Aufgabe 1: Kommandozeilenargumente

a) Schreiben Sie ein Programm `calc`, mit dem die vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division durchgeführt werden können. Das Programm soll mit der Syntax
`./calc op1 operation op2`

aufgerufen werden können. Die Operanden op1 und op2 können Ganzzahlen oder Gleitpunktzahlen sein. Für die Operation sollen die Zeichen +, -, * oder / verwendet werden. Bei einer fehlerhaften bzw. unvollständigen Eingabe soll eine Fehlermeldung erscheinen. Das Ergebnis der Berechnung soll zusammen mit der Eingabe ausgegeben werden, z.B.:

```
$./calc 1 + 3
1 + 3 = 4
```

Für jede Operation soll eine eigene Funktion geschrieben werden, die die Berechnung und Ausgabe durchführt. Diese 4 Funktionen sollen im Modul `operation.c` implementiert werden.

Aufgabe 2: Vektor zwischen zwei Punkten

Schreiben Sie ein Programm, in dem zwei dreidimensionale Punkte im Raum definiert sind. Das Programm soll eine Funktion enthalten, die den Vektor zwischen den Punkten bestimmt und den Betrag des Vektors ausrechnet. Der Betrag des Vektors soll als Rückgabewert der Funktion zurückgegeben werden. Nach dem Funktionsaufruf sollen die beiden Punkte und der Vektor, sowie der Betrag des Vektors auf den Bildschirm ausgegeben werden.

Vektor $\overrightarrow{P_1P_2}$ und Abstand d zwischen Punkte $P_1(x_1, y_1, z_1)$und $P_2(x_2, y_2, z_2)$

$$\overrightarrow{P_1P_2} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$
$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Aufgabe 3: Präprozessor-Direktiven

a) Welches Ergebnis liefert die Ausführung des folgenden Codes und wie erklären Sie sich das Ergebnis?

```
#define SUM(a,b) a+b
printf("%d", SUM(3,4) * SUM(2,1));
```

b) Definieren Sie ein Makro, das die eulersche Zahl e auf 4 Stellen gerundet darstellt.

c) Definieren Sie ein Makro `FUNKTION(X)`, das $e^x + 5x + 3$ berechnet. Zur Berechnung von e^x können Sie `pow (math.h)` verwenden.

d) Definieren Sie ein Makro

```
#define ausgabe(variable)
```

das für eine Integer-Variable deren Namen und Wert ausgibt.

Aufrufbeispiele:

```
int y = 15; ausgabe(y); => Ausgabe: y = 15
int z = -7; ausgabe(z); => Ausgabe: z = -7
```

Aufgabe 4: Personendatenbank

Erweitern Sie die Personendatenbank vom letzten Übungszettel um die Möglichkeit, Personendaten als CSV-Datei (Comma Separated Value) auf der Festplatte zu speichern bzw. diese wieder einzulesen. CSV-Dateien enthalten pro Zeile jeweils einen Datensatz, dessen Werte (Attribute) durch Kommata getrennt sind. Ignorieren Sie den Fall, dass Kommata in Ihren Nutzdaten auftauchen und verzichten Sie dementsprechend auf Quoting. Beachten Sie auch die Reihenfolge der Werte innerhalb einer CSV-Zeile: Anrede, Vorname, Nachname, Strasse, Postleitzahl, Ort.

Beispiel CSV-Datei mit zwei Datensätzen:

Herr,Heinz,Becker,Cauchy-Folge 42,12345,Hilbertraum
Frau,Hilde,Becker,Cauchy-Folge 42,12345,Hilbertraum

Können Sie die aus Ihrer Anwendung exportierte Datei fehlerfrei in eine Tabellenkalkulation importieren (LibreOffice, MS-Office, etc.)?
Können Sie die CSV-Datei aus LEA in Ihre Anwendung importieren? Soweit möglich tauschen Sie die CSV-Dateien mit Kommilitonen und prüfen Sie, ob sich die Dateien in Ihre Datenbank einlesen lassen.

Benutzen Sie zur Implementierung der Dateioperationen die Streaming-Funktionen der C-Bibliothek.

Hinweis: Zum Einlesen der CSV-Datei müssen die Zeilen in ihre Einzelteile zerlegt werden. Dafür gibt es verschiedene Ansätze (fast alle sind in C mühsam ;)): Suchen nach dem Trennzeichen und zerschneiden mit Zeigeroperationen. Oder die Funktion strtok(3). Diese arbeitet ähnlich wie der aus Java bekannte StringTokenizer, nur mit wesentlich mehr Handarbeit. Wir empfehlen, mit einer Variante von scanf(3) zu starten und den folgenden Format-String zu verwenden:

```
int ret = sscanf(buf, "%[^,\n]%*c%[^,\n]%*c%[^,\n]%*c%[^,\n]%*c%d,%[^,\n]%*c",
                      &a[0], &b[0], &c[0], &d[0], &x, &e[0]);
```

Das liefert die 6 Komponenten einer CSV-Zeile (5 Strings und eine Zahl).

Hinweis: `fopen(3), fclose(3), fflush(3), fread(3), fwrite(3), fscanf(3), sscanf(3), scanf(3)`

Empfohlene Aufgabe:

Aufgabe 5: Stack Speicher

Erweitern Sie Ihr Programm zu Aufgabenblatt 6, Aufgabe 2„Verkettete Listen“ und entwickeln Sie einen Stack Speicher.

a) Implementieren Sie die Funktionen `create(), discard(), push(), pop()` und `is_empty()` aus der Header-Datei `stack.h` (s. Download lea). Dort finden Sie auch vorgegebene Strukturen zum Aufbau der Datenstruktur und zum Speichern der Daten auf dem Stack.
b) Überlegen Sie sich auch einige Testfälle und implementieren Sie diese in einem Testprogramm.