Hausübung 2XHIF

Einfache Bücherverwaltung:

Mit Hilfe eines C - Programms ist eine einfache Bücherverwaltung zu realisieren.

Die Daten liegen zunächst als CSV-Datei (CSV steht für ”comma separeted values”) vor. Dabei handelt es sich um eine Textdatei, bei der jeder einzelne Datensatz in einer eigenen Zeile steht und die Felder durch Strichpunkte getrennt sind. Die erste Zeile in der Datei gibt die Feldüberschriften an. Jede weitere Zeile beinhaltet die Daten eines Buches, wobei die einzelnen Komponenten durch Strichpunkte getrennt sind. (Man beachte: es können auch ungültige Datensätze vorhanden sein!)
Schreiben Sie ein C-Programm, das eine CSV-Datei einliest und die Daten in ein dynamisches Strukturfeld (Initialgröße 10) umspeichert. Sollte die aktuelle Feldgröße mit jeweils 80% von 10 (später 20, 30, …) beschrieben sein, so ist das Feld um weitere 10 Strukturen dynamisch zu vergrößern.
Der Name der zu öffnenden CSV-Datei ist über die Kommandozeile zu übernehmen.

Jeder Datensatz entspricht folgender Struktur:

```
struct buch{
  char buchNummer[11];    // Feld von 10 char und binäre 0
  char *titel;            // dynamisches Textfeld geeigneter Größe
  int bestand;            // Lagerbestand - Stückzahl
  double preis;           // Preis (in Euro)
}
```

Eine gültige Buch-Nummer besteht aus zuerst 3 Großbuchstaben (engl. Alphabet) gefolgt von 7 Zifferzeichen. Sollte eine Buch-Nummer, welche aus einem Datensatz der CSV – Datei in eine Struktur und somit in das Strukturfeld übertragen werden soll, nicht korrekt sein, so wird dieser Datensatz einfach ausgelassen. Sollte die Buch-Nummer bereits einmal vorgekommen sein (**eindeutige** Werte!) und der Titel passt, wird die Anzahl zum Bestand hinzugefügt, und der Preis überschrieben. Passt der Titel nicht zur Buch-Nummer wird der Datensatz verworfen. Ebenso wird jeder Datensatz, welcher als korrupt zu werten ist (negative Anzahl oder Preis, zu wenige, zu viele oder nicht konvertierbare Werte) verworfen.

Das Programm sollte mit der zur Verfügung gestellten Textausgangsdatei (BuchAusgang.csv) getestet werden.
Als Zugangsdatei kann BuchZugang.csv verwendet werden.

Das Programm soll folgende Funktionalitäten realisieren:

- Ordentliche Menüführung
  Der Benutzer wird mittels eines sinnvollen Menüs durch die Möglichkeiten des Programms geleitet.
  Das Programm wird erst beendet, wenn der Benutzer den Menüpunkt "EXIT" wählt.

- Klare Hilfestellungen
  Sollten diverse Fehler auftreten (Dateneingabe, falsche Werte, etc.) so ist mit einer entsprechenden "sinnvollen" Fehlermeldung darauf zu reagieren.

- Einzelne Buchzugänge erfolgen über die Konsole mittels einer geeigneten Funktion. Bei richtiger Eingabe der entsprechenden Werte ist der entsprechende Datensatz im Bücherfeld anzupassen bzw. bei noch nicht vorhandener (gültiger) Buch-Nummer ist der neue Datensatz im Feld anzuhängen.

- Entnahmen von Büchern erfolgen ebenso über die Konsole mittels einer geeigneten Funktion. Der Bestand der entnommenen Bücher ist dabei entsprechend anzupassen.

- Hinzufügen einer Bücherliste durch Lesen einer neuen Bücherliste aus einer Datei.

- Löschen einer Buchnummer und somit Entfernen eines Buches aus dem Inventar.

- Suchen eines bestimmten Buches, soll wahlweise über Angabe des Titels oder der Buch-Nummer möglich sein.

