

Beispielangaben zu OOP (Objekt-Orientierte Programmierung)

SS 2009

Beispiel 1

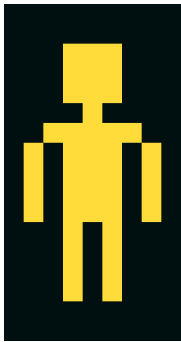
Relevante Themen:

- Ressourcen-Allokation
- Fehlerbehandlung
- Dynamic Objects
- Klassen zur Informationskapselung
- Standard I/O von C++

Design und Implementierung:

Schreiben Sie in C++ ein skriptgesteuertes Bildbearbeitungsprogramm `imgsynth`. Das Programm liest als einzige Kommandozeilenoption den Namen einer Skriptdatei ein, welche die Befehle, die vom Programm durchzuführen sind, beinhaltet. Jede Zeile in dieser Skriptdatei enthält genau einen kompletten Befehl. Diese Befehle werden sequentiell abgearbeitet, wobei jeder Befehl auf den Bildspeicher angewendet wird. Der erste Befehl muss allerdings immer ein Lesebefehl sein, um eine Bilddatei vom Dateisystem einzulesen und der letzte Befehl muss immer ein Schreibbefehl sein, um die aktuellen Bilddaten in eine Datei zu schreiben. Jeder Befehl kann eine fix definierte Anzahl von Argumenten übernehmen.

Die Synopsis des Programms ist folgende: `imgsynth -i <scriptfile>`



Beispiel: Angenommen, `imgsynth` wird aufgerufen mit `imgsynth -i bsp.isc`, wobei die Skriptdatei `bsp.isc` folgende Befehle enthält:

```
read(BMP, yellow_man.bmp)
fillrect(0,3,6,5,0,255,0)
fillrect(2,13,7,4,0,0,255)
write(BMP, "yellow_man f.bmp")
```

... so wird damit die Bilddatei `yellow_man.bmp` (Bild links) eingelesen, weiters werden zwei Rechtecke in das Bild gemalt und schlussendlich wird das modifizierte

Bild unter dem Filenamen "`yellow_man f.bmp`" (Bild rechts) gespeichert. White spaces innerhalb von Dateinamen erfordern, dass der Dateiname mit Anführungszeichen umgeben wird.



Im folgendem werden die zu unterstützenden Befehle nochmals genauer erklärt:

`read(typ, name)` ... Öffnet eine Grafikdatei „name“ und interpretiert das Format entsprechend dem spezifizierten Typ „typ“ und liest den Bildinhalt in den Hauptspeicher. Der Name der Grafikdatei kann optional unter Anführungszeichen gestellt sein. Als Typ muss nur der Dateityp „BMP“ unterstützt werden, was für eine vereinfachte Version des Windows Bitmap Dateiformates (auch als OS/2 Bitmap Dateiformat bekannt) steht (siehe Beschreibung am Beispielenende).

`fillrect(x, y, w, h, r, g, b)` ... Zeichnet ein gefülltes Rechteck an die Koordinate (x,y), mit der angegebenen Größe (w, h) und der angegebenen Farbe (r, g, b). Die Koordinate (0,0) bezeichnet dabei die Bildecke „links oben“. Die Größe (w, h) beschreibt die Breite „w“ (nach rechts) und die Höhe „h“ (nach unten) des zu malenden Rechteckes in Pixel. Die Farbe (r, g, b) gibt die Farbanteile Rot, Grün und Blau jeweils im Wertebereich von 0..255 der Füllfarbe des Rechteckes an.

`write(typ, name)` ... Speichert das aktuelle Bild als Datei „name“ mit Typ „typ“ im Dateisystem. Als Formatstring braucht nur „BMP“ unterstützt werden, was dieselbe Bedeutung hat wie beim Befehl `read()`.

Whitespaces wie beispielsweise Leerzeichen und Tabulatoren sind beim Einlesen der Skriptdatei zu ignorieren.

Erstellen Sie dazu eine Klasse CBitmap, welche von einem Eingabestream Grafiken vom Format „BMP“ dekodieren kann. Der benötigte Bildspeicher ist dabei dynamisch anzufordern. Falls der Eingabestream nicht dem Bildformat entspricht, soll eine Fehlerbehandlung basierend auf Ausnahmen (Exceptions) eingebaut werden. Da es im Allgemeinen unterschiedliche Pixelformate gibt (Anzahl der Bits pro Farbkanal), soll das Abspeichern der Bildpunkte in den Bildspeicher über eine separate Klasse CPixelFormat durchgeführt werden. Der pro Pixel allokierte Speicher soll dem konkreten Pixelformat entsprechen und nicht beispielsweise allgemein als eine 32-bit-Zahl gespeichert werden. Die Klasse CPixelFormat soll beispielsweise eine Methode `setPixel(char* data, x, y)` enthalten, wobei „data“ ein Zeiger auf den Farbwert darstellt und „x“ und „y“ die Pixelkoordination darstellen, wo das Pixel abgespeichert werden soll. Die Klasse CPixelFormat ist in die Klasse CBitmap mittels Komposition einzubinden.

