Inhaltsverzeichnis

1. Aufgabe: Extremwerte ausgeben

1.1 MinMaxGanz (Extremwerte von Ganzzahltypen ausgeben)

Schreiben Sie ein C-Programm namens `MinMaxGanz` welches die folgende Ausgabe produziert:

```
1   ------------------------------------
2                            +8   CHAR_BIT
3                            +2 MB_LEN_MAX
4                          -128  SCHAR_MIN
5                          +127  SCHAR_MAX
6                           255  UCHAR_MAX
7                          -128   CHAR_MIN
8                          +127   CHAR_MAX
9                        -32768   SHRT_MIN
10                       +32767   SHRT_MAX
11                        65535  USHRT_MAX
12                  -2147483648    INT_MIN
13                  +2147483647    INT_MAX
14                   4294967295   UINT_MAX
15                  -2147483648   LONG_MIN
16                  +2147483647   LONG_MAX
17                   4294967295  ULONG_MAX
18         -9223372036854775808  LLONG_MIN
19         +9223372036854775807  LLONG_MAX
20         18446744073709551615 ULLONG_MAX
21  ------------------------------------
22                             8   CHAR_BIT
23                             2 MB_LEN_MAX
24                          -128  SCHAR_MIN
25                          +127  SCHAR_MAX
26                           255  UCHAR_MAX
27                          -128   CHAR_MIN
28                          +127   CHAR_MAX
29                        -32_768   SHRT_MIN
30                        +32_767   SHRT_MAX
31                         65_535  USHRT_MAX
32                -2_147_483_648    INT_MIN
33                +2_147_483_647    INT_MAX
34                 4_294_967_295   UINT_MAX
35                -2_147_483_648   LONG_MIN
36                +2_147_483_647   LONG_MAX
37                 4_294_967_295  ULONG_MAX
38   -9_223_372_036_854_775_808  LLONG_MIN
39   +9_223_372_036_854_775_807  LLONG_MAX
40   18_446_744_073_709_551_615 ULLONG_MAX
41  ------------------------------------
42  Loesung01: Das war's erstmal!
```

Benützen Sie dazu die in der Kopfdatei `limits.h` definierten Konstanten `CHAR_BIT`, `MB_LEN_MAX`, `SCHAR_MIN`, ... etc. Im ersten Abschnitt (Zeile 2 bis 20) sollen Sie die Werte dieser Konstanten direkt mit dem Befehl `printf` (Kopfdatei `stdio.h`) ausgeben. Für den zweiten Abschnitt (Zeile 22 bis 40) sollen Sie zwei Funktionen mit folgenden Prototypen definieren:

```
43  char * lesbarULL(unsigned long long zahl);
44     // Wandelt zahl in eine entsprechende Folge von Dezimalziffern um
45     // und liefert die Adresse dieser Zeichenfolge. Die Dezimalziffern
46     // sind durch Unterstriche '_' in Dreiergruppen eingeteilt, z.B. so:
47     // 12_345_678, 10_000_000_000 oder 100_000_000_000 etc.
48  char * lesbarSLL(signed long long zahl);
49     // Wie lesbarULL, aber das Ergebnis beginnt mit einem Vorzeichen z.B.
50     // so: +12_345_678, -10_000_000 oder +100_000_000_000 etc.
```

Können Sie beim Ausgeben von `ULLONG_MAX` mit `printf` einen kleinen Unterschied zwischen den Compilern `gcc` und `lcc` feststellen?

