NTNU
Institutt for datateknikk
og informasjonsvitenskap

**Eksamensoppgave i**

# TDT4102 - Prosedyre- og objektorientert programmering

## Lørdag 19 mai 2012, 09:00

## Kontaktperson under eksamen: Trond Aalberg (97631088)

***Eksamensoppgaven er utarbeidet av Trond Aalberg***
***og kvalitetssikret av Hallvard Trætteberg***

**Språkform:** Bokmål

**Tillatte hjelpmidler (hjelpemiddelkode C):**
Walter Savitch, Absolute C++ eller Lyle Loudon, C++ Pocket Reference.
Bestemt, enkel kalkulator tillatt.

**Sensurfrist:** Mandag 11 juni.

## Generell introduksjon

Les gjennom oppgavetekstene nøye. Noen av oppgavene har lengre tekst, men dette er for å gi kontekst, introduksjon og eksempler til oppgaven.

Når det står "implementer" eller "lag" skal du skrive en implementasjon. Hvis det står "vis" eller "forklar" står du fritt i hvordan du svarer, men bruk enkle kodelinjer og/eller korte forklaringer og vær kort og presis. I noen oppgaver er det brukt nummererte linjer for koden slik at det skal være lett å referere til spesifikke linjer. Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger som du finner nødvendig.

Hver enkelt oppgave er ikke ment å være mer omfattende enn det som er beskrevet. Noen oppgaver fokuserer bare på enkeltfunksjoner og da er det utelukkende denne funksjonen som er tema. Andre oppgaver er "oppskriftsbasert" og vi spør etter forskjellige deler av en klasse, samarbeidende klasser eller et program. Du kan velge selv om du vil løse dette trinnvis ved å ta del for del, eller om du vil lage en samlet implementasjon. Sørg for at det går tydelig frem hvilke spørsmål du har svart på hvor i koden din.

Det er ikke viktig å huske helt korrekt syntaks for biblioteksfunksjoner. Oppgaven krever ikke kjennskap til andre klasser og funksjoner enn de du har blitt kjent med i øvingsopplegget. All kode skal være i C++.

Hele oppgavesettet er arbeidskrevende og det er ikke foreventet at alle skal klare alt. Tenk strategisk i forhold til ditt nivå og dine ambisjoner! Velg ut deloppgaver som du tror du mestrer og løs disse først. Slå opp i boka kun i nødsfall. All tid du bruker på å lete i boka gir deg mindre tid til å svare på oppgaver. Deloppgavene i de "tematiske" oppgavene er organisert i en logisk rekkefølge, men det betyr IKKE at det er direkte sammenheng mellom vanskelighetsgrad og nummereringen av deloppgavene.

Oppgavene teller med den andelen som er angitt i prosent. Den prosentvise uttellingen for hver oppgave kan/vil likevel bli justert ved sensur basert på hvordan oppgavene har fungert. De enkelte deloppgaver kan også bli tillagt forskjellig vekt.

## Oppgave 1: Kodeforståelse (20%)

**a)** Gitt følgende variabler og tilordninger

```
int a = 5;
int b = 5;
int *c = &a;
int *d = &b;
```

Hva blir resultatet av disse uttrykkene: true eller false?

```
a == b
&a == c
c == d
*c == *d
```

**b)** Hva skrives ut av følgende kode?

```
int a = 5.8 * 2;
cout << "a = " << a << endl;

double b = 5 / 2;
cout << "b = " << b << endl;

int x = 5;
int y = 10;
int c = x++ + ++y;
cout << "c = " << c << endl;
```

**c)** Klassen **Node** under brukes til å lage en dobbeltlenket liste. Den har en medlemsfunksjon **void insertAfter(Node *n)** som skal sette inn noden **n** på plassen etter objektet funksjonen kalles i kontekst av. Her blir det litt kluss med pekertilordningen, noe du oppdager når du prøver å iterere over alle nodene i en liste du har laget (uendelig løkke).

1) Vis hva som blir feil med en tegning av node- og pekerstrukturen for eksempelet i boksen under (bruk bokser for noder og piler for pekere).
2) Vis hvilke endringer som må gjøres i koden for å få riktig oppførsel.

```
1:  class Node {
2:  private:
3:    string value;
4:    Node* next;
5:    Node* prev;
6:  public:
7:    Node(const string &value): value(value), next(NULL), prev(NULL){}
8:    void insertAfter(Node* n);
9:    string getValue();
10: };
11:
12: void Node::insertAfter(Node *n){
13:   if (next == NULL){
14:     next = n;
15:     n->prev = this;
16:   }else{
17:     next = n;
18:     n->prev = this;
19:     next->prev = n;
20:     n->next = next;
21:   }
22: };
```

```
Node *first = new Node("First");
Node *middle = new Node("Middle");
Node *last = new Node("Last");

first->insertAfter(last);
first->insertAfter(middle);
```

**Introduksjon til oppgavene 1d - 1h:**

Bubblesort (boblesortering) er en velkjent algoritme for sortering av en sekvens av elementer (for eksempel verdiene i en array/tabell eller en vector). Den brukes fordi den er enkel å forstå og implementere, men er kun egnet til små sorteringsoppgaver siden den har dårlig ytelse.

