COM SYS Communication & Distributed Systems

Betriebssysteme und Systemsoftware
Sommersemester 2014

RWTH AACHEN UNIVERSITY

Übungsblatt 2

Abgabe: 12. Mai 2014

Aufgabe 2.1: C-Programmierung: Lineare Listen (4+3+0.5+1+1+0.5 = 10 Punkte)

Zeiger sind gut zum Aufbau komplexer Datenstrukturen geeignet. Ein Beispiel dafür sind **lineare Listen**. Eine lineare Liste ist eine Menge von Objekten gleichen Typs, die über Zeiger miteinander verkettet sind. Ein Beispiel ist in Abbildung 1 gegeben. Hier wird eine Liste verwaltet, in der Studierende mit Name und Matrikelnummer erfasst sind. Die Liste ist nach Matrikelnummern aufsteigend sortiert.

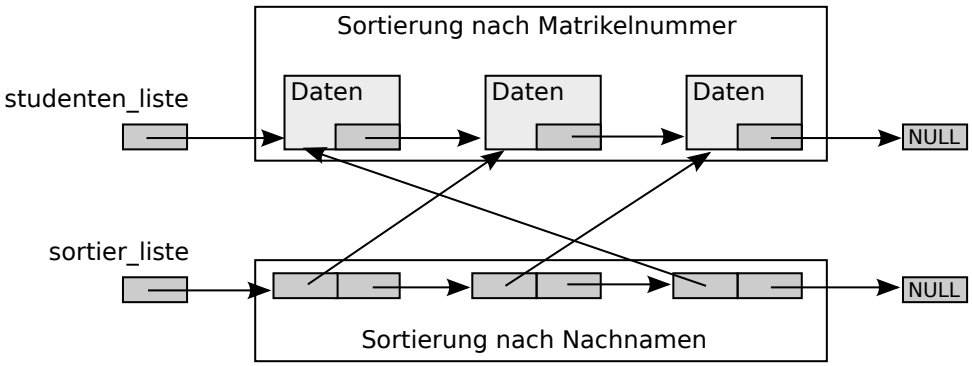

Abbildung 1: Aufbau von verketteten Listen.

Folgende Zugriffsfunktionen stehen für die lineare Liste zur Verfügung:

`is_empty()` testet, ob die Liste leer ist

`enqueue()` fügt einen neuen Studierenden in die Liste ein

`dequeue()` löscht einen Studierenden aus der Liste

`get_student()` liest die Daten zu einem Studierenden aus der Liste aus

a) Als Anhang zum Übungsblatt finden Sie die Quellcodedatei `u2_01.c`. Studieren Sie den Quellcode, compilieren Sie das Programm und testen Sie es. Sie werden feststellen, dass die Funktionen `enqueue` und `dequeue` noch nicht implementiert sind. Implementieren Sie beide Funktionen mit Hilfe der im Codegerüst angegeben Kommentare. Ihr Programm muss mit `gcc -Wall -O OUTFILE infile.c` ohne Warnungen kompilieren. Dokumentieren Sie in der Abgabe die Ausgabe des Programms.

b) Es soll ermöglicht werden, die Liste statt nach Matrikelnummer nach einem beliebigen Merkmal zu sortieren. Schreiben Sie hierzu eine Funktion, die eine weitere lineare Liste `sortier_liste` erzeugt. Die Elemente dieser Liste sollen keine eigenen Einträge des jeweiligen Datums enthalten, sondern lediglich auf die ursprünglichen Elemente der Datenbank verweisen (vgl. Abbildung 1). Nutzen Sie zur Implementierung Funktionspointer um die jeweilige Vergleichsmethode zur Sortierung einfach anzupassen. Übergeben Sie den Funktionspointer als Parameter zu Ihrer Funktion, die die neue `sortier_liste` erzeugt. Schreiben Sie zwei Vergleichsfunktionen, die als Parameter übergeben werden können, um die Liste entweder nach Vor- oder Nachname sortieren zu können.

Hinweis: Ihre Vergleichsmethoden müssen alle die selbe Methodensignatur aufweisen; halten Sie es also allgemein und verwenden sie stets diese Signatur: `int compare(stud_type* a, stud_type* b)`. Typischerweise nutzt man einen `int` als Rückgabewert um $a < b$, $a == b$ und $a > b$ durch -1, 0 und 1 darzustellen.

c) Nehmen Sie an, Sie müssten als nächste Aufgabe eine verkettete Liste erstellen, die anstelle von Datensätzen mit Namen und Matrikelnummern von Studierenden nun Hörsaalnamen und ihre GPS-Koordinaten (als zwei float-Zahlen für die Koordinaten) erfassen soll. Wie viele der vier Zugriffsunktionen müssten geändert werden, um eine solche Liste zu unterstützen? Warum?

Hinweis: Sie müssen die neue Liste und die dazugehören Funktionen nicht implementieren, lediglich die obigen Fragen beantworten.

d) Implementieren Sie Funktionen, um die im Speicher befindliche Datenbank permanent auf das Dateisystem zu schreiben bzw. von dort in den Hauptspeicher zurückzuladen.

e) Erklären Sie die Funktion des folgenden Makros:

```
#define position(typ, el) ((unsigned long)(&((typ *)0)->el))
```

Das Makro wird z.B. folgendermaßen verwendet: `position(stud_type, next_student)`.

