Heinrich Heine Universität Düsseldorf

Hardwarenahe Programmierung - Übungsblatt 3

- Hinweise:
  - In diesem Übungsblatt geht es um dynamische Speicherverwaltung. Nutzen Sie deshalb unbedingt `valgrind`, um Speicherzugriffsfehler zu finden.
  - `valgrind` und `fsanitize` können nicht jede Art von Fehlern finden. Machen Sie sich deshalb auch selber Gedanken darüber, ob Ihr Programm richtig sein kann und verlassen Sie sich nicht zu sehr darauf, dass die Tests Ihre Fehler aufdecken.

  Termine für Einzelgespräche:
  - Mittwoch, der 27. Oktober, von 12:30 Uhr bis 16:00 Uhr.
  - Donnerstag, der 28. Oktober, von 10:30 Uhr bis 14:00 Uhr.
  - Freitag, der 29. Oktober, von 8:30 Uhr bis 12:00 Uhr.

Pflichtaufgabe 1 Structs, Enums und Typedefs

Lesen Sie sich zunächst alle Schritte durch und wählen Sie die Namen Ihrer Datenstrukturen so dass diese zu den vorgegebenen Signaturen passen. Eventuell müssen Sie ein `typedef` nutzen.
Sie sollen nicht prüfen ob der Wochentag zum Datum passt.

1. Schreiben Sie in `date.h` im Ordner `struct` ein `enum`, in dem ein Wochentag gespeichert werden kann. Das `enum` soll so gestaltet sein, dass der Sonntag den Wert 0 hat, Montag den Wert 1, usw. bis zum Samstag.
2. Nutzen Sie dieses `enum` in einem `struct` in dem ein Datum mit Wochentag gespeichert werden kann.
3. Implementieren Sie die folgenden Funktionen in `date.c`
   - `date* create_date(int day, int month, int year, enum weekday weekday)` welche ein Datum erstellt. Fehlerhafte Parameter (z.B. negative Werte oder Monat größer als 12) müssen nicht beachtet werden.
   - `char* date_to_string(date* d)` welche ein Datum übergeben bekommt und einen String mit diesem Datum im Format `dd.mm.yyyy` zurückgibt. Dabei ist d = day, m = month und y = year. Die Zahlen müssen entsprechend mit Nullen aufgefüllt werden, wie zum Beispiel `09.10.2021` oder `01.01.0001`.
     Der Speicher für diesen String muss dynamisch alloziert werden (zum Beispiel mit den Funktionen `malloc` oder `calloc`).
4. Erstellen Sie in `exam.h` eine Struktur worin Daten für eine Klausur gespeichert werden können. Eine Klausur hat ein Datum und eine maximal erreichbare Höchstpunktzahl `points`.
5. In `exam.c` sind folgende Funktionen zu schreiben
   - `struct exam create_exam(date* d, int points)` erstellt eine Klausur. Negative Punkte müssen nicht abgefangen werden.

- `int pass_limit(struct exam *exam)` gibt die Anzahl der Punkte zurück welche man mindestens zum Bestehen braucht. Diese Beträgt normalerweise 50% der Gesamtpunkte. Die Fachschaft hat jedoch durchgesetzt, dass für Klausuren welchen an einem Freitag den 13ten geschrieben werden nur 42% der Punkte zum Bestehen benötigt sind. Die Punktzahl wird zum Vorteil der Studierenden stehts abgerundet.

6. Erweitern Sie dafür die `date.c` um eine Funktion `int is_friday_13th(date* d)` welches einen Wert **ungleich 0** zurückgibt wenn das Datum an einem Freitag den 13ten ist und **0** sonst.

7. Sie können ihre Abgabe mit den Skripten und Tests welche wir zur Verfügung stellen testen:

   - `make test` testet Ihre Abgabe.
   - `./codestyle.py` (oder `make codestyle`) führt nur den codestyle Test aus.
   - `make run_exam_tests` bzw. `make run_date_tests` führt nur die jeweiligen Tests einzeln aus.

Pflichtaufgabe 2 2D-Arrays

In dieser Aufgabe sollen Sie durch Leerzeichen (*whitespace*) getrennte Wörter in ein zweidimensionales `char`-Array einlesen und durchnummeriert wieder ausgeben.

Eine Eingabe könnte beispielsweise so aussehen:

```
Duck        Goose
  Chicken
Owl
```

Erwartete Ausgabe:

```
There are 4 words:
1 Duck
2 Goose
3 Chicken
4 Owl
```

Hierfür sollen die folgenden Funktionen in der Datei `words/words.c` implementiert werden:

- `char** read_words(size_t *num_words)` liest Wörter von der Standardeingabe in ein zweidimensionales `char`-Array ein. Dieses muss dynamisch alloziert werden (zum Beispiel mit den Funktionen `malloc`, `realloc` und/oder `calloc`), da die Anzahl der einzulesenden Wörter zu Anfang nicht bekannt ist. Die Anzahl soll über den Zeiger `num_words` zurückgegeben werden.

  Wörter sollen nicht länger als 13 Zeichen sein. Falls längere Wörter übergeben werden darf eine beliebige Ausgabe produziert werden, aber das Programm darf *nicht* abstürzen! Damit der Nullterminator passt muss ein entsprechendes Array mindestens für 14 Zeichen Platz bieten. Die Funktion `scanf` bietet sich zum Einlesen an. Aus Sicherheitsgründen **muss** beim Format-String die Anzahl der einzulesenden Zeichen angegeben werden.

