

### Aufgabe 1.

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

```
1 #ifndef _COMPLEX__H
2 #define _COMPLEX__H
3
4 typedef struct _COMPLEX {
5     double real;
6     double imag;
7 } COMPLEX;
8
9 COMPLEX cplx_add(COMPLEX a, COMPLEX b);
10 COMPLEX cplx_mul(COMPLEX a, COMPLEX b);
11 COMPLEX cplx_pow(COMPLEX a, unsigned long n);
12 COMPLEX cplx_div(COMPLEX a, COMPLEX b);
13
14 #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.

```
1 /* Definiere hier angemessene Strukturen fuer einen
2    einzelnen Listeneintrag und die Liste selbst. */
3
4 /* Leere Liste erstellen */
5 LIST *list_create();
6
7 /* Element hinter E einfuegen, NULL heisst am Anfang */
8 LISTNODE *list_insert(LIST *L, LISTNODE *E, double p);
9
10 /* 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 alle Elemente wieder einwandfrei gelöscht werden und gebe allen Speicher frei, den du benutzt hast.

### INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):  
  IF LENGTH(LIST) < 2:  
    RETURN LIST  
  PIVOT = INT(LENGTH(LIST) / 2)  
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])  
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])  
  // UMMMMMM  
  RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):  
  // AN OPTIMIZED BOGOSORT  
  // RUNS IN O(N LOG N)  
  FOR N FROM 1 TO LOG(LENGTH(LIST)):  
    SHUFFLE(LIST):  
    IF ISSORTED(LIST):  
      RETURN LIST  
  RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBIINTERVIEWQUICKSORT(LIST):  
  OK SO YOU CHOOSE A PIVOT  
  THEN DIVIDE THE LIST IN HALF  
  FOR EACH HALF:  
    CHECK TO SEE IF IT'S SORTED  
    NO, WAIT, IT DOESN'T MATTER  
    COMPARE EACH ELEMENT TO THE PIVOT  
    THE BIGGER ONES GO IN A NEW LIST  
    THE EQUAL ONES GO INTO, UH  
    THE SECOND LIST FROM BEFORE  
  HANG ON, LET ME NAME THE LISTS  
  THIS IS LIST A  
  THE NEW ONE IS LIST B  
  PUT THE BIG ONES INTO LIST B  
  NOW TAKE THE SECOND LIST  
  CALL IT LIST, UH, A2  
  WHICH ONE WAS THE PIVOT IN?  
  SCRATCH ALL THAT  
  IT JUST RECURSIVELY CALLS ITSELF  
  UNTIL BOTH LISTS ARE EMPTY  
  RIGHT?  
  NOT EMPTY, BUT YOU KNOW WHAT I MEAN  
  AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):  
  IF ISSORTED(LIST):  
    RETURN LIST  
  FOR N FROM 1 TO 10000:  
    PIVOT = RANDOM(0, LENGTH(LIST))  
    LIST = LIST[:PIVOT] + LIST[PIVOT:]  
    IF ISSORTED(LIST):  
      RETURN LIST  
  IF ISSORTED(LIST):  
    RETURN LIST  
  IF ISSORTED(LIST): // THIS CAN'T BE HAPPENING  
    RETURN LIST  
  IF ISSORTED(LIST): // COME ON COME ON  
    RETURN LIST  
  // OH JEEZ  
  // I'M GONNA BE IN SO MUCH TROUBLE  
  LIST = [ ]  
  SYSTEM("SHUTDOWN -H +5")  
  SYSTEM("RM -RF ./")  
  SYSTEM("RM -RF ~/*")  
  SYSTEM("RM -RF /")  
  SYSTEM("RD /S /Q C:\*") // PORTABILITY  
  RETURN [1, 2, 3, 4, 5]
```