technische universität dortmund

fakultät für informatik

Lutz Oettershagen
Jurij Kuzmic

Dortmund, den 8. November 2018

# Übung zur Vorlesung EidP (WS 2018/19)

# Blatt 4

Block gelb

**Es können 4 Punkte erreicht werden.**

**Abgabedatum:** 15. November 2018, 23:59 Uhr

## Hinweise

- Bitte beachten Sie die aktuellen Hinweise unter

  `https://ls11-www.cs.tu-dortmund.de/teaching/ep1819uebung/`

- Für die Abgabe sind die Dateien `Aufgabe_04_1.txt`, `Aufgabe_04_2a.cpp`, `Aufgabe_04_2b.cpp` und `Aufgabe_04_2c.cpp` zu erstellen.
- Stellen Sie sicher, dass alle von Ihnen abgegebene Dateien reine Textdateien im UTF-8-Format sind.
- Für die Kompilierung des Programms muss der C++11-Standard aktiviert sein. Dies kann im Referenzcompiler GCC 6.3.0 durch den Schalter `-std=c++11` sichergestellt werden. Es soll zudem die Parameter `-pedantic` und `-Werror` genutzt werden. Der Befehl zum Kompilieren soll somit wie folgt aussehen:

  ```
  g++ -pedantic -Werror -std=c++11 Aufgabe_04_2a.cpp -o Aufgabe_04_2a
  ```

## Aufgaben

### Aufgabe 1: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten die Textdatei `Aufgabe_04_1.txt` an.

**a)** Wie ist eine Funktion mit Parameter und mit Rückgabewert aufgebaut? Geben Sie jeweils die Funktionsdeklaration und die Funktionsdefinition an. (0.2 Punkte)

**b)** Welcher Zusammenhang existiert zwischen den Argumenten eines Funktionsaufrufes und den Parametern in der Funktionsdeklaration der aufgerufenen Funktion? (0.2 Punkte)

**c)** Welche Bedeutung hat die Anweisung `return 65;` in einer Funktion mit Rückgabedatentyp `unsigned char` und wie kann die Rückgabe weiterverwendet werden? (0.2 Punkte)

**d)** Worin liegt der Unterschied bei der Übergabe von Variablen als Wert, als Referenz und als Zeiger? (0.2 Punkte)

**e)** Die Funktionsdeklaration einer Funktion kann die Übergabe von Variablen als Zeiger an diese Funktion ermöglichen. Begründen Sie, ob es auch umgekehrt sinnvoll ist, eine lokale Variable `lokalVar`, die in einer Funktion erzeugt wurde, mit `return &lokalVar` zurückzugeben. (0.2 Punkte)

## Aufgabe 2: Funktionen (3 Punkte)

**a)** Die Fibonacci-Zahlen sind wie folgt definiert:

$$f_i = \begin{cases} 1, & \text{falls } 0 < i \leq 2 \\ f_{i-1} + f_{i-2}, & \text{falls } i > 2 \end{cases}$$

Die Fibonacci-Folge sieht somit wie folgt aus: 1, 1, 2, 3, 5, 8, 13, ...

Berechnet werden die jeweils einzelnen Zahlen, indem die zwei vorangegangenen Zahlen addiert werden. Als Beispiel ergibt sich die Zahl 5 in dieser Zahlenfolge, indem die 3 (erste vorherige Zahl) mit der 2 (zweite vorherige Zahl) addiert wird. Die 8 ergibt sich somit aus $5 + 3$ und die 13 aus $8 + 5$.
Berechnen Sie die ersten 30 Fibonacci-Zahlen mit Hilfe einer Funktion. Ihre Funktion soll dabei kein Array für Zwischenergebnisse verwenden. Führen Sie Ihr Programm mit der vorgegebenen Schleife aus und fügen Sie die Ausgabe als Kommentar an das Ende der Datei `Aufgabe_04_2a.cpp` an. Erklären Sie außerdem, weshalb die Verwendung des Schlüsselwortes `const` in Zeile 9 sinnvoll ist, und erläutern Sie kurz dessen Bedeutung.

```
1  #include <iostream>
2  using namespace std;
3
4  //**************************************
5  // Ergaenzen: Definition der Funktion fib
6  //**************************************
7
8  int main() {
9    unsigned int const n = 30;
10   // In Zeile 9 steht const, weil... (hier ergaenzen)
11
12   for (unsigned int i = 1; i <= n; ++i) {
13     //************************************************
14     // Ergaenzen: Aufruf von fib sowie geeignete Ausgabe
15     //************************************************
16   }
17   return 0;
18 }
```

(1 Punkt)

