MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller, Dr. K. Fuerlinger

Systempraktikum

Einführung in die C-Programmierung

Hinweise

Schreiben Sie zu jedem Ihrer Programme stets ein Makefile, das Ihre Programme übersetzt. Als Referenzumgebung zählt der CIP-Pool, alle Ihre Programme müssen hier übersetzbar und ausführbar sein. Eine Beispielimplementierung zu den Aufgaben finden Sie unter `http://www.nm.ifi.lmu.de/sysprak`. Beachten Sie hierbei, dass es sich nicht immer um die effizienteste und / oder eleganteste Variante der Implementierung handelt. Ziel der Beispielimplementierung ist vielmehr Ihnen als Hilfestellung und zum Verständnis zu dienen.

Wenn Sie Probleme bei der Implementierung der Ubungsaufgaben haben, so steht Ihnen beginnend mit Montag, 25.10.2021 bis einschließlich Freitag, 12.11.2021 ein Tutor zur Hilfestellung bereit. Die aktuelle Einteilung der Tutor Termine und die Online Zugangsmoeglichkeiten finden Sie auf der Homepage der Veranstaltung unter `https://www.nm.ifi.lmu.de/sysprak`.

Anleitungen und Hilfestellungen zu C finden Sie in großen Mengen im Internet und / oder der Bibliothek. Die Angaben zu Buchkapiteln auf den nachfolgenden Seiten beziehen sich auf folgende frei verfügbare Quellen:

- Buch 1: C von A bis Z, Jürgen Wolf, Galileo Computing
  verfügbar als *OpenBook* unter: `http://openbook.galileocomputing.de/c_von_a_bis_z/`
- Buch 2: Linux-Unix-Programmierung, Jürgen Wolf, Galileo Computing
  verfügbar als *OpenBook* unter: `http://openbook.galileocomputing.de/linux_unix_programmierung/`

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller, Dr. K. Fuerlinger

Systempraktikum — Aufgabenblock 1

Aufgabe 1.1: *temperaturUmrechner.c*

Schreiben Sie ein Programm, das Temperaturangaben zwischen verschiedenen Einheiten konvertieren kann. Als Quell- und Zieleinheit sollen jeweils Grad Celsius, Grad Delisle, Grad Fahrenheit, Kelvin und Grad Rankine unterstützt werden. Sowohl die Quell- und Zieleinheit als auch der zu konvertierende Temperaturwert soll als Kommandozeilenparameter übergeben werden. Denken Sie an eine geeignete Fehlerbehandlung, und implementieren Sie auch eine Hilfefunktion, die die Benutzung Ihres Programm im Falle einer nicht korrekten Bedienung erörtert.

Hinweise:

- Um Kommandozeilenparameter einfach einlesen zu können, schauen Sie sich die Funktion `getopt(...)` an.
- Überlegen Sie, wie Sie die Aufgabenstellung möglichst einfach und um weitere Temperaturskalen leicht erweiterbar implementieren können.
- Zur Information: Umrechnung von Temperaturskalen

von → / nach ↓	Celsius $[^\circ C]$	Delisle $[^\circ De]$	Fahrenheit $[^\circ F]$	Kelvin $[K]$	Rankine $[^\circ Ra]$
$[^\circ C]$	$= T_C$	$= 100 - T_{De} \cdot \frac{2}{3}$	$= (T_F - 32) \cdot \frac{5}{9}$	$= T_K - 273.15$	$= T_{Ra} \cdot \frac{5}{9} - 273.15$
$[^\circ De]$	$= (100 - T_C) \cdot 1.5$	$= T_{De}$	$= (212 - T_F) \cdot \frac{5}{6}$	$= (373.15 - T_K) \cdot 1.5$	$= (671.67 - T_{Ra}) \cdot \frac{5}{6}$
$[^\circ F]$	$= T_C \cdot 1.8 + 32$	$= 212 - T_{De} \cdot 1.2$	$= T_F$	$= T_K \cdot 1.8 - 459.67$	$= T_{Ra} - 459.67$
$[K]$	$= T_C + 273.15$	$= 373.15 - T_{De} \cdot \frac{2}{3}$	$= (T_F + 459.67) \cdot \frac{5}{9}$	$= T_K$	$= T_{Ra} \cdot \frac{5}{9}$
$[^\circ Ra]$	$= T_C \cdot 1.8 + 491.67$	$= 671.67 - T_{De} \cdot 1.2$	$= T_F + 459.67$	$= T_K \cdot 1.8$	$= T_{Ra}$

