

# Übung zur Vorlesung EidP (WS 2020/21)

## Blatt 6

Block rot

**Es können 4 Punkte erreicht werden.**

**Abgabedatum:** 07. Januar 2021, 23:59 Uhr

### Hinweise

- Bitte beachten Sie die aktuellen Hinweise unter

<https://ls11-www.cs.tu-dortmund.de/teaching/ep2021uebung/>

- Die Aufgaben sind **in Dreiergruppen** zu bearbeiten.
- Für die Abgabe sind die jeweils genannten Dateien zu erstellen.
- Stellen Sie sicher, dass alle von Ihnen abgegebene Dateien reine Textdateien im UTF-8-Format sind.
- Die Verwendung von zusätzlichen **Bibliotheken** zur Lösung der Aufgaben ist **nicht erlaubt!**
- Für die Programmieraufgaben kopieren Sie immer die Ergebnisse als Block-Kommentar an das Ende der Datei, welche das jeweilige Hauptprogramm enthält.
- Am 17.12. wird eine **Probeklausur** auf der Übungswebseite veröffentlicht.
- Am 11.01. wird ein **Zusatzblatt mit 6 Bonuspunkten** auf der Übungswebseite veröffentlicht.

### Aufgaben

#### Aufgabe 1: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten eine Textdatei `Aufgabe_06_1.txt` an.

- a) Erklären Sie den Unterschied zwischen einer lokalen und einer globalen Variable inklusive der Sichtbarkeitsbereiche. (0.2 Punkte)
- b) Wie verhält sich eine statische Variable in einer Funktion? Welche Auswirkung hat dies auf den Gültigkeitsbereich und die Sichtbarkeit? (0.2 Punkte)

- c) Was sind rekursive Funktionen? (0.1 Punkte)
- d) Wie funktioniert das Prinzip der Rekursion in einer Funktion einschließlich Rekursionsverankerung? (0.3 Punkte)
- e) Welches Problem wird mit Namensräumen gelöst? Geben Sie ein Beispiel für einen beliebigen Namensraum an. (0.2 Punkte)

## Aufgabe 2: Rekursion (3 Punkte)

In dieser Aufgabe sollen Sie zunächst einen rekursiven Sortieralgorithmus implementieren, um ein vorgegebenes Array sortieren zu können. Danach sollen Sie einen rekursiven Suchalgorithmus implementieren, um innerhalb des vorgegebenen Arrays einen bestimmten Wert zu finden. Eine Art zu sortieren haben Sie bereits im Aufgabenblatt 0 kennengelernt, diese Art des Sortierens wird *Bubblesort* genannt.

### a) Rekursives Sortieren

Legen Sie für diese Teilaufgabe die Dateien `Aufgabe_06_2a.h` und `Aufgabe_06_2a.cpp` an. Implementieren Sie dazu den rekursiven Bubblesort-Algorithmus und beschreiben Sie die Header-Dateien korrekt, sodass die `bubblesort`-Funktion in der Teilaufgabe c) über eine Header-Datei eingebunden werden kann. Der Funktionskopf soll dabei wie folgt aussehen:

```
void bubblesort(int array[], unsigned int const start,
               unsigned int const end);
```

Benutzen Sie für die Implementierung eine Rekursion. Das Benutzen von Schleifen (egal, ob `do-while`, `while` oder `for`) ist nicht erlaubt. Als Tipp erhalten Sie eine wörtliche Beschreibung des rekursiven Bubblesort-Algorithmus:

- Falls `start` kleiner `end - 1` ist, tue Folgendes:
  - Falls `Element[start]` größer als `Element[start+1]` ist, vertausche diese Elemente und wiederhole Bubblesort auf dem gesamten Array.
  - Ansonsten (`Element[start]` nicht größer als `Element[start+1]`) erhöhe `start` um 1 und wiederhole Bubblesort.

Für das Vertauschen zweier Elemente wird Ihnen die `swap`-Funktion vorgegeben. Diese Funktion tauscht die Werte der beiden Eingabeparameter `int &a` und `int &b`. Sie können die `swap`-Funktion in Ihre `bubblesort` Implementierung einbauen und sich somit Zeilen sparen.

```
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Allerdings sollen Sie die `swap`-Funktion an die richtige Stelle in eine der drei Dateien (`Aufgabe_06_2a.cpp`, `Aufgabe_06_2b.cpp` oder `Aufgabe_06_2c.cpp`) setzen, damit das Programm kompiliert werden kann. Überlegen Sie sich also, welche Funktion `swap` nutzt und sie daher beim Kompilieren kennen sollte.

**Ganz wichtig ist jedoch, dass die Implementierungen Rekursionen enthalten muss und keine Schleifen genutzt werden dürfen!** (1 Punkt)

### b) Rekursives Suchen

Legen Sie für diese Teilaufgabe die Dateien `Aufgabe_06_2b.h` und `Aufgabe_06_2b.cpp` an. Implementieren Sie als nächstes den Such-Algorithmus und beschreiben Sie die Header-Dateien

korrekt, sodass die `finde`-Funktion in der Teilaufgabe c) über eine Header-Datei eingebunden werden kann. Der Funktionskopf soll dabei wie folgt aussehen:

```
int finde(int wert, int folge[],
          unsigned int const intervallLaenge);
```

Bei der Implementierung teilt man ein sortiertes Array in zwei etwa gleich große Arrays und sucht im jeweils zum Suchwert passenden Array weiter, bis der Wert gefunden wird, oder festgestellt wird, dass der Wert nicht vorhanden ist. Der Algorithmus soll den Wert -1 zurückgeben, falls der Suchwert im Array nicht vorhanden ist. Ansonsten soll die Position (Index im Array), an dem sich der Suchwert befindet, zurückgegeben werden.

Falls Sie Teilaufgabe a) nicht lösen konnten, können Sie die folgende iterative Implementierung von `bubblesort` verwenden, um Aufgabenteil b) zu lösen.

