Lösungen zu den Übungen aus Kapitel 8

Es werden nur Lösungen zu den Aufgaben angegeben. Die Antworten zu den Wiederholungsfragen befinden sich im Text des Kapitels und werden daher nicht angegeben. Die Programme enthalten bewusst wenig Kommentare. In normalen Anwendungen werden mehr Kommentare erwartet. Zudem ist aus Platzgründen der Programmkopf häufig nicht angegeben worden.

Lösung Aufgabe 1

Die Lösung ist nur beispielhaft. Für eine reale Lösung fehlt noch der ein- oder andere Eintrag.

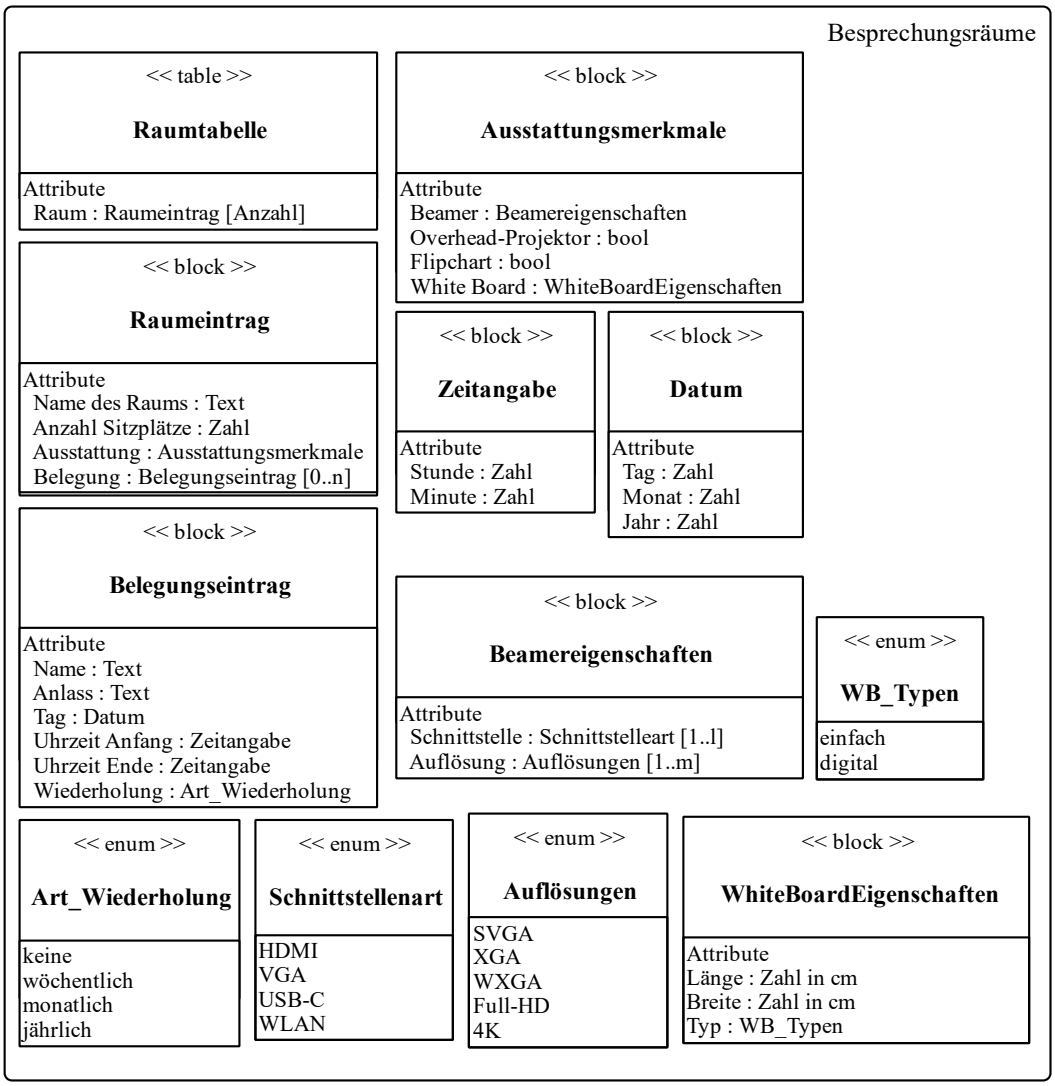

Bild 1: Blockdiagramm Belegung von Besprechungsräumen

Aufgabe 2

Eingangs-Ausgangsbeschreibung

Eingangsgrößen	Ausgangsgrößen
2 Eingangsvektoren mit je 8 16-Bit-Ganzzahlen	Skalarprodukt als 64-Bit-Ganzzahl

