Heinrich Heine Universität Düsseldorf

Hardwarenahe Programmierung - Übungsblatt 4

Auf diesem Übungsblatt geht es um Kommandozeilenparameter, Lesen aus Dateien und Schreiben in Dateien.

- Hinweise:
  - Durch das Ändern einer Dateiendung ändert sich der Inhalt einer Datei **nicht**. Tatsächlich sind Dateiendungen unter Linux oft egal.
  - Testen Sie Ihre Programme *zusätzlich* mit `valgrind`. Die Tests übernehmen diese Arbeit nicht für Sie.

  Termine für Einzelgespräche:
  - Mittwoch, der 3. November, von 12:30 Uhr bis 16:00 Uhr.
  - Donnerstag, der 4. November, von 10:30 Uhr bis 14:00 Uhr.
  - Freitag, der 5. November, von 8:30 Uhr bis 12:00 Uhr.

Vorbereitungsaufgabe 1 Taschenrechner

In `calculator/calculator.c` finden Sie ein Programm, das einen mathematischen Operator und eine Reihe von Zahlen als Kommandozeilenparameter entgegen nimmt, den Operator auf die Zahlen anwendet und das Ergebnis auf fünf Stellen gerundet ausgibt.

Eingaben, die keine gültigen Zahlen sind, werden dabei übersprungen.

Bisher beherrscht das Programm den Operator `*`. Der Aufruf `./calculator '*' 2 3 4` würde also beispielsweise das Produkt $2 \cdot 3 \cdot 4 = 24$ berechnen.

Erweitern Sie das Programm, sodass es zusätzlich auch noch den den Operator `+` beherrscht. Der Aufruf `./calculator '+' 2 3 3.5` sollte dann `8.50000` ausgeben.

Sie können Ihr Programm mit dem Skript `test.sh` testen.

Pflichtaufgabe 2 Crypto-Währungsrechner

Implementieren Sie in der Datei `crypto/crypto.c` ein Programm, dass über die Kommandozeile verschiedene Crypto-Währungen entgegen nimmt, zu Euro umrechnet, aufsummiert und auf zwei Stellen gerundet ausgibt. Der Einfachheit halber werden Punkte anstatt Kommas als Trennzeichen verwendet.

Es sollen die folgenden Optionen mit zugehörigen Wechselkursen verarbeitet werden können wobei `<zahl>` jeweils durch eine eingegebene Gleitkommazahl (`double`) zu ersetzen ist:

- `--bitcoin <zahl>` oder `--btc <zahl>` mit 52586.02 € je Bitcoin
- `--dogecoin <zahl>` oder `--doge <zahl>` mit 0.27 € je Dogecoin
- `--ethereum <zahl>` oder `--eth <zahl>` mit 3590.12 € je Einheit
- `--filecoin <zahl>` oder `--fil <zahl>` mit 49.74 € je Filecoin

- `--monero <zahl>` oder `--xmr <zahl>` mit 227.73 € je Einheit

Zusätzlich zu der langen Optionsform soll es noch die zugehörigen Kurzformen bestehend aus den Anfangsbuchstaben der langen Optionsformen mit nur einem Bindestrich geben. Zum Beispiel sollte man statt `--xmr <zahl>` auch `-x <zahl>` schreiben können.

Beispielaufrufe und -ausgaben

```
$ ./crypto --bitcoin 1                          //"Summiert" eine Bitcoin auf.
52586.02                                        //Eine Bitcoin kostet 52586.02 €.

$ ./crypto --bitcoin 1 --btc 0.5 -b 0.5         //Summiert 1 + 0.5 + 0.5 = 2 Bitcoin.
105172.04                                       //Zwei Bitcoin kosten 105172.04 €.

$ ./crypto -b 0.5 -b 0.5                        //Summiert 1 + 1 = 2 Bitcoin.
105172.04

$ ./crypto --dogecoin 3.50 --bitcoin 0.001 //Summiert 3.5 Dogecoin u. 0.001 Bitcoin.
53.53

$ ./crypto                                      //Summiert nichts.
0.00
```

Fehlerfälle

In einem Fehlerfall soll sich das Programm beenden und eine aussagekräftige Fehlermeldung auf der Standardfehlerausgabe ausgegeben. Zusätzlich soll das Programm folgende Fehlercodes zurückgeben:

- 1 bei einer ungültigen Anzahl an Parametern.
- 2 bei einer unbekannten Option.
- 3 bei einer ungültigen Zahl.

Falls mehrere Fehler gleichzeitig auftreten sollten dürfen Sie sich den Fehler aussuchen, den Sie behandeln möchten.

