Datenbank

Allgemeine Anmerkungen zur Aufgabenstellung:

- Beantworten Sie die Kontrollfragen in `TUWEL` bevor Sie mit der Übung beginnen.
- Legen Sie für `Teil A` der Angabe ein neues `Code::Blocks`-Projekt an und erweitern Sie dieses schrittweise für die darauffolgenden Punkte.
- Stellen Sie während der Übung laufend sicher, dass Ihr Programm kompilierbar ist und richtig funktioniert.
- Die Abgabe erfolgt durch Hochladen Ihres vollständigen `Code::Blocks`-Projektordners als zip-Datei (`Matrikel-Nr_UE5.zip`) in `TUWEL`. Anschließend melden Sie sich für das Angabegespräch in `TUWEL` an. Dieses erfolgt über die Telekonferenz-App `Zoom` (www.zoom.us). Stellen Sie sicher, dass diese App rechtzeitig vor Beginn des Abgabegespräches auf Ihrem PC installiert ist und dass Sie Ihr aktuelles Übungsprogramm in `Code::Blocks` für die Abgabe geöffnet haben. Testen Sie bitte auch die korrekte Funktionalität Ihrer Videokamera, da Sie zu Beginn Ihres Abgabegespräches Ihren Studentenausweis herzeigen müssen und auch während des Gesprächs eine aktive Videoverbindung erforderlich ist.
- Beachten Sie, dass Upload und Terminauswahl für das Abgabegespräch erst möglich sind, wenn Sie 80% der Kontrollfragen richtig beantwortet haben. Diese müssen, genauso wie die Übung, bis spätestens **07.06.2020 23:59** durchgeführt werden.
- Inhaltliche Fragen stellen Sie bitte im zugehörigen Forum in `TUWEL`. Das Betreuungsteam steht Ihnen in den jeweiligen Übungswochen von Mo.-Fr. in der Zeit von 09:00-17:00 zur Verfügung. Außerhalb dieser Zeiten sind wir auch bemüht Ihre Anfragen möglichst zeitnah zu beantworten.
- Für eine positive Beurteilung Ihrer Abgabe **muss** diese kompilierbar sein, Ihr Programm fehlerfrei terminieren und die in der Angabe angeführten Funktionen mit den vorgegebenen Funktionsköpfen enthalten. Des Weiteren müssen die Funktionen hinreichend getestet sein. Sie müssen in der Lage sein, Ihr Programm beim Abgabegespräch zu erklären!

Folgende Hilfsmittel stehen Ihnen in `TUWEL` zur Verfügung:

- Alle bisherigen Referenzbeispiele sowie Vorlesungsfolien
- Interaktive Jupyter-Notebooks auf `prog1.iue.tuwien.ac.at`
- Das Buch zur LVA "Programmieren in C" (Robert Klima, Siegfried Selberherr)
- Kurzanleitung für die Entwicklungsumgebung `Code::Blocks`

Hinweis: Schreiben Sie Ihr Programm selbstständig! Das Kopieren der Lösung stellt einen groben Verstoß gegen die Durchführungsbestimmungen der LVA (siehe LVA–Beschreibung im TISS) dar, welcher zu einer negativen Beurteilung der Übung führt.

Einleitung
In dieser Übung werden Sie eine einfache Datenbank programmieren. Ihre Datenbank soll dabei das Erstellen, Löschen, Suchen und Sortieren von Einträgen ermöglichen. Weiters kann die Datenbank in eine Datei gespeichert oder von dort geladen werden. Dabei erlernen Sie den Umgang mit Verbunddatentypen sowie das Lesen und Schreiben von Dateien in der Programmiersprache C.

Anmerkungen zum Programmaufbau
Das Programm soll aus der Datei `main.c` sowie dem Modul `dbfunc`, welches die Dateien `dbfunc.c` und die zugehörigen Header-Datei `dbfunc.h`enthält, bestehen. Beachten Sie hierbei, dass die Datei `main.c` *nur* die Funktion `main` enthalten darf. Alle anderen Funktionen sollen in dem Modul `dbfunc` implementiert werden.