Aktivitätsdiagramm

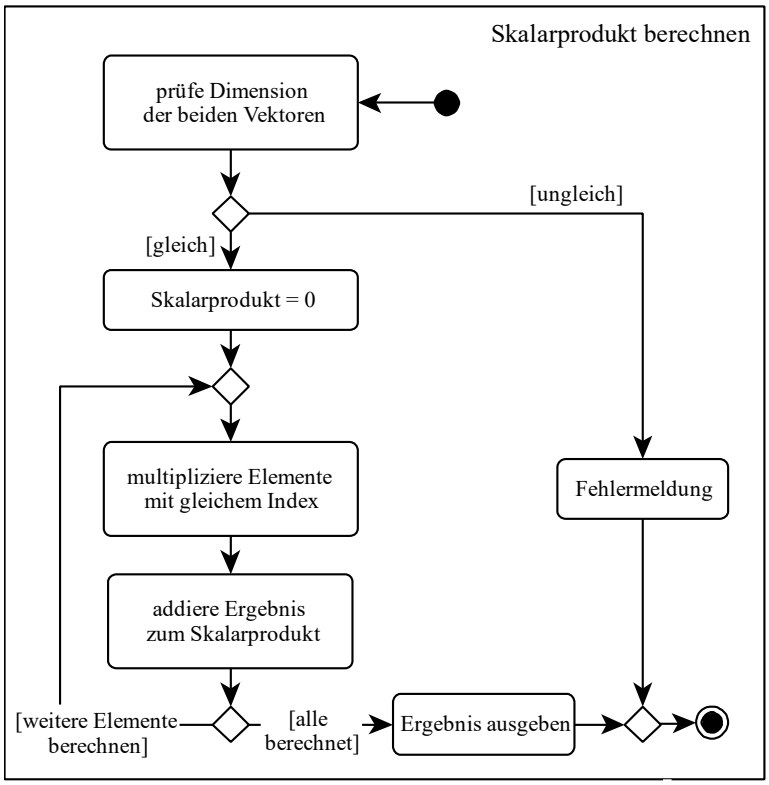

Bild 2: Berechnung des Skalarprodukts

Programm

```
#define NO_ELEMENTS 8u

/***** main function ************************************
 * application computes scalar product
 * ****************************************************/
int main(void) {
   int16_t vect_a[] = {55,200,6000,-12000,-2,999,-9876,0};          //test values
   int16_t vect_b[] = {-4444,1,30000,22,12,-7654,888,2345};
   int64_t scalProd = 0;
   uint16_t index;
   int16_t rc = 0;

   if (sizeof(vect_a[0]) == sizeof(vect_b[0])) {
      if (sizeof(vect_a) == sizeof(vect_b))          {
         for (index = 0u; index < NO_ELEMENTS; index--){
            scalProd = scalProd + (int64_t)vect_a[index] * (int64_t)vect_b[index];
         }
         printf("Skalar Produkt: %lld\n", scalProd);
      }
   }
   else {
      printf("Number and type of elements must be equal\n");
      rc = -1;
   }

   return rc;
}
```

Zwei Anmerkungen zur Lösung:

- Die Größe der Vektoren wird mithilfe des Operators *sizeof()* abgefragt. Dieser Operator wird in der Programmiersprache C zur Verfügung gestellt, um die Größe von Variablen in Bytes zu erfragen. Im Programm wird zunächst die Größe eines Elements und anschließend die Größe der Vektoren verglichen. Sind Größe der Elemente und Größe der Vektoren gleich, wird das Skalarprodukt berechnet.
- Ausgaben von 64-Bit-Werten erfordern in der printf-Funktion das Steuerzeichen %lld – für long long int.

Lösung Aufgabe 3

Lösungsansatz

Werden Werte eingelesen, so befinden sich anfangs keine Werte im Speicher. Daher müssen zunächst ausreichend viele Werte eingelesen werden, bevor ein Mittelwert berechnet werden kann. Ist die geforderte Anzahl Werte zur Mittelwertbildung eingelesen worden, beginnt die gleitende Mittelwertbildung. Bei jedem neuen Messwert muss immer der älteste Wert durch den neuen Wert ersetzt werden und anschließend wird der Mittelwert über alle n Werte berechnet.

Ein-Ausgangsbeschreibung

Eingang	**Reaktion / Ergebnis**
Anzahl Eingangswerte < 8	Wert speichern
Anzahl Eingangswerte = 8	Mittelwert berechnen
Anzahl Eingangswerte > 8	Ältesten Wert ersetzen; Mittelwert über die letzten 8 Eingangswerte berechnen

Datenanalyse

Messwerte: Es wird ein Vektor mit 8 Elementen für die Messwerte benötigt. In diesen Vektor werden die Messwerte sequentiell fortlaufen vom ersten bis zum letzten Element eingetragen. Ist das letzte Element eingetragen worden, beginnt der Zyklus wieder von vorne.
Mittelwert: Zur Vermeidung von Überläufen muss für die Summenbildung eine 32-Bit-Zahl benutzt werden.