Prüfen Sie Ihr Programm mit dem Skript `test.sh`

Pflichtaufgabe 3 Datei Ein- und Ausgabe

In dieser Aufgabe sollen sie in `container/containers.c` ein Programm für die Planung der Lagerung von Containern an einem Seehafen schreiben.

Ihr Programm bekommt die folgenden zwei Kommdandozeilenparameter:

1. Eine Datei mit einer Liste an Containerbezeichnungen und Lager IDs
2. Eine Datei mit einer Liste an Lager-IDs und Lagerbezeichnungen. In der ersten Zeile dieser Datei steht die Anzahl der Paare.

Statt maximal einem Dateinamen kann auch `--` angegeben werden. Dann soll diese Eingabe über die Standardeingabe erfolgen. Fehlerhafte Kommandozeilenparameter müssen nicht überprüft werden.

Ihre Aufgabe ist es die Container auf die Lager aufzuteilen. Dabei sollen Sie folgendes beachten:

- Ordnen Sie mit Hilfe der IDs jedem Container einem Lager zu und schreiben Sie die Container zeilenweise in die entsprechenden Dateien.

- Die Datei für ein Lager hat den selben Namen wie die Lagerbezeichnung
- Die erstellten Lagerdateien sollen im Unterordner `output` liegen.
- Wenn es nicht möglich ist eine ID zuzuordnen geben Sie den Container mit einer entprechenden Meldung auf der **Standardfehlerausgabe** aus. Nutzen Sie hierfür den vorgegebenen Formatstring `UNKNOWN_ID_STRING` aus `container/containers.h`. Ihr Programm soll danach aber normal weiterlaufen, am Ende jedoch `EXIT_MISSING_WAREHOUSE` zurückgeben.
- Sämtliche Fehlermeldungen müssen über die Standardfehlerausgabe ausgegeben werden.
- Die Reihenfolge der Container ist beizubehalten.
- Auch für leere Lager müssen Lager-Dateien erstellt werden.

Beispiel: `./containers container/example warehouse/example`

Der erste Container `ax367412ef` in der Datei `container/example` soll in das Lager mit ID 2.

Dem Lager mit ID 2 ist in `warehouse/example` die Bezeichnung `Busan` zugeordnet.

`container/example`

```
ax367412ef 2
bx64c6e95e 1
cxa7c839b 2
dx3b501308 7
ex1d9df296 2
fx9823d880 6
gx9913c3a5 7
hx8d14db84 2
```

`warehouse/example`

```
3       // Anzahl steht in erster Zeile
1 Rotterdam
2 Busan
7 Hongkong
```

`output/Busan`

```
ax367412ef
cxa7c839b
ex1d9df296
hx8d14db84
```

`output/HongKong`

```
dx3b501308
gx9913c3a5
```

`output/Rotterdam`

```
bx64c6e95e
```

Standardfehlerausgabe

```
ERROR: Harbour ID 6 unknown. Container fx9823d880 could not be assigned!
```

Hinweise:

- Es steht immer genau ein Container mit ID pro Zeile in der Eingabedatei. Genauso immer ein Lager mit ID pro Zeile. Das Format muss nicht überprüft werden.
- Es ist nicht nötig zwei verschiedene Funktionen zu schreiben, um in eine Datei und die Standardausgabe zu schreiben. Wenn Sie geschickt sind kann sämtliche Fallunterscheidung zwischen Datei und Standardausgabe beim öffnen der Dateien geschehen. Schauen Sie sich den Datentyp von `stdin`, `stderr` und `stdout` welche in `stdio.h` definiert sind an.
- Falls Sie die Standardausgabe mit `fclose(stdout)` schließen beachten Sie, dass danach das Verhalten von `printf` undefiniert ist. Bei `stdin` und `scanf` verhält es sich ähnlich.
- Die Bezeichnungen von Containern und Lagern sind maximal 100 Zeichen lang.
- Alle Eingabedateien könnten (bis auf die Lageranzahl) leer sein.
- Für die Anzahl der Container oder Lager gibt es kein Maximum. Verlassen Sie sich nicht auf die maximale Anzahl in den Tests. Wir testen Ihre Abgabe möglicherweise mit größeren Zahlen.
- Testen Sie ihr Programm mit dem Skript `test.sh`. Dies sollte unter fünf Minuten dauern.
- Achten Sie darauf, bei der Abgabe das Datei-Limit nicht zu überschreiten.