```
1 void bubblesort(int array[], unsigned int const start,
2                 unsigned int const end) {
3     for (unsigned int i = start; i < (end - 1); ++i) {
4         for (unsigned int j = start; j < (end - i - 1); ++j) {
5             if (array[j] > array[j + 1]) {
6                 int tmp = array[j];
7                 array[j] = array[j + 1];
8                 array[j + 1] = tmp;
9             }
10        }
11    }
12 }
```

**Ganz wichtig ist, dass die Implementierung von `finde` Rekursionen enthalten muss und keine Schleifen genutzt werden dürfen!** (1.7 Punkte)

c) Testen der Algorithmen

Benutzen Sie für die Teilaufgabe c) die auf der Übungsseite vorgegebene Datei `Aufgabe_06_2c.cpp`. Testen Sie Ihre Implementierungen auf der vorgegebenen `main`-Funktion. Als Hilfe bekommen Sie die Funktion `printArray`. Die Funktion `printArray` gibt ein Array auf dem Terminal aus; damit können Sie überprüfen, ob das Array richtig sortiert wurden. Im Folgenden wird Ihnen die Vorlage für `Aufgabe_06_2c.cpp` gegeben. Nach erfolgreicher Implementierung der Funktionen `bubblesort` und `finde` sollten ab Zeile 28 die Positionen der gesuchten Werte ausgegeben werden. Am Ende sollten Sie 5 Dateien haben (`Aufgabe_06_2a.h`, `Aufgabe_06_2a.cpp`, `Aufgabe_06_2b.h`, `Aufgabe_06_2b.cpp`, `Aufgabe_06_2c.cpp`).

**Aufgabe:** Binden Sie die Header-Dateien für `bubblesort` und `finde` korrekt ein. Kopieren Sie die Ausgabe des Programms an das Ende dieser Datei.

(0.3 Punkte)

### Präsenzaufgabe 3: Rekursionen (0 Punkte)

In dieser Aufgabe sollen drei Probleme **rekursiv** gelöst werden. Für diese Aufgabe wird Ihnen die `main`-Funktion und die Ausgabe vorgegeben. Ihre Aufgabe ist es, die drei folgenden Probleme zu implementieren.

a) Implementieren Sie die `summenformel`-Funktion, so dass diese die  $n$ -te Zahl der Summenformel ausgibt. Die Summenformel ist definiert durch:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} = \frac{n^2+n}{2} = 1 + 2 + 3 + 4 \dots$$

(0 Punkte)

b) Implementieren Sie die `fibonacci`-Funktion, so dass diese Funktion die  $n$ -te Fibonaccizahl zurückgibt. Die Fibonaccizahlen sind definiert durch:

$$f(i) = \begin{cases} 0 & \text{falls } i \leq 0 \\ 1 & \text{falls } 0 < i \leq 2 \\ f(n-1) + f(n-2) & \text{falls } i > 2 \end{cases}$$

(0 Punkte)

c) Implementieren Sie die `fakultaet`-Funktion, so dass diese die  $n$ -te Fakultät ausgibt. Die Fakultät ist definiert durch:

$$\prod_{i=1}^n i = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$

(0 Punkte)

Lösen Sie diese Aufgaben mit Hilfe von Rekursionen. Das Verwenden von Schleifen (egal, ob `do-while`, `while` oder `for`) oder eine direkte Berechnung ist nicht erlaubt. Benutzen Sie folgende Vorlage.

```
1  /** Aufgabe_06_3.cpp */
2  #include <iostream>
3  using namespace std;
4
5  int summenformel(int x) {
6      // hier Ihre Implementierung
7  }
8
9  int fibonacci(int x) {
10     // hier Ihre Implementierung
11 }
12
13 int fakultaet(int x) {
14     // hier Ihre Implementierung
15 }
16
17 int main() {
18     int const n = 10;
19     for (int i = 0; i < n; ++i)
20         cout << fibonacci(i) << " ";
21     cout << endl;
```

```
22     for (int i = 0; i < n; ++i)
23         cout << fakultaet(i) << " ";
24     cout << endl;
25     for (int i = 0; i < n; ++i)
26         cout << summenformel(i) << " ";
27     cout << endl;
28     return 0;
29 }
30 /* Ausgabe:
31 0 1 2 3 4 5 6 7 8 9
32 0 1 2 3 4 5 6 7 8 9
33 0 1 2 3 4 5 6 7 8 9
34 */
35 /*** Ende Aufgabe_06_3.cpp ***/
```