1.6.A Programmstrukturen/Abweisende Schleife – Arbeitsblatt

Aufgabe 1: Fehlerquote

`(fehlerquote.c)`
Die Fehlerquote eines Produkts liegt bei 30,0%. Jedes Jahr soll sie um ein Zehntel reduziert werden. Ein Programm soll illustrieren, wie die Fehlerquote in den nächsten Jahren absinkt. Bei f=1,0% soll das Programm abbrechen.

Aufgabe 2: Abkühlen eines Schmelzofens durch Materialentnahme

`(schmelzofen.c)`
Bei jedem Öffnen eines Schmelzofens gehen 5% der Energie verloren. Mit Hilfe eines Programms soll herausgefunden werden, wie oft der Ofen nachein- ander geöffnet werden kann, bevor 70% der Energie entwichen sind, so dass eine Warnung ausgegeben werden muss (Formel: $E_x = E_{x-1} \cdot 0,95$).

Hinweise zum programmtechnischen Lösungsweg (*Algorithmus*):

- Bei $m = 0$ und $E = 1,0$ anfangen (Ofen wurde bisher null mal geöffnet, Anfangs-Energie zu 100% vorhanden)
- Neues E ausrechnen und ausgeben
- m erhöhen
- Testen, ob $E = 0,3$ erreicht wurde
- Falls nicht, weitermachen

Aufgabe 3: Entladen eines Kondensators über einen Widerstand

`(c_entladen.c)`
Mit Hilfe eines Programms soll herausgefunden werden, wie lange es dauert, bis ein Kondensator ($C = 100\,\mu\text{F}$), der an einer Betriebsspannung von $U_0 = 350\,\text{V}$ liegt, über einen Widerstand ($R = 100\,\text{k}\Omega$) entladen werden muss, bis die Kondensatorspannung auf einen ungefährlichen Wert von $U_u = 24\,\text{V}$ abgesunken ist.

Formel: $U_c = U_0 \cdot e^{\frac{-t}{\tau}}$ mit $\tau = R \cdot C$

Hinweise zum programmtechnischen Lösungsweg (*Algorithmus*):

- Bei $t = 0$ anfangen
- U_c für t ausrechnen und ausgeben
- t um t_x erhöhen (z.B. um $t_x = 1\,\text{s}$)
- Testen, ob $U_c = 24\,\text{V}$ erreicht wurden
- Falls nicht, weitermachen

Hinweise zur Programmerstellung:

- Die e-Funktion heißt in C `exp()`. Für $y = e^x$ schreibt man also `y=exp(x);` `x` und `y` sind beide vom Typ `double`.
- Für die Funktion `exp()` muss man `<math.h>` einbinden. In derselben Headerdatei sind auch `sin()`, `cos()` und viele andere mathematische Funktionen verfügbar.
- Der GNU-C-Compiler `gcc` bindet diese mathematischen Funktionen standardmäßig *nicht* ein. Um sie zu benutzen, muss der Compiler-Aufruf um die Option `-lm` ergänzt werden (`-l` steht für *library* und `m` für den Dateinamen der Mathe-Funktionsbibliothek):

Terminal
```
schueler@debian964:~$ gcc entladen.c -lm
```

Aufgabe 4: Zahlenumwandlung von Basis 10 nach Basis 2

a) `(zahlwand2.c)`
Um eine Zahl (eingegeben z.B. zur Basis 10) im Dualzahlensystem darstellen zu können, ist eine Folge von Divisionen durch 2 notwendig. Mit jeder Division wird eine weitere Ziffer ermittelt. Die Folge bricht ab, wenn der Rest null ist Schreiben Sie ein entsprechendes Programm! Hinweis: Die Ausgabe des Programmes darf in umgekehrter Reihenfolge stattfinden; mit Tricks wie der Verwendung des Backspace-Zeichens (`\b`) kann man allerdings auch eine normale Ausgabereihenfolge erreichen.

b) `(zahlwand2-10.c)`
Erweitern Sie das Programm aus der vorigen Teilaufgabe so, dass statt der festen Basis 2 (Dualzahlen) eine beliebige Basis zwischen 2 und 10 möglich ist!

c) `(zahlwand2-36.c)`
Erweitern Sie das Programm aus der vorigen Teilaufgabe so, dass jetzt die Umwandlung in eine beliebige Basis zwischen 2 und 36 möglich ist! Ziffern zwischen 10 und 35 sollen durch die Buchstaben A bis Z dargestellt werden.