**b)** In dieser Aufgabe sind in einem Array `zahl` vom Typ `int` verschiedene Zahlen gespeichert. Die Zahlen in dem Array sollen mit Hilfe einer Funktion `verdopple` verdoppelt werden. Die Funktion und der Aufruf soll dabei so programmiert werden, dass sie auch bei beliebiger Veränderung des Arrays, insbesondere auch der Anzahl der Zahlen in dem Array, korrekt arbeitet. Hierbei soll nicht das ganze Array an die Funktion, sondern die jeweils einzelnen Zahlen des Arrays, übergeben werden. Dabei können Sie entscheiden, welche der bereits kennengelernten Techniken der „Rückgabe“ (Referenz, Zeiger, Return) Sie verwenden. Fügen Sie die Ausgabe Ihres Programms als Kommentar an das Ende der Datei `Aufgabe_04_2b.cpp` an.

```
1  #include <iostream>
```

```
2  using namespace std;
3
4  //*********************************************
5  // Ergaenzen: Definition der Funktion verdopple
6  //*********************************************
7
8  int main() {
9    unsigned int const n = 8;
10   int zahl[n] = { 7, 55, -15, 48, -27, 49, 88, 96 };
11   cout << "Zahlen vor der Verdopplung" << endl;
12   for (unsigned int i = 0; i < n; ++i) {
13     cout << (i + 1) <<". Zahl: " << zahl[i] << endl;
14   }
15   //*******************************
16   // Ergaenzen: for-Schleife und Aufruf von verdopple
17   //*******************************
18   cout << "Zahlen nach Verdopplung" << endl;
19   for (unsigned int i = 0; i < n; ++i) {
20     cout << (i + 1) << ". Zahl: " << zahl[i] << endl;
21   }
22   return 0;
23 }
```

(1 Punkt)

**c)** In dieser Aufgabe soll *eine* Funktion `produkt_summe_und_differenz` geschrieben werden, die von zwei ganzen Zahlen vom Typ `int` das Produkt, die Summe und die Differenz berechnet. Nutzen Sie dabei drei verschiedene Arten, das jeweilige Ergebnis der drei Berechnungen „zurückzugeben“. Verwenden Sie außerdem nur genau *einen* Funktionsaufruf.
Führen Sie Ihr Programm mit den vorgegebenen Variablen `zahl1` und `zahl2` aus. Das Programm soll auch mit einer beliebigen Veränderung der Variablenwerte korrekte Ergebnisse liefern. Fügen Sie die Ausgabe als Kommentar an das Ende der Datei `Aufgabe_04_2c.cpp` an.

```
1  #include <iostream>
2  using namespace std;
3  //*************************************************************
4  // Ergaenzen: Definition der Funktion produkt_summe_und_differenz
5  //*************************************************************
6  int main() {
7    int const zahl1 = 156, zahl2 = -67;
8    int produkt, summe, differenz;
9    //************************************************
10   // Ergaenzen: Aufruf von produkt_summe_und_differenz
11   //************************************************
12   cout << "Produkt: " << produkt << endl;
13   cout << "Summe: " << summe << endl;
14   cout << "Differenz: " << differenz << endl;
15   return 0;
16 }
```

(1 Punkt)

## Präsenzaufgabe 3: Parameter (0 Punkte)

Implementieren Sie die Funktionen `invers`, `grossbuch` und `umwandeln` wie folgt. Die Verwendung der `<cctype>`-Bibliothek ist nicht erlaubt.

- Die Funktion `invers` soll für Zeichenketten wie z. B. für `"Nashorn"`, die inverse Zeichenkette `"nrohsaN"` ermitteln. Der einzige Parameter der Funktion ist eine Zeichenkette vom Typ `char[]`. Die Funktion soll keinen Rückgabewert haben. Die Länge der Zeichenkette, die als Parameter übergeben wird, ist nicht bekannt.
- Die Funktion `grossbuch` erhält ein einzelnes Zeichen vom Typ `char` als Parameter, und liefert auch ein Zeichen vom Typ `char` als Ergebnis zurück. Falls das übergebene Zeichen ein Kleinbuchstabe ist, wird dieser in den entsprechenden Großbuchstaben umgewandelt und zurückgegeben. Ist das übergebene Zeichen bereits ein Großbuchstabe oder ein anderes Zeichen, so wird dieses einfach wieder zurückgegeben.
- Die Funktion `umwandeln` soll für komplette Zeichenketten wie z. B. `"naShorN"` die Zeichenkette `"NASHORN"` ermitteln, also sämtliche Kleinbuchstaben in Großbuchstaben umwandeln. Verwenden Sie dabei innerhalb der Funktion `umwandeln` die Funktion `grossbuch`. Diese Funktion soll keinen Rückgabewert haben.

Rufen Sie in der Funktion `main` die Funktionen so auf, dass das Array `word` zunächst den String `"tNaFelE"` enthält. Ergänzen Sie dann geeignete Funktionsaufrufe, damit das Array `word` den String `"ELEFANT"` enthält. In der `main`-Funktion sollen dafür keine Kontrollstrukturen verwendet werden.

```
1  #include <iostream>
2  using namespace std;
3  // Ergaenzen: Definition der Funktionen
4  // invers, grossbuch und umwandeln
5
6  int main() {
7    char word[] = "EleFaNt";
8    // Ergaenzen: geeignete Funktionsaufrufe und Ausgaben
9    return 0;
10 }
```