Aufgabe 5.1 Multiplikationstabelle verallgemeinern

In Aufgabe 1.3 hattest Du eine Funktion geschrieben, die die Multiplikationstabelle generiert:

```
void generiere_multiplikationstabelle();
```

Diese Funktion hat intern 100 Multiplikationen durchgeführt. Schreibe nun eine Funktion, die nicht hart die beiden Zahlen x und y miteinander multipliziert, sondern stattdessen eine beliebige Funktion auf x und y anwendet, welche der Klient beim Aufruf übergibt:

```
void generiere_tabelle(int (* f)(int, int));
```

Teste diese Funktion mit mindestens zwei Funktionen, z.B. Addition und Multiplikation.

Aufgabe 5.2 Zahlen sortieren

In `<stdlib.h>` ist eine Sortierfunktion `qsort` definiert. Das folgende Beispiel verwendet diese Sortierfunktion, um ein Array von `int`-Elementen aufsteigend zu sortieren:

```
int ascending(const void * x, const void * y)
{
    const int * p = x;    /* Was passiert hier? */
    const int * q = y;

    int i = *p;           /* Was passiert hier? */
    int j = *q;

    return (i > j) - (i < j);
}

int a[] = {3, 7, 1, 2, 1, 8, 3};
qsort(a, sizeof(a) / sizeof(a[0]), sizeof(a[0]), ascending);
```

Schreibe eine Vergleichsfunktion, mit der man ein `int`-Array *absteigend* sortieren kann. Schreibe eine weitere, mit der alle geraden Zahlen vor die ungeraden sortiert werden.

Aufgabe 5.3 Zeichenketten sortieren

Ein Array von Stringliteralen kann in C wie folgt deklariert werden:

```
const char * words[] = {"hello", "beautiful", "world"};
```

Mache Dir die Struktur dieses Arrays bei Bedarf anhand einer Zeichnung klar.

Sortiere ein Array von Zeichenketten lexikographisch (d.h. in derselben Reihenfolge wie im Wörterbuch). Warum kannst Du als Vergleichsfunktion nicht einfach `strcmp` an `qsort` übergeben, sondern musst eine eigene schreiben? Was für Zeiger erwartet `strcmp`, und was für Zeiger bekommst Du in der Vergleichsfunktion tatsächlich übergeben?

Hinweis: In Aufgabe 5.2 mussten die `const void*` in `const int*` umgewandelt werden, weil die zu sortierenden Elemente vom Typ `int` waren. Jetzt besteht das Array dagegen aus Zellen vom Typ `const char*`. Was ist also der korrekte Typ für `p` und `q`?

Schreibe außerdem eine Vergleichsfunktion für eine lexikalische Sortierung (d.h. primär nach Länge sortieren und gleich lange Zeichenketten lexikographisch).

Aufgabe 5.4 Eigene Typen sortieren

Eine Person habe einen Vornamen, einen Nachnamen, ein Alter und ein Geschlecht. Definiere eine passende Struktur und implementiere die beiden folgenden Funktionen:

```
void person__print(const struct person * self);
void print_persons(const struct person * persons, int n);
```

Schreibe vier Vergleichsfunktionen für das Sortieren von Personen mittels `qsort`:

a) alphabetisch

b) nach Alter, von jung nach alt

c) nach Alter, von alt nach jung

d) nach Geschlecht

Deine Testdaten sollten natürlich auch verschiedene Personen mit gleichem Nachnamen/ gleichem Alter enthalten, um die Korrektheit der Vergleichsfunktionen sicherzustellen.

Aufgabe 5.5 Binäre Suche

In der Datei `t9.c` findest Du eine Musterlösung zu Aufgabe 3.6, in der das Wörterbuch nicht mehr als zweidimensionales Array von Zeichen gespeichert wird, sondern als ein Array von Zeigern auf Zeichen. So sind die Wörter nicht mehr in ihrer maximalen Länge beschränkt, und kurze Wörter verschwenden nicht mehr so viel Speicher.

Die lineare Suche aus `in_dictionary` ist verhältnismäßig langsam. Da das Wörterbuch bereits in geordneter Form vorliegt, bietet sich die binäre Suche als effizientere Alternative an. Verwende dazu die in `<stdlib.h>` definierte Suchfunktion `bsearch`. Welchen Schlüssel übergibst du an `bsearch`, und wie muss die Vergleichsfunktion aussehen?