Inhalte, die dieses Beispiel vermitteln soll:

- Basisdatentypen (Buch 1: 5)
- Formatierte Ein- und Ausgabe (Buch 1: 4.2)
- Kontrollstrukturen (Bedingungen, Schleifen) (Buch 1: 8)
- Behandlung von Kommandozeilenparametern / Argumenten
- Fehlerbehandlungen, auch durch fehlerhaftes Nutzerverhalten

Aufgabe 1.2: *matrixMult.c*

Sie möchten zwei $n \times n$-Matrizen einer beliebigen, aber festen (apriori bekannten) Größe miteinander multiplizieren. Erstellen Sie ein Programm, das zwei $n \times n$-Matrizen mit Zufallszahlen des Typs `int` zwischen 0 und 9 initialisiert und miteinander multipliziert. Sowohl die beiden erzeugten Matrizen als auch das Ergebnis der Multiplikation sollen abschließend in einem menschenlesbaren Format auf der Standardausgabe ausgegeben werden.

Inhalte, die dieses Beispiel vermitteln soll:

- Zufallszahlen und ihre Erzeugung (Buch 1: 20.4.4)
- Formatierte Ein- und Ausgabe (Buch 1: 4.2)
- Menschenlesbare Aufbereitung von Informationen
- Mehrdimensionale Felder / Arrays (Buch 1: 11)

Aufgabe 1.3: *findMaxOfFloats.c*

Gegeben ist eine Datei, die beliebig viele Zahlen des Typs `float` enthält, wobei je Zeile genau eine Zahl enthalten ist. Schreiben Sie ein Programm, das eine solche Datei öffnet und die darin enthaltenen Zahlen in einem Feld / Array speichert. Der Dateiname der zu öffnenden Datei soll dabei entweder als Kommandozeilenparameter übergeben werden können, oder, falls kein Kommandozeilenparameter übergeben wurde, im Programmverlauf abgefragt werden. Bestimmen Sie anschließend das Maximum der gelesenen Zahlen und geben Sie es bis auf zwei Stellen nach dem Komma genau auf der Standardausgabe aus.
Hinweis: Sie finden eine Datei mit 10.000 Zufallszahlen zu Testzwecken im Umfang der Beispielimplementierung enthalten.

Inhalte, die dieses Beispiel vermitteln soll:

- Speicherverwaltung
- Felder / Arrays (Buch 1: 11)
- Dateiein- und -ausgabe (Buch 1: 16)
- Formatierte Ein- und Ausgabe (Buch 1: 4.2)
- Kontrollstrukturen (Bedingungen, Schleifen) (Buch 1: 8)

Aufgabe 1.4: *gameOfLife.c*

Implementieren Sie das *game of life*. Die Größe des Spielfeldes soll 16×16 Felder betragen, optional soll der Anwender per Kommandozeilenparameter eine alternative Größe des Spielfeldes angeben können. Die initiale Belegung des Spielfeldes mit lebenden Zellen soll zufällig geschehen, wobei obligatorisch die Dichte der Belegung des Spielfeldes mit lebenden Zellen per Kommandozeilenparameter vom Nutzer angegeben werden muss. Geben Sie die erzeugte Population und jeweils mit einer Sekunde Verzögerung die Population der nächsten Spielrunde auf der Standardausgabe aus. Stagniert eine Population, d. h. es gibt von einer Runde zur nächsten keine Änderung, soll das Programm terminieren.

