In der letzten Woche habt ihr die elementaren Grundlagen der Programmiersprache C kennengelernt. Nun wollen wir mit euch ein kleines Projekt über 2 Tage durcharbeiten. Die folgende Aufgabe unterteilt sich daher in 4 Teilaufgaben. Wir werden jeden Tag eine Musterlösung dazu hochladen, damit ihr an dem darauf folgenden Tag immer auf dem gleichen Stand seid. Am Ende der Woche soll ein lauffähiges Programm erstellt werden, welche die wesentlichen Elemente der Sprache C beinhaltet.
Uns ist es **wichtig**, dass Ihr in **Teams zu 2 - 3 Studenten arbeitet**.

Aufgabe 1. Miniprojekt - Verwaltungssoftware für **MyBookForYou**
Tag 1 ▲ – ▲▲
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. ▲

Aufgabe 2. Umgang mit Strings (Teil 2): ▲ - ▲ ▲
Erstellen Sie eine eigene String-Lib und implementieren Sie die folgenden Funktionen.

a) Schreiben Sie eine `int strlen(char * __str)` Funktion, die die Länge eines Strings zurück gibt.

b) Schreiben Sie eine `int atoi(char * __str)` Funktion, die einen eingegeben Zahl als String in eine Integer umwandelt.

c) Schreiben Sie eine `int strncpy(char * __src, char * __dest, int len)` Funktion, die einen String kopiert.

d) Schreiben Sie eine `int strcmp(char * __st char * __str2)` Funktion, die einen String mit einem anderen String vergleicht.

e) Schreiben Sie eine `char * strcat(char * __str1, char * __str2)` Funktion, die zwei Strings zusammenfügt.

Aufgabe 3. Dynamisches Feld: ▲ - ▲ ▲
In dieser Aufgabe erstellen wir unsere erste Datenstruktur: Das dynamische Feld, auch oft Vektor genannt.
Diese Datenstruktur ist im wesentlichen ein Array, dessen Größe zur Laufzeit unserer Programme veränderbar ist.
Bevor wir uns aber an die eigentliche Implementierung wagen, kümmern wir uns um eine saubere Schnittstelle:

a) Erstellen Sie eine .c and eine .h Datei für unsere Datenstruktur. Deklarieren Sie eine Struktur IntVector sowie eine Funktion..

   - zur Allokation unserer Struktur:
     `IntVector *IntVector_new(size_t capacity)`
   - zur Deallokation:
     `void IntVector_delete(IntVector *vector)`
   - um ein Integer in den Vektor einzufügen:
     `int IntVector_push_back(IntVector *vector, int val)` die 0 bei Erfolg zurückgibt und 1 bei einem Fehler.
   - um einen Wert zu erhalten:
     `int IntVector_get(IntVector *vector, size_t index)`
   - um einen Wert zu setzen:
     `void IntVector_set(IntVector *vector, size_t index, int value)`
   - die die aktuelle Größe des Vektors zurückgibt:
     `size_t IntVector_size(IntVector *vector)`

b) Definieren Sie IntVector in der .h Datei. Die Datenstruktur funktioniert intern wie folgt: Man hat eine Kapazität, die sagt für wie viele Objekte tatsächlich Speicher zur verfügung steht. Dann hat man eine Größe, die besagt wieviele Objekte aktuell im Vektor liegen. Schlussendlich hat man einen Verweis auf den Speicher des Vektors.

c) Definieren Sie die Funktionen in der .c Datei. Sie dürfen dazu die Funktion realloc benutzen (siehe man realloc für Details). Eine Test-Main liegt im Repository für Sie bereit.

d) (Optional) Nennen Sie das struct IntVector um in IntVectorImpl und fügen Sie die Zeile `#typedef IntVectorImpl *IntVector` in die Header Datei ein. Verschieben Sie anschließend die Definition des structs

in die .c Datei und hinterlassen Sie in der Header-Datei lediglich eine Deklaration. Passen Sie abschließend alle Signaturen und Funktionskörper an.
Wir nennen dies das Opaque Pointer Idiom. Wir haben in unserem Header ausschließlich eine Schnittstelle. Alle Implementationsdetails befinden sich in der .c Datei. Wer nur die .h Datei sieht, hat keine Informationen darüber wie IntVector implementiert ist. Kaum eine Programmiersprache bietet derart perfektes Information Hiding wie C.