Aufgabe 1. Denke dir einen sinnvollen Namen aus für ein Modul, dass Vektorrechnung auf $\mathbb{R}^n$ implementiert. Vektoren sollen `double`-Arrays mit der Länge n sein. Implementiere die nachfolgenden Funktionen:

a) eine Funktion, die genügend Speicher für einen Vektor reserviert und einen Pointer darauf zurück gibt

b) Vektoraddition und –subtraktion

c) Produkt eines Vektors mit einer skalaren Größe

d) Skalarprodukt zweier Vektoren

e) eine Funktion, die prüft, ob zwei Vektoren orthogonal zueinander stehen

f) eine Funktion, die prüft, ob zwei Vektoren parallel zueinander sind

g) eine Funktion, die einen Vektor auf der Konsole aus gibt

Aufgabe 2. Implementiere einige Funktionen um mit quadratischen Matrizen umzugehen:

a) Eine Funktion, die Speicher für eine quadratische Matrix allokiert, eine um ihn freizugeben, eine um sie auszugeben und eine um sie zur Einheitsmatrix zu initialisieren (das ist die Matrix mit 1en auf der Hauptdiagonale und 0en sonst):

```
1 double **matrix_alloc(int n);
2 void     matrix_free(double **A, int n);
3 void     matrix_print(double **A, int n);
4 double **matrix_id(double **A, int n);
```

b) Eine Funktion um eine Matrix zu transponieren (d.h. an der Hauptdiagonale "zu spiegeln")

c) Eine Funktion, die zwei solche Matrizen miteinander multipliziert und eine neue Matrix zurück gibt. Für zwei $n \times n$-Matrizen $A = (a_{ij})$ und $B = (b_{ij})$ ist $A \cdot B = C = (c_{ij})$ über folgende Formel definiert:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Aufgabe 3. Implementiere eine c-Datei zu folgender Header-Datei:

```c
/* gibt die Länge eines Strings zurück */
int str_len(char *s);

/* gibt 0 zurück, wenn zwei strings gleich
 * sind und 1 sonst */
int str_cmp(char *s1, char *s2);

/* kopiert s nach d und gibt d zurück */
char *str_cpy(char *d, char *s);

/* hänge s2 and s1 an und gib s1 zurück */
char *str_cat(char* s1, char* s2)
```

Aufgabe 4. Diese Aufgabe läuft auf die Implementierung des Merge-Sort Algorithmus hinaus.

a) Implementiere eine Funktion `merge`, die zwei bereits sortierte (eventuell verschieden große) Arrays als Argumente erhält, diese zu einem sortieren Array kombiniert und dieses zurück liefert.

b) Die Funktion `mergesort` selbst soll ein Array in zwei (möglichst gleich große) Teilarrays zerlegen, sich für diese Teilarrays selbst aufrufen und danach die dann sortierten Teilarrays mit der `merge`-Funktion kombinieren. Erhält die Funktion ein Array mit keinem oder einem Element so belässt es dieses Array wie es ist, dann ist es nämlich bereits sortiert.

c) Besorge dir die Datei `daten.h`, sortiere das darin definierte Array und schreibe es sortiert in eine Datei.

Hier als Tipp ein Vorschlag für die Signaturen der beiden Funktionen:

```c
int *merge(int *list1, int n, int *list2, int m);
void mergesort(int *list, int n);
```