- Händisches Ändern des Buchtitels nach Eingabe der Buch-Nummer

- Bildschirmausgabe des gesamten Buchbestandes.
  Alle Bücher sind in Listenform (Tabelle) auf dem Bildschirm auszugeben.

- Bildschirmausgabe des sortierten Buchbestandes.
  Alle Bücher sind in Listenform (Tabelle) sortiert auf dem Bildschirm auszugeben.
  Sortierung kann nach Buchnummer, Titel (ohne Berücksichtigung der Klein- Groß-schreibung oder nach Preis erfolgen.

- Bildschirmausgabe von Bestelllisten.
  Alle nachzubestellenden Bücher (Bestand unter 5) sind in Listenform auf dem Bildschirm auszugeben.

- Ausgabe des aktuellen Wertes des Gesamtlagers.

- Filtern
  Ausgabe von Büchern, welche im Titel einen Suchstring beinhalten
  oder in der Buchnummer eine bestimmte Zeichenkette besitzen oder
  mehr (weniger) als einen bestimmten Preis kosten

- Freigeben aller angeforderten dynamischen Speicher.

- Speichern des aktuellen Feldes in einer neuen csv-Datei.
  Die Eingabe des Dateinamens erfolgt über die Konsole.
  Jede einzelne Buchstruktur wird in einen passenden csv-String verwandelt und in eine eigene Zeile der Ausgangsdatei gespeichert.

Dazu sind mindestens folgende Funktionen zu implementieren:

```
buch_t *fuegeBuecherFeldHinzu(char *dateipfad, buch_t *feld,
                              int *n, int *maximal);
```
Öffnet die Datei, liest und konvertiert die Daten und liefert das übergeben und eventuell dynamisch veränderte und beschriebene Bücherfeld zurück. Die Anzahl der aktuellen Bücher sowie die Größe des momentanen Feldes werden in den übergebenen Pointern rückgeschrieben. Sollte ein Datei-Öffnugns- oder Schließfehler auftreten, so wird mit einer entsprechenden Fehlermeldung reagiert.

```
int erstelleNeuesBuch(char *textzeile, buch_t *buecher, int anzahlAkt);
```
Konvertiert die Daten aus der übergebenen Textzeile. Überprüft dabei ob ein Buch erstellt werden kann (oder nicht) und ob es eventuell bereits vorhanden ist. Entsprechend wird ein neues Buch am Ende des Feldes hinzugefügt oder der entsprechende Datensatz korrigiert. Die Funktion liefert zurück, ob sich die Länge des Feldes verändert hat (Buch hinzugefügt) oder ob sie gleichgeblieben ist (kein Buch oder Daten eines vorhandenen wurden nur verändert).

```
int gueltigeBuchNummer(char *text);
```
Überprüft ob es sich um eine gültige Buchnummer handelt.

```
buch_t *sucheBuch(buch_t *buecher, int n, char *suchText,int titelOderNummer);
```
Durchsucht das übergebene Feld, ob der `suchText` in einer Struktur vorkommt. Die Option `titelOderNummer` gibt dabei an, ob der `suchText` als Titel oder Buchnummer auftreten kann. Rückgegeben wird die Adresse jenes Buches, für welches der `suchText` gefunden wurde – ansonst `NULL`.

```
int buchZugang(buch_t *buecher, int anzahlAkt);
```
List über die Konsole die Daten eines "neuen" Buches ein und speichert diese in das übergebene Bücherfeld. Liefert zurück ob das Hinzufügen möglich war oder nicht bzw. sich die Anzahl der Datensätze verändert hat.

```
void buchEntnahme(buch_t *buecher, int anzahl);
```
List über die Konsole die Entnahme eines bestimmten Buches ein und verändert den vorhandenen Bestand. (Bestand kann nicht unter 0 sinken!)

```
int buchEntfernen(buch_t *buecher, int anzahl, char *buchNummer);
```
Überprüft ob die übergebene Buchnummer im Feld vorhanden ist und löscht dieses Buch aus dem Feld (andere Elemente nachrücken!). Liefert zurück ob ein Buch gelöscht wurde oder nicht.

```
void aendereTitel(buch_t *buchAdresse);
```
Ändert den Titel eines bestimmten Buches auf einen neuen.

```
void listeBuchbestand(buch_t *buecher, int anzahl);
```
Gibt den aktuellen Bücherstand in Form einer tabellenartigen Liste aus.

```
void listeBuchbestandGeordnet(buch_t *buecher, int anzahl, int ordnungNach);
```
Erzeugt eine Kopie des Datenfeldes und ordnet dieses entsprechend der übergebenen Option `ordnungNach` nach Titel, Buchnummer oder Preis. Gibt den so sortierten aktuellen Bücherstand in Form einer tabellenartigen Liste aus.

```
void bestellListe(buch_t *buecher, int anzahl);
```
Gibt alle Bücher in Listenform aus, welche einen Bestand kleiner 5 besitzen.

```
double lagerWert(buch_t *buecher, int anzahl);
```
Bestimmt den Warenwert aller Bücher und liefert diesen zurück.

```
void titelFilter(buch_t *buecher, int anzahl, char *suchText);
```
Gibt alle Bücher aus, in denen der übergebene Suchtext im Titel vorkommt.

```
void nummerFilter(buch_t *buecher, int anzahl, char *suchText);
```
Gibt alle Bücher aus, in denen der übergebene Suchtext in der Buchnummer vorkommt.

```
void preisFilter(buch_t *buecher, int anzahl, double vergleichsPreis,
                 int mehrOderWeniger);
```
Gibt alle Bücher aus, in denen der Preis kleiner oder größer (Option `mehrOderWeniger`) als der übergebene Vergleichspreis ist.

```
int speichereBuecher (char *dateipfad, buch_t *buecher, int anzahl);
```
Speichert alle Bücher des aktuellen Feldes in geeigneter Form als "CSV-Datei" in eine entsprechende Datei. Sollte die Datei bereits existieren, so wird der Benutzer gefragt, ob die Datei überschrieben wird, die Daten an die bestehende angehängt werden oder ob eine neue Datei angelegt werden soll.
Der Rückgabewert gibt an ob die Daten hinausgeschrieben wurden oder nicht.

```
void toCSVText(const buch_t* buch, char *csvText);
```
Verwandelt das mittels Pointer übergebene Buch in eine passende CSV-Zeile, welche auf den übergebenen char-Pointer geschrieben wird..

```
int menue();
```
Stellt ein geeignetes Auswahlmenü dar und liefert den gewählten Auswahlpunkt zurück.

```
void freeData(buch_t *buecher, int anzahl);
```
Gibt alle dynamisch angeforderten Daten frei.