Hinweis: Regeln des *game of life*

- Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren.
- Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an Einsamkeit.
- Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Folgegeneration lebend.
- Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Überbevölkerung.
- Hinweis: alle anderen Zellen leben nicht

Inhalte, die dieses Beispiel vermitteln soll:

- Verwendung von Zufallszahlen
- Formatierte Ein- und Ausgabe (Buch 1: 4.2) sowie menschenlesbare Aufbereitung von Informationen
- Kommandozeilenparameter (Buch 1: 13)
- Dateiein- und -ausgabe (Buch 1: 16)
- Felder / Arrays (auch mehrdimensional) (Buch 1: 11)
- Zeiger / Pointer (Buch 1: 12)
- Speicherverwaltung

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller, Dr. K. Fuerlinger

Systempraktikum — Aufgabenblock 2

Aufgabe 2.5: *zahlenraten.c*

In diesem Beispiel soll das „Spiel“ Zahlenraten implementiert werden.

1. Der Computer erzeugt eine Zufallszahl zwischen 1 und 100, die es zu erraten gilt. Die Spielerin wird solange aufgefordert eine Zahl einzugeben (zu raten), bis die Zahl erraten wurde. Nach jeder Eingabe gibt der Computer an, ob die Zufallszahl kleiner, größer oder gleich der geratenen Zahl ist. Das Spiel ist beendet, sobald die Zahl erraten wurde. Die Anzahl der Rateversuche wird mitgezählt und am Spielende ausgegeben. Nach jedem Spiel wird die Spielerin gefragt, ob noch eine Runde gespielt werden soll. Wird diese Abfrage verneint, wird das Programm beendet.
2. Das Zahlenraten soll nun um eine Bestenliste (der besten 10), die in einer Datei abgespeichert ist, erweitert werden. Die Einträge in der Bestenliste sind nach der Anzahl der Rateversuche geordnet. Nummer 1 in der Bestenliste hat die wenigsten Versuche gebraucht, um eine Zahl korrekt zu raten. Bei Gleichstand wird der alte Eintrag nach unten verschoben (der neue Eintrag vor dem älteren eingefügt). Um einen Überblick über die abgeschlossenen Spiele zu haben, wird in eine Logdatei die jeweils zu erratende Zahl und die Anzahl der benötigten Rateversuche protokolliert.

Inhalte, die dieses Beispiel vermitteln soll:

- Basisdatentypen (Buch 1: 5)
- Formatierte Ein- und Ausgabe (Buch 1: 4.2)
- Dateiein- und -ausgabe (Buch 1: 16)
- Felder / Arrays (Buch 1: 11)
- Zeiger / Pointer (Buch 1: 12)
- Kontrollstrukturen (Bedingungen, Schleifen) (Buch 1: 8)

Aufgabe 2.6: *complex.c*

In dieser Aufgabe soll eine Bibliothek geschrieben werden, die das Rechnen mit komplexen Zahlen ermöglicht. Die Bibliothek soll für die „Grundrechenarten“ und die formatierte Ausgabe einer komplexen Zahl Methoden bereitstellen. Schreiben Sie eine solche Bibliothek und binden Sie sie in ein Testprogramm, das alle Funktionen überprüft, über ein Headerfile ein.

Hinweis: Eine komplexe Zahl x ist in ihrer algebraischen Form darstellbar als $x = a + bi$, wobei $x \in \mathbb{C}$ und $a, b \in \mathbb{R}$. a wird Realteil und b Imaginärteil der komplexen Zahl genannt. Für komplexe Zahlen (hier $x = a + bi$ und $y = c + di$) gelten folgende Rechenregeln:

- Addition:

  $z = (a + bi) + (c + di) = (a + c) + (b + d)i$

- Subtraktion:

  $z = (a + bi) - (c + di) = (a - c) + (b - d)i$

- Multiplikation:

  $z = (a + bi)(c + di) = (ac - bd) + (ad + bc)i$

- Division ($y = c + di \neq 0$):

  $z = \frac{a+bi}{c+di} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$

  Hinweis: Vor einer Division muss überprüft werden, ob der Nenner gleich 0 ist.

