Aufgabe 1. Implementiere die Signumsfunktion `sgn(x)`, den Absolutbetrag `betrag(x)`, `cos(x)` und die Wurzelfunktion `wurzel(x)` (mit dem Heron-Verfahren vom ersten Zettel) als Funktionen und lagere sie in ein eigenes Modul aus.

Aufgabe 2. Schreibe Funktionen `square_to` und `root_to`, die einen `double`-Pointer entgegen nehmen, die dort stehende Variable quadrieren bzw. daraus die Wurzel ziehen und das Ergebnis sowohl zurück geben als auch an die gleiche Speicherstelle schreiben Schreibe diese Funktionen auch in das mmymathModul.

Aufgabe 3. Erweitere das „mymath"-Modul noch um eine Funktion, die zu den drei Koeffizienten $a, b, c \in \mathbb{R}$ einer quadratischen Gleichung

$$a \cdot x^2 + b \cdot x + c = 0$$

die Lösungen berechnet.

Aufgabe 4. Schreibe ein Modul `arrayhelpers`, das einige nützliche Funktion zum Umgang mit statischen `int`-Arrays enthält:

a) Array zeilenweise oder mit Kommata getrennt ausgeben

b) Alle Felder eines Arrays mit einem Wert initialisieren

c) Array um 1 rotieren (d.h. das hinterste Element an erste Stelle schreiben und alle anderen Elemente um eins nach hinten schieben)

d) Array um k rotieren

e) Array umdrehen

f) Ein Array in einem anderen suchen und die Position zurück geben. Sollte das Array nicht im anderen enthalten sein, so soll der Rückgabewert -1 sein.

Beispiel:

```
1 int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
2 int B[3] = {4, 5, 6}
3 int C[2] = {5, 7}
4 int D[2] = {9, 10}
```

Hier gilt: `B` ist an 3-ter Stelle in `A` enthalten und `D` an 8-ter. Das Array `C` ist garnicht in `A` enthalten, darum wird der Rückgabewert -1 sein.

Aufgabe 5. In dieser Aufgabe geht es um Sortieralgorithmen. Definiere dir ein Test-Array mit einer *festen* Anzahl von Einträgen mit denen du den Algorithmus testest.

a) Implementiere folgenden Sortieralgorithmus: Sortiere das kleines Element an die erste Stelle, dann das zweitekleinste Element an die zweite Stelle usw. Dieser Algorithmus ist in vielen Fällen relativ langsam (aber sehr schnell bei kurzen Listen) – man spricht auch von einer Komplexität von $\mathcal{O}(n^2)$ (wobei n die Anzahl der Elemente ist).

b) Aus theoretischer Sicht sind Sortieralgorithmen bis zu $\mathcal{O}(n \log(n))$ realisierbar (von denen wir in den nächsten Tagen auch noch einige kennen lernen werden). Wenn man nun aber die Wertemenge der zu sortierenden Eintrag einschränkt (z.B. sei die größte zu sortierende Zahl 20000) ist es sogar möglich einen *linearen* Sortieralgorithmus zu implementieren, also $\mathcal{O}(n)$. Dazu stellt man sich für jede Zahl einen leeren "Bucket" (Korb) vor. Dann geht man die Liste der zu sortierenden Einträge durch und für ein Verkommen der Zahl k einen Ball in den k-ten Bucket. Danach hat man alle Zahlen aus der Liste statt dessen in die Buckets sortiert und kann die ursprüngliche Liste überschreiben. Diese machen wir auf folgende Weise: Gehe die Buckets vom ersten bis zum letzten durch. Liegen im k-ten Bucket j Bälle, schreibe j mal die Zahl k hintereinander in die Liste. Da die Bälle genau den zu sortierenden Zahlen entsprechen ist die Liste nachher sortiert. Diese Algorithmus heißt "Bucket-Sort".

   Implementiere den Bucket-Sort-Algorithmus für `int`-Arrays mit Werten zwischen 0 und 20000.