Aufgabe 1. Verwende `qsort` um ein Array von rationalen Zaheln (gestern implementiert) zu sortieren.

Aufgabe 2. Implementiere doppelt verkettete Listen, die `double`-Variablen speichern.

```
/* 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 */
LISTNODE *list_unshift(LIST *L, double p);
LISTNODE *list_push(LIST *L, double p);

/* Element am Anfang bzw. Ende entfernen und
   die Daten zurueck geben */
double list_shift(LIST *L);
double list_pop(LIST *L);

/* eine Element aus der Liste entfernen */
void list_delete(LIST *L, LISTNODE *E);

/* zwei Listen zusammenfuegen */
LIST *list_merge(LIST *L, LIST *M);

/* Liste inklusive allen Elementen frei geben */
void list_free(LIST *L);
```

Für die Allgemeinbildung, hier, was ihr gerade gemacht habt:

Ein *Stack* ist eine Datenstruktur zum speichern mehrerer Einträge, welche nur die Operationen push und pop unterstützt: das heißt, man kann immer nur am Ende des Stacks neue Daten anfügen und auch nur von dort Daten entfernen (Natürlich würde man einen Stack für gewöhnlich mit einer Array implementieren)

Eine *Queue* ist eine Datenstruktur, welche ausschließlich die Operationen unshift und pop unterstützt: Man kann nur am Anfang Daten anfügen und nur am Ende Daten entfernen.

Aufgabe 3. Das 8-Damen Problem ist wie folgt definiert: Platziere 8 Damen auf einem gewöhnlichen Schachbrett so, dass sie sich paarweise nicht bedrohen.

Das n-Damen Problem ist analog definiert (auf einem $n \times n$-Schachbrett). Schreibe ein Programm, dass das n-Damen Problem löst (falls möglich). Es hilft bei solchen Fragestellung häufig, sich Spezialfälle aufzumalen (betrachte beispielsweise das 3- oder 4-Damen Problem).