1.2 MinMaxBruch (Extremwerte von Gleitpunkttypen ausgeben)

Schreiben Sie ein C-Programm namens `MinMaxBruch`, welches folgende Ausgabe produziert:

```
1   MinMaxBruch: Jetzt geht es los!
2   ------------------------------------------------------------
3           1.17549e-038 FLT_MIN
4           3.40282e+038 FLT_MAX
5           2.22507e-308 DBL_MIN
6           1.79769e+308 DBL_MAX
7           3.3621e-4932 LDBL_MIN
8          1.18973e+4932 LDBL_MAX
9   ------------------------------------------------------------
10  12345678901234567890123456789012345678901234567890123456789012345678901
11  0         1         2         3         4         5
12  ------------------------------------------------------------
13  FLT_MIN
14  0.00000000000000000000000000000000000000117549
15  ------------------------------------------------------------
16  FLT_MAX
17  340282346638528860000000000000000000000.00000
18  ------------------------------------------------------------
19  DBL_MIN
20  0.00000000000000000000000000000000000000000000000000000000000000000000000
21  00000000000000000000000000000000000000000000000000000000000000000000000000
22  00000000000000000000000000000000000000000000000000000000000000000000000000
23  00000000000000000000000000000000000000000000000000000000000000000000000000
24  00000000000000000000000000000000000000000000000000000000000000000000000000
25  00000000000000000000000000000000000000000000000000000000000000000000000000
26  00000000022250738585072014000000000000000000000000000000000000000000000000
27  ------------------------------------------------------------
28  DBL_MAX
29  17976931348623157000000000000000000000000000000000000000000000000000000000
30  00000000000000000000000000000000000000000000000000000000000000000000000000
31  00000000000000000000000000000000000000000000000000000000000000000000000000
32  00000000000000000000000000000000000000000000000000000000000000000000000000
33  00000000000000000000000000000000000000000000000000000000000000000000000000
34  00000000000000000000000000000000000000000000000000000000000000000000000000
35  000000000.00000000000000000000
36  ------------------------------------------------------------
37  MinMaxBruch: Das war's erstmal!
```

Benützen Sie dazu die in der Kopfdatei float.h definierten Konstanten `FLT_MIN`, `FLT_MAX`, `DBL_MIN`, ... etc. Die Zeilen 10 und 11 sind nur eine Art **Lineal**. Sie können als Stringliterale ausgeben werden, etwa so: `printf("12345678901234567890123 ... \n");`.

In den Zeilen 20 bis 26 wird `DBL_MIN` in einer Länge von 330 Zeichen (mit 328 Stellen nach dem Dezimalpunkt) ausgegeben. In den Zeilen 29 bis 35 wird `DBL_MAX` in einer Länge von 330 Zeichen (mit 20 Stellen nach dem Dezimalpunkt) ausgegeben. Die Zeilen 20 bis 26 sollen vom Programm `MinMaxBruch` als **eine** Zeile (der Länge 330) ausgegeben werden (diese Zeile wurde nur für die obige Darstellung "von Hand" in mehrere Zeilen zu je 50 Zeichen umgebrochen). Ganz Entsprechendes gilt für die Zeilen 29 bis 35.

2. Aufgabe: Lies Zahlen oder Namen für Zahlen

2.1 LiesInt (int-Werte oder min, max einlesen)

Schreiben Sie ein C-Programm namens `LiesInt`, welches wiederholt 2 Ganzzahlen von der Standardeingabe einliest, in Werte des Typs `int` umwandelt und ihre Summe ausgibt, bis der Benutzer `"q"` eingibt. Anstelle von Ganzzahlen soll der Benutzer auch die Namen `"min"` und `"max"` (für die Zahlen `INT_MIN` bzw. `INT_MAX`) eingeben können.

Die `main`-Funktion Ihres Programms soll "ganz klein und simpel" sein, alle "wichtigen Arbeiten" sollen in anderen Funktionen erledigt werden. Die Eingaben des Benutzers sollten Sie nicht als int-Werte, sondern als Strings einlesen (mit der Funktion `scanf`, Kopfdatei `stdio.h`). Eine Funktion namens `tolower` wird in der Kopfdatei `ctype.h` deklariert. Mit der Standardfunktion `atoi` (ascii to int, Kopfdatei `stdlib.h`) kann man einen String wie etwa `"-123"` in den `int`-Wert `-123` umwandeln.

