Übung 3

Algorithmen und Programmierung

Jonas Henschel, Jens Pönisch, Dominik Gorgosch, Bastian Felix Bachmann, Arvid Horn, Billy Naumann

Aufgabe 1 (Programmierung von C Funktionen)

Schreiben Sie jeweils eine C-*Funktion* mithilfe des im letzten Übungsblatt vorgegebenen C-Codes, um folgende Algorithmen umzusetzen:

a) Bestimmen Sie das **Maximum von** 3 Eingabewerten. Überlegen Sie im Anschluss, wie Sie die Funktion für 4 Werte anpassen müssten.

b) Bestimmen Sie ob eine gegebene erste Zahl durch eine gegebene zweite Zahl restlos geteilt werden kann. Die Funktion soll 1 zurückgeben, wenn die Division keinen Rest hat und den Wert 0, wenn es einen Rest gibt. In der main() Funktion sollen Sie dann eine Textausgabe entsprechend des Ergebnisses der Funktion umsetzen. Sie können davon ausgehen, dass beide Zahlen positive ganze Zahlen sind und die zweite Zahl ungleich 0 ist.

Aufgabe 2 (Rekursion - Aufgaben)

a) Sie wollen eine Funktion schreiben, welche Zahlen von $1 \dots n$ aufsummiert. Dies könnte wie folgt geschehen:

```
1 int sum1() { return 1; }
2 int sum2() { return 3; }
3 int sum3() { return 6; }
4 int sum4() { return 10;}
```

Wie Sie sehen, gibt diese Programm lediglich von Ihnen vorbereitete Summen zurückgibt. Ohne weitere Erklärung ist es schwer nachzuvollziehen was die Funktion tut und vor allem ist es auch unnötige Arbeit das Ergebnis im Vorfeld selbst auszurechnen. Das könnte man auch abstrakter darstellen:

```
1 int sum1() { return 1 ;}
2 int sum2() { return 2 + 1;}
3 int sum3() { return 3 + 2 + 1; }
4 int sum4() { return 4 + 3 + 2 + 1; }
```

Hier ist besser ersichtlich was passiert und vor allem hat der Rechner den Aufwand der Berechnung. Trotzdem ist es immer noch sehr aufwendig den Code zu schreiben. Das geht noch besser:

```
1  int sum1() { return 1; }
2  int sum2() { return 2 + sum1();}
3  int sum3() { return 3 + sum2();}
4  int sum4() { return 4 + sum3();}
```

Jetzt nutzen wir bereits die Ergebnisse der jeweiligen kleineren Funktion. Aber geht das noch allgemeiner?

```
1  int sum1(int number) {
2      return number;
3  }
4
5  int sum2(int number) {
6      return number + sum1(number - 1);
7  }
8
9  int sum3(int number) {
10     return number + sum2(number - 1);
11 }
12
13 int sum4(int number) {
14     return number + sum3(number - 1);
15 }
```

Mit der letzten Lösung sind wir schon sehr nahe einer rekursiven Lösung. Können Sie den letzten Schritt machen?

b) Schreiben Sie eine ***rekursive*** Funktion zur Berechnung der **Fakultät** einer gegebenen Zahl. Lassen Sie für jeden Aufruf die Parameter und jeden Rückgabewert der Funktion auf der Kommandozeile ausgeben.

c) Schreiben Sie eine ***rekursive*** Funktion, die die Quersumme einer übergebenen positiven Zahl berechnet und zurückgibt. Die Quersumme ist die Summer aller Ziffern einer Zahl (quersumme(1234) == 1+2+3+4 == 10).

d) Gegeben ist eine positive ganze Zahl n. Geben Sie nacheinander alle möglichen Zahlen im Dezimalsystem aus, die sich aus n ergeben, wenn man für jede Stelle in n alle Stellen rechts von der gewählten Stelle auf 0 setzt.
Beispiel: n=4175, Mögliche Zahlen: 4175, 4170, 4100, 4000. (Die Reihenfolge der ausgegebenen Zahlen ist nicht wichtig)

Aufgabe 3 (Sichtbarkeit und Lebensdauer von Variablen)

a) Welche Variablen sind direkt **nach** den Anweisungen in den Quelltextzeilen [4, 6, 8, 10 und 14] lebendig? Geben Sie zusätzlich ihre Sichtbarkeit und ihren Wert an.

```
1  double durchschnitt(double, double);
2
3  int main() {
```

```
4       double a = 1.0, b = 10.0, c;
5       {
6           double b = 40.0;
7           double d = 60.0;
8           c = durchschnitt(d, b);
9       }
10      return c;
11  }
12
13  double durchschnitt(double e, double f) {
14      return (e + f) / 2;
15  }
```

b) Welche Aussagen können Sie zur Lebendigkeit und Sichtbarkeit der Variablen `n` und `res` treffen?

```
1   int fak(int n) {
2       if (n == 0) {
3           return 1;
4       } else {
5           int res = n;
6           res = res * fak(n-1);
7           return res;
8       }
9   }
```