Abgabefrist Übungsaufgabe 5

Die Lösung muss abhängig von der Tafelübung abgegeben werden, an der ihr teilnehmt:

- **W1** – eigene Übung U5 am 29./30.06.: Abgabe bis **Donnerstag, den 08.07.2021 08:00**
- **W2** – eigene Übung U5 am 06./07.07.: Abgabe bis **Dienstag, den 13.07.2021 08:00**

In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch ein Lösungsvorschlag aus dem Tutorenteam.

Allgemeine Hinweise zu den BS-Übungen

- Ab jetzt ist es *nicht* mehr möglich, Einzelabgaben im AsSESS zu tätigen. Falls ihr (statt in einer Dreiergruppe) zu zweit oder zu viert abgeben möchtet, klärt dies bitte *vorher* mit eurem Übungsleiter! Der Lösungsweg und die Programmierung sind gemeinsam zu erarbeiten.
- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die Zusatzaufgaben sind ein Stück schwerer als die „normalen“ Aufgaben und geben zusätzliche Punkte.
- Die Aufgaben sind über AsSESS (`https://ess.cs.tu-dortmund.de/ASSESS/`) abzugeben. Dort gibt **ein** Gruppenmitglied die erforderlichen Dateien ab und nennt dabei die anderen beteiligten Gruppenmitglieder (Matrikelnummer, Vor- und Nachname erforderlich!). Namen und Anzahl der abzugebenden C-Quellcodedateien[2] variieren und stehen in der jeweiligen Aufgabenstellung; Theoriefragen sind grundsätzlich in der Datei `antworten.txt`[3] zu beantworten. Bis zum Abgabetermin kann eine Aufgabe beliebig oft abgegeben werden – es gilt die letzte, vor dem Abgabetermin vorgenommene Abgabe.
- Sobald eine Abgabe korrigiert wurde, kann das Ergebnis ebenfalls im AsSESS eingesehen werden.

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.
[2] codiert in UTF-8
[3] reine Textdatei, codiert in UTF-8

Betriebssysteme Übungsaufgabe 5 Dateioperationen

Aufgabe 5: Dateioperationen (10 Punkte)

In dieser Aufgabe wird der Umgang mit dem Dateisystem geübt.

ACHTUNG: Zu dieser Aufgabe existiert eine Vorgabe in Form von C-Dateien mit einem vorimplementierten Code-Rumpf, die ihr in der Programmieraufgabe erweitern sollt. Diese Vorgaben sind von der Veranstaltungswebseite herunterzuladen, zu entpacken und zu vervollständigen! Die Datei `vorgaben-A5.tar.gz` lässt sich unter Linux/UNIX mittels `tar -xzvf vorgaben-A5.tar.gz` auspacken.

Theoriefragen (4 Punkte)

1. Unix-Zugriff auf Peripheriegeräte (2 Punkte)

Wie werden unter Unix Peripheriegeräte repräsentiert? Welche zwei Klassen von Geräten gibt es? Beschreibt in eigenen Worten den Unterschied zwischen diesen Klassen von Geräten und nennt für jede Klasse ein Beispiel.

2. write und fwrite (1 Punkt)

Erklärt in eigenen Worten den Unterschied zwischen write(2) und fwrite(3).

3. E/A-Scheduling (1 Punkt)

Weshalb ist FIFO-Scheduling (First In First Out) bei Festplatten (HDDs) im Allgemeinen nicht sinnvoll (herkömmliche Bauweise mit magnetischen Platten, keine SSD)? Welche alternative Scheduling-Strategie ist eurer Meinung nach vorzuziehen?

`⇒antworten.txt`

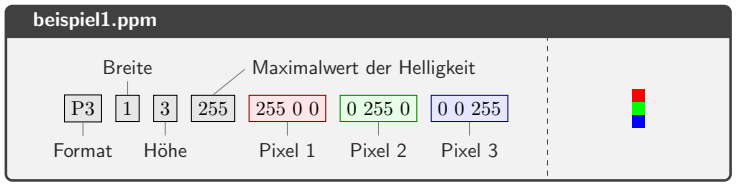
Programmieraufgabe: Portable Anymap (6 Punkte)

Für die Programmieraufgabe sollt ihr euch mit dem *Portable Anymap*-Dateiformat beschäftigen. Eine Erklärung des Formats findet ihr hier:

- Übungsfolien U5
- `https://de.wikipedia.org/wiki/Portable_Anymap`

Dateien in diesem Format könnt ihr beispielsweise in GIMP öffnen und anzeigen lassen.