Geben Sie alle Fehlermeldungen Ihres Programms auf stderr aus. **Hinweis**: Verwenden Sie dazu die Funktion `fprintf`.

Stoffgebiete
Zusätzlich zum Stoff aller vorangegangenen Übungen benötigen Sie für diese Übung folgende neue Themengebiete aus den Vorlesungen 9-10 (Kapitel 16-17):

- Strukturen (Kapitel 16.2)
- Typdefinitionen (Kapitel 16.7)
- Datei-Manipulationen (Kapitel 17)
- Öffnen und Schließen von Dateien (Kapitel 17.2)
- Ein- und Ausgabe mit Text- und Binärdateien (Kapitel 17.3)

Teil A

✘ Erstellen Sie zuerst im Modul `dbfunc` die Struktur `Data_s` welche einen Datenbankeintrag repräsentiert und wie folgt definiert ist:

```
struct Data_s {
        char name[TEXT_LEN];
        char brand[TEXT_LEN];
        char invNr[TEXT_LEN];
        long year;
};
typedef struct Data_s Data_t;
```

Diese Struktur speichert den Namen, die Marke, die Inventar-Nr. sowie das Anschaffungsjahr eines Objekts in Ihrem Lagerbestand ab. Definieren Sie zusätzlich eine maximale Textlänge `TEXT_LEN` in geeigneter Form im Modul `dbfunc`.

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
void initItems(Data_t *items, long len);
```

Parameter:

- `items`: Zu initialisierendes Feld von Elementen des Typs `Data_t`
- `len`: Anzahl der Elemente des Feldes `items`

Beschreibung:

Die Funktion `initItems` initialisiert alle Datenbankeinträge als leere Einträge indem der Inventar-Nr. (`invNr`) eine leere Zeichenkette zugewiesen wird.

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
void printItem(Data_t *item);
```

Parameter:

- `item`: Zeiger auf den darzustellenden Datenbankeintrag

Beschreibung:

Die Funktion `printItem` soll den Datenbankeintrag `item` ausgeben. Die Formatierung soll dabei an folgendem Beispiel angelehnt sein:

```
Name:                   Deskjet_4200K
Marke:                  Hewlett_Packard
Inventar-Nr.:           42AD9345ER
Anschaffungsjahr:       2012
```

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
void printDB(Data_t *items, long len);
```

Parameter:

- `items`: Feld von Elementen des Typs `Data_t` welches ausgegeben werden soll
- `len`: Anzahl der Elemente des Feldes `items`

Beschreibung:

Die Funktion `printDB` soll alle *gültigen* Datenbankeinträge des angegeben Feldes `items` ausgeben. Verwenden Sie dazu die zuvor implementierte Funktion `printItem`. Die Ausgabe soll dabei folgende Form haben:

```
Datenbankeintraege:
---------------Eintrag  1---------------
Name:                   Deskjet_4200K
Marke:                  Hewlett_Packard
Inventar-Nr.:           42AD9345ER
Anschaffungsjahr:       2012
---------------Eintrag  2---------------
Name:                   Laserjet_5200D
...
```

✘ Erstellen Sie im Hauptprogramm ein Feld von Datenbankeinträgen, welches zehn Elemente vom Typ `Data_t` speichern kann. Initialisieren Sie dieses Feld mit Hilfe Ihrer Funktion `initItems` und weisen Sie anschließend den ersten beiden Elementen des Feldes geeignete Testdaten zu. Geben Sie danach die Datenbankeinträge aus.

Hinweis: Für eine einfachere Implementierung der Funktionen in den kommenden Teilen empfehlen wir in Zeichenketten für Namen oder Marken *keine* Leerzeichen zu verwenden.

Teil B

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long readValue(char *name);
```

Parameter:

- `name`: Name des Parameters welcher eingelesen wird

Rückgabewert:

Erfolgreich eingelesene Zahl

Beschreibung:

Die Funktion `readValue` soll einen Ganzzahlenwert von der Tastatur einlesen. Dazu soll der Parameterbezeichner `name` ausgegeben werden und danach der Wert eingelesen werden. Bspw. würde der Aufruf `readValue("Jahr")` mit der anschließenden Eingabe 2020 zu folgender Ausgabe führen:

```
Jahr: 2020
```

Im Fall einer ungültigen Eingabe soll eine Fehlermeldung auf `stderr` ausgegeben werden und die Eingabe erneut abgefragt werden.

