Übungsserie 3b

Programmierung mit C

Damit für alle die gleichen Regeln gelten, müssen Ihre C-Programme in Standard-C (C11 oder C99) mit dem GCC oder Clang ohne Warnungen kompilieren. Vermeiden Sie insbesondere plattformspezifische Includes, z.B. `windows.h`*. Schalten Sie alle Warnungen Ihres Compilers an um hilfreiche Informationen zu erhalten, Warnungen zu vermeiden und Fehler aufzuspüren. GCC und Clang unterstützen dies beispielsweise mit den Optionen:* `-Wall -Wextra -pedantic -std=c11`*. Bitte beachten Sie eventuell benötigte Bibliotheken zum Linken, z.B.* `-lm` *für das Linken gegen die Mathebibliothek. Eine ausgezeichnete Referenz zu C finden Sie unter: http://en.cppreference.com/w/c.*

Bitte beachten Sie den Abgabezeitpunkt in Moodle. Serien per E-Mail oder verspätete Abgaben können nicht gewertet werden. Die Textaufgaben bitte per PDF mit den Programmieraufgaben zusammen in ein Zip-Archiv bündeln.

Aufgabe 1 (5 Punkte) Erweitern Sie das Matrix-Modul aus der letzten Aufgabenblatt 3a, indem Sie eine Funktion `toStringMatrix` (Signatur vgl. unten) hinzufügen, welche eine beliebige Matrix in eine Zeichenkette (`char*`) schreibt. Dazu soll diese Funktion den nötigen Speicher selbstständig allozieren. Ferner soll für jedes Element der folgende Formatspezifizierer `"%g"` genutzt werden. Alle Elemente einer Matrix Zeile sollen durch ein Leerzeichen von einander getrennt in einer Zeile stehen. (Am Ende einer Zeile darf auch ein Leerzeichen sein.)
Die Schwierigkeit dabei ist den Speicherbereich für die Zeichenkette passend zu allozieren, da man a priori nicht weiss wieviele Zeichen ein `%g` für beliebige double Werte benötigt. Nutzen Sie dazu den Funktionsaufruf `snprintf( NULL, 0, "%g", element )` um die Länge zu bestimmen.

```
/**
 * Constructs and allocates a string (aka char array) where each row is rendered
 * into a separate line, where the elements are separted by a single space.
 *
 * @param m Pointer to the matrix which should be rendered into a string.
 * @param rows Number of rows.
 * @param cols Number of columns.
 * @return Pointer to the string (aka char array).
 */
char* toStringMatrix( Matrix m, size_t rows, size_t cols );
```

Beispiel Output für eine 5×5 Matrix:

```
4.39319e-118 2.34668e+95 -1.08341e-284 1.27306e+81 -2.75487e+22
-1.25391e-133 2.53813e+242 -7.69633e+135 -1.26383e+288 -1.7891e+71
-1.63089e+85 -1.96931e-65 -6.49907e-51 -3.41815e+81 -5.80093e+31
6.31693e+184 -4.40666e+161 -7.79765e-257 -6.00756e-280 -7.50165e-230
8.24975e+134 6.90185e-10 -3.69296e+262 9.83129e+187 1.37848e+196
```

Sie können Ihr Programm mit der Funktion `randomMatrix` exemplarisch testen.

```
// includes needed: inttypes.h stdlib.h time.h
typedef union { double dbl; uint64_t ull; } RndPattern;

double randomDouble() {
  RndPattern result;
  // RAND_MAX is 7FFFFFFF hence only 31 bits randomized, leaving the signum
```

```
7   // always positive instead we take only the last two bytes and paste them
8   // into the memory
9   for( size_t i = 0; i < sizeof( double ) / 2; ++i ) {
10    result.ull = result.ull << 16 | ( rand() % 65536 );
11  }
12  return result.dbl;
13 }
14
15 Matrix randomMatrix( size_t rows, size_t cols ) {
16  srand(time(0));
17  Matrix m = newMatrix( rows, cols );
18  if( m ) {
19    for( size_t i = 0; i < rows; ++i ) {
20      if( m[i] ) {
21        for( size_t j = 0; j < cols; ++j ){
22          m[i][j] = randomDouble();
23        }
24      }
25    }
26  }
27  return m;
28 }
```