Der read-Befehl soll die Pixel nicht auf ein gemeinsames Pixelformat konvertieren. Die einzelne Filter/Befehle sollen die Operationen direkt im ursprünglichen Pixelformat durchführen.

Die in der Spezifikation nicht genauer ausgeführten Teile können selbst sinnvoll ausgestaltet werden, wobei jedoch die folgenden Anforderungen einzuhalten sind.

Das Windows Bitmap Format:

Um diese Aufgabe zu lösen müssen Sie ein Bild im Standardformat „Windows Bitmap“ einlesen, interpretieren und schreiben können. Das Format ist auch als „OS/2 Bitmap“ bekannt.

Sie brauchen im Zuge dieses Beispiels allerdings nicht alle Details des Windows Bitmap Formates zu unterstützen. Als Pixelformat brauchen Sie beispielsweise nur RGB24 zu unterstützen. Weiters brauchen Sie nur unkomprimierte Bildspeicherung zu unterstützen. Im Folgenden ist das eingeschränkte Windows Bitmap Format näher beschrieben.

Eine Grafikdatei vom Typ „Windows Bitmap“ enthält am Beginn einen sogenannten Header, gefolgt von den eigentlichen Bilddaten. Der Header erlaubt es einerseits, zu erkennen, ob tatsächlich ein „Windows Bitmap“ Format vorliegt und gibt weiters an, wie die Bilddaten zu interpretieren sind.

Der Header besteht aus mehreren Feldern, welche in folgender Tabelle zusammengefasst sind. Die Spalte „Offset“ beschreibt die fixe Position des jeweiligen Feldes innerhalb der Datei und die Spalte „Länge“ gibt die Länge des Feldes in Bytes an. Manche der Felder haben für „Windows Bitmap“ Grafiken einen konstanten Wert.

Offset (Position in der Datei)	Länge (Bytes)	Funktion/Wertebereich
0 (erstes Byte)	1	→ Muss den Wert 0x42 haben
1	1	→ Muss den Wert 0x4D haben
2	4	Gesamte Dateigröße
6	4	→ Nicht verwendet
10	4	Position des Bildinhaltes in der Datei (= Offset zu den eigentlichen Daten)
14	4	→ Muss den Wert 0x28 haben
18	4	Anzahl der Pixel pro Zeile
22	4	Anzahl von Pixelzeilen
26	2	→ Muss den Wert 0x01 haben
28	2	Pixelformat (siehe weiter unten)
30	4	→ Muss den Wert 0x00 haben
34	4	Größe der Bilddaten (es gilt: Wert des Feldes = Dateigröße – 54)
38	4	Horizontale Auflösung (Pixel pro Meter)
42	4	Vertikale Auflösung (Pixel pro Meter)
46	4	→ Muss den Wert 0x00 haben
50	4	→ Muss den Wert 0x00 haben

Bilddaten:

Nach dem Header kommen die Bilddaten, d.h., die Beschreibung der Pixelfarben für jeden Pixel. Diese Bilddaten sind eine einfache Sequenz von Pixelwerten; pro Pixel wird die Farbe separat angegeben. Die Pixelwerte sind zeilenweise gespeichert. Ist die Zeilenanzahl im Header positiv, so ist das Bild um die X-Achse gespiegelt gespeichert, ist die Zeilenanzahl im Header negativ, so beginnt die Aufzählung mit der obersten Zeile.

Die Länge jeder Zeile in Bytes muss ein Vielfaches von 4 sein. Sollte die Byteanzahl der Pixeldaten in einer Zeile kein Vielfaches von 4 sein, so werden Padding-Bytes am Ende der Zeile benutzt, um auf ein Vielfaches von 4 zu kommen. Diese Padding-Bytes enthalten keine Information und sind beim Einlesen einfach zu überspringen, sie müssen aber beim Schreiben unbedingt geschrieben werden.

Pixelformate:

Ein Pixel kann auf verschiedene Weise gespeichert werden; das Format wird durch das Feld „Pixelformat“ im Header festgelegt. Für dieses Beispiel ist nur der Wert Pixelformat= 24 wichtig, wodurch ein Pixel durch drei Bytes dargestellt wird: ein Byte für die blaue, ein Byte für die grüne und ein Byte für die rote Farbkomponente. Dieses Format nennt man auch B8R8G8.