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
void readString(char *name, char *text, long len);
```

Parameter:

- `name`: Name des Parameters welcher eingelesen wird
- `text`: Feld, in dem die eingegebene Zeichenkette gespeichert werden soll
- `len`: Maximale Länge der Zeichenkette `text`

Beschreibung:

Die Funktion `readString` soll eine Zeichenkette von der Tastatur einlesen. Analog zu `readValue` soll der Parameterbezeichner `name` ausgegeben werden und danach ein Text eingelesen und in `text` gespeichert werden.

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long addItem(Data_t *items, long len);
```

Parameter:

- `items`: Feld von Elementen des Typs `Data_t` an dessen ersten freien Platz ein neuer Eintrag hinzugefügt werden soll.
- `long len`: Anzahl der Elemente des Feldes `items`

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn ein Fehler aufgetreten ist

Beschreibung:

Die Funktion `addItem` soll einen neuen Datenbankeintrag von der Tastatur einlesen und an den ersten freien Platz im Feld `items` speichern. Verwenden Sie zum Einlesen die zuvor implementierten Funktionen `readValue` und `readString`. Nehmen Sie dabei an, dass die eingegebenen Zeichenketten keine Leerzeichen enthalten. Geben Sie eine Fehlermeldung aus, sollten keine freien Einträge mehr vorhanden sein oder sollte der hinzuzufügende Eintrag ungültig sein (leere Zeichenkette für die Inventar-Nr. `invNr`).

Ein Aufruf könnte beispielhaft zu folgender Abfrage führen:

```
----------Neuen Eintrag eingeben----------
Name:                   Deskjet_4200K
Marke:                  Hewlett_Packard
Inventar-Nr.:           82AD9345ER
Anschaffungsjahr:       2020
```

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long deleteItem(Data_t items, long len);
```

Parameter:

- `items`: Feld von Elementen des Typs `Data_t` welches die Datenbankeinträge enthält
- `len`: Anzahl der Elemente des Feldes `items`

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn ein Fehler aufgetreten ist und der Eintrag nicht gelöscht werden konnte

Beschreibung:

Die Funktion `deleteItem` soll zunächst den Index des zu löschenden Elements von der Tastatur einlesen. Verwenden Sie dafür die zuvor implementierte Funktion `readValue`. Das entsprechende Element der Datenbank `items` soll ungültig gemacht werden, indem der Inventar-Nr. eine leere Zeichenkette zugewiesen wird. Geben Sie eine Fehlermeldung aus, falls der gewählte Index schon ungültig ist oder wenn der

gewählte Index außerhalb der Feldgröße liegt. Wurde nicht das letzte Element gelöscht, soll die entstandene Lücke durch Vorschieben der darauffolgenden Elemente geschlossen werden.

✘ Implementieren Sie in Ihrem Hauptprogramm ein Menü folgender Form um Ihre Funktionen wiederholt aufrufen zu können:

```
1 - Alle Datensätze ausgeben
2 - Eintrag hinzufügen
3 - Eintrag löschen
0 - Programm beenden
```

Teil C

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long saveDB(Data_t *items, long len, char *outputFile);
```

Parameter:

- `items`: Feld von Elementen des Typs `Data_t` welches die Datenbankeinträge enthält
- `len`: Anzahl der Elemente des Feldes `items`
- `outputFile`: Pfad der Datei in der die Datenbank gespeichert werden soll (ohne Leerzeichen)