Aufgabe 2 (14 Punkte)

a) (3 Punkte) Gegeben sei eine Zahlenfolge in expliziter Bildungsvorschrift:

$$g(n) = \frac{2}{3} \cdot \left(1 - \left(-\frac{1}{2}\right)^n\right), \; n \in \mathbb{N}_0$$

Notieren Sie (mindestens) die ersten fünf Werte der Zahlenfolge. Konstruieren Sie nur auf Grundlage dieser Werte eine *rekursive* Funktion `double recursive(unsigned int n)` in dem C-Modul `folge.c`, welche für alle $n \geq 0$ ein zu $g(n)$ identisches Ergebnis liefert.

b) (5 Punkte) Gegeben ist eine Binärdatei `exercise3b.bin`, welche einige Werte der Zahlenfolge enthält. Erweitern Sie das C-Modul `folge.c` so, dass die Werte aus der Binärdatei gelesen werden können. Gegeben ist die Binärdatei `exercise3b.bin` in Moodle unter der URL: https://moodle2.uni-leipzig.de/mod/resource/view.php?id=524355. Die Binärdatei ist dabei aus einem Header fester Länge und einer Zahlenfolge, den eigentlichen Daten, aufgebaut. Schreiben Sie zunächst eine Funktion `FileHeader *readBinaryHeaderFile(const char *fileName)` die den Header liest und in der Standardausgabe (`stdout`) ausgibt. Dabei genügt es so viele Bytes einzulesen wie das `struct` groß ist. Generell kann beim Lesen der Datei davon ausgegangen werden, dass genügend Daten zur Verfügung stehen.

Die Datenstruktur des Headers ist wie folgt aufgebaut:

```
typedef struct {
    char fileName[60];
    char endian[20]; // either "big endian" or "little endian"
    char dataType[20];
    char sizeOfDataType[10];
    char elementCount[10];
} FileHeader;
```

Beachten sie, dass die Funktion `readBinaryHeaderFile` den eingelesenen Header

auch als `FileHeader`-Zeiger zurückgibt. Schreiben Sie alle Fehlermeldungen, z.B. wenn das Öffnen der Datei fehlschlägt, in die Standardfehlerausgabe (`stderr`).

c) (4 Punkte) Erweitern Sie das C-Modul um die Funktion `double *readBinaryFile(const char *fileName)`. Diese soll mithilfe der Funktion aus Teilaufgabe a) die eigentlichen Daten lesen und als Feld zurückgeben. Nutzen Sie die gegebene Funktion `endianDoubleSwap`, die gegebenenfalls die Byte-Reihenfolge dreht wenn die Endianness der Datei und des Systems nicht übereinstimmt. Die Endian und andere wichtige Informationen sind im Header angegeben und sollen verwendet werden.

Hinweis: Hilfreich könnte die Funktion `fseek` sein.

Gegebene Funktionen zum Testen der Endianness des Systems und zum Drehen der Byte-Reihenfolge:

```c
// includes needed: stdlib.h string.h
typedef enum { bigEndian, littleEndian } Endianness;

double doubleSwap(const double value)
{
    union {
        double value;
        unsigned char byte[8];
    } endian1, endian2;

    endian1.value = value;

    for (size_t i = 0; i < 8; ++i) {
        endian2.byte[i] = endian1.byte[7 - i];
    }

    return endian2.value;
}

Endianness systemEndian()
{
    int x = 1;

    if (*((char *)&x) == 1) {
        return littleEndian;
    }
    return bigEndian;
}

double *endianDoubleSwap(double *result, const char *endian,
                         const int elementCount)
{
    Endianness fileEndian =
        strcmp("big endian", endian) == 0 ? bigEndian : littleEndian;

    if (systemEndian() != fileEndian) {
        for (int i = 0; i < elementCount; ++i, ++result) {
            *result = doubleSwap(*result);
            // printf("%2zu %.23f\n", i, *result);
        }
        result -= elementCount;
    }

    return result;
}
```

d) (2 Punkte) Schreiben Sie eine Funktion `int equalTest(const double *content,`

`const size_t len)`, welche die ersten 50 Werte aus der Binärdatei mit den Ergebnissen der Funktion `recursive` aus Teilaufgabe **a)** vergleicht.