Pflichtaufgabe 4 Binärdateien

In dieser Aufgabe geht es darum, Binärdateien zu lesen. Sie erhalten einige binäre Bild-Dateien mit folgendem Inhalt:

- Eine Bildbreite, gegeben als 8-Byte vorzeichenloser Integer.
- Eine Bildhöhe als ein 4-Byte vorzeichenloser Integer.
- Eine Bildtiefe (die Anzahl der Farbkanäle eines Bildes, zum Beispiel 3 bei RGB-Bildern) als 2-Byte vorzeichenloser Integer.
- Eine zusammenhängende Folge von Bildpixeln. Ein Pixel besteht aus so vielen Bytes wie es Farbkanäle gibt. Beispielsweise besteht ein blauer RGB-Pixel aus den drei Bytes $(0, 0, 255)$. Die Pixel sind Zeilenweise gespeichert.

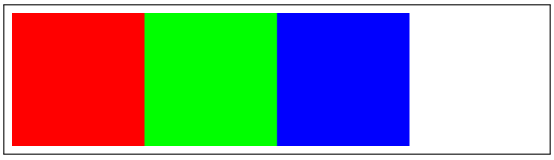

Abbildung 1: Ein Bild mit Breite 4, Höhe 1 und Tiefe 3 bestehend aus einem roten, grünen, blauen und weißen Pixel. Die Datei `binary/example` entspricht dem Bild `binary/example.bmp`.

```
04 00 00 00 00 00 00 00       // Breite = 4
01 00 00 00                   // Höhe = 1
03 00                         // Tiefe = 3
ff 00 00                      // Ein rotes Pixel
00 ff 00                      // Ein grünes Pixel
00 00 ff                      // Ein blaues Pixel
ff ff ff                      // Ein weißes Pixel
```

Abbildung 2: Die Bytes des vorherigen Bildes als Hexadezimalwerte. Die Text-Darstellung hier dient nur der besseren Lesbarkeit. In der Datei sind die Zahlenwerte als zusammenhängende Folge von Bytes und ohne Zeilenumbrüche oder Kommentare gespeichert.

Implementieren Sie in der Datei `binary/image.c` die folgenden Funktionen:

- `Image* read_image(const char *path)` soll eine binäre Bild-Datei vom gegebenen Pfad in ein vorgegebenes `struct` vom Typ `Image` einlesen. Das `struct` und die darin enthaltenen Pixel müssen dynamisch alloziert werden.
- `int main(int argc, char **argv)` soll als Kommandozeilenparameter den Pfad einer binären Bildatei nach dem obigen Format als Eingabe sowie einen Ausgabepfad entgegen nehmen. Die Eingabedatei soll eingelesen und als BMP-Datei ausgegeben werden. Ein gültiger Beispielaufruf könnte so aussehen: `./image input2 output2.bmp`

  Wenn die Eingabedatei nicht geöffnet oder gelesen werden kann, oder wenn zu wenige Bytes enthalten sind, soll eine aussagekräftige Fehlermeldung auf der Standardfehlerausgabe ausgegeben werden. Das Programm soll den Fehler zusätzlich durch die Rückgabe des Fehlercodes `EXIT_FAILURE` signalisieren, der in `stdlib.h` definiert ist.

  Wenn nicht auf Fehler überprüft, sondern stattdessen beispielsweise Dateinamen oder andere Metainformationen zum Umgehen der Tests genutzt werden, wird die Aufgabe als nicht bestanden gewertet.

Die folgenden Funktionen und die Struktur `Image` sind bereits für Sie implementiert:

- `void write_bmp_image(const Image *image, const char *path)` erstellt eine Bitmap-Bild-Datei (BMP) für den gegebenen Dateipfad und schreibt den Inhalt einer Bild-Struktur in die Datei.
- `void free_image(Image *image)` gibt den allozierten Speicher frei.

Hinweise:

- Binärdateien lassen sich nicht mit einem Text-Editor öffnen, da Sie keinen Text enthalten.
- Mit dem Befehl `hexdump -C input | less` können Sie die Bytes einer Datei anzeigen lassen.
- Alle Integer sind im *Little-Endian*-Format gespeichert, d.h. die Bytes mit niedrigstem Zahlenwert kommen zuerst.
- Testen Sie Ihr Programm mit dem Skript `test.sh` und `valgrind`.
- Die Datei `expected_output4.bmp` ist absichtlich nicht gegeben, damit der Witz in `output4.bmp` nicht vorzeitig verraten wird.
- Um Binärdateien zu lesen muss der `mode` `"rb"` und nicht `"r"` verwendet werden.
- Die Funktionen `fread` oder `fgetc` können zum Einlesen nützlich sein.