Institut für Informatik
Prof. Dr. Stefan Conrad

Sommersemester 2020
Übung 6

Hardwarenahe Programmierung
Gruppe 14 (Lars)

Es ist soweit: In dieser Übung nutzen Sie dynamische Speicherverwaltung in C! Außerderm verwenden Sie `valgrind`*, um Ihre Programme auf Speicherlecks zu untersuchen.*

- *Denken Sie wie immer daran, Ihre Lösungen im ILIAS hochzuladen und den Test im ILIAS zu absolvieren!*

Aufgabe 1 *Studenten-Daten (**Pflichtaufgabe**)*

Einige Listen von Matrikelnummern und Studenten sind durcheinandergeraten. Eine Liste sieht Beispielsweise so aus:

```
78901 Mini
64578 Track
23456 Donald
34567 Gundel
```

Es darf angenommen werden, dass Zeilen nicht länger als 100 Zeichen sind und dass in der ersten Spalte nur Zahlen stehen.

Um die Listen wieder in Ordnung zu bringen, gehen Sie folgendermaßen vor:

(a) Die Studenten sollen nacheinander aus einer Datei in eine verkettete Liste eingelesen werden.

(b) Fügen Sie einen Studenten direkt vor anderen Studenten ein, die eine größere Matrikelnummer haben.

(c) Danach sollen die Daten aus der verketteten Liste in eine weitere Datei ausgegeben werden.

(d) Implementieren Sie dazu die nötigen Funktionen in `liste.c`. Die `main`-Funktion ist Ihnen in der Datei `liste.c` schon vorgegeben und darf nicht verändert werden.

(e) Der Speicher der Listenknoten muss dabei dynamisch mit `malloc` und `free` verwaltet werden.

(f) Abschließend muss aller allozierter Speicher freigegeben werden.

(g) Nutzen Sie `valgrind`, um ihre Speicherverwaltung zu überprüfen.

(h) Testen Sie Ihr Programm mit dem Test-Skript `test.sh`.

Aufgabe 2 *Ringpuffer (**Pflichtaufgabe**)*

In dieser Aufgabe sollen Sie einen Ringpuffer für die Speicherung von Strings implementieren. Ein Puffer ist ein linearer Speicher: Elemente werden beginnend bei Position 0 nacheinander eingefügt, bis der Puffer voll ist. Jedes Element belegt eine Position (unabhähnig vom Speicherbedarf des Elements). Die Besonderheit eines Ringpuffers ist, dass hier wieder bei Position 0 angefangen wird, wenn der Puffer voll ist. Alte Elemente können dabei überschrieben werden.

Sowohl der Speicher für das Array `strings` als auch der Speicher für die darin verwalteten Strings muss dynamisch alloziert werden.

Achten Sie auf eine sorgfältige Speicherverwaltung und -freigabe sowie die Erfüllung aller Unittests!

(a) Definieren Sie in der Datei `ringpuffer.h` die Datenstruktur `buffer`. Die Datenstruktur muss mindestens die folgenden Felder verwenden:

- `size`: Größe des Puffers.
- `pos`: Position im Puffer, an die als nächstes geschrieben werden soll; anfangs 0.
- `strings`: Ein Array von dynamisch allozierten Strings.

Wählen Sie für die Felder geeignete Datentypen.

(b) Implementieren Sie die in `ringpuffer.c` vorgegebenen Funktionsprototypen und prüfen Sie Ihre Implementierungen mit den vorgegebenen Unittests.

(c) Überprüfen Sie Ihr Programm mit `valgrind` und `make run`. Achten Sie besonders auf den Fall, wo bereits belegte Positionen überschrieben werden. Die Speicherlecks der Kategorie `still reachable`, die durch das `check`-Framework verursacht werden, dürfen hierbei ausnahmsweise ignoriert werden.