Ein Dialog mit dem Programm `LiesInt` soll etwa so aussehen (die Eingaben des Benutzers sind fett hervorgehoben):

```
1   LiesInt: Jetzt geht es los!
2   ------------------------------------------------------------
3   Bitte geben Sie jeweils 2 Ganzzahlen oder min oder max ein
4   (oder q zum Beenden)
5   ------------------------------------------------------------
6           100 +         200 ==         300 | Zwei Zahlen: 25 -12
7            25 +         -12 ==          13 | Zwei Zahlen: 999999999 1
8     999999999 +           1 ==  1000000000 | Zwei Zahlen: max 1
9    2147483647 +           1 == -2147483648 | Zwei Zahlen: 0 min
10            0 + -2147483648 == -2147483648 | Zwei Zahlen: max max
11   2147483647 +  2147483647 ==          -2 | Zwei Zahlen: min min
12  -2147483648 + -2147483648 ==           0 | Zwei Zahlen: q
13  ------------------------------------------------------------
14  LiesInt: Das war's erstmal!
```

2.2 LiesFloat (float-Werte oder min, max, inf, nan etc. einlesen)

Ebenso wie `LiesInt`, aber anstelle von `int`- sollen `float`-Werte eingelesen werden. Ausser Bruchzahlen wie `1.5` oder `-234.56e+5` etc. soll man auch die folgenden Namen für die entsprechenden `float`-Werte eingeben dürfen: `min`, `max`, `inf`, `nan`, `-min`, `-max`, `-inf` und `-nan`. Ein Dialog mit dem Programm `LiesFloat` soll etwa so aussehen:

```
1   LiesFloat: Jetzt geht es los!
2   ------------------------------------------------------------
3   Bitte geben Sie jeweils 2 Bruchzahlen oder min, -min, max,
4   -max, inf, -inf, nan oder -nan ein (oder q zum Beenden).
5   ------------------------------------------------------------
6     1.100000e+00 +   2.200000e+00 ==   3.300000e+00 | Zwei Zahlen: 12.5 3.4
7     1.250000e+01 +   3.400000e+00 ==   1.590000e+01 | Zwei Zahlen: max max
8     3.402823e+38 +   3.402823e+38 ==            Inf | Zwei Zahlen: -max -max
9    -3.402823e+38 +  -3.402823e+38 ==           -Inf | Zwei Zahlen: inf inf
10            Inf +            Inf ==            Inf | Zwei Zahlen: inf -inf
11            Inf +           -Inf ==            NaN | Zwei Zahlen: nan 1
12            NaN +   1.000000e+00 ==            NaN | Zwei Zahlen: q
13  ------------------------------------------------------------
14  LiesFloat: Das war's erstmal!
```

3. Aufgabe: Zahlen in Bittabellen speichern

3.1 BitTab01 (eine "feste" Bittabelle programmieren)

Schreiben Sie ein C-Programm namens `BitTab01`, welches Ganzzahlen (zwischen 0 und 3000) in einer Bittabelle speichert. Die Bittabelle ist eine Reihung von (3001 / 8 sind gleich) 376 Bytes.

Das Programm soll den Benutzer wiederholt nach einem Kommando fragen und das Kommando ausführen, bis der Benutzer ein `q` eingibt. Folgende Kommandos sollen erlaubt sein:

```
15   e123    // Einfuegen (die Zahl 123 in die Bittabelle einfeugen)
16   s45     // Suchen    (die Zahl  45 in der Bittabelle suchen)
17   l678    // Loeschen  (die Zahl 678 in der Bittabelle loeschen)
18   a       // Ausgeben  (alle Zahlen aus der Bittabelle, zum Bildschirm)
19   w       // write     (die Bittabelle in  eine  Datei namens BitTab01.bin)
20   r       // read      (die Bittabelle aus einer Datei namens BitTab01.bin)
```