Was ist der Rückgabewert? Was bedeutet dieser?

f) Was macht diese Funktion falsch, bzw. warum ist sie, egal was sie konkret implementiert, in den meisten Fällen nutzlos?

```
int* was_mach_ich_falsch(void) {
    int x;
    /* Hier wird etwas sinnvolles
     * mit der Variable gemacht
     * z.B. x = 42; */
    return &x;
}
```

Hinweis: Es hat nichts damit zu tun, dass hier auf einem Integer gearbeitet wird. `int` bzw. `int*` sind durch einen beliebigen Typen und seinen Pointer (also `typ` bzw. `typ*`) ersetzbar.

Aufgabe 2.2: Bash (2+1+2 = 5 Punkte)

a) Schreiben Sie ein Shell-Skript, dass alle `x` Sekunden ausgibt ob ein durch eine `PID` identifizierter Prozess läuft oder nicht. Übergeben Sie die Anzahl an Sekunden und die PID als Parameter an das Skript, sodass es wie folgt aufgerufen werden kann: `$ script [PID] [SECONDS]`. Ihnen sind keine Einschränkungen gegeben, welche Befehle Sie dafür verwenden.

b) Was bewirkt das folgende Skript?

```
S=0
for f in $(find . -name "*.c"); do S=$(($S + $(wc -l $f | awk '{ print $1 }'))); done
echo $S
```

c) Schreiben Sie ein Shell-Skript, das die Verzeichnisstruktur des aktuellen oder eines als Parameter angegebenen Verzeichnisses auflisten kann. Schreiben Sie dazu eine rekursive Funktion, die in Unterverzeichnisse herabsteigt. Geben Sie das Verzeichnis und die Dateien auf dem Bildschirm aus. Nutzen Sie Einrückungen um die Zugehörigkeit von Dateien und Verzeichnissen zueinander zu kennzeichnen. Dies kann dann zum Beispiel so aussehen:

```
File: linux-3.14/COPYING
File: linux-3.14/CREDITS
Directory: linux-3.14/Documentation
  File: linux-3.14/Documentation/00-INDEX
  Directory: linux-3.14/Documentation/ABI
    File: linux-3.14/Documentation/ABI/README
    Directory: linux-3.14/Documentation/ABI/obsolete
```

```
  File: linux-3.14/Documentation/ABI/obsolete/proc-sys-vm-nr_pdflush_threads
  File: linux-3.14/Documentation/ABI/obsolete/sysfs-bus-usb
  File: linux-3.14/Documentation/ABI/obsolete/sysfs-class-rfkill
  File: linux-3.14/Documentation/ABI/obsolete/sysfs-driver-hid-roccat-koneplus
  File: linux-3.14/Documentation/ABI/obsolete/sysfs-driver-hid-roccat-kovaplus
  File: linux-3.14/Documentation/ABI/obsolete/sysfs-driver-hid-roccat-pyra
Directory: linux-3.14/Documentation/ABI/removed
  File: linux-3.14/Documentation/ABI/removed/devfs
  File: linux-3.14/Documentation/ABI/removed/dv1394
  ...
```

Hinweis: es ist möglich, in der bash Funktionen zu definieren. Die Syntax finden Sie z.B. hier:

`http://tldp.org/LDP/abs/html/functions.html`

Aufgabe 2.3: Bash: Textverarbeitung (0.5+0.5+3+1 = 5 Punkte)

Im Folgenden sollen Sie die Textbearbeitung mittels bash betrachten. Beantworten Sie dazu die folgenden Fragen. Alle diese Probleme können und sollen als "Einzeiler"-Shell-Skripte durch das Kombinieren verschiedener Kommandozeilenprogramme, aber ohne Schleifen und bash-Variablen gelöst werden!

a) Schreiben Sie ein Skript, welches Ihnen zu allen Dateien (und Verzeichnissen) im aktuellen Verzeichnis nur die Dateigröße und den Dateinamen ausgibt. Die Ausgabe braucht nicht schön formatiert zu sein, beide Angaben zu einer Datei können einfach durch ein einzelnes Leerzeichen getrennt ausgegeben werden.

b) Modizieren Sie die Ausgabe aus dem vorherigen Aufgabenteil so, dass Datum und Dateiname in umgekehrter Reihenfolge ausgegeben werden.

c) Bei der Anmeldung zu den Übungen konnten Sie einen String als "Teampräferenz" festlegen. Personen mit gleichem String wurden bevorzugt der gleichen Übungsgruppe zugeordnet. Im Lernraum finden Sie die Datei `teamnamen.txt`, die in jeder Zeile einen Teamnamen enthält, der von einem Studierenden vergeben wurde. Schreiben Sie ein Skript, das ausgibt, wie viele Dreiergruppen sich aus den Teampräferenzen ergeben. Wie müssen Sie ihr Skript ändern, um die Anzahl der Zweiergruppen auszugeben? Wie viele Einergruppen (Teamname, der nur einmal auftaucht) gibt es? Wie viele Studierende haben keine Teampräferenz angegeben?

   Hinweis: `uniq`

d) Schreiben Sie ein Skript, das mit einem Aufruf die Anzahl der Einer-, Zweier- und Dreiergruppen, wie sie im vorherigen Aufgabenteil definiert wurde, ausgibt. Dabei ist sowohl eine Ausgabe der Form "<Gruppengröße> <Anzahl>" (oder umgekehrt) in drei Zeilen erlaubt, als auch die reine Ausgabe der Anzahlen in aufsteigender Gruppengröße.

   Hinweis: Sie dürfen, müssen aber nicht die Anzahl der Studierenden ohne Teampräferenz ("Nullergruppen") ausgeben.