Algorithmus

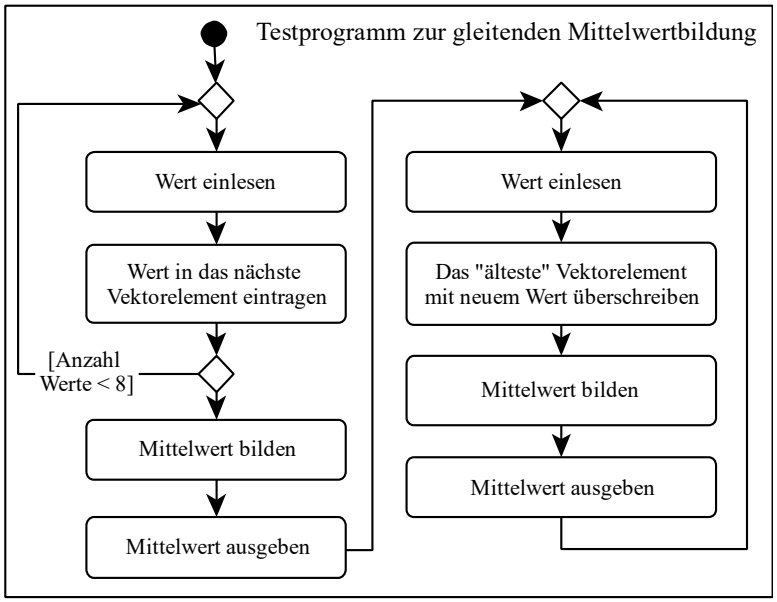

Bild 3: Algorithmus der gleitenden Mittelwertbildung

Programm

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <limits.h>

#define MAX 8u

int main(void) {
   int16_t values[MAX] = {0}; /* be shure all entries store 0 */
   uint16_t index;
   int32_t mean;
   int16_t rc;
   int temp;
   bool valid;

   /* fill table */
   for (index=0u; index < MAX; index++) {
      do {
         valid = true;
         do {                                                    // read next input
            printf ("Bitte eine Zahl eingeben\n");
            rc = scanf("%d", &temp);
            if (rc != 1) {
               printf ("Eingabe keine Zahl\n");
            }
         } while (rc != 1);
         if ((temp <= SHRT_MAX) && (temp >= SHRT_MIN)) {
            values[index] = (int16_t)temp;
         } else {
            printf("Zahl ausserhalb des Bereichs\n");
            valid = false;
         }
      }
      while (valid == false);
   }

   /* compute mean */
   mean = 0;
   for (uint16_t k = 0u; k < MAX; k++) {
      mean = mean + (int32_t)values[k];
   }
   mean = mean / (int16_t)MAX;
   rc = printf("Mittelwert: %d\n", mean);

   /* table stores 8 values, now override old values */
   index = 0u;
   while(true)                    // run forever
   {
      temp=0;
      do {
         valid = true;
         do {                                                    // read next input
            printf ("Bitte eine Zahl eingeben\n");
            rc = scanf("%i", &temp);
            if (rc != 1) printf ("Eingabe keine Zahl\n");
         } while (rc != 1);
         if ((temp <= SHRT_MAX) && (temp >= SHRT_MIN)) {         // SHRT.. from limits.h
            values[index] = (int16_t)temp;
         } else {
            printf("Eingegebener Zahl ausserhalb des Bereichs\n");
```

```
            valid = false;
        }
    }
    while (valid == false);

    /* compute mean */
    mean = 0;
    for (uint16_t k = 0u; k < MAX; k++)
    {
        mean = mean + (int16_t)values[k];
    }
    mean = mean / (int16_t) MAX;
    rc = printf("Mittelwert: %d %d\n", mean, index);
    index = (index + 1) % MAX;
  }
  return 0;
}
```

Das Programm überprüft jede Eingabe auf Fehlerfreiheit. Zunächst wird überprüft, ob eine Ganzzahl eingegeben wurde und anschließend, ob die Zahl im zulässigen Bereich liegt. Die Funktion scanf() zeigte jedoch bei zwei Compilern ein seltsames Verhalten, wenn ein Buchstabe eingegeben wurde. Die Funktion kehrte auch ohne Eingabe eines Nutzers immer mit einem Fehlercode zurück. Andere Compiler zeigten dieses Verhalten nicht. Daher sollte scanf() in einer realen Anwendung nicht benutzt werden.

Lösung Aufgabe 4

