



Norges teknisk-naturvitenskapelige
universitet
Institutt for datateknikk og
informasjonsvitenskap

TDT4102 Prosedyre
og Objektorientert
programmering
Vår 2013

Øving 8

Frist: 2013-04-12

Aktuelle temaer for denne øvingen:

- Klasser, arv, polymorfi og virtuelle funksjoner
- Unsigned og const

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner som er gitt i oppgaven.
- Det er valgfritt om du vil bruke en IDE (Visual Studio, XCode), men koden må være enkel å lese, kompilere og kjøre.

Anbefalt lesestoff:

- Kapittel 14 og 15, Absolute C++ (Walter Savitch)
- It's Learning-notater

NOTAT

- Alle **uthevede** spørsmål skal besvares.
- Eventuelle andre spørsmål i teksten er ment som refleksjonsoppgaver, men anbefales å gjøres da de er en del av øvingen og øvingsopplegget er eksamensrelevant.

Introduksjon:

Denne øvingen er annerledes enn de andre øvingene ettersom du nå skal skrive et program som er noe mer visuelt engasjerende enn tidligere. Du skal implementere en bildeklasse som representerer en Bitmap Image File.

Utdelte filer:

Disse filene skal lastes ned, lagres sammen med de andre .h og .cpp i øvingen og legges til prosjektet. Øvingen skal løses uten å gjøre endringer i de utdelte filene, og det forventes at du har satt deg inn i koden i filen `Image.h`.

- `Image.h`: prototypen til bildeklassen du skal implementere. Ikke endre, men **ta en titt på denne**.
- `ImageIO.cpp`: Inneholder funksjonalitet for å lagre Image-objekter til disk.

Andre relevante kilder:

- RGB fargemodellen - Wikipedia
<http://en.wikipedia.org/wiki/RGB>.
- Hvordan representere et todimensjonalt array med en endimensjonal liste. Se forelesningsnotater.

1 Introduksjon

Denne oppgaven kan hoppes over hvis du er kjent med begrepene «piksel», «RGB», «bitmap» og «unsigned» datatyper. Er du i tvil om du er kjent med disse begrepene, gjør oppgaven!

- a) Bilder kan representeres digitalt som et nett av punkter, hvor hvert punkt har en bestemt farge. I datamaskinverdenen kalles disse punktene for «pikslar», og størrelsen på bilder blir som regel oppgitt på formen $X \times Y$, der X er bildets bredde og Y er dets høyde i pikslar. Disse verdiene omtales som bildets oppløsning («dimensjoner» er også brukt, dog mindre ofte) og er angitt i pikslar. Altså har et bilde med en oppløsning på 1366×768 1366 pikslar i bredden og 768 i høyden.

Hva er dimensjonene til dette bildet

<http://www-ee.uta.edu/dip/courses/ee5356/lena512.bmp>

- b) I digitale bilder har hver av pikslene en gitt farge. For å kunne representere en farge digitalt trenger vi en fargemodell, som for eksempel RGB. RGB er en fargemodell hvor fargene **R**ød, **G**rønn og **B**lå blandes sammen i forskjellige forhold for å lage andre, mer spennende, farger. I motsetning til den subtraktive fargemodellen vi lærte om på barneskolen (hvor rød, gul og blå var primærfargene) er RGB-modellen additiv. I en additiv fargemodell får man hvitt hvis man blander alle primærfargene; i en subtraktiv fargemodell får man sort ved å blande alle primærfargene.

Hvilken farge får man hvis man ikke blander sammen noen farger i en additiv fargemodell?

- c) For å representere en farge i RGB-modellen (Rød, Grønn og Blå) trengs det en separat verdi for hver av fargene. Altså en verdi for å indikere hvor mye rød vi skal ha, en annen verdi som indikerer hvor mye blått og en tredje verdi for å indikere mengden grønt. Det er vanlig å bruke verdier mellom 0 og 255 for å indikere mengden av hver farge. For eksempel: «R=0, G=0, B=255» er en måte vi kan definere en farge i RGB-modellen (i dette tilfelle er fargen «blå», eller «det blir ikke mer blå enn dette»). Vi kan også uttrykke dette på formen «0, 0, 255» eller hexadesimalt med «0x0000FF». Uansett så er den første verdien rød, den andre verdien grønn og den siste verdien blå.

Hvordan kan vi representere rød i RGB-modellen?

- d) Å representere en farge i RGB-modellen krever altså tre verdier. I C++ kan vi for eksempel bruke tre variabler, en for hver farge. I `Image.h` finnes det en struct ved navn «Color» som gjør akkurat dette: den har tre `unsigned char`-variabler som representerer mengden av hver primærfarge i fargen. Unsigned er engelsk for «uten fortegn». En vanlig (`signed`) `char` kan lagre verdier mellom -128 og 127 (husk at bokstaver egentlig bare er tall). Unsigned `char` kan lagre verdier mellom 0 og 255, noe som passer ypperlig til vårt formål da hver fargeverdi i RGB-modellen vår kan ha verdier fra og med 0, til og med 255!

