Übung 10

Ziel der Übung:

- Umwandlung von Zahlensystemen programmieren
- Wiederholung: Algorithmus, Funktionen und Schleifen

Aufgabe 1: Umwandlung von Dezimalzahlen in andere Zahlensysteme

In der Vorlesung „Zahlensysteme“ haben Sie gelernt, dass man natürliche Zahlen mit Hilfe von Stellenwertsystemen darstellen kann. Schreiben Sie ein **C++-Programm**, das eine natürliche Zahl $n \geq 1$ einliest und mit Hilfe der folgenden Funktionalität in ein **anderes Zahlensystem** umwandelt.

Teil a) Schreiben Sie eine C++-Funktion `int konvertierung(int n, int a[])`, die eine natürliche Zahl **n** aus dem **Dezimalsystem in das Binärsystem** (Basis 2) umwandelt und auf dem Bildschirm ausgibt. Für die Umwandlung soll eine **Schleife** verwendet und das Ergebnis in ein Array **a[ ]** gespeichert werden. Die Funktion soll die Anzahl der belegten Arrayelemente zurückliefern.

Hinweis: Sehen Sie sich noch einmal das Verfahren aus der Vorlesung an und überlegen Sie sich, welche Werte Sie benötigen. Die Binärziffern können innerhalb der Funktion auch in umgekehrter Reihenfolge ausgeben werden.

Teil b) Schreiben Sie eine rekursive C++-Funktion `void rekursiveKonvertierung(int n)`, die eine natürliche Zahl **n** aus dem **Dezimalsystem in das Binärsystem** (Basis 2) umwandelt und auf dem Bildschirm ausgibt. Überlegen Sie hier, wie man innerhalb der Funktion die Binärziffern in der richtigen Reihenfolge ausgeben kann.

Teil c) Erweitern Sie ihre Funktion **`rekursiveKonvertierung`** so, dass eine Zahl **n** auch in ein Stellenwertsystem mit der Basis 8 (Oktal) und der Basis 16 (Hexadezimal) umwandeln kann.

Teil d) Verändern Sie Ihr C++-Programm so, dass Sie mit Hilfe eines Auswahlmenüs die Konvertierung in verschiedene Zahlensysteme (Binär, Oktal und Hexadezimal) durchführen können. Testen Sie das Programm für Beispiele und deren Konvertierungen.

Beispiel: Die Aus- und Eingaben Ihres Programms können wie folgt aussehen (die Ausgaben sind schwarz und die Eingaben grau dargestellt):

```
Geben Sie eine Zahl ein: 101
Waehlen Sie (1) Basis 2, (2) Basis 8, (3) Basis 16 oder (0) beenden: 1
1100101
Waehlen Sie (1) Basis 2, (2) Basis 8, (3) Basis 16 oder (0) beenden: 2
145
Waehlen Sie (1) Basis 2, (2) Basis 8, (3) Basis 16 oder (0) beenden: 3
65
```

Aufgabe 2: Bestimmung von Armstrong-Zahlen

Eine natürliche Zahl heißt Armstrong-Zahl, wenn die Summe ihrer Ziffern, jeweils potenziert mit der Stellenanzahl der Zahl, wieder die Zahl selbst ergibt:

$$153 = 1^3 + 5^3 + 3^3 \Rightarrow 153 \text{ ist eine Armstrong-Zahl}$$

Schreiben Sie ein **C++-Programm**, das eine Zahl einliest und für diese Zahl ausgibt, ob es sich um eine Armstrong-Zahl handelt oder nicht. Um festzustellen, ob es sich bei der eingelesenen Zahl um eine Armstrong-Zahl handelt, entwickeln und verwenden Sie die folgenden Funktionen:

1. Schreiben Sie eine C++-Funktion `int stellenanzahl(int n)`, die für eine beliebige natürliche Zahl **n** die Anzahl ihrer Dezimalziffern zurückliefert.
2. Schreiben Sie eine C++-Funktion `bool armstrong(int n)`, die überprüft, ob eine beliebige natürliche Zahl **n** eine Armstrong-Zahl ist. Verwenden Sie dazu die Funktion **`stellenanzahl`**. Wenn es sich um eine Armstrong-Zahl handelt, soll die Funktion `true` zurückgeben, sonst `false`.