Aufgabe 5: Größter gemeinsamer Teiler zweier Zahlen

`(ggt.c)`
Der größte gemeinsame Teiler zweier Zahlen wird wie folgt berechnet: Sind die Zahlen a und b gleich, ist a bereits die Lösung. Solange sie nicht gleich sind, kann man einen Schritt weiter kommen, indem man die größere der beiden Zahlen durch die Differenz $|a - b|$ der beiden ersetzt. Hier eine Beispielausgabe des Programms:

Terminal

```
schueler@debian964:~$ a.out
Eingabe a: 35
Eingabe b: 56
a=35    b=21
a=14    b=21
a=14    b=7
a=7     b=7
GGT:7
```

Aufgabe 6: Folgen

Eine Zahlenfolge besteht aus mehreren Zahlen, die aufeinander folgen (daher der Name). Die Reihenfolge der Zahlen liegt fest: 4;3;2;1 ist eine andere Folge als 1;2;3;4. Es gibt endliche Folgen (1;2;3;4) und unendliche Folgen (1;2;3;4;...). Es gibt regelmäßige Folgen (1;2;3;4) und unregelmäßige Folgen (16;33;45;78). Unregelmäßige Folgen können z. B. durch Zufall entstehen oder durch gezielte Eingabe von Messwerten.

a) `(afolge00.c)`
Eine bekannte Art der regelmäßigen Folgen sind arithmetische Folgen. Jede Zahl ist um einen bestimmten Betrag größer (oder kleiner) als die vorhergehende. Ein Beispiel: 12;14;16;18;...

```
1  zahl=12;
2  while(1)
3  {
4      printf("%d\n", zahl);
5      zahl=zahl+2;
6  }
```

Soll die Folge enden, muss man die Ende-Bedingung im Schleifenkopf einbauen. Ein Beispiel: 12;14;16;18.

```
1  zahl=12;
2  while(zahl<20)
3  {
4      printf("%d\n", zahl);
5      zahl=zahl+2;
6  }
```

Erstellen Sie ein Programm `afolge00.c`, das die unendliche Folge 1;2;3;4;... ausgibt. Für die Zahl soll der Datentyp `short int` verwendet werden. Was fällt auf?

b) `(afolge01.c)`
Jetzt sollen Sie die unendliche Folge 1,5;2,5;3,5;4,5;... ausgeben. Für die Zahl soll der Datentyp `float` verwendet werden. Was fällt auf?

c) `(afolge10.c)`
Beim Programm `afolge10.c` soll noch einmal die Folge 1;2;3;4;... ausgegeben werden (wieder mit Datentyp `short int`). Diesmal soll die Folge aber nur so weit laufen (ausgegeben werden), bis die größte Zahl im Datentyp `short int` erreicht ist. Man kann für diese Zahl die Konstante `SHRT_MAX` verwenden, wenn man die Headerdatei `<limits.h>` eingebunden hat.

d) `(gfolge00.c)`
Eine andere bekannte Art sind geometrische Folgen. Hier erhält man eine Zahl, indem man die vorige Zahl mit einem Faktor multipliziert. Ein Beispiel: 3;6;12;24.

```
1  zahl=3;
2  while(zahl<48)
3  {
4      printf("%d\n", zahl);
5      zahl=zahl*2;
6  }
```

Erstellen Sie das Programm `gfolge00.c`, das die unendliche Folge 1;2;4;8;16;... ausgibt. Für die Zahl soll der Datentyp `long int` verwendet werden. Was fällt auf?

e) `(gfolge10.c)` Das Programm soll die unendliche Folge $\frac{1}{2};\frac{1}{4};\frac{1}{8};\frac{1}{16};\ldots$ ausgeben. Was fällt auf?