Weitere Informationen über dieses Grafikformat finden Sie beispielsweise im Internet unter folgenden Adressen:

http://en.wikipedia.org/wiki/BMP_file_format

<http://atlc.sourceforge.net/bmp.html>

Als Beispiel für das „Windows Bitmap“ Bildformat wird gezeigt, welche Binärdarstellung das obige Beispielbild yellow_man.bmp hat. Der Header der Bilddatei hat folgenden Inhalt:

	00	01	02	03	04	05	06	07	08	09
0	0x42	0x4d	532				not used			
10	54 (of f set)				0x28				9	
20	(Zei l e)		17 (#Zei l en)				1		24 (PF)	
30	0				0 (???)				2834	
40	[Pi xel / m]		2834 [Pi xel / m]				0			
50	0									

Die Bilddaten für das Bild sehen folgendermaßen aus (die einzelnen Bytewerte sind als hexadezimale Werte angegeben):

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
54	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	P
82	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	P
110	10	10	00	10	10	00	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	10	10	00	10	10	00	P
138	10	10	00	10	10	00	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	10	10	00	10	10	00	P
166	10	10	00	10	10	00	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	10	10	00	10	10	00	P
194	10	10	00	10	10	00	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	10	10	00	10	10	00	P
222	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	P
250	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	P
278	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	P
306	10	10	00	3a	dc	ff	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	3a	dc	ff	10	10	00	P
334	10	10	00	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	10	10	00	P
362	10	10	00	10	10	00	10	10	00	10	10	00	3a	dc	ff	10	10	00	10	10	00	10	10	00	10	10	00	P
390	10	10	00	10	10	00	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	10	10	00	10	10	00	P
418	10	10	00	10	10	00	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	10	10	00	10	10	00	P
446	10	10	00	10	10	00	10	10	00	3a	dc	ff	3a	dc	ff	3a	dc	ff	10	10	00	10	10	00	10	10	00	P
474	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	P
502	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	10	10	00	P
530	00	00																										

Anforderungen:

Kommentieren Sie den Code, um das Verstehen des Codes zu vereinfachen. Zusätzlich soll zu Beginn jeder Klasse eine Klassenbeschreibung stehen. Am Beginn jeder Datei soll außerdem Name, MNR, KZ der Gruppenmitglieder sowie die Beispielnnummer stehen. Ein nicht kommentierter Code wird negativ bewertet!

Ausnahmen (Exceptions) müssen immer abgefangen werden. Überlegen Sie sich dabei aber auch, welche Ausnahmen zu einer Programmbeendigung führen müssen und in welchen Fällen es ausreicht, eine Warnung bzw. Fehlermeldung auszugeben. Bei jeder Ausnahmebehandlung ist auf jeden Fall eine Fehlermeldung auf die Fehlerausgabe (cerr) auszugeben. Als Orientierung seien einige mögliche Quellen für Ausnahmen aufgezählt: Speichermangel, inkorrektter Input, Schreiben auf Datei fehlgeschlagen, etc.

Benutzen Sie zur Programmentwicklung auch ein Makefile, welches zumindest die Targets „clean“, „all“ und „run“ enthält. Mittels „make run“ soll das Programm direkt aufgerufen werden können (mit sinnvollen Argumenten versehen).

Schreiben Sie zu dem Programm ein Protokoll, das folgende Informationen beinhaltet:

- Identifikation (Name, MNR, Kz)
- Aufgabenstellung (kann direkt von dieser Angabe übernommen werden, für das konkrete Beispiel reicht es, den Text der Aufgabenstellung zu übernehmen; Beispiele und Spezifikation der Bildformate sind nicht zu inkludieren)
- Klassendiagramme (inkl. Beschreibung)
- Schematische Beschreibung der Allokation und Freigabe von Ressourcen
- Schematische Beschreibung der Ausnahmebehandlung (mögliche Ursachen von Exceptions, sowie die Strategie bei deren Behandlung (u.a., ob lokal oder eher global behandelt))
- Dokumentation des Arbeitsaufwandes
- Dokumentation von ev. aufgetretenen Problemen
- Kommentierter Programmcode

Der Umfang des Protokolls soll (ohne Mitzählung der Codeseiten) zwischen 5 bis 10 Seiten sein.

Abgabe:

Legen Sie ein Verzeichnis „Beispiel1“ an, in dem Sie alle Ihre Programmdateien hineingeben. Geben Sie auch das Protokoll in dieses Verzeichnis hinein.

Packen Sie das Verzeichnis folgendermaßen ein:

tar -zcvf Beispiel1.tgz Beispiel1

und geben Sie dieses Archivfile elektronisch ab. Ohne diese elektronische Abgabe sowie das Mitbringen eines schriftlichen Ausdruckes des Protokolls wird die Abgabe nicht angenommen.

Der relevante Stoff für das Abgabegespräch ist das Kapitel 1 des Skriptums. Es ist dabei auch notwendig, dass Sie den Text, wo zutreffend, für die Lösung des Beispiels anwenden.