- `https://www.gimp.org/downloads/`



Für die Bearbeitung dieser Aufgabe sind die Funktionen **fopen(3)**, **fclose(3)**, **fseek(3)**, **ftell(3)**, **fscanf(3)** und **fprintf(3)** hilfreich.

© 2021 TU Dortmund, Informatik 12, Arbeitsgruppe Systemsoftware

a) Format lesen (2 Punkte)

Ziel dieser Teilaufgabe ist es ein C-Programm (`a5_a.c`) zu entwickeln, das eine Datei im *Portable Anymap*-Format einlesen kann und die Breite und Höhe des enthaltenen Bildes, sowie die Dateigröße (in Bytes) in der Konsole ausgibt. Der Pfad für die zu lesende Datei soll als Kommandozeilenparameter beim Aufruf übergeben werden. Der Aufruf muss wie folgt aussehen:

./a5_a <Dateipfad>

Beispiel

In den Vorgaben zu dieser Aufgabe finden sich einige Beispieldateien, sowie eine Codevorgabe für den Ausgabestring.

Es sollen hier nur trivial formatierte Dateien betrachtet werden, Sonderfälle wie Kommentarzeilen müsst ihr NICHT beachten.

Denkt an eine ordentliche Fehlerbehandlung, zum Beispiel bei falscher Nutzereingabe, und achtet darauf, dass euer Programm ordnungsgemäß hinter sich aufräumt.

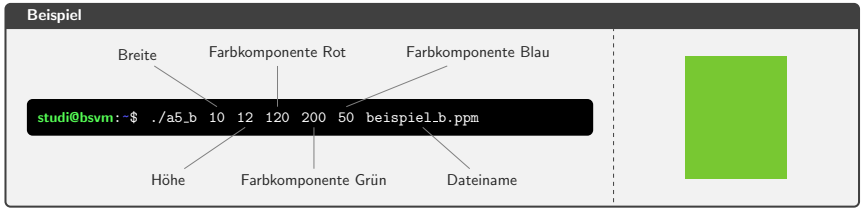
⇒ `a5_a.c`

b) Format schreiben (2 Punkte)

In dieser Teilaufgabe soll ein C-Programm (`a5_b.c`) implementiert werden, bei dem ihr eine Bilddatei im *Portable Pixmap*-Format mit einer Farbe durchgehend füllt. Hierbei sollen die Pixeldaten in ASCII kodiert werden, der Maximalwert für die Helligkeit soll 255 sein. Die Breite, Höhe und zu verwendende Farbe sollen als Kommandozeilenparameter beim Aufruf übergeben werden; zusätzlich soll noch der Pfad, in dem das Bild abgespeichert wird, angegeben werden. Der Aufruf muss wie folgt aussehen:

./a5_b <Breite> <Höhe> <Farbkomponente rot> <Farbkomponente grün> <Farbkomponente blau> <Dateiname>.ppm

Beispiel



Die Ausgabedatei `beispiel_b.ppm` findet ihr in der Vorlage.

Denkt an eine ordentliche Fehlerbehandlung, zum Beispiel bei falscher Nutzereingabe, und achtet darauf, dass euer Programm ordnungsgemäß hinter sich aufräumt.

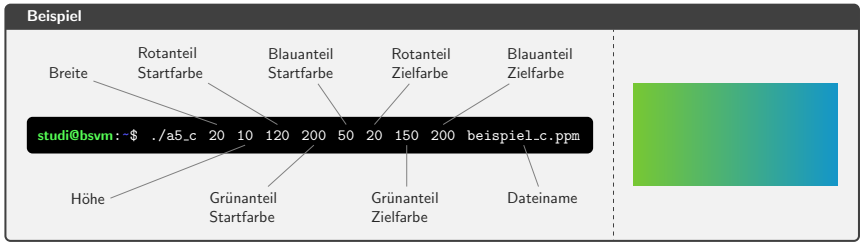
⇒ `a5_b.c`

c) Farbgradient (2 Punkte)

