Aufgabe 1.

a) Implementiere die Addition, Multiplikation, Potenzen und Division komplexer Zahlen. Verwende dazu folgende Header-Datei:

```c
#ifndef _COMPLEX__H
#define _COMPLEX__H

typedef struct _COMPLEX {
        double real;
        double imag;
} COMPLEX;

COMPLEX cplx_add(COMPLEX a, COMPLEX b);
COMPLEX cplx_mul(COMPLEX a, COMPLEX b);
COMPLEX cplx_pow(COMPLEX a, unsigned long n);
COMPLEX cplx_div(COMPLEX a, COMPLEX b);

#endif
```

b) * Implementiere die Addition, Multiplikation und Division sowie das Kürzen rationaler Zahlen. Schreibe dazu erst die Header-Datei.

Aufgabe 2. Bearbeite nun erst einmal die Aufgaben der vorherigen Zettel, falls du sie nicht schon alle gelöst hast.

Aufgabe 3. Lese noch einmal im Skript die Sektion 7.5 und implementiere doppelt verkettete Listen, die `double`-Variablen speichern.

```c
/* Definiere hier angemessene Strukturen fuer einen
   einzelnen Listeneintrag und die Liste selbst. */

/* Leere Liste erstellen */
LIST *list_create();

/* Element hinter E einfuegen, NULL heisst am Anfang */
LISTNODE *list_insert(LIST *L, LISTNODE *E, double p);

/* Element am Anfang bzw. Ende einfuegen */
```

```
11 LISTNODE *list_unshift(LIST *L, double p);
12 LISTNODE *list_push(LIST *L, double p);
13
14 /* Element am Anfang bzw. Ende entfernen und
15    die Daten zurueck geben */
16 double list_shift(LIST *L);
17 double list_pop(LIST *L);
18
19 /* ein Element aus der Liste entfernen */
20 void list_delete(LIST *L, LISTNODE *E);
21
22 /* zwei Listen zusammenfuegen */
23 LIST *list_merge(LIST *L, LIST *M);
24
25 /* Liste inklusive allen Elementen frei geben */
26 void list_free(LIST *L);
```

Aufgabe 4. In dieser Aufgabe sollst du eine Datenbank von Studenten programmieren. Dazu verwenden wir doppelt verkettete Listen.

a) Schreibe ein struct `student`, welches Einträge für Name, Vorname, Matrikelnummer, Notendurchschnitt, Fachsemester und ein Array der Länge 20 mit den Noten aus all seinen Prüfungen enthält. Wenn du magst, kannst du dir natürlich weitere Einträge überlegen.

b) Schreibe Funktionen um einige oder alle diese Werte für einen gegebenen `studenten` zu setzen und schreibe eine Funktion, um die Werte eines `studenten` auf der Konsole auszuprinten. Achte dabei auf eine schöne Formatierung.

c) Schreibe eine doppelt verkettete Liste, welche als Daten Pointer auf den struct `student` speichern kann.

d) Schreibe eine Funktion, welche Speicher für einen neuen `studenten` allokiert und einen Pointer auf diesen zurückgibt. Fülle diesen `studenten` mit deinen (Wunsch-)Daten und ~~schicke diese an Google~~ füge diesen Eintrag der Liste hinzu.

e) Wiederhole den vorherigen Schritt mit den Daten deiner Freunde. Wenn du keine Freunde hast, denke dir welche aus.

f) Schreibe eine Funktion, um dir den gesamten Inhalt der Liste auf der Konsole auszugeben.

g) Ein `student` wird exmatrikuliert und muss deshalb aus der Liste gelöscht werden.

h) Sorge dafür, dass alles Elemente wieder einwandfrei gelöscht werden und gebe allen Speicher frei, den du benutzt hast.