Hvorfor bruker vi unsigned char til å lagre fargedata i bildeklassen?

Hvorfor kan det være lurt å bruke unsigned variabler til å lagre høyden og bredden til bildeklassen?

2 Grunnarbeidet - Image klassen

a) **Implementer konstruktøren for Image-klassen.** Husk at konstruktører brukes til å sette opp objektet før bruk, og den må derfor initialisere alle medlemsvariablene. Arrayen som holder pikseldataene må allokeres dynamisk.

1. Det trengs 3 `unsigned char` for å lagre fargen til en piksel. Hvor mange `unsigned chars` trenger man for et helt bilde? *Hint: se Figur 1.*

Figur 1: Et eksempel på hvordan fargedataen til et 3×1 bilde kan representeres i et array.

R _{0,0}	G _{0,0}	B _{0,0}	R _{1,1}	G _{1,1}	B _{1,1}	R _{2,2}	G _{2,2}	B _{2,2}
------------------	------------------	------------------	------------------	------------------	------------------	------------------	------------------	------------------

2. *Hint: Selv om bilder gjerne har to dimensjoner, kan vi representere det med et en-dimensjonalt array gitt at det har konstant bredde. Se Figur 2 og 3.*

Figur 2: Et to-dimensjonalt array med plass til 9 elementer.

0	1	2
3	4	5
6	7	8

Figur 3: Det samme arrayet som i Figur 2 presentert som et en-dimensjonalt array.

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

b) **Implementer destruktøren for klassen.**

c) **Skriv medlemsfunksjonene `GetWidth()` og `GetHeight()` for Image-klassen.**

1. Hva betyr `const` som står etter disse funksjonene?
2. Hvorfor burde vi bruke `const` her?

d) **Skriv `GetData()`-medlemsfunksjonene.** Disse skal begge returnere en peker til bildets dataområde.

1. Én av disse er `const`, den andre er ikke. Hvorfor?

e) **Implementer `SetColor()`-medlemsfunksjonen.** Denne skal sette fargeverdien til én piksel i bildet. Husk at fargeverdien består av tre forskjellige komponenter som skal lagres etter hverandre i bildedataene.

1. Pass på at `SetColor()` ikke prøver å endre på punkter utenfor bildet!
2. Hvordan får man tak i array-indeksen til pikselen på posisjon (x, y) i bildetabellen?

f) **Skriv `GetColor()`-medlemsfunksjonen.** Denne skal returnere fargeverdien til en piksel i bildet.

1. Test også at `GetColor()` returnerer den samme fargen som ble satt av `SetColor()`

g) **Skriv `Fill()`-funksjonen.** Denne skal fylle hele bildet med samme farge.

h) **Bruk `SaveImage()`-funksjonen, deklart i `Image.h`, til å lagre et bilde med bakgrunnsfargen `R=193, G=84, B=193`. Denne fargen kalles *crayola fuchsia*. Vi sa RGB kunne brukes til å lage mange spennende farger!**

1. Eksperimenter selv med andre, spennende fargekombinasjoner. Bare fantasien setter grenser!
2. Bruk også `SetColor()` til å sette ett rødt punkt på en hvit bakgrunn. Hvis punktet tegnes på feil sted har du en feil i `SetColor()`.

3 Abstrakte klasser - Shape-klassen

Du kan nå fylle et bilde med farger og sette enkeltpunkter til andre farger; nå skal du lage et sett med klasser som representerer forskjellige geometriske figurer som kan tegnes inn i et bilde. Siden C++ er et objektorientert språk skal vi løse dette på en objektorientert måte ved å lage en klasse som inneholder fellestrekkene til de geometriske figurene vi har lyst til å kunne tegne.

a) Opprett klassen Shape

- b) Shape** skal ha en medlemsvariabel for å lagre farge. Denne variabelen skal være privat, og skal settes i konstruktøren. **Legg til denne medlemsvariabelen.**

Tips: Bruk `Color`-structen fra `Image.h`.

1. Hvordan kan vi hente ut fargen til en **Shape** når medlemsvariabelen er `private`?

- c) Shape** skal ha en pure virtual medlemsfunksjon kalt **Draw()**; denne funksjonen skal ta inn en referanse til et **Image**-objekt som argument. I form-klassene som skal lages etterpå skal denne funksjonen implementeres og brukes til å tegne formen på et bilde. **Legg til medlemsfunksjonen Draw().**

1. Hva vil det si at en funksjon er pure virtual?
2. En klasse som inneholder en pure virtual-medlemsfunksjon kalles en abstrakt klasse. Hva kan vi ikke gjøre med abstrakte klasser?