In dieser Teilaufgabe soll ein C-Programm (`a5_c.c`) implementiert werden, das einen Farbgradienten in einer Datei im *Portable Pixmap*-Format abspeichert. Hierbei sollen die Pixeldaten in ASCII kodiert werden, der Maximalwert für die Helligkeit soll 255 sein. In der Mitte des Gradienten sollen Start- und Zielfarbe im gleichen Verhältnis vorkommen. Der Gradient soll von einer Startfarbe gleichmäßig in eine Zielfarbe übergehen. Der Gradient soll von links nach rechts verlaufen. Hierbei sollen Breite und Höhe, sowie die Start- und Zielfarben als Kommandozeilenparameter beim Aufruf übergeben werden; zusätzlich soll noch der Pfad, in dem das Bild abgespeichert wird, angegeben werden. Der Aufruf muss wie folgt aussehen:

./a5_c <Breite> <Höhe> <Rotanteil Startfarbe> <Grünanteil Startfarbe> <Blauanteil Startfarbe> <Rotanteil Zielfarbe> <Grünanteil Zielfarbe> <Blauanteil Zielfarbe> <Dateiname>.ppm

Beispiel



Die Ausgabedatei `beispiel_c.ppm` findet ihr in der Vorlage.

Denkt an eine ordentliche Fehlerbehandlung, zum Beispiel bei falscher Nutzereingabe, und achtet darauf, dass euer Programm ordnungsgemäß hinter sich aufräumt.

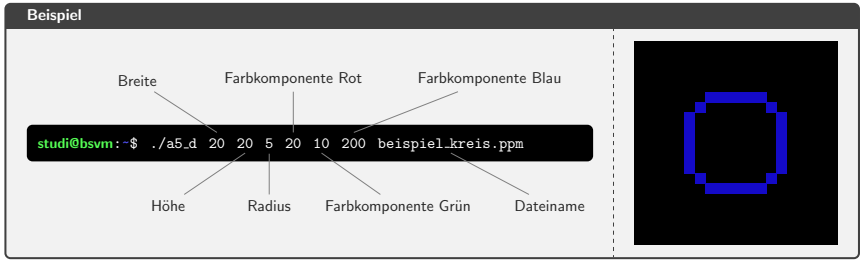
⇒ `a5_c.c`

d) Zusatzaufgabe: Kreis zeichnen (2 Sonderpunkte)

Für die Zusatzaufgabe sollt ihr ein C-Programm (`a5_d.c`) erstellen, das ein Bild mit einem Kreis zeichnet und in einer Datei im *Portable Pixmap*-Format abspeichert. Die Breite und Höhe des Bildes, sowie der Radius des Kreises in Pixeln und dessen Farbe sollen als Kommandozeilenparameter beim Aufruf übergeben werden; zusätzlich soll noch der Pfad an dem das Bild abgespeichert wird angegeben werden. Der Aufruf muss wie folgt aussehen:

./a5_d <Breite> <Höhe> <Radius> <Farbkomponente rot> <Farbkomponente grün> <Farbkomponente blau> <Dateiname>.ppm

Beispiel



Die Ausgabedatei `beispiel_kreis.ppm` findet ihr in der Vorlage.

Der Kreis soll zentriert im Bild gezeichnet werden, die Mitte des Kreises ist also die Mitte des Bildes.

Denkt an eine ordentliche Fehlerbehandlung, zum Beispiel bei falscher Nutzereingabe, und achtet darauf, dass euer Programm ordnungsgemäß hinter sich aufräumt.

⇒ `a5_d.c`

© 2021 TU Dortmund, Informatik 12, Arbeitsgruppe Systemsoftware

Tipps zu den Programmieraufgaben:

- Kommentiert euren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denkt daran, dass viele Systemaufrufe fehlschlagen können! Fangt diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), gebt geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beendet euer Programm danach ordnungsgemäß.
- Die Programme sollen sich mit dem gcc auf den Linux-Rechnern im IRB-Pool übersetzen lassen. Der Compiler ist mit den folgenden Parametern aufzurufen:
  `gcc -Wall -D_GNU_SOURCE`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind: `-std=c11 -Wpedantic -Werror -D_POSIX_SOURCE`
- Achtet darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt, z.B. durch Nutzung von `-Werror`.
- Alternativ kann auch der GNU C++-Compiler (*g++*) verwendet werden.