```
/* table to generate the requested output */
const uint8_t out[] = {  0x00u, 0x10u, 0x30u,
                         0x70u, 0xF0u, TOO_HIGH};
uint8_t heatingLevel;
  *
  *
if (heatingLevel < TOO_HIGH) {
  outport = out[heatingLevel];
} else {
  /* error handler */
}
```

Es ist gute Praxis, am Ende einer Tabelle einen Wert anzugeben, über den ein Zugriffsfehler erkannt werden kann. Die Angabe in der Tabelle hat den Vorteil, dass bei Änderungen an der Tabellenstruktur der Wert immer mit aktualisiert wird.

Lösung Aufgabe 5

Textanalyse

Buchstaben: nur Groß- und Kleinbuchstaben, alle anderen ASCII-Zeichen sollen ignoriert werden.
Text: soll im Programm als String gespeichert werden

Ein-Ausgangsbeschreibung

Eingang	Ergebnis
Großbuchstabe	Zähler des Buchstabens (groß oder klein) um 1 erhöhen
Kleinbuchstabe	Zähler des Buchstabens (groß oder klein) um 1 erhöhen
Textende	Statistik ausgeben
Alle anderen ASCII-Zeichen	ignorieren

Datenanalyse

1. String für den Text. Da dieser nur gelesen wird, ist es ein konstanter String.
2. Tabelle mit 26 Elementen für die Statistik.

Aktivitätsdiagramm

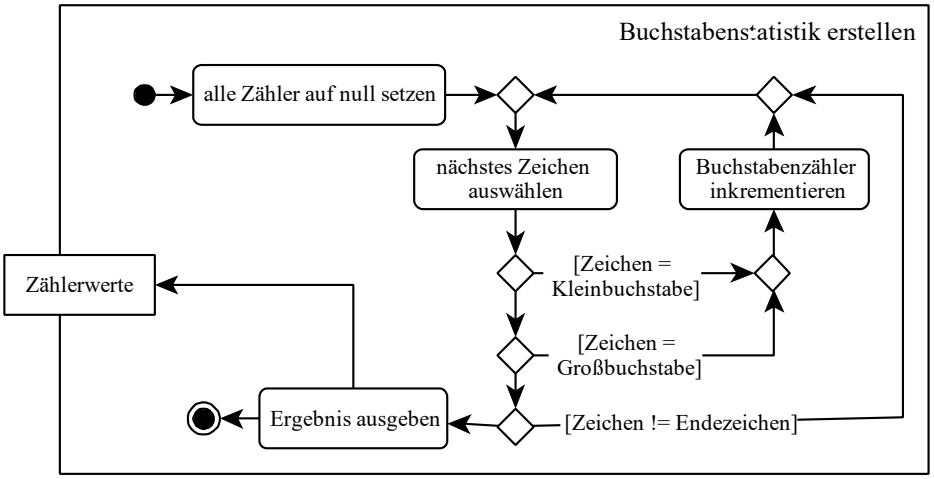

Bild 4: Buchstabenstatistik erstellen

Programm

```
#include <stdio.h>

int main(void) {
    const char text[] =          "\
        Schreiben Sie ein Programm, dass die Haeufigkeit   von Buchstaben in einem Text ermittelt.          \
        Dabei ist es unerheblich, ob es ein Gross- oder Kleinbuchstabe ist. Der Text soll sich im Programm    \
        befinden und nur ASCII-Zeichen des 7-Bit-ASCII-Codes enthalten. Als Ergebnis soll die Haeufigkeit fuer  \
        jeden Buchstaben in Tabellenform ausgegeben werden.
    ";

    uint16_t countLetter[26] = {0u};     // all values zero
    uint16_t index = 0;
    uint8_t cIndex;

    do  {
        if((text[index] >= 'a') && (text[index] <= 'z')) {
          cIndex = text[index] – 'a';                    // get index of letter
          countLetter[cIndex]++;
        }
        else if((text[index] >= 'A') && (text[index] <= 'Z')) {
          cIndex = text[index] - 'A';
          countLetter[cIndex]++;
        }
    } while (text[index++] != '\0');

    /* print result */
    printf("Ergebnis:\n");
    for (index = 0u; index < 26u; index++) {
        printf("%c - %i\n", 'a'+index, countLetter[index]);
    }

    return 0;
}
```

Längere Strings über mehrere Zeilen benötigen ein Zeilen-Fortsetzungszeichen. Ansonsten wurde in diesem Beispiel wieder die Kenntnis über die Codierung der ASCII-Zeichen und die Tatsache, dass Rechner Bitketten verarbeiten, ausgenutzt.

Lösung Aufgabe 6

```
int16_t matrix [8][6];          /* first index: column */
int16_t test;

test = matrix[1][4];            /* index is one less than element number */
```