Ein Dialog mit dem Programm `BitTab01` soll etwa so aussehen können:

```
21  Kommando? e38
22  Kommando? e538
23  Kommando? e17
24  Kommando? a
25  ------------------------------------------------------------
26  Zur Zeit sind folgende Zahlen in der Tabelle:
27     17    38   538
28  Insgesamt 3 Stueck!
29  ------------------------------------------------------------
30  Kommando? w
31  Kommando? l38
32  Kommando? a
33  ------------------------------------------------------------
34  Zur Zeit sind folgende Zahlen in der Tabelle:
35     17   538
36  Insgesamt 2 Stueck!
37  ------------------------------------------------------------
38  Kommando? r
39  Kommando? a
40  ------------------------------------------------------------
41  Zur Zeit sind folgende Zahlen in der Tabelle:
42     17    38   538
43  Insgesamt 3 Stueck!
44  ------------------------------------------------------------
45  Kommando? q
46  BitTab01: Das war's erstmal!
```

3.2 BitTab02 (Bittabellen als Objekte eines Verbundtyps realisieren)

Schreiben Sie zu der folgenden Kopfdatei eine passende Implementierungsdatei `BitTab02.c`:

```
1   // Datei BitTab02.h
2   /* ------------------------------------------------------------------------
3   Kopfdatei zur Implementierungsdatei BitTab02.c.
4   Realisiert BitTabellen, in denen man Ganzzahlen des Typs ganz "sammeln"
5   (d.h. einfuegen, suchen, loeschen) kann.
6
7   Erlaeuterungen zu den wichtigsten Funktionen:
8
9   Funktion      | Was man damit machen kann
10  --------------+---------------------------------------------------------
11  newBitTab02   | Eine neue BitTabelle erzeugen
12  freeBitTab02  | Gibt eine BitTabelle an die Speicherverwaltung zurueck
13  fuegeEin      | Eine Zahl in  eine  BitTabelle einfuegen
14  suche         | Eine Zahl in  einer BitTabelle suchen ("ist sie drin?")
15  loesche       | Eine Zahl aus einer BitTabelle entfernen
16  schreibInDatei| Eine BitTab. in eine Datei schreiben ("persistent machen")
17  liesAusDatei  | Eine BitTab. aus einer Datei lesen
18  ------------------------------------------------------------------------ */
19  #include <stddef.h>  // Typen ptrdiff_t, size_t, wchar_t, Konstante NULL
20
21  #define TEST
22  #undef  TEST
23
24  // In die BitTabellen kann man Zahlen des folgenden Typs ganz einfuegen
25  // (Vorsicht: Wenn man ganz als vorzeichenlosen Typ vereinbart (z.B.
26  // als unsigned int) und beim Erzeugen einer neuen BitTabelle (mit
27  // newBitTab02) als kleinste und/oder groesste Zahl eine negative Zahl
28  // angibt, passieren Fehler, die vom Compiler nicht entdeckt werden!)
29  typedef int ganz;
30  // Zum Typ ganz passendes Umwandlungszeichen fuer printf:
31  #define PG "d"
32
33  // Eine BitTabelle ist ein Verbund des folgenden Typs BitTab02:
34  typedef struct {
35     char * NAME;       // Eine BitTab02 hat einen Namen (z.B. "meineTab03")
36     ganz   MIN_ZAHL;   // Kleinste Zahl, die einfuegbar sein soll
37     ganz   MAX_ZAHL;   // Groesste Zahl, die einfuegbar sein soll
38     ganz   ANZ_ZAHLEN; // Anzahl der einfuegbaren Zahlen (MAX-MIN+1)
39
40     size_t TAB_LEN;    // Laenge der tab-Komponente in Bytes
41
