Ludwig-Maximilians-Universität München
LFE Medieninformatik
Prof. Dr. Andreas Butz
Dominikus Baur, Sebastian Boring

Sommersemester 2009
Übungsblatt 1
24.April 2009

Computergrafik 1

Abgabetermin:
Die Lösung zu diesem Übungsblatt ist bis zum Freitag den **1. Mai 2009, 12:00 Uhr s.t.** über UniWorx abzugeben.

Inhalt:
Diese Blatt dient zum Bekanntmachen mit der Programmiersprache C++. Dazu werden Funktionen, Kontrollstrukturen, Zeiger und Arrays anhand praktischer Beispiele betrachtet. Pro Aufgabe können maximal zehn Punkte erreicht werden. Zum Bestehen des Übungsblatts müssen in **JEDER** Aufgabe mindestens fünf Punkte erreicht werden.

Geben Sie zu jeder Aufgabe alle benötigten Header- und Sourcedateien mit ab. Abgaben die nicht kompilieren werden mit 0 Punkten bewertet. Fassen Sie alle Aufgaben zu einer zip-Datei zusammen und laden Sie sie bei UniWorx hoch. Für jede Aufgabe soll ein Ordner mit dem Namen „auf1-X“ angelegt werden. Für die Mathematik-Aufgabe (Aufgabe 4) fügen Sie ein eingescanntes Bild oder ein PDF (falls mit Word oder LaTeX erstellt) in den Ordner „auf1-4“ ein.

Aufgabe 1 Funktionen (10 Punkte)

In dieser Aufgabe soll die Programmierung von Funktionen und die Integration von Kontrollstrukturen geübt werden. Dafür wird eine Funktion zur Berechnung der Fibonacci-Folge als Beispiel verwendet.

a) Schreiben Sie eine Applikation, die über eine Funktion `Fibonacci` verfügt, die die Fibonacci-Folge einer vom Benutzer eingegebenen Zahl berechnet und ausgibt. Erstellen Sie hierzu lediglich eine Datei `Fibonacci.cpp` ohne header-Datei mit nur zwei Funktionen, einer `main`-Funktion und der oben genannten. Verwenden Sie für die Berechnung der Fibonacci-Folge die Rekursion. Definieren Sie zuerst die `main`-Funktion, dann die Funktion `Fibonacci` und verzichten Sie auf eine explizite Deklaration der Funktionen (**5 Punkte**).

b) Drehen Sie nun die Definition der beiden Funktionen um, so dass zuerst `Fibonacci`, dann `main` definiert wird. Versuchen Sie die Quelldatei zu kompilieren.

c) Fügen Sie vor den Funktionsdefinitionen nun die Deklarationen ein, also die Funktionsprototypen. Kompilieren Sie erneut die Quelldatei.

d) Erstellen Sie nun eine header-Datei `Fibonacci.h` und setzen Sie dort die Funktionsprototypen ein. Löschen Sie diese aus der `.cpp`-Datei und fügen Sie stattdessen ein `# include`-Statement ein, das die header-Datei einbindet (**5 Punkte**).

Erstellen Sie ein Verzeichnis „auf1-1“, das `Fibonacci.h` und `Fibonacci.cpp` enthält und fügen Sie es Ihrer Abgabe hinzu.

Aufgabe 2 Klassen, Vererbung, Abstraktion (10 Punkte)

C++ ist vollständig objektorientiert und bietet daher vielfältige Möglichkeiten Klassenbeziehungen auszudrücken. Die folgenden Klassen liegen im Namensraum `CGGeometry`: Die Klasse `Point` stellt eine abstrakte Oberklasse für die spezielleren Klassen `Point1D, Point2D, Point3D, etc.` dar. Ein `Point` hat eine Position im Raum sowie eine Farbe vom Typ `Color`. `Color` ist eine einfache Klasse die einen RGBA-Farbwert speichert. Sie bietet die Methoden **`void`** `setColor(int, int, int, int)` und **`int*`** `getColor` an, die jeweils die aktuelle Farbe setzen bzw. ein Array mit vier Einträgen (Rot, Grün, Blau, Alpha der aktuellen Farbe) zurückgeben. Instanzen des Typs `Color` sollen allerdings auch über einen speziellen Konstruktor initialisiert werden können.