I boblesortering brukes en nøstet løkke. I den innerste løkka itererer vi over alle elementer og hvis **`arr[j] > arr[j+1]`** så bytter vi om på disse to verdiene. Hvis vi starter med sekvensen `{5,4,3,2,1,0}`, vil første iterasjon av den innerste løkka føre til at **`5`** forflyttes til siste posisjon `{4,3,2,1,0,`**`5`**`}`. Ved andre iterasjon vil den nest høyeste verdien (**`4`**) forflyttes til sin rette plass `{3,2,1,0,`**`4`**`,5}` etc. Algoritmen kalles boblesortering fordi de minste verdiene langsomt "bobler" fremover til sine rette posisjoner.

Funksjonen under er et første utkast til implementasjon av boblesortering for typen **`vector<int>`**. I de følgende delspørsmål skal du identifisere feil og endre koden.

```
1:   void bubblesort(vector<int> arr){
2:     for(int i = arr.size(); i > 0; --i){
3:       for (int j = 0; j < arr.size(); ++j){
4:         if (arr[j] > arr[j+1]){
5:           int temp = arr[j];
6:           arr[j] = arr[j+1];
7:           arr[j+1] = temp;
8:         }
9:       }
10:    }
11:  }
```

**d)** Hvilken C++ biblioteksfunksjon kunne du brukt i stedet for kodelinjene 5-7?

**e)** For å teste implementasjonen lager du en variabel **`vector<int> testvec`** i main, legger inn noen verdier ved hjelp av **`push_back`** og gjør kallet **`bubblesort(testvec)`**. Når du etterpå skriver ut verdiene i **`testvec`** til **`cout`**, ser det ut som **`testvec`** fortsatt er usortert! Hva er problemet og hva må endres for å rette denne feilen?

**f)** Etter å ha rettet feilen i forrige deloppgave, prøver du å sortere en vector med innholdet **`{5,4,3,2,1}`**. Etter sortering er innholdet blitt **`{0,1,2,3,4}`**.
Hva er feil i implementasjonen og hvordan rette opp? Hint: hvor havnet tallet 5 etter første gjennomløp.....

**g)** Ytelsen til bubblesort kan forbedres ved å redusere antallet iterasjoner i den innerste løkka. Etter første gjennomløp (iterasjon) av den innerste løkka vil største verdi være plassert sist. Ved andre gjennomløp kan du derfor ignorere siste element. Ved tredje gjennomløp kan du ignorere de to siste elementene etc. Vis hvordan koden kan endres til å implementere denne forbedringen.

**h)** En annen ytelsesforbedring kan oppnåes ved å avslutte når du vet at elementene er sortert. For en sekvens bestående av **`{1,2,3,5,4}`** vil implementasjonen gjøre mange unødvendige iterasjoner. Forklar hvordan du underveis kan teste om sekvensen er sortert og avslutte hvis det ikke er behov for flere iterasjoner.

## Oppgave 2: Funksjoner (20%)

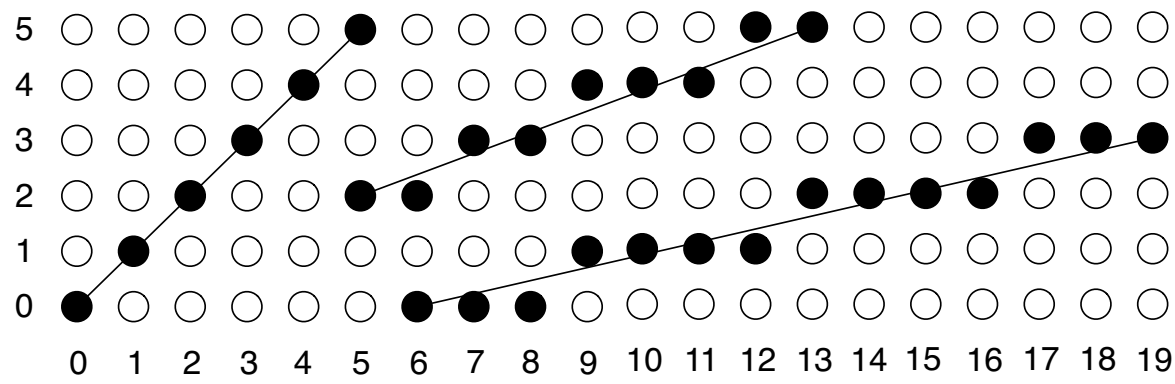
**a)** Implementer en funksjon **`int round(double d)`** som runder av et flyttall til nærmeste hele tall (returnerer en **`int`**). Funksjonen skal også håndtere negative tall.
*Eksempler:* ***`round(3.45)`*** *gir* ***`3`***, ***`round(3.60)`*** *gir* ***`4`***, ***`round(3.50)`*** *gir* ***`4`*** *(vi runder oppover ved .50),* ***`round(-3.45)`*** *gir* ***`-3`***, ***`round(-3.6)`*** *gir* ***`-4`***.