Inhalte, die dieses Beispiel vermitteln soll:

- Modularisierung und Headerfiles
- Erstellen eigener Bibliotheken

Aufgabe 2.7: *matrixBib.c*

Das Ziel dieser Aufgabe ist es eine Bibliothek zu schreiben, die Funktionen zum Rechnen mit Matrizen bereitstellt. Implementieren Sie die Bibliothek auf Basis des nachfolgenden Headerfiles `matrixBib.h`. Schreiben Sie ebenfalls ein Testprogramm, das alle Funktionen Ihrer Bibliothek nutzt und für Ihre Tests herangezogen werden kann.

Inhalte, die dieses Beispiel vermitteln soll:

- Zusammenfassung und Wiederholung der bisher erlernten Inhalte
- Präzision und Ungenauigkeiten bei der Datenverarbeitung:
  Wenn Sie zwei unterschiedliche Verfahren zur Berechnung der Determinanten implementieren, können Sie die ermittelten Determinanten vergleichsweise großer Matrizen miteinander vergleichen. Sie werden feststellen, dass durch Rundungsfehler und Ungenauigkeiten bei der Zahlendarstellung die Ergebnisse, die durch Ihre Algorithmen ermittelt wurden, schnell differieren können.

matrixBib.h

```c
#include <float.h>
#include <stdbool.h>

typedef struct {
  double *eintraege;
  unsigned int breite;
  unsigned int hoehe;
} matrix;

/*
Initialisiert eine neue Matrix:
- reserviert lediglich den notwendigen Speicher
*/
matrix initMatrix(unsigned int breite, unsigned int hoehe);

/*
Initialisiert eine neue Matrix:
- reserviert den notwendigen Speicher
- befuellt die Matrix mit 0
*/
matrix initMatrixZero(unsigned int breite, unsigned int hoehe);

/*
Initialisiert eine neue Matrix:
- reserviert den notwendigen Speicher
- befuellt die Matrix mit Zufallszaheln
*/
matrix initMatrixRand(unsigned int breite, unsigned int hoehe);

```

```
/*
Kopiert eine Matrix und gibt die Kopie zurueck
*/
matrix copyMatrix(matrix toCopy);

/*
"Zerstoert" eine Matrix
- gibt reservierten Speicher wieder frei
- setzt alle Werte auf NULL bzw. "0"
*/
void freeMatrix(matrix toDestroy);

/*
Gibt den Eintrag der Matrix an der Stelle (xPos, yPos) zurueck, DBL_MAX im Fehlerfall
*/
double getEntryAt(matrix a, unsigned int xPos, unsigned int yPos);

/*
Setzt den Eintrag der Matrix an der Stelle (xPos, yPos)
Rueckgabe: true, false im Fehlerfall
*/
bool setEntryAt(matrix a, unsigned int xPos, unsigned int yPos, double value);

/*
Gibt eine Matrix auf der Kommandozeile aus
*/
void prettyPrint(matrix a);

/*
Addiert zwei Matrizen
Rueckgabe:
- Ergebnis der Addition in neu erzeugter Matrix
- Matrix der Groesse "0" im Fehlerfall
*/
matrix addMatrix(matrix a, matrix b);

/*
Subtrahiert zwei Matrizen:
Rueckgabe: "a - b"
- Ergebnis der Subtraktion in neu erzeugter Matrix
- Matrix der Groesse "0" im Fehlerfall
*/
matrix subMatrix(matrix a, matrix b);

/*
Multipliziert zwei Matrizen:
Rueckgabe: "a * b"
- Ergebnis der Multiplikation in neu erzeugter Matrix
- Matrix der Groesse "0" im Fehlerfall
*/
matrix multMatrix(matrix a, matrix b);

/*
Transponiert eine Matrix:
Rueckgabe: "a^T"
*/
matrix transposeMatrix(matrix a);

/*
Gibt die Determinante der Matrix a zurueck, DBL_MAX im Fehlerfall
*/
double determinante(matrix a); // ein einfacher Algorithmus reicht fuer kleine Matrizen
double detQuick(matrix a);     // wer moechte kann ein effizientes Verfahren implementieren
```

