muss nichts geschehen und ist es größer als 0 muss token im rechten Teilbaum eingefügt werden.

d) Implementieren Sie die Funktion `create_bleep_tree`, die aus einer Zeichenkette einen Baum erstellt. Nutzen Sie Ihre `insert_in_tree` Funktion.

e) Lesen Sie von der Standardeingabe eine Zeile mit der Funktion `get_line` aus der Programmieren 1 Bibliothek ein. Geben Sie die eingelesene Zeichenkette wieder aus und zensieren Sie mithilfe des Baums. Sie müssen nur Wörter bzw. Sätze verarbeiten können, in denen die Wörter durch Leerzeichen getrennt sind und die mit einem Leerzeichen enden. Beispiel: "`Der Affe sitzt im Baum `". Wird zu: "`Der A*** sitzt im Baum `"

f) Geben Sie den gesamten allokierten Speicher wieder frei. Implementieren Sie dazu geeignete Funktionen und verifizieren Sie mithilfe von `report_memory_leaks(true)`.

Aufgabe 3: Wunschbaum

Der Weihnachtsmann investiert dieses Jahr in seine IT und möchte von handgeschriebenen Wunschzetteln und Papierlisten mit Geschenken auf einen modernen binären Wunschbaum umstellen. Glücklicherweise wurden bereits in diesem Jahr moderne Scanner mit Texterkennung angeschafft, die auch mühelos Kinderhandschrift erkennen können. D.h. Sie bekommen die Wunschzettel bereits digital in einer Textdatei. Ihre Aufgabe besteht nun darin, die Datei mit den Wunschzetteln einzulesen und die Wünsche in einen binären Baum einzutragen. Der binäre Baum soll nach dem Wunschtext sortiert sein. Jeder Knoten im Baum, soll sowohl den Wunsch, als auch die Häufigkeit speichern wie oft der Wunsch insgesamt von Kindern gewünscht wurde. Jeder Knoten soll zudem eine Liste mit Kindernamen enthalten, die den Wunsch geäußert haben.

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

a) Die Elfen aus der IT-Abteilung haben bereits eine Baumstruktur `TreeNode` erstellt. Prüfen Sie, ob diese Ihre Anforderungen erfüllen. Machen Sie sich auch mit dem weiteren Template-Code vertraut.

b) Implementieren Sie eine Struktur `Element`, in der der Wunschtext, die Häufigkeit sowie eine Liste von Kindern gespeichert wird, die diesen Wunsch haben. Erstellen Sie auch eine Konstruktorfunktion `new_element`, die alle Elemente übergeben bekommt und dann eine dynamisch allokierte Struktur `Element` zurückgibt. Die Struktur `Node` kann Ihnen helfen.

c) Implementieren Sie eine rekursive Funktion `add_wish`. Diese soll den Binärbaum nach dem Wunschtext durchsuchen und entweder einen vorhandenen Knoten aktualisieren oder einen neuen Knoten geordnet einfügen. Nutzen Sie für die Sortierung innerhalb des Binärbaums die Funktion `strcmp`, die Ihnen `0` zurückgibt, wenn der Wunschtext des Elements gleich dem gesuchten Wunschtext ist.

d) Schreiben Sie eine rekursive Funktion `print_tree_as_list`, die den Baum auf der Konsole im Listenformat lexikographisch sortiert nachdem Wunschtext wie nachfolgend dargestellt ausgibt:

```
                                  Wunsch Anzahl  Kinder
                          Barbie Meerjungfrau     13  Finn, Paul, Theresa, Noah, Juna, Leon, Romy, Luise, Niklas, Elias, Jonas, Emely, Konstantin
            Barbie und Ken Geschenkset mit Huendchen     12  Finn, Julia, Luis, Theresa, Juna, Romy, David, Luise, Amelie, Ben, Konstantin, Lukas
                                     Haarkreide      9  Paul, Theresa, Juna, Philipp, David, Luise, Ben, Amy, Ella
                                  Hasbro Kroko Doc      9  Julia, Theresa, Juna, Luise, Niklas, Finja, Jonas, Ben, Amy
                                      Kidisecrets      5  Finn, Theresa, Luise, Emely, Lukas
                                Lego Das Disney Schloss     10  Finn, Luis, Theresa, Philipp, Frieda, Elias, Jonas, Emely, Ella, Lukas
                                 Lego Millenium Falcon     14  Paul, Eva, Luis, Theresa, Noah, Romy, Philipp, Oskar, Amelie, Elias, Emely, Konstantin, Ella, Lucy
Mattel Exklusiv Hot Wheels Doppel-Looping Wettrennen     10  Finn, Luis, Juna, Philipp, Frieda, Luise, Jonas, Emely, Ben, Lukas
                                  My Fairy Garden     13  Finn, Eva, Luis, Noah, Juna, Philipp, Luise, Amelie, Elias, Jonas, Ben, Amy, Lukas
                                MyToys Holzeisenbahn     12  Finn, Paul, Juna, Romy, Philipp, David, Luise, Elias, Jonas, Amy, Ella, Benjamin
          MyToys Puppenhaus mit Garten und Moebel     12  Finn, Julia, Luis, Theresa, Philipp, Katharina, Luise, Amelie, Finja, Ben, Amy, Benjamin
        MyToys-Collection Puppenwagen Trendy     11  Luis, Theresa, Philipp, Oskar, David, Luise, Niklas, Emely, Ben, Amy, Lukas
                                   Nintendo Switch      9  Katharina, David, Amelie, Finja, Ben, Amy, Ella, Lukas, Benjamin
                      PLAYMOBIL Polizei-Einsatzwagen      8  Juna, Romy, Philipp, David, Luise, Ben, Konstantin, Ella
                                       Piano Matte      9  Finn, Theresa, Juna, Philipp, David, Luise, Jonas, Ben, Konstantin
                          Play-Doh Verrueckte Haufen     14  Finn, Luis, Theresa, Juna, Romy, David, Niklas, Amelie, Elias, Finja, Jonas, Konstantin, Lukas, Lucy
                                     Pop up Pirate      6  Juna, Philipp, Luise, Elias, Jonas, Konstantin
                        Schleich Mobile Tieraerztin     12  Finn, Theresa, Juna, Romy, Philipp, Oskar, David, Finja, Ben, Amy, Konstantin, Ella
       Spin Master Monster Jam - Grave Digger     14  Finn, Paul, Theresa, Juna, Leon, Philipp, David, Luise, Elias, Emely, Ben, Amy, Konstantin, Lukas
           Tonies 30 Lieblings-Kinderlieder      9  Finn, Paul, Luis, Juna, Philipp, Elias, Emely, Ben, Ella
```

