Aufgabe 1. ▲ – ▲▲
Queue als Adapterstruktur
In der echten Welt benutzen Sie bereits vorhandene Bibliotheken und schreiben auf diesen aufbauend Ihre eigenen Funktionalitäten.
Probieren wir dies einmal. Gegeben ist eine Listenimplementation als Header und C-Files (zum Download bei precampus). Machen Sie sich mit dem Interface vertraut.

a) Eine Queue ist anschaulich eine Warteschlage. Definieren Sie die Funktionsdeklarationen in queue.h mithilfe der Listenimplementation.

b) Implementieren Sie die Queue noch einmal in einem seperaten .c File (nutzen Sie aber die selbe Header-Deatei) mithilfe des IntVectors vom letzten Aufgabenblatt. Sie sehen: Die Implementation ist ein nebensächliches Detail, das z.B. über Performanz entscheidet. Die Semantik wird durch das Interface bestimmt.

c) Angenommen die maximale Anzahl der Elemente in der Queue stünde bereits fest. Fällt Ihnen eine (effiziente) Implementation ein, die ohne dynamische Speicherverwaltung auskommt? Diskutieren Sie gerne untereinander wie eine solche Implementation aussehen könnte.

Aufgabe 2. ▲▲ – ▲▲▲
Wiederholung / Anwendung von dynamischer Speicherverwaltung
Implementiere einige Funktionen um mit quadratischen Matrizen umzugehen:

a) Eine Funktion, die Speicher für eine quadratische Matrix allokiert, eine um ihn freizugeben, eine um sie auszugeben, eine um sie zur Einheitsmatrix zu initialisieren (das ist die Matrix mit 1en auf der Hauptdiagonale und 0en sonst), eine um einen Wert auszulesen und schließlich eine um einen Wert zu setzen.

```
1 IntMatrix *IntMatrix_new(int n);
2 void IntMatrix_delete(IntMatrix *m);
3 void IntMatrix_print(IntMatrix *m);
4 void IntMatrix_id(IntMatrix *m);
5 int IntMatrix_get(IntMatrix *m, int x, int y);
6 void IntMatrix_set(IntMatrix *m, int x, int y,
```

```
7                                   int value);
```

b) Eine Funktion um eine Matrix zu transponieren (d.h. an der Hauptdiagonale "zu spiegeln")

c) Eine Funktion, die zwei solche Matrizen miteinander multipliziert und eine neue Matrix zurück gibt. Für zwei $n \times n$-Matrizen $A = (a_{ij})$ und $B = (b_{ij})$ ist $A \cdot B = C = (c_{ij})$ durch $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ definiert.

d) Nonplusultra: Implementieren Sie die Potenzfunktion nach Vorbild der ägyptischen Multiplikation für Matrizen. Testen Sie Ihre Implementation mit dem Test aus dem Repository. Dieser berechnet eine beliebige Fibonaccizahl und dank effizienter Matrizenpotenzierung ausgesprochen schnell.

Aufgabe 3. ▲▲
Schreibe eine Funktion, die mithilfe von `qsort` ein Array von Strings lexikographisch (wie im Telefonbuch) sortiert. Das heißt, dass erst nach der ersten Stelle sortiert wird, dann nach der Zweiten usw. In der `<string.h>` liegt eine Vergleichsfunktion für diese Situation vor: `strcmp`. Als Beispiel hier eine lexikographisch sortierte Liste:

```
1  1
2  10
3  133
4  2
5  2344
6  Hallo
7  Thor
8  Tor
```

Aufgabe 4. Miniprojekt - Verwaltungssoftware für **MyBookForYou**
Tag 2 ▲▲ – ▲▲▲
Vervollständigen Sie die Funktionalitäten des Programms **MyBookForYou**. Eine erste Version ist in den Musterlösungen zu finden
Zur Erinnerung:
Ein in Bonn ansässiges Unternehmen **MyBookForYou** bietet Bücher zum

Verkauf an. **MyBookForYou** verwaltet ihre Bücher, Kunden sowie die Rechungen derzeitig über eine großen Excel-Tabelle. Da die Tabelle immer unübersichtlicher wird, hat sich die Inhaberin dazu entschieden, eine neue Verwaltungssoftware entwickeln zulassen. Im Gespräch mit dem Projektleitern (Thorbörn und Sirko) sind folgende **Requirements** für die Software entstanden.

- Bücher abspeichern, sie beinhalten folgende Daten
  - ISBN (char[])
  - Buchname (char[])
  - Author (char[])
- Funktionalität: Bücher anlegen, Bücher löschen, Bücher bearbeiten
- Kunden abspeichern, sie beinhalten folgende Daten
  - Kunden-Nr (unsigned int)
  - Name (char[])
  - Vorname (char[])
- Funktionalität: Kunden anlegen, Kunden löschen, Kunden bearbeiten

Es sollen Platz für jeweils 20 Bücher, 20 Kunden geben, dass heisst ihr benötigt ein ßtruct-Array"von 20 Elementen.
Ihre Augabe ist es :

a) Macht Euch gedanken, wie man das Projekt **modular** organisiern kann. Erstellt Header- und C-Dateien, um eine gute Struktur für das Projekt zu bekommen. (z.B: main.c, buecher.c, buecher.h, ...)▲

b) Erstellt in den jeweiligen Dateien die erforderlichen Datenstrukturen, um die erforderlichen Daten zu speichern (Definition in der Header-Datei) ▲

c) schreibt Funktionen, um Bücher und Kunden anzulegen, zu löschen und zu bearbeiten ▲▲

d) schreibt Ausgabefunktionen um alle Bücher und alle Kunden auszugeben. ▲

e) (optional!) Erstellt ein "Haupt-Menü", um auf die ensprechenden Funktionalitäten zuzugreifen. ▲