  Lesen Sie die Dokumentation der Funktion `scanf`! Am Rückgabewert der Funktion können Sie erkennen, wann die Eingabe beendet ist.

- `void print_words(char **words, size_t num_words)` gibt die Wörter aus.

- `void free_words(char **words, size_t num_words)` gibt den allozierten Speicher frei.

Hinweise:

- Die erste Zeile der Ausgabe wird schon von der `main`-Funktion ausgegeben.

- Die Anzahl der Wörter ist nur durch die Größe des Datentyps `size_t` begrenzt. Ihr Programm sollte also auch mit zum Beispiel 2^{40} Eingabewörtern korrekt funktionieren, wenn der Computer einen ausreichend großen Hauptspeicher hätte. Der Datentyp `int` ist in der Regel **nicht** geeignet, um bis 2^{31} oder höher zu zählen.
- Bei dieser Aufgabe sind nur die Zahlen 0, 1, 13 und 14 erlaubt, damit niemand auf die Idee kommt, ein Array fester Größe für `words` zu allozieren.
- Die Datei `main.c` darf nicht verändert werden.
- Testen Sie Ihr Programm mit dem Test-Skript `test.sh`.
- Testen Sie Ihr Programm zusätzlich nochmal selber mit dem Programm `valgrind`, um weitere Speicherzugriffsfehler zu erkennen.

Pflichtaufgabe 3 Verkettete Liste

In dieser Aufgabe geht es um verkettete Listen und wie man mit ihnen arbeitet. Sie haben eine nach *internationaler Standardbuchnummer* (ISBN) sortierte Reihe von Büchern gegeben.

Leider sind die ISBNs dieser Bücher durcheinander geraten. Alle Zahlen sind um eine bestimmte Anzahl verschoben worden. Bei einer Verschiebung um den Wert 2 würde beispielsweise die ISBN `660-4-8764-7593-5` zu `882-6-0986-9715-7`, d.h. einzelne Zahlen wurden um 2 inkrementiert, aber die Zahlen 8 und 9 "laufen über" zu 0 und 1.

Ihre Aufgabe ist es nun diese Verschiebung rückgängig zu machen. Dazu erhalten Sie eine Eingabe mit Zeilen in folgendem Format:

- Ein Buchtitel von höchstens 10 Zeichen der Sorte A – Z bzw. a – z (d.h. ohne Leerzeichen).
- Das Jahr, in dem das Buch veröffentlicht wurde (0 bis 9999 einschließlich).
- Eine ISBN der Form `XXX-X-XXXX-XXXX-X`, wobei `X` eine Zahl von 0 bis 9 ist.
- Eine Zahl von 0 bis 9, die angibt, um wie viel die Zahlen in der ISBN verschoben wurden.

Einzelne Einträge dieser Zeilen sind durch ein oder mehrere Leerzeichen getrennt. Das Format muss von Ihnen nicht überprüft werden.

Sie erhalten von uns die Datei `books/books.c`, in der Sie die folgenden Funktionen implementieren müssen:

- Die Funktion `Book* read_library(void)` muss die Bücher von der Standardeingabe in eine dynamisch allozierte verkettete Liste einlesen, die ISBN korrigieren und danach wieder in eine (nach ISBN) sortierte Reihenfolge bringen. Das `struct` mit Namen `Book` ist ihnen dazu bereits vorgegeben. Ob Sie die einzelnen Bücher gleich beim Einlesen an der richtigen Stelle einfügen oder ob sie die Liste danach sortieren bleibt Ihnen überlassen.
- Die Funktion `void free_library(Book *list)` soll den allozierten Speicher wieder freigeben.

Die Datei `books/main.c` darf nicht verändert werden und enthält bereits eine vorgegebene Funktion `void print_library(Book *list)`.

Eine Eingabe könnte so aussehen:

```
Third      2010     082-6-1575-4271-1    3
Last       2012     197-5-7351-5540-4    2
FirstBook  2021     529-2-7323-8153-2    4
Second     1999     790-3-7871-5099-1    2
```

Die zugehörige Ausgabe würde dann so aussehen:

```
ISBN: 185-8-3989-4719-8, Year: 2021, Title: FirstBook
ISBN: 578-1-5659-3877-9, Year: 1999, Title: Second
ISBN: 759-3-8242-1948-8, Year: 2010, Title: Third
ISBN: 975-3-5139-3328-2, Year: 2012, Title: Last
```

Hinweise:

- Beachten Sie, dass die Ausgabe nach ISBN sortiert sein muss.
- Benutzen Sie die Funktionen `malloc` oder `calloc`, um die Knoten der verketteten Liste einzeln zu allozieren und anschließend `free`, um die Knoten einzeln freizugeben.
- Die Länge der Liste ist nicht beschränkt, d.h. Ihr Programm muss eine beliebige Anzahl an Zeilen verarbeiten können.
- Denken Sie daran, ihre Variablen zu initialisieren. Die Funktion `malloc` initialisiert den allozierten Speicherbereich **nicht** automatisch auf den Wert 0.
- Testen Sie Ihr Programm mit dem Test-Skript `test.sh`
- Testen Sie Ihr Programm zusätzlich nochmal selbst mit dem Programm `valgrind`, um weitere Speicherzugriffsfehler zu erkennen.