Beispiele: Die Aus- und Eingaben Ihres Programms können wie folgt aussehen (die Ausgaben sind schwarz und die Eingaben grau dargestellt):

```
Geben Sie eine Zahl ein: 153
Die Zahl 153 ist eine Armstrong-Zahl.
```

```
Geben Sie eine Zahl ein: 287
Die Zahl 287 ist keine Armstrong-Zahl.
```

Hausaufgaben Serie 10

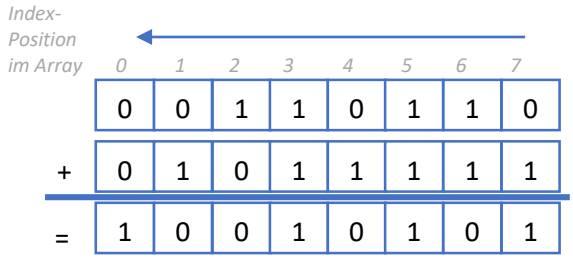
Hausaufgaben mit Namen, Studiengang und Matrikelnummer unter **„Aufgaben“** auf StudIP hochladen.
Abgabe bis **11.01.2022** (ÜG-1), **12.01.2022** (ÜG-4), **13.01.2022** (ÜG-2) und **14.01.2022** (ÜG-3).

Aufgabe 1: Binäre Zahlen addieren

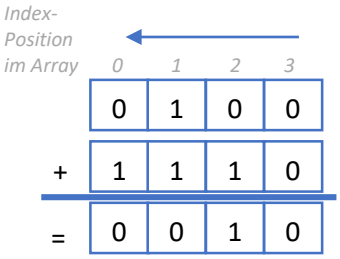
In der Vorlesung haben Sie Binärzahlen und deren Addition kennengelernt. Bilden Sie nun die schriftliche Addition von Binärzahlen **mit Hilfe von Arrays** nach. Dabei wird eine Binärzahl in einem Array aus Nullen und Einsen gespeichert. Die einzelnen Stellen der Zahlen werden bei der Addition von hinten nach vorne addiert, wobei folgende Rechenregeln gelten:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 Überlauf 1
1 + 1 + 1 (Überlauf) = 1 Überlauf 1

Folgende Beispiele zeigen die Addition von zwei Binärzahlen mit einer Bitbreite von 8 bzw. 4:



Beispiel 1: *Bit-Breite (=8) des Ergebnisses ist ausreichend, kein Überlauf bzw. 0*



Beispiel 2: *Bit-Breite (=4) des Ergebnisses reicht nicht, es ergibt sich ein Überlauf von 1*

Teil a) Schreiben Sie ein **C++ Programm**, in dem Sie zwei Arrays vom Typ **`int`** definieren und jeweils mit einer beliebigen Binärzahl initialisieren. Dabei können Sie die Bitbreite (Anzahl der Elemente) frei wählen. Zusätzlich sollen Sie ein drittes Array definieren, das für das Ergebnis der binären Addition benötigt wird. Beachten Sie, dass alle drei Arrays die gleiche Bitbreite haben. Entwickeln Sie dann die in Teil b angeforderte Additionsfunktion. Führen Sie abschließend die binäre Addition mittels Ihrer Funktion und der drei Arrays durch und geben Sie das Ergebnis und den Überlauf aus.

Teil b) Schreiben Sie eine **C++-Funktion** `int binaereAddition (int a[], int b[], int bits, int erg[])`, die nach den oben genannten Rechenregeln eine binäre Addition durchführt. Dabei repräsentieren die Eingabeparameter **a** und **b** die als Arrays gespeicherten Binärzahlen, die zu addieren sind. Beide Binärzahlen haben die gleiche Anzahl von Stellen (Ziffern), welche der Funktion als weiterer Eingabeparameter **bits** übergeben wird. In dem Parameter **erg** soll das Ergebnis der Addition, ebenfalls als Binärzahl, gespeichert werden. Gehen Sie davon aus, dass die Arrays nur Nullen und Einsen enthalten. Ergibt sich im letzten Additionsschritt ein Überlauf, der aufgrund mangelnder Bit-Breite nicht im Ergebnis-Array gespeichert werden kann, so liefert die Funktion eine **1**, sonst **0** zurück.

(12 Punkte)

Aufgabe 2: Palindrom-Test einer Binärzahl

Zahlenpalindrome sind Zahlen, die von vorne wie von hinten gelesen denselben Wert ergeben. Schreiben Sie ein **C++-Programm**, das überprüft, ob eine Binärzahl ein Palindrom ist. Dafür sollen Sie zunächst eine beliebige Binärzahl in einem **Array** speichern. Prüfen Sie dann, ob die im Array gespeicherte Binärzahl von vorne und von hinten gelesen die gleiche Binärzahl ist. Geben Sie anschließend die Binärzahl und das Ergebnis aus.

Beispiele: (die Ausgaben ihres Programms sollen wie folgt aussehen.)

```
01100110
Palindrom

0111110
Palindrom

0111
kein Palindrom
```

(8 Punkte)