42     char * tab;        // Die eigentliche BitTabelle
43     ganz   anzZahlen;  // Wieviele Zahlen enthaelt diese BitTab02 momentan?
44     ganz   aktZahl;    // Welche Zahl wurde zuletzt bearbeitet?
45     ganz   meldNr;     // Nr einer Fehler- oder Erfolgsmeldung
46     char * meldText;   // Text der Fehler- oder Erfolgsmeldung
47  } BitTab02;
48
49  // Deklarationen der oeffentlichen Funktionen:
50  BitTab02 * newBitTab02(char * name, ganz min_zahl, ganz max_zahl);
51  // Erzeugt eine neue BitTabelle mit dem angegebenen Namen. In diese
52  // Tabelle kann man Ganzzahlen zwischen min_zahl und max_zahl einfuegen
53
54  void freeBitTab02(BitTab02 * bt);
55  // Gibt die BitTab02 *bt an die Speicherverwaltung zurueck (mit free).
56
57  void gibBitTab02Aus    (BitTab02 * bt);
```

```
58  // Gibt ein paar Zeilen mit wichtigen Informationen zu der BitTab02 *bt
59  // aus (aber nicht die Zahlen, die sich momentan in *bt befinden).
60
61  void gibAlleZahlenAus(BitTab02 * bt);
62  // Gibt die Zahlen aus, die sich momentan in der BitTab02 *bt befinden
63  // (Achtung: Moeglicherweise sind das sehr viele!).
64
65  ganz fuegeEin        (BitTab02 * bt, ganz zahl);
66  // Stellt sicher, dass die zahl sich in der BitTab02 *bt befindet.
67  // Liefert ALLES_OK wenn die zahl sich noch nicht in *bt befand,
68  // liefert sonst ERR_FUEGEEINZAHL_SCHON_IN_TAB.
69
70  ganz suche           (BitTab02 * bt, ganz zahl);
71  // Prueft, ob die zahl sich momentan in der BitTab02 *bt befindet.
72  // Liefert entsprechend ZAHL_IST_IN_TAB bzw. ZAHL_IST_NICHT_IN_TAB.
73
74  ganz loesche         (BitTab02 * bt, ganz zahl);
75  // Stellt sicher, dass die zahl sich nicht (mehr) in der BitTab02 *bt be-
76  // findet. Liefert ALLES_OK wenn die zahl sich in *bt befand,
77  // liefert sonst ERR_LOESCHEZAHL_NICHT_IN_TAB.
78
79  ganz schreibInDatei  (BitTab02 * bt);
80  // Schreibt die BitTab *bt in eine Datei. Wenn bt->NAME gleich
81  // "Otto" ist, dann heisst die Datei "BitTab02_Otto.bin".
82  // Liefert ALLES_OK bzw. ERR_BEIM_SCHREIBEN_IN_DATEI.
83
84  ganz liesAusDatei    (BitTab02 * bt);
85  // Liest eine BitTab aus einer Datei nach *bt. Wenn bt->Name gleich
86  // "Otto" ist, wird aus der Datei namens "BitTab02_Otto.bin" gelesen.
87  // Liefert ALLES_OK bzw. ERR_BEIM_LESEN_AUS_DATEI.
88
89  // Die folgenden Fehlernummern werden von den einzelnen Funktionen
90  // als Ergebnis geliefert und in der Komponenten bt->meldNr einer
91  // BitTab02 *bt gespeichert (in der Komponenten  bt->meldText wird
92  // der Name der zugehoerigen Konstanten, z.B. "ALLES_OK" gespeichert):
93  extern const ganz ALLES_OK;
94  extern const ganz ERR_ZAHL_ZU_KLEIN;
95  extern const ganz ERR_ZAHL_ZU_GROSS;
96  extern const ganz ZAHL_IST_IN_TAB;
97  extern const ganz ZAHL_IST_NICHT_IN_TAB;
98  extern const ganz ERR_LOESCHEZAHL_NICHT_IN_TAB;
99  extern const ganz ERR_FUEGEEINZAHL_SCHON_IN_TAB;
100 extern const ganz ERR_BEIM_SCHREIBEN_IN_DATEI;
101 extern const ganz ERR_BEIM_LESEN_AUS_DATEI;
102 // ---------------------------------------------------------------------
```

Schreiben Sie ausserdem ein C-Programm namens `BitTab02Tst`, welches den Modul `BitTab02` (möglichst "vollautomatisch") testet.