a) Legen Sie sowohl die Header- als auch die Sourcedatei für `Color` an und deklarieren und implementieren Sie die Klasse. Achten Sie darauf, dass bei der Methode `setColor` die Farbkomponenten Werte von 0 bis 255 haben sollen. Ist dies nicht der Fall, geben Sie eine kurze Fehlermeldung an der Kommandozeile aus und setzen Sie als Farbe Schwarz (d.h. Rot = Grün = Blau = 0) (**4 Punkte**).

b) Erstellen Sie jetzt sowohl Header- als auch Sourcedateien für die Klassen `Point` und dessen Erben `Point2D` und `Point3D`. Überlegen Sie sich einen geeigneten Weg die Position des Punkts im Raum intern zu speichern und fügen Sie Getter- und Setter-Methoden für dieses Attribut hinzu (**3 Punkte**).

c) Deklarieren und implementieren Sie eine weitere Methode **`void`** `output` für die beiden Punktklassen, die eine kurze aber lesbare Beschreibung des Punkts (d.h. Position + Farbe) auf die Kommandozeile ausgibt (**1 Punkt**).

d) Fügen Sie schließlich noch eine Sourcedatei namens `CGAuf1-2.cpp` hinzu, die eine `main`-Methode hat die folgendes leistet:

- Erstellt einen `Point2D` an der Position (4.6, 2.4) mit einem knalligen Rotton.
- Erstellt einen `Point3D` an der Position (4.6, 2.4, 1.0) in Blau.
- Erstellt einen weiteren `Point3D` an der Position (4.6, 2.4, -1.0) in Grau.
- Gibt nacheinander die Beschreibung der drei Punkte aus.

(**2 Punkte**).

Erstellen Sie ein Verzeichnis „auf1-2“, das alle Header- und Sourcedateien enthält und fügen Sie es Ihrer Abgabe hinzu.

Aufgabe 3 Zeiger (pointer) und Felder (arrays) (10 Punkte)

Laden Sie sich das auf der Homepage zur Verfügung gestellte Framework für ein Betriebssystem Ihrer Wahl herunter. Dieses lädt automatisch die beiden Bilder „test1.png“ und „test2.png“ mit 24-Bit Farbtiefe. Die Methoden `extractChannel`, `scaleImage` und `diffImages` sind nicht implementiert und deren aufrufende Funktionen `extractColors`, `produceScaledImages` und `produceDiffImages` entsprechend in der `main`-Methode auskommentiert.

a) Kompilieren Sie das Projekt und starten Sie die entstandene ausführbare Datei. Werfen Sie auch einen Blick auf den Quellcode. In den Feldern `imgData` und `imgData2` werden die Pixel der Bilder gespeichert. Denken Sie daran, dass das Feld eindimensional ist, die gespeicherte Information allerdings zweidimensional. Die Position eines Pixels `(x, y)` in diesem Feld lässt sich für ein 24-Bit RGB-Bild über folgende Formel berechnen:

$Position = (y \cdot Breite\ des\ Bilds + x) \cdot 3$

An dieser Position im Feld sind drei aufeinanderfolgende Bytes für den Rot-, Grün- und Blauwert zu finden.

b) Die Funktion `extractChannel` soll dazu dienen, ein RGB-Bild in die Farbkanäle zu zerlegen. Der Parameter `offset` gibt dabei an, welcher Farbkanal zurückgegeben werden soll (Rot: 0, Grün: 1, Blau: 2). Reparieren Sie diese Funktion indem Sie ein eigenes `byte`-Feld mit nur einem Farbkanal erstellen und einen Pointer darauf zurückgeben. Entfernen Sie den Kommentar in `main` bei `extractColors`. Wenn alles klappt sollten die drei erzeugten Bilder „red.png“, „green.png“ und „blue.png“ die jeweiligen Farbkanäle des Bildes enthalten (**2 Punkte**).

c) Eine zweite Funktion, `scaleImage`, soll das Bild skalieren. Reparieren Sie ebenfalls diese Methode und benutzen Sie dazu das *Nearest Neighbor Interpolationsverfahren*. Bei diesem Algorithmus wird für jeden Bildpunkt im Ergebnis einfach die Farbe des nächstgelegenen Bildpunkts im Ursprungsbild verwendet. Reparieren Sie auch diese Funktion indem Sie ein eigenes `byte`-Feld in der passenden (Ziel-)Größe erstellen und einen Pointer darauf zurückgeben. Entfernen Sie wieder den Kommentar bei `produceScaledImages` in `main` und überprüfen Sie das Ergebnis anhand der erzeugten Dateien „half.png“, „twice.png“ und „skew.png“ (**4 Punkte**).