Die Ermittlung der einzelnen Textzeilen aus der CSV – Datei kann mittels nachgestellter Codebeschreibung erfolgen:

```
int main(int argc, char *argv[])
{
 FILE *fp;
 char ein[2000];
 int buchAnzahl = 0, aktuelleFeldgroesse = 10;
 buch_t *buecher;
 // weitere Variablendeklaration

 // auf korrekte Anzahl von Kommandozeilenparameter pruefen
 // Erzeugung eines dynamischen Initialfeldes der Größe 10

 // mittels Funktion fuegeBuecherFeldHinzu
        // Öffnen der Datei zum Lesen und Erzeugung des FILE-Pointers
        // bei Öffnungsfehler → Fehlermeldung und Programm beenden
        // eventuelles Vor-Lesen der Überschritenzeile (überspringen)
        // Schleife → man liest Zeile für Zeile aus der Datei
                // Wegschneiden des Zeilensprungs wenn dieses das letzte Zeichen der Zeile
                // mittels Funktion erstelleNeuesBuch
                        // Buch in Feld hinzufügen
                // Überprüfung ob Bücheranzahl schon 80% der Feldgroesse erreicht
                // wenn dies der Fall, so wird das Feld dynamisch um 10 Einheiten erhoeht
        // Schließen der Datei
        // dynamisches Bücherfeld rückliefern

 // Feld von Büchern wurde erzeugt und mit diesem wird nun gearbeitet
 return 0;
}
```