4. Aufgabe: Random01 (den Modul Konsole benutzen)

Schreiben Sie ein C-Programm namens `Random01`, welches Buchstaben (`'A'` bis `'Z'` und dann wieder von vorn) zu zufällig gewählten Stellen auf dem Bildschirm ausgibt. Nach jeweils 200 Ausgaben soll die Hintergrundfarbe gewechselt werden.

Die Stelle, zu der eine Ausgabe erfolgt, soll zufällig gewählt werden, aber "in der Nähe" der Stelle der vorigen Ausgabe: null oder eine **Zeile** (rauf oder runter) und null bis drei **Spalten** (nach links oder rechts) entfernt. Nach Ausgabe von 'N' an eine bestimmte Stelle des Bildschirms soll der nächste Buchstabe also zu einer der mit `'O'` gekennzeichneten Stellen (oder an die alte, mit `'N'` gekennzeichnete Stelle) ausgegeben werden:

```
............
...OOOOOOO...
...OOONOOO...
...OOOOOOO...
............
```

Benützen Sie zur Lösung dieser Aufgabe den Modul `Konsole` (bestehend aus den Dateien `Konsole.h` und `Konsole.c`). Zufallszahlen kann man mit den Funktion `srand` und `rand` erzeugen (deklariert in der Kopfdatei `stdlib.h`).

Das fertige Programm soll 4000 Buchstaben ausgeben. Während Sie das Programm noch entwickeln und testen empfiehlt es sich, deutlich weniger Buchstaben auszugeben (z.B. 500 oder 800).

Tip: Programmieren Sie zuerst einen "minimalen Prototypen" des Programms `Random01`. Dieser Prototyp gibt nur **einen** Buchstaben (z.B. `'X'`) aus, und zwar in die Mitte des aktuellen Fensters.

Anmerkung: Mit dieser Aufgabe sollen Sie sich vor allem mit dem Modul `Konsole` vertraut machen, der auch zur Lösung weiterer Aufgaben (`Hexer03` und `Kalk01`) zu verwenden ist.

5. Aufgabe: Daten hexadezimal darstellen

5.1 Hexer01 (den Inhalt beliebiger Speicheradressen ausgeben)

Schreiben Sie ein C-Programm namens `Hexer01`, welches eine kleine Bedienungsanleitung und ein paar Adressen von Variablen ausgibt. Danach kann der Benutzer wiederholt eine Adresse eingeben und bekommt den Inhalt dieser Adresse (8 Bytes) zweimal angezeigt: Einmal hexadezimal (pro Byte zwei Hexziffern, siehe unten Zeile 18, 21 und 24)) und einmal als Zeichen (pro Byte ein Zeichen, siehe unten Zeile 19, 22 und 25). Anstelle von nicht-druckbaren Zeichen wird dabei das Zeichen Punkt `'.'` ausgegeben. Gibt der Benutzer einen leeren String oder ein `'v'` ein, werden ihm die nächsten 8 Bytes angezeigt, gibt er ein `'z'` ein, werden im die vorigen 8 Bytes angezeigt. Gibt er `'q'` ein, beendet sich das Programm `Hexer01`. Ein Dialog mit dem Programm soll etwa so aussehen:

```
1   ----------------------------------------
2   Hexer01, Benutzung:
3   Beenden:                 'q'
4   Diese Hilfe ausgeben:    'h'
5   8 Bytes vorwaerts:       'v' oder Return
6   8 Bytes zurueck:         'z'
7   ---------------------------
8   Oder eine Adresse eingeben,
9   z.B. eine der folgenden:
10  &v1: 0x404094
11   c1: 0x404098
12   p1: 0x4040a0
13  &v2: 0x12ff24
14   c2: 0x12ff1c
15   p2: 0x12ff18
16  ----------------------------------------
17  Eine Adresse (oder 'q', 'h', 'v', 'z' oder nur Return)? 404096
18  0x404096: FF FF 41 42 43 31 32 33
19  0x404096:  .  .  A  B  C  1  2  3
20  Eine Adresse (oder 'q', 'h', 'v', 'z' oder nur Return)?
21  0x40409e: 61 62 FF FF FF FF A0 40
22  0x40409e:  a  b  .  .  .  .  .  @
23  Eine Adresse (oder 'q', 'h', 'v', 'z' oder nur Return)? z
24  0x404096: FF FF 41 42 43 31 32 33
25  0x404096:  .  .  A  B  C  1  2  3
26  Eine Adresse (oder 'q', 'h', 'v', 'z' oder nur Return)? q
27  Hexer01: Das wars erstmal!
```

