Bevor du diesen Zettel machst, solltest du die Aufgaben 1 und 4 vom gestrigen Zettel bearbeiten.

Aufgabe 1. Das folgende Programm sollte ein Array mit n Zahlen allokieren. In den i-ten Eintrag $1/(i+1)$ schreiben und das Array ausgeben. Dieses Programm benutzt die Funktion `printarray` aus dem Modul `"arrayhelpers"`. Leider haben wir dabei 5 Fehler gemacht. Schnapp sie dir alle!

```c
/* Array erstellen, mit 1/(i+1) fuellen und ausgeben.
 * (c) 2015 Clelia und Johannes */

#include <stdio.h>
#include "arrayhelpers.h"

int main () {
   double *array;
   int n,i;

   n = 42;
   /* Hole Speicher fuer n Eintraege */
   array = malloc (n);
   if (array == NULL) {
      printf ("Fehler, nicht genug Speicher.\n");
   }
   for (i = 0;i < sizeof (array);i++){
      /* Schreibe 1/(i+1) in Array */
      array[i] = 1/(i+1);
   }
   printarray (array, n); /* Gebe Array aus */
   free (array);
   return 0;
}
```

Aufgabe 2. Gegeben sei eine Datei, in der ausschließlich Zahlen stehen. In der ersten Zeile stehe eine natürliche Zahl, die angibt wie viele Zahlen noch folgen. Schreibe ein Programm, dass diese Datei einliest, die Zahlen sortiert und die Datei mit der sortierten Liste überschreibt.

Aufgabe 3. a) Schreibe ein Programm, welches eine Datei im folgenden

Format ausliest: In der ersten Zeile steht die Anzahl der folgende Zeilen. Die Zeilen sehen dann so aus:

```
cos(0.4) = 0.921
cos(0.45) = 0.9014
cos(0.6) = 0.82533
```

Wenn in einer Zeile das Ergebnis um mehr als 10^{-3} von dem tatsächlichen Wert abweicht, so soll diese Differenz mit Zeilennummer auf der Konsole ausgegeben werden.

b) * Modifiziere das Programm so, dass in der ersten Zeile nicht mehr die Anzahl der Einträge stehen muss.

Aufgabe 4. 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):

```
double **matrix_alloc(int n);
void     matrix_free(double **A, int n);
void     matrix_print(double **A, int n);
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 $A \cdot B = C = (c_{ij})$ durch $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ definiert.

Aufgabe 5. * Implementiere den Bucket-Sort Algorithmus: Unter der Annahme, dass die zu sortierenden Daten aus einem endlichen Wertebereich stammen kann man dies theoretisch sehr schnell machen (man sagt: in Linearzeit). Ohne große Einschränkung der Universalität des Algorithmus betrachten wir hier nur ein `unsigned short`-Array A. Bucketsort besteht nun aus zwei Durchläufen:

a) Sei n das Maximum aus A, erstelle dann ein `unsigned`-Array B mit $n+1$ Elementen und sorge dafür, dass an der i-ten Stelle die Anzahl der is steht, die in A vorkommen.

b) Gehe nun B durch und überschreibe A. Schreibe dabei k mal das Element i, wenn an der i-ten Stelle von B ein k steht.