4 Spesialisering av baseklasser - Line-klassen

Den første figuren vi skal lage er en av de aller eldste geometriske formene, nemlig linjen. Eller, da en linje strengt tatt er uendelig i lengde er det linjestykker vi skal tegne. Et linjestykke kan defineres ut i fra et startpunkt og et endepunkt. I denne oppgaven kan du gå ut fra at startpunktet alltid vil ligge til venstre for endepunktet. (Hvis du vil ha en ekstra utfordring, kan du gå ut fra at dette ikke alltid er tilfellet, og skrive koden din deretter. Dette er imidlertid ikke påkrevd.)

a) Lag klassen **Line**. Denne skal arve **Shape**.

1. Hvilke argumenter trenger konstruktøren til en linjeklasse å ta inn? Husk at alle figurer arver fargen fra **Shape**!
2. Hvordan kaller man konstruktøren til **Shape** fra konstruktøren til **Line**?

b) Implementer **Draw()**-funksjonen for **Line**-klassen.

1. Hvordan tegner man en linje? Her må du blåse støvet av matte 1-boken din og finne det ut! Alternativt kan du bruke topunktsformelen fra ungdomskolen:

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

NB! Pass på at du får korrekt oppførsel mtp. heltallsdivisjon i formelen over.

c) Test klassen din ved å tegne linje med startpunkt (0, 0) og slutt punkt (100, 50). Lagre bildet for å se resultatet.

d) Test klassen din igjen ved å tegne en linje med startpunkt (0, 0) og slutt punkt (100, 200).

1. Ble denne linjen like fin som den forrige? Hvis du har fulgt oppgaveteksten er svaret nei. Årsaken til dette er fordi implementasjonen av formelen $y = xm + a$ tegner linjen mens den beveger seg langs x-aksen. Dette fungerer helt fint for forholdsvis horisontale linjer, men fører til krøll hvis linjen er for bratt (altså hvis stigningstallet til linjen er større enn 1).

e) Skriv om **Draw()**-funksjonen til **Line**-klassen slik at den sjekker stigningstallet til linjen før den begynner å tegne. Hvis stigningstallet er større enn 1 skal **Draw()**-funksjonen bevege seg langs y-aksen.

1. *Hint: Vi kan skrive om topunktsformelen til å gi x som en funksjon av y:*

$$x = \frac{x_2 - x_1}{y_2 - y_1}(y - y_1) + x_1$$

5 Flere spesialiseringer - Rectangle klassen

Linjer kan brukes til å lage rektangler. Hvordan, lurer du? Det er slik at et rektangel har fire sider og fire hjørner, men kan defineres ut i fra dets diagonal, som kan defineres som en linje! Husk at et rektangel har fire sider som er parvis like.

a) Lag klassen **Rectangle**. Denne skal, i likhet med **Line**, arve **Shape**.

1. Hvilke argumenter trenger **Rectangle**-klassen sin konstruktør å ta inn?

b) Skriv **Draw()**-funksjonen for klassen **Rectangle**.

1. *Hint: Prøv å tegne diagonalen først.*

6 Enda flere spesialiseringer - Circle klassen

Rette linjer er i grunnen ganske kjedelig, synes du ikke? Vel hold deg fast for nå skal vi tegne sirkler, geometriens rareste objekt. En sirkel er definert ved et midtpunkt og en radius. Alle punktene langs en sirkels periferi er like langt fra dets midtpunkt.

a) **Lag klassen Circle.** I likhet med de fleste andre klassene i denne øvingen skal den arve Shape.

1. Hvilke argumenter trenger Circle-klassen sin konstruktør å ta inn?

b) **Skriv Draw()-funksjonen for klassen Circle.** Formelen for punktene i en sirkel med radius r og sentrum i (x_0, y_0) er:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

1. *Hint: her kan det være en idé å iterere over både x- og y-aksen.*

7 Polymorfi med virtuelle funksjoner

For å lage fantastiske bilder kan det være greit å kunne samle flere forskjellige figurer på et «lerret». Du får nå se hvor praktisk det er når alle klassene arver samme klasse.

a) **Skriv klassen Canvas.** Denne klassen skal inneholde en `std::vector` av Shape-pekere.

b) **Skriv medlemsfunksjonen AddShape()** som tar inn en Shape-pekere og legger den til i vektoren.

c) **Skriv medlemsfunksjonen RasterizeTo()** som skal ta inn en referanse til et Image-objekt. Funksjonen skal tegne alle Shape-objektene i vektoren inn i bildet.

d) **Bruk Canvas-objektet ditt til å tegne et smilefjes.** Bruk linje-segmenter til å lage munnen, eller (valgfritt) skriv en ny klasse for å tegne halvsirkler.

e) *Valgfri: Bruk LoadImage()-funksjonen til å laste inn bildet referert til i Oppgave 1a. Du må først lagre det på harddisken. Bruk Canvas-klassen til å tegne en bart på damen.*