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn ein Fehler aufgetreten ist und die Daten nicht gespeichert werden konnten

Beschreibung:

Die Funktion `saveDB` soll die gültigen Datenbankeinträge des Feldes `items` in die Textdatei `outputFile` speichern. Die Formatierung soll dabei an folgendes Beispiel angelehnt sein:

```
#Name           #Marke            #Inventar-Nr.     #Jahr
Deskjet_4200K   Hewlett_Packard   42AD9345ER        2012
Printmaster     Canon             3960FGA798        2010
...
```

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long loadDB(Data_t *items, long len, char *inputFile);
```

Parameter:

- `items`: Feld von Elementen des Typs `Data_t` welches die Datenbankeinträge enthält
- `len`: Anzahl der Elemente des Feldes `items`
- `inputFile`: Pfad der Datei von der die Datenbank geladen werden soll (ohne Leerzeichen)

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn ein Fehler aufgetreten ist und die Daten nicht geladen werden konnten

Beschreibung:

Die Funktion `loadDB` soll die Datenbankeinträge von der Textdatei `inputFile` laden und in das Feld `items` speichern. Die Funktion erwartet, dass `inputFile` so formatiert ist, wie in der Funktion `saveDB` festgelegt. Anderenfalls wird eine Fehlermeldung ausgegeben und das Feld `items` mittels der Funktion `initList` neu initialisiert.

✘ Erweitern Sie Ihr Menü um die folgenden Funktionen:

```
4 - Datei speichern
5 - Datei laden
```

Teil D

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long searchFile(char *inputFile, char *key);
```

Parameter:

- `inputFile`: Pfad der Datei die durchsucht werden soll (ohne Leerzeichen)
- `key`: Schlüssel nach dem gesucht werden soll

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn ein Fehler aufgetreten ist und die Daten nicht gelesen werden konnte

Beschreibung:

Die Funktion `searchFile` soll die Datei `inputFile` durchsuchen und *alle* Zeilen ausgeben, die den Schlüssel `key` enthalten. Wenn aus der Datei `inputFile` nicht gelesen werden kann oder der Schlüssel `key` nicht gefunden wurde, soll eine entsprechende Fehlermeldung ausgegeben werden. Bei einer gültigen Suche soll die Ausgabe an folgendes Beispiel (`key = "Canon"`, `inputFile = "test.txt"`) angelehnt sein:

```
test.txt Zeile: 2, Printmaster Canon 3960FGA798 2010
```

✘ Erweitern Sie Ihr Menü um die folgende Funktion:

```
6 - In Datei suchen
```

Teil E

✘ Passen Sie Ihr Hauptprogramm so an, dass die bei Programmstart an die Funktion `main` übergebenen Parameter `argc` und `argv` genutzt werden, um eine Suche durchzuführen. Dabei soll auf die im vorigen Teil implementierte Funktion `searchFile` zurückgegriffen werden. Als erstes Argument soll Ihr Programm den Schlüssel `key` erwarten, gefolgt von einer oderer mehrerer Dateien in der nach diesem Schlüssel gesucht werden soll. Wird gar keiner oder nur ein Parameter an die Funktion `main` übergeben, soll das Programm mit dem zuvor implementierten Menü ausgeführt werden.

Führen Sie zum Beispiel unter Linux Ihr Programm `Uebung05` mit folgendem Befehl aus

```
./Uebung05 Canon test.txt test2.txt test3.txt
```

so soll mit Hilfe der Funktion `searchFile` nach dem Schlüssel `key = "Canon"` in den Dateien `test.txt`, `test2.txt` und `test3.txt` gesucht werden.