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller, Dr. K. Fuerlinger

Systempraktikum — Aufgabenblock 3

Aufgabe 3.8: *checkprime.c*

Große Primzahlen sind vor allem in modernen kryptographischen Verfahren unerläßlich. Die Basis für z. B. das RSA-Verfahren bilden zwei (möglichst große, zufällig gewählte) Primzahlen.

1. Schreiben Sie ein Programm, das testet, ob eine gegebene Zahl p vom Typ `unsigned long long` prim ist. Als einfaches Testverfahren können Sie prüfen, ob es mindestens eine Zahl $x \in [2; \sqrt{p}]$ gibt, durch die p ohne Rest teilbar ist. Es ist hinreichend, wenn Sie für x die Werte 2 und alle ungeraden Zahlen des Intervalls prüfen. Nach Belieben kann auch ein anderer (performanterer) Test gewählt werden.
2. Erweitern Sie Ihr Testverfahren dahingehend, dass die Kandidatenmenge für x in zwei gleichgroße Teilmengen aufgeteilt wird, die parallel in zwei Prozessen verarbeitet werden. Die Interprozesskommunikation soll dabei über eine Pipe geschehen.

Inhalte, die dieses Beispiel vermitteln soll:

- Erzeugung von Prozessen (Buch 2: 7.8)
- Kommunikation über Pipes (Buch 2: 9.3)

Aufgabe 3.9: *checkPrimeMultiThreaded.c*

Erstellen Sie ein zweites Programm zum Primzahlentest, das das vorherige Testverfahren in beliebig vielen Threads durchführt. Zur Kommunikation können Sie hier auf Pipes verzichten. Um wirklich große Zahlen testen zu können soll die Bibliothek GMP (*The GNU Multiple Precision Arithmetic Library*, `http://gmplib.org/`) verwendet werden.

Inhalte, die dieses Beispiel vermitteln soll:

- Threads (Buch 2: 10)
- Gemeinsam genutzte Speicher- und Adressräume
- Arbeiten mit externen Bibliotheken

Aufgabe 3.10: *chat.c*

Schreiben Sie ein „Chat“-Programm, das es ermöglicht zwischen zwei Teilnehmern abwechselnd Nachrichten über ein Verbindungsnetz auszutauschen.

- Das Programm soll nach dem *Client-Server-Prinzip* gestaltet sein. Das heißt, dass eine Instanz (der *Server*) auf eine eingehende Verbindung wartet und ein *Client* die Verbindung zum wartenden Server herstellt. Ein Aufruf des Programmes ohne Kommandozeilenparameter soll einen Server starten. Ein Aufruf des Programmes mit einer IP-Adresse als Kommandozeilenparameter soll einen Client starten, der sich zu einem (schon gestarteten) Server verbindet.

- Der TCP-Port 4711 soll zur Kommunikation verwendet werden.
- Der Chat soll als „Frage-Antwort-Spiel“ organisiert sein. Auf eine Nachricht des Servers kommt immer eine Nachricht des Clients und umgekehrt.
- Der Einfachheit halber soll die Nachrichtenlänge auf ein bestimmtes Maximum (z. B. 256 Zeichen) begrenzt sein.
- Mit einer bestimmten Zeichenkette muss das Programm beendet werden können (z. B. `quit`).

Inhalte, die dieses Beispiel vermitteln soll:

- Sockets (Buch 2: 11) und einfache (eigene) Protokolle
- Client-Server-Prinzip