Tip: Programmieren Sie zuerst einen "minimalen Prototypen", etwa so:

```
28  int main() {
29     char * text = "ABC123abc";
30     char * adr;
31
32     printf("text: %X\n", text);
33     while (1) {
34        printf("Eine Adresse?  ");
35        scanf ("%X", &adr);
36        printf("An der Adresse %X steht %d gleich '%c'\n", adr, *adr, *adr);
37     }
38     return 0;
39  } // main
```

5.2 Hexer03 (den Inhalt einer beliebigen Datei ausgeben)

Schreiben Sie ein C-Programm namens `Hexer03`, welches einen Pfadnamen einer Datei von der Standardeingabe einliest, die Datei einliest und ihren Inhalt in hexadezimaler Darstellung und in zeichenweiser Darstellung zur Standardausgabe ausgibt, etwa so:

```
1   Hexer03: Pfadname einer Datei? Hexer03.c
2   Laenge der Datei: 15768 Bytes
3
4        0: 2F 2F 20 44 61 74 65 69 20 48 65 78 65 72 30 32 //.Datei.Hexer03
5       16: 2E 63 0A 2F 2A 20 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D .c./*.----------
6       32: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ----------------
7       48: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ----------------
8       64: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ----------------
9       80: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 0A 4C --------------.L
10      96: 69 65 73 74 20 76 6F 6E 20 64 65 72 20 53 74 61 iest.von.der.Sta
11     112: 6E 64 61 72 64 65 69 6E 67 61 62 65 20 64 65 6E ndardeingabe.den
12     128: 20 50 66 61 64 6E 61 6D 65 6E 20 65 69 6E 65 72 .Pfadnamen.einer
13     144: 20 44 61 74 65 69 20 44 20 65 69 6E 2C 20 6C 69 .Datei.D.ein,.li
14     160: 65 73 74 20 64 69 65 0A 44 61 74 65 69 20 44 20 est.die.Datei.D.
15     176: 75 6E 64 20 67 69 62 74 20 69 68 72 65 6E 20 49 und.gibt.ihren.I
16     192: 6E 68 61 6C 74 20 69 6E 20 68 65 78 61 64 65 7A nhalt.in.hexadez
17     208: 69 6D 61 6C 65 72 20 44 61 72 73 74 65 6C 6C 75 imaler.Darstellu
18     224: 6E 67 20 75 6E 64 20 69 6E 20 7A 65 69 63 68 65 ng.und.in.zeiche
19     240: 6E 2D 0A 77 65 69 73 65 72 20 44 61 72 73 74 65 n-.weiser.Darste
20     256: 6C 6C 75 6E 67 20 7A 75 72 20 53 74 61 6E 64 61 llung.zur.Standa
21  Seite nach unten
22
23  Tasten: Pfeiltasten rauf/runter (Zeilen), Bild-rauf/runter (Seiten)
24          Pos1 (Dateianfang), Ende (Dateiende)
25          'q' (quit), Return-Taste (letztes Kommando noch mal)
```