f) `(gfolge10.c)`
Das Programm soll die unendliche Folge $\frac{1}{2};-\frac{1}{4};\frac{1}{8};-\frac{1}{16};\ldots$ ausgeben.

g) `(mersenne.c)`
Andere Folgen erfordern etwas mehr Berechnung. Bei der Mersenne-Folge erhält man eine Zahl, indem man den Vorgänger verdoppelt und anschließend eins addiert. Erste Zahl ist die Null. Schreiben Sie das Programm `mersenne.c`, das aufhört, sobald 70000 überschritten wird!

h) `(fibo.c)`
Bei den bisherigen Folgen reichte es aus, die vorige Zahl zu kennen, um die aktuelle Zahl zu erhalten. Anders ist es, wenn man die vorletzte Zahl auch noch mit einbeziehen möchte. Ein Beispiel ist die Jacobsthal-Folge, bei der jede Zahl gleich dem Vorgänger plus dem Doppelten des Vor-Vorgängers ist. Die ersten Elemente sind dabei 0 und 1:

```
1  alt=0;
2  zahl=1;
3  while(zahl<200000)
4  {
5      merker=zahl;
6      zahl=zahl+2*alt;
7      alt=merker;
8  }
```

Erstellen Sie das Programm `fibo.c`, das die Fibonacci-Folge ausgibt (Elemente<70000). Bei der Fibonacci-Folge ist jede Zahl gleich der Summe aus Vorgänger und dem Vor-Vorgänger. Die ersten Elemente sind dabei 0 und 1.

Aufgabe 7: Reihen

Jede Zahlenreihe basiert auf einer Zahlenfolge. Wenn man die Zahlen einer Folge nach und nach aufaddiert, erhält man die zugehörige Reihe:

- Die Folge 1;2;3;4 gehört zur Reihe 1;1+2;1+2+3;1+2+3+4 (arithmetische Reihe).
- Die Folge 1;2;4;8 gehört zur Reihe 1;1+2;1+2+4;1+2+4+8 (geometrische Reihe).

Bei manchen Reihen werden die Zahlen im Laufe der Additionen immer größer, bei anderen Reihen bewegen sie sich gegen einen bestimmten Wert (Grenzwert).

Bei der Programmierung unterscheidet sich die Reihe von der Folge dadurch, dass man eine weitere Variable benötigt, in der die Summe abgelegt wird:

```
1  int zahl=1;
2  int summe=0;
3  while(summe<20)
4  {
5      summe=summe+zahl;
6      zahl=zahl+1;
7  }
```

a) `(georeihe00.c)`
Geben Sie die Reihe $1+10+100+1000+\ldots$ aus! Sobald die Summe 1000000 überschritten hat, soll die Reihe beendet werden.

b) `(georeihe10.c)`
Geben Sie die Reihe $1-10+100-1000+\ldots$ aus! Sobald die Summe 1000000 überschritten oder -1000000 unterschritten hat, soll die Reihe beendet werden.

c) `(georeihe20.c)`
Geben Sie die unendliche Reihe $1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots$ aus!

d) `(georeihe30.c)`
Geben Sie die unendliche Reihe $1-\frac{1}{2}+\frac{1}{4}-\frac{1}{8}+\ldots$ aus!

e) `(harmreihe00.c)`
Außer der arithmetischen und der geometrischen Reihe gibt es noch viele andere. Erstellen Sie ein Programm, bei dem Sie die unendliche Reihe $\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\ldots$ ausgeben! Was fällt auf?

f) Zusatzaufgabe `(sinreihe00.c)`
Geben Sie die unendliche Reihe $\frac{1}{1!}-\frac{1}{3!}+\frac{1}{5!}-\frac{1}{7!}+\ldots$ aus!

g) Zusatzaufgabe (`sinreihe10.c`)
Erstellen Sie ein Programm, bei dem Sie nach Eingabe der Zahl `x` durch den Benutzer die unendliche Reihe $\frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$ ausgeben!