**b)** Implementer en funksjon **`int adjacent_find(int arr[], int first, int last)`** som leter i tabellen **`arr`** etter naboelementer med lik verdi. Funksjonen skal returnere indeksen til første element av disse. Argumentene **`first`** og **`last`** angir området i tabellen den skal lete i. Funksjonen skal returnere **`-1`** hvis den ikke finner noen like naboelementer.
*Eksempel: Gitt* ***`int x[] = {0,4,4,0,3,3};`*** *så skal* ***`adjacent_find(x,0,6)`*** *returnere* **`1`** *siden dette er indeksen til det første tallet i første par funksjonen finner.*

**c)** Implementer en funksjon **`void print_adjacent(int arr[], int size)`** som bruker funksjonen fra oppgaven over og skriver ut til cout alle par av naboverdier som finnes i **`arr`**, hvor **`size`** er antallet elementer i tabellen.
*Eks: Gitt* ***`int x[] = {0,4,4,5,3,3,3,7}`*** *så skal funksjonen skrive ut: 4 4 - 3 3 - 3 3. Merk at vi får skrevet ut 3 3 to ganger siden både posisjon 4 og 5 har et neste element som er samme verdi. Formatteringen av utskriften er uvesentlig i denne oppgaven.*

**d)** Digitale bilder er bygd opp av punkter i et jevnt mønster (se under). Når du skal tegne opp (plotte) en linje fra et startpunktet **`(x1,y1)`** til et sluttpunktet **`(x2,y2)`** må du regne ut hvilke punkter som skal tegnes opp mellom disse to punktene. Linja du tegner opp vil bli en tilnærming, men siden punktene er små og tett plasserte vil det se ut som ei jevn linje.
En enkel (og ikke så veldig effektiv) algoritme for dette baserer seg på den velkjente formelen for linjer: **`y = a*x + b`** hvor **`a`** er stigningstallet **`(y2-y1)/(x2-x1)`** og **`b`** er linjas skjæringspunkt på y-aksen. Tallet **`b`** (skjæringspunktet) kan regnes ut ved å sette inn en av koordinatene som er input til funksjonen. Figuren under viser 3 forskjellige linjer og punktene som algoritmen har regnet ut for disse.

1) Implementer funksjonen **`void plotline(int x1,int y1,int x2,int y2)`** som skriver til cout alle x,y-punktene som skal plottes for linja fra **`(x1,y1)`** til **`(x2,y2)`**. Du kan iterere over x og regne ut y, men husk at x og y til et punkt skal være det koordinat som er nærmest den ideelle linja. Det holder om løsningen dekker stigningstall fra 0 til 1.
2) Hva skjer hvis stigningstallet til linja er større enn 1?
Forklar med kode eller tekst hvordan du kan håndtere dette i funksjonen din.
3) Hvilket problem kan du få hvis (**`x2-x1) == 0`** ?
4) Er det andre ting som implementasjonen bør ta hånd om?

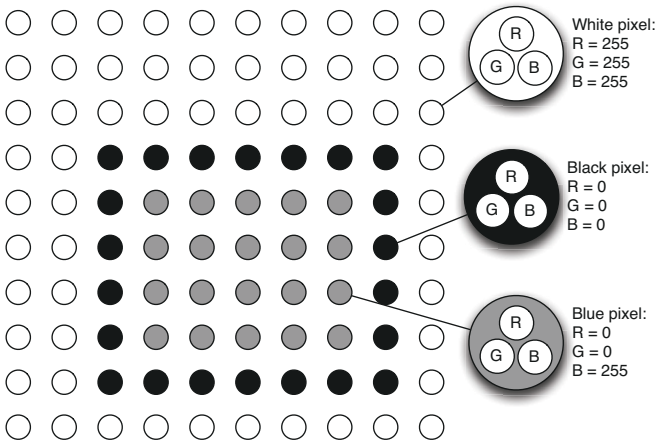


## Oppgave 3: Minnehåndtering (25%)

I denne oppgaven skal du implementere en klasse **`Image`** som brukes for å representere fargebilder internt i et program. Et bilde har en gitt bredde (**`w`**) og høyde (**`h`**) og består av **`w * h`** bildepunkter. Hvert bildepunkt kan adresseres med **`(x, y)`**-koordinater. Fargen til et bildepunkt angis med tre enkeltverdier; en verdi for hver av primærfargene rød, grønn og blå (**RGB**). Verdien for hver av primærfargene angis med et heltall mellom 0-255. Slike bildepunkter i digitale bilder kalles for **pixler**.