Die Zeilen 1 bis 2 und 23 bis 25 sollen immer unverändert bleiben und auch durch weitere Eingaben des Benutzers nicht "nach oben wandern" oder sonstwie verschoben werden. In den Zeilen 4 bis 20 soll immer ein Abschnitt der Datei (pro Zeile 16 Zeichen) angezeigt werden, und zwar in drei Spalten: Spalte 1 enthält eine dezimale Zeilen-Nummer, Spalte 2 die hexamdezimale Darstellung von (maximal) 16 Zeichen Spalte 3 die zeichenweise Darstellung der selben Zeichen. In Spalte 3 werden nicht-druckbare und unsichtbare Zeichen wie Blanks und Tabs durch je einen Punkt '.' dargestellt. Mit Hilfe bestimmter Tasten kann der Benutzer diesen Abschnitt innerhalb der Datei verschieben (wie in der kurzen Bedienungsanleitung in den Zeilen 23 bis 25 erläutert wird). In Zeile 21 steht immer der letzte Befehl, den der Benutzer mit Hilfe der Tasten eingegeben hat. Dieser Befehl wird erneut ausgeführt, wenn der Benutzer auf die Return-Taste drückt.

Auf dem Bildschirm werden also immer (maximal) 17 Zeilen mit je 16 Zeichen der Datei angezeigt. Ein Druck auf die Bild-runter-Taste soll den sichtbaren Abschnitt um 16 Zeilen (und nicht um 17 Zeilen!) verschieben. In der obigen Darstellung würde dadurch die Bildschirmzeile 20 (die mit der Zeichen-Nr. 256: beginnt) nach oben in die Bildschirmzeile 4 verschoben und in den Bildschirmzeilen 5 bis 20 würden 16 "neue" Zeilen erscheinen.

Tip: Definieren Sie im Programm einen Puffer mit Platz für z.B. eine Million Zeichen. Lesen Sie die anzuzeigende Datei mit einem einzigen Lesebefehl in diesen Puffer. Falls die Datei länger ist als der Puffer, sollten Sie einen kleinen Hinweis auf diese Tatsache ausgeben (z.B. am Ende von Zeile 2), dann aber nur den Pufferinhalt (hexadezimal und zeichenweise) auf dem Bildschirm darstellen.

6. Aufgabe: Kalk01 (ein Tischkalkulator)

Schreiben Sie ein C-Programm namens `Kalk01`, welches auf dem Bildschirm einen einfachen Kalkulator zum Rechnen mit Ganzzahlen darstellt, etwa so:

```
1
2
3
4
5
6                                Tischrechner
7
8                             +123 <-- Erster Operand
9                                + <-- Operator
10                            +45 <-- Zweiter Operand
11                            +168 <-- Ergebnis
12
13
14
15
16
17
18
19
20
21
22   Beenden: 'q' oder esc,  Vorzeichen rumdrehen: 'm',  Weitergehen: Return
23   Beide Operanden loeschen (auf 0): 'c'
24   Erlaubte Operatoren: '+', '-', '*' und '/',  Erlaubte Ziffern: '0' bis '9'
25
```

Auf dem Bildschirm soll in jedem Moment eine korrekte Berechnung zu sehen sein (am Anfang z.B. die Berechnung 123 plus 45 ist gleich 168). Der Benutzer kann die beiden Operanden und den Operator (in den Bildschirmzeilen 8 bis 10) verändern, und nach jeder Veränderung wird das Ergebnis (in der Bildschirmzeile 11) so angepaßt, dass wieder eine korrekte Berechnung zu sehen ist.

Alle Eingaben des Benutzers sollen (mit Hilfe des Moduls `Konsole`) "sofort" eingelesen werden und (falls sie erlaubt sind) eine sofortige Reaktion des Programms auslösen. Nicht erlaubte Eingaben sollen eine "Nullreaktion" (d.h. keine Reaktion) auslösen.

Das Programm soll möglichst benutzerfreundlich gestaltet werden. Insbesondere sollen Eingaben des Benutzers möglichst "wohlwollend" interpretiert werden. Beispiel: Wenn der Bildschirmzeiger (Cursor) sich in Zeile 8 (d.h. im ersten Operanden) befindet und der Benutzer einen Operator (z.B. `'*'` oder `'/'` etc.) eingibt, dann soll das nicht als ein Fehler des Benutzers behandelt werden. Statt dessen soll der eingegebene Operator in Zeile 9 angezeigt und der Bildschirmzeiger dorthin bewegt werden.