4 Speicherverwaltung

Bearbeiten Sie auch weitere Aufgaben des gestrigen Aufgabenzettels.

Aufgabe 4.1. (Speicherverwaltung – leicht) Nehmen Sie eines der Programme, dass Sie gestern geschrieben haben und bauen Sie es von Arrays fester Größe auf **malloc**(3) und **free**(3) um. Stellen Sie sicher, dass aller von Ihnen allozierter Speicher beim Programmende wieder dealloziert wird. Verwenden Sie **valgrind**(1), um zu prüfen, ob Sie damit Erfolg hatten.

Aufgabe 4.2. (Kopierprogramm – leicht) Schreiben Sie ein Programm, mit dem Sie eine Datei in eine andere Kopieren können. Der Nutzer soll in der Lage sein, das Programm wie folgt aufzurufen:

```
./kopiere Quelle Ziel
```

Folgende Funktionen sind zur Lösung der Aufgabe hilfreich: **fopen**(3), **fread**(3), **fwrite**(3), und **fclose**(3).

Implementieren Sie den Kopiervorgang, indem Sie wiederholt in einer Schleife ein Stück der Eingabedatei in einen Puffer einlesen und dieses in die Ausgabedatei schreiben. Lesen Sie die Dokumentation von **fread**(3), spezifisch den Abschnitt *Return Value*, um herauszufinden, wie Sie das Ende der Eingabedatei erkennen.

Experimentieren Sie mit verschiedenen Puffergrößen. Können Sie einen Unterschied in der Geschwindigkeit feststellen?

Aufgabe 4.3. (sort(1) – leicht) Schreiben Sie eine eigene Implementierung des Programmes **sort**(1). Ihre Implementierung soll in der Lage sein, Dateien mit beliebig vielen beliebig langen Zeilen zu sortieren. Sie müssen keine Optionen implementieren und nur die Eingabe von Text aus der Standardeingabe unterstützen.

Für diese Aufgabe nützliche Funktionen: **getline**(3), **qsort**(3), und **strcmp**(3).

Aufgabe 4.4. (PPM-Bilder lesen / schreiben – mittel) PPM (portable pixmap) ist eine Familie einfacher Bildformate. Recherchieren Sie im Internet, wie PPM-Bildformate funktionieren und schreiben Sie eine Funktion, die ein PPM-Bild einließt, und als Datenstruktur im Computer ablegt. Sie brauchen nur PPM-Bilder vom Typ `P3` zu unterstützen.

Das Bild soll durch folgende Struktur dargestellt werden:

```c
/* ein einzelnes Pixel */
struct pixel {
    int rot, blau, gruen;
};

/* ein PPM Typ P3 Bild */
struct ppm_bild {
    /* Kopfdaten */
    int breite, hoehe;
    int max_helligkeit;

    /* mit malloc() allozieren */
    struct pixel *leinwand;
};
```

Anschließend schreiben Sie eine Funktion, die eine `ppm_bild` Struktur als Typ `P3` PPM-Bild in eine Datei schreibt.

Prüfen Sie die Korrektheit Ihrer Funktion, indem Sie diese in einem Programm verwenden, welches ein PPM-Bild einließt und wieder in eine Datei schreibt. Das entstehende Bild muss identisch zum Original sein.

Aufgabe 4.5. (PPM-Bildverarbeitung – mittel/schwer) Schreiben Sie ein einfaches Bildverarbeitungsprogramm, welches auf Typ `P3` Bildern arbeitet. Das Bildverarbeitungsprogramm soll in der Lage sein, einfache Transformationen auf dem Bild auszuführen. Hier sind ein paar Beispiele nach aufsteigender Schwierigkeit sortiert:

- Konvertierung von Farbe nach Graustufen oder Schwarzweiß
- Zuschneiden des Bildes
- Drehung um 90°, 180°, und 270°, sowie Spiegelung
- Filterung mit Faltungsmatrizen (Gaußfilter, Schärfungsfilter, Kantenfilter, etc.)
- Skalierung
- Drehung um beliebige Winkel

Sie können sich auch gerne eigene Operationen ausdenken.

Aufgabe 4.6. Sudokulöser – schwer Schreiben Sie ein Programm, das ein Sudoku-Puzzle ließt und versucht, eine Lösung zu finden. Geben Sie eine Fehlermeldung aus, wenn es keine oder mehr als eine Lösung gibt. Sonst geben Sie die korrekte Lösung des Puzzles ein.

Verwenden Sie die Methode des *rekursiven Backtrackings* oder denken Sie sich ggf. eine eigene Methode zur Lösung von Sudoku-Puzzlen aus.