e) Der Weihnachtsmann hat leider nicht genug Elfen um alle Geschenke zu produzieren. Als Heuristik nimmt er an dass es ausreichend ist die 11 häufigsten Geschenke herzustellen, sodass alle Kinder (hoffentlich) versorgt sind. Erstellen Sie als erstes eine Liste, in der die Wünsche absteigend nach Ihrer Häufigkeit sortiert sind. Implementieren Sie dafür die Funktion `insert_ordered_by_count`, die Ihnen eine Liste aus Strukturen vom Typ `ElementNode` erstellt, diese soll die Strukturen vom Typ `Element` aus dem Baum `tree` absteigend sortiert nach der Wunschhäufigkeit enthalten. Die Funktion soll für die Strukturen vom Typ `Element` keinen neuen Speicher allokieren.

f) Schreiben Sie in der `main` Funktion eine Routine um zu überprüfen, ob alle 29 Kinder Geschenke bekommen. Geben Sie das Resultat auf der Konsole aus. Nutzen Sie die Funktion aus *e)* und die bestehenden Funktionen und Strukturen im Template.

g) Implementieren Sie Funktionen, um den gesamten Speicher wieder freizugeben. Am Ende soll es keine Memory Leaks geben.

Aufgabe 4: Dateisystem

In dieser Aufgabe sollen Sie die Datenstruktur eines Dateisystems implementieren. Ein Dateisystem ist ein *Baum*, dessen Knoten entweder Ordner (`NT_DIR`) oder Dateien (`NT_FILE`) sind. Jeder Knoten besitzt einen Namen mit maximal 64 Zeichen. Ordner enthalten ein Array mit enhaltenen Knoten, Dateien enthalten einen Zeiger auf ihren Inhalt. Das Array eines Ordners ist stets lexikographisch sortiert. Als Wurzelknoten für ein Dateisystem dient immer ein Ordner mit leerem Namen.

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

a) Machen Sie sich mit der gegebenen Struktur `Node` vertraut und implementieren Sie die Konstrukturfunktionen `make_file` und `make_directory`.

b) Implementieren Sie die Funktion `free_node`, die den allokierten Speicher eines Knotens und gegebenenfalls aller enthaltener Knoten freigibt.

c) Implementieren Sie die Funktion `insert_into_directory`, die einen Knoten in einen Ordner einfügt. Erzeugen Sie hier keine Kopie des Knotens, sondern fügen Sie den Zeiger auf den Knoten in das Dateisystem ein. Beachten Sie, dass die Position des neuen Knotens in dem Array so gewählt werden muss, dass das Array alphabetisch sortiert bleibt. Nutzen Sie dafür die `strcmp` Funktion.

d) Implementieren Sie die Funktion `print_node`, die die enthaltenen Dateien des Dateisytems ausgibt. Für das in der `main` Funktion definierte Dateisystem sollten Sie folgende Ausgabe erzeugen:

```
/archive.a
/hello.c
/home/user/config.cfg
/home/user/game.exe
/home/user/image.jpg
/system/4_processes.txt
/system/8_configuration.txt
/system/secret/flag.txt
/test.txt
/world.c
```

e) Implementieren Sie die Funktion `find_node`, die mit einer Pfadangabe den entsprechenden Knoten zurückgibt. Dabei sollen sowohl Dateien (z.B. "`/system/4_processes.txt`") und Ordner (z.B. "`/home/user`" und "`/home/user/`") zurückgegeben werden können. Existiert der Pfad nicht, soll `NULL` zurückgegeben werden. Sie können davon ausgehen, dass sie nur korrekt formatierte Eingaben behandeln müssen. Erzeugen Sie auch hier keine Kopie des Knotens, sondern geben Sie den Zeiger auf den enthaltenen Knoten zurück.

f) Kommentieren Sie `report_memory_leaks(true)` am Beginn der `main` Funktion aus. Stellen Sie sicher, dass der komplette Speicher wieder korrekt freigegeben wird.

Hinweis:

Mit der Funktion `snprintf` können Sie eine formatierte Ausgabe in einen String speichern. Die Parameter für die Formatierung entsprechen denen der `printf` Funktion.

```
int snprintf(char * str, size_t size, const char * format, ...);
```

- `str` ist der String, in dem der formatierte Text gespeichert wird.
- `size` ist die Größe des Strings (mit dem terminierenden Null-Byte).
- `format` gibt das Format der Ausgabe wie in `printf` an.
- Auf `format` folgen die zu formatierenden Werte.