Für Windows-Systeme verwenden Sie analog dazu den Befehl

```
Uebung05.exe Canon test.txt test2.txt test3.txt
```

Teil F

✘ Erweitern Sie die Funktion `searchFile` um einen Parameter:

```
long searchFile(char *inputFile, char *key, long ignoreCase);
```

Parameter:

- `inputFile`: Pfad der Datei die durchsucht werden soll
- `key`: Schlüssel nach dem gesucht werden soll
- `ignoreCase`: Groß/Kleinschreibung beachten wenn `ignoreCase == 0`, in allen anderen Fällen Groß/Kleinschreibung ignorieren

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn ein Fehler aufgetreten ist und die Datei nicht gelesen werden konnte

Beschreibung:

Die Funktion `searchFile` soll die Datei `inputFile` durchsuchen und die Zeilen, die den Schlüssel `key` enthalten, ausgeben. Wenn aus der Datei `inputFile` nicht gelesen werden kann, oder der Schlüssel `key` nicht gefunden wurde, soll eine entsprechende Fehlermeldung ausgegeben werden. Wenn `ignoreCase == 0` ist, soll die Suche Groß/Kleinschreibung beachten. In allen anderen Fällen soll die Groß/Kleinschreibung ignoriert werden.

✘ Steuern Sie die erweiterte Funktion `searchFile` bei Programmaufruf mittels Parameterübergabe an die Funktion `main`. Verwenden Sie dafür das zusätzliche optionale Argument `-i`, das bei Programmaufruf noch vor dem Schlüssel `key` angegeben werden kann. Wird dieses Argument gesetzt, soll bei der Suche die Groß/Kleinschreibung ignoriert werden.

Bei einem Aufruf des Programmes `Uebung05` mit

```
./Uebung05 -i canon test.txt
```

könnte die Ausgabe zum Beispiel so aussehen:

```
test.txt Zeile: 2, Printmaster Canon 3960FGA798 2010
test.txt Zeile: 3, Printmaster canon 3960FGA798 2010
```

Teil G

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long sortItems(Data_t *items, long len, long which);
```

Parameter:

- `items`: Feld von Elementen des Typs `Data_t` welches die Datenbankeinträge enthält

- `len`: Anzahl der Elemente des Feldes `items`
- `which`: Attribut nach dem sortiert werden soll: 1 für die Inventar-Nr. und 2 für das Anschaffungsjahr

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn `which` nicht 1 oder 2 ist oder ein anderer Fehler auftritt

Beschreibung:

Die Funktion `sortItems` soll alle Einträge der Datenbank `items` in aufsteigender Reihenfolge sortieren. Ungültige Einträge sollen dabei ignoriert werden und am Ende des Feldes stehen bleiben. Das Attribut nach dem sortiert werden soll wird dabei mit dem Argument `which` festgelegt. Beachten Sie, dass in dieser Funktion keine Datenbankeinträge ausgegeben werden, sondern die Einträge des Feldes `items` lediglich sortiert und eventuelle Fehlermeldungen ausgegeben werden.

✘ Erweitern Sie Ihr Menü um die folgende Funktion:

```
7 - Datensätze sortieren
```

Teil H

✘ Implementieren Sie im Modul `dbfunc` die Funktion

```
long addFromFile(Data_t *items, long len, char *inputFile, char *key);
```

Parameter:

- `items`: Feld von Elementen des Typs `Data_t` an dessen ersten freien Platz ein neuer Eintrag hinzugefügt werden soll.
- `len`: Anzahl der Elemente des Feldes `items`
- `inputFile`: Pfad der Datei, die durchsucht werden soll
- `key`: Schlüssel nach dem gesucht werden soll

Rückgabewert:

0 ... bei Erfolg

-1 ... wenn ein Fehler aufgetreten ist und die Datei nicht gelesen werden konnte

Beschreibung:

Die Funktion `addFromFile` soll die Datei `inputFile` nach dem Schlüssel `key` durchsuchen und die gefundenen Einträge in das Feld `items` speichern. Dabei ist darauf zu achten, dass in dem Feld `items` Platz für die neuen Einträge ist und dass es zu keinen Duplikaten kommt (keine Einträge mit gleicher Inventar-Nr.).

✘ Erweitern Sie Ihr Menü um die folgende Funktion:

```
8 - Eintraege aus Datei hinzufuegen
```