Human-Computer Interaction Group

Leibniz Universität Hannover

Assignment 10

Programmieren 1 – WiSe 25/26

Prof. Dr. Michael Rohs, Falk Stock, M.Sc.

Alle Assignments (bis auf das erste) müssen in Zweiergruppen bearbeitet werden. Ein Gruppenmitglied kann dabei die Lösung der Zweiergruppe gebündelt abgeben. Einzige Ausnahme ist dabei das erste Assignment, wo jedes Gruppenmitglied einzeln abgeben muss. 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 Donnerstag, den 15.01. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2025/Prog1. Die Abgabe muss aus einer einzelnen ZIP-Datei bestehen, die den Quellcode, eine PDF für Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen auf.

Zum Bestehen der Studienleistung müssen Sie mindestens zwei von vier Punkten pro Assignment erzielen. Möchten Sie zusätzlich den Klausurbonus erreichen, müssen Sie über alle Assignments hinweg 75% der Punkte erreichen.

Hinweis: prog1lib

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

Aufgabe 1: Mensa-Simulator (1 Punkt)

In dieser Aufgabe wird eine interaktive Mensasimulation implementiert. Ziel des Spiels ist es, die richtigen Gerichte zu servieren, um so die Reputation der Mensa zu erhöhen.

Die Mensa bietet auf der Tageskarte 5 verschiedene Gerichte (`const String menu[]`) an. Bevor die ersten Studierenden kommen, kocht die Küche bereits 5 zufällige Gerichte von der Tageskarte. Jedes Mal, wenn ein Gericht an einen Studierenden ausgegeben wurde, kocht die Küche ein weiteres zufälliges Gericht. Die fertigen Gerichte werden in der Liste `food` gespeichert. Es stehen zunächst 3 Studierende in der Schlange und warten auf ihr Essen. Die Studierenden sind in der Liste `students` durch ihre zufällig gewählten Essenswünsche repräsentiert.

Der Benutzer der Simulation spielt die Rolle einer/eines Mensa-Bediensteten und hat das Ziel, durch Ausgabe der gewünschten Gerichte die Reputation der Mensa zu maximieren. Wenn ein Essenswunsch erfüllt wurde, steigt die Reputation um 1. Jedes Mal, wenn die Reputation der Mensa steigt, kommt ein weiterer Studierender dazu. Wenn ein falsches Gericht ausgegeben wurde, sinkt die Reputation um 1 und das Essen wird zurückgenommen. Wenn ein Essenswunsch nicht erfüllt werden kann, weil das entsprechende Gericht gerade nicht fertig ist, sinkt die Reputation der Mensa um 2. Der Studierende verlässt daraufhin die Mensa ohne gegessen zu haben.

Die Template-Datei für diese Aufgabe ist `mensa_game.c`. Bearbeiten Sie die mit `todo` markierten Stellen.

a) Implementieren Sie die Funktion **length_list()**, die die Anzahl der Elemente der Liste zurückgibt.

b) Implementieren Sie die Funktion **get_list()**, die das Listenelement an der Indexposition zurückgibt. Das erste Listenelement befindet sich an Indexposition 0.

c) Implementieren Sie die Funktion **free_list()**, die eine `List` mitsamt Inhalt freigibt. Beachten Sie, dass die Liste Besitzer ("owner") der Listenelemente (`value`) ist und alle Listenelemente dynamisch allokiert sind. Diese müssen also auch freigegeben werden.

d) Implementieren Sie die Funktion **append_list()**, die ein neues Listenelement hinten an die Liste anfügt.

e) Implementieren Sie die Funktion **print_situation()**, die den aktuellen Zustand der Simulation wie im obigen Beispiel ausgibt.

f) Implementieren Sie die Funktion **finish()**, die eine abschließende Meldung ausgibt, allen dynamisch allokierten Speicher freigibt und das Programm beendet. Achten Sie darauf, dass keine Memory Leaks auftreten. Die Leaks, welche durch `i_input()` entstehen, dürfen dabei jedoch ignoriert werden.

g) Implementieren Sie die Funktion **run_mensa()**, die es erlaubt, interaktiv gewünschte Essen auszugeben. Es gibt nur eine Essensausgabe vor der die Studierenden in einer Schlange warten und nacheinander bedient werden. Dazu wird der Index des gewünschten Gerichts in der Liste der fertiggestellten Essen eingegeben. Falls ein Essenswunsch nicht erfüllt werden kann, soll `-1` eingegeben werden. Die Eingabe von `-2` beendet das Programm. Implementieren Sie die Funktion entsprechend dem gezeigten Beispiel.

Hier ein möglicher Simulationsablauf:

```
Vielen Dank! Ich liebe die Mensa!
fertige Essen: [Schnitzel, Pasta, Dessert, Schnitzel, Salat]
nächster Essenswunsch: Dessert (2 hungrige Studierende warten)
Reputation der Mensa: -1
> 2
Vielen Dank! Ich liebe die Mensa!
fertige Essen: [Schnitzel, Pasta, Schnitzel, Salat, Pasta]
nächster Essenswunsch: Pasta (2 hungrige Studierende warten)
Reputation der Mensa: 0
> 1
Vielen Dank! Ich liebe die Mensa!
fertige Essen: [Schnitzel, Schnitzel, Salat, Pasta, Veggie]
nächster Essenswunsch: Veggie (2 hungrige Studierende warten)
Reputation der Mensa: 1
> 4
Vielen Dank! Ich liebe die Mensa!
fertige Essen: [Schnitzel, Schnitzel, Salat, Pasta, Veggie]
nächster Essenswunsch: Veggie (2 hungrige Studierende warten)
Reputation der Mensa: 2
> 1
Schnitzel möchte ich nicht! Ich möchte Veggie!
fertige Essen: [Schnitzel, Schnitzel, Salat, Pasta, Veggie]
nächster Essenswunsch: Schnitzel (1 hungrige Studierende warten)
Reputation der Mensa: 1
> 2
```

```
Salat möchte ich nicht! Ich möchte Schnitzel!
Fertig für heute. Die Mensa schließt.
Finale Reputation der Mensa: 0
```

Aufgabe 2: Wunschbaum (1 Punkt)

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. Das heißt 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`. Die Datei, welche die Wünsche enthält ist `wishes.txt`.

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 zuerst 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 3: Dateisystem (2 Punkte)

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 63 Zeichen. Ordner enthalten ein Array mit enthaltenen Knoten, Dateien enthalten einen Pointer 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 **new_file()** und **new_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 Pointer 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`") zurückgegeben werden können. Existiert der Pfad nicht, soll `NULL` zurückgegeben werden. Erzeugen Sie auch hier keine Kopie des Knotens, sondern geben Sie den Pointer auf den enthaltenen Knoten zurück.

f) 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.