d) „test2.png“ ist eine stark komprimierte Version von „test1.png“ mit sichtbaren Artefakten. Implementieren Sie die Funktion `diffImages`, die zwei Bilder miteinander vergleichen und das Differenzbild zurückgeben soll (d.h. für jeden Farbkanal und jeden Bildpunkt soll der Unterschied berechnet und zurückgegeben werden). Erstellen Sie ein eigenes `byte`-Feld und geben Sie einen Pointer darauf zurück. Entfernen Sie den letzten Kommentar bei `produceDiffImages` und überprüfen Sie anhand des erzeugten „diff.png“ ob Ihre Funktion läuft (**2 Punkte**).

e) Achten Sie bei den vorhergehenden Aufgaben darauf in den Schleifen Zeiger zu erhöhen und nicht mit Feldern zu arbeiten um die Effizienz Ihrer Implementierung zu erhöhen (**2 Punkte**).

Erstellen Sie ein Verzeichnis „auf1-3“, das alle Header- und Sourcedateien enthält und fügen Sie es Ihrer Abgabe hinzu.

Aufgabe 4 Polarkoordinaten (10 Punkte)

Ein nützliches Werkzeug in der Computergrafik und Navigation sind Polarkoordinaten. Im Gegensatz zum kartesischen Koordinatensystem, bilden sie ein kreisförmiges Koordinatensystem, in dem jeder Punkt durch ein Tupel der Form (φ, r) dargestellt wird. Hierbei beschreibt φ den Winkel des Punktes (Winkel zwischen dem Ortsvektor des Punktes und der positiven x-Achse), und r den Abstand des Punktes vom Ursprung. Durch Polarkoordinaten lassen sich *Drehung* und *Skalierung* sehr einfach realisieren.

a) Polarkoordinaten lassen sich folgendermaßen in kartesische Koordinaten umwandeln:
$\vec{r} = r \cdot cos\varphi \cdot \vec{e_x} + r \cdot sin\varphi \cdot \vec{e_y}$

- Was bedeutet die *Drehung* (um den Ursprung) für die Umwandlung? Geben Sie hierzu die Umwandlung in kartesische Koordinaten für die Zahl (α, r) an, nachdem sie um den Winkel β (gegen den Uhrzeigersinn) gedreht wurde.

- Was bedeutet die *Skalierung* (in Bezug auf den Ursprung)? Geben Sie hierzu die Umwandlung in kartesische Koordinaten für die Zahl (α, r) an, nachdem sie um den Faktor s skaliert wurde.
- Warum lässt sich die *Translation* (d.h. die Verschiebung des Punktes) nicht so trivial durch Polarkoordinaten beschreiben?
- Wie lässt sich ein Vektor $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$ in Polarkoordinaten allgemein umwandlen?

(**5 Punkte**).

b) Gegeben sei der Vektor $\vec{v} = \begin{pmatrix} -4 \\ -2 \end{pmatrix}$ in kartesischen Koordinaten:

- Wandeln Sie diese Zahl in Polarkoordinaten um.
- Drehen Sie diese Zahl nun um den Winkel $32,4^\circ$ (im Uhrzeigersinn). Geben Sie das Ergebnis in **beiden** Koordinatensystemen an.
- Skalieren Sie die Zahl nun mit dem Faktor $1,75$ (in Bezug auf den Ursprung) und geben Sie das Ergebnis erneut in **beiden** Koordinatensystemen an.

(**3 Punkte**).

c) Welchen Einfluss hat die Reihenfolge der Transformationen (in diesem Fall *Drehung* und *Skalierung*) auf das Ergebnis? Begründen Sie Ihre Antwort (**2 Punkte**)!

Erstellen Sie ein Verzeichnis „auf1-4“, das die Lösung als eingescanntes Bild, oder als PDF (aus Word bzw. LaTeX generiert) enthält und fügen Sie es Ihrer Abgabe hinzu.