En foreløpig deklarasjon av klassen **`Image`** er gitt under. Illustrasjonen viser eksempel på et 10x10 bilde og hvordan det logisk er bygd opp. Siden vi er interessert i effektiv minnebruk, rask skriving til fil etc. skal vi bruke en "lavnivå" datastruktur hvor vi allokerer et sammenhengende minneområde av typen **`unsigned char`** (som ett enkelt dynamisk allokert array). Du får med andre ord ikke bruke en sammensatt datatype for lagring av de enkelte bildepunkt.

```
class Image{
private:
   int width, height;
   unsigned char *pixels;
public:
   Image();
   Image(int w, int h);
   int getWidth();
   int getHeight();
   void resize(int w, int h);
   void save(const char *name);
   ~Image();
};
```




**a)** Medlemsvariabelen **`pixels`** skal peke til det dynamisk allokerte minnet vi bruker. Hvorfor bruker vi datatypen **`unsigned char`** og hva betyr **`*`** i denne sammenhengen?

**b)** For hver pixel (bildepunkt) trenger vi separate verdier for rød, grønn og blå. Gitt et bilde med bredde **`w`** og høyde **`h`**, hvor stort minne trenger du (antall unsigned char)?

**c)** Implementer klassens to konstruktører og destruktøren.

**d)** Implementer følgende get- og set-funksjoner for å sette/lese fargeverdiene til en pixel. For å forenkle litt bruker vi en struct-type **`Colour`** som parameter og returtype, men merk at datatypen for det dynamisk allokerte minnet er unsigned char.

```
struct Colour{ unsigned char red, green, blue;};
void setPixel(int x, int y, const Colour &c);
Colour getPixel(int x, int y);
```

*Tips: For å regne om fra todimensjonal syntaks basert på (x,y) til endimensjonal indeks kan du bruke formelen* ***`indeks = w * y + x`***. *Merk at denne må tilpasses oppgaven.*

**e)** Hva betyr parameterdeklarasjonen **`const Colour &c`** i set-funksjonen beskrevet i deloppgave over og hvorfor bruker vi denne deklarasjonen?

**f)** Implementer medlemsfunksjonen **`void resize(int w, int h)`**. Denne brukes for å endre størrelse på bildet. Hvis bredde og/eller høyde økes skal det legges til nye punkter uten at eksisterende bilde blir endret (forvrengt). Hvis høyde og eller bredde reduseres skal effekten være at du "klipper kantene" av eksisterende bilde.

## Oppgave 4: Klasser, objekter og et litt større prosjekt (35%)

I denne oppgaven skal du jobbe med en klasse **`Drawing`** for tegninger med forskjellige typer figurer. Vi lar **`Drawing`**-klassen arve fra **`Image`** (oppgave 3) og bruker også **`Colour`** (fra 3d).

Denne oppgavesiden er en introduksjon til oppgaven, deloppgavene står på neste side!

Merk at det ikke er nødvendig å ha løst noen av deloppgavene i 3) for å gjøre denne oppgaven. Det er bare nødvendig å forstå spesifikasjonen av **`Image`** og hvordan medlems-funksjonene skal/kan brukes.

Koden i den følgende forklaringen er bare ment som et utgangspunkt (et første utkast) for oppgavene du skal løse, og du kan legge til klasser og endre på de som er deklarert.

```
class Drawing: public Image{
private:
  Colour bg;
  void update();
public:
  Drawing(int width, int height, const Colour &bg);
  void setBackground(const Colour &c);
  void addLine(int x1, int y1, int x2, int y2, const Colour &c);
  void addRectangle(int x, int y, int width, int height, const Colour &c);
  ~Drawing();
};
```

Vi ønsker å kunne legge til linjer og rektangler til et **`Drawing`**-objekt med forskjellige medlems-funksjoner (**`addLine`**, **`addRectangle`**). Disse funksjonene tar argumenter som spesifiserer hvordan objektet som skal tegnes. For linjer må vi ha startpunkt (x1,y1), endepunkt (x2,y2) og farge. For rektangler må vi ha et hjørnekoordinat (x,y), bredde, høyde og farge.

**`Drawing`**-klassen skal være i stand til å huske hvilke figurer du har lagt til tegningen. Når du bruker add-funksjonene skal det instansieres objekter som skal "administreres" av Drawing-klassen. I første utkast til kode har vi laget følgende klasser for linje og rektangel:

```
class Line{
public:
  Line(int x1, int y1, int x2, int y2, const Colour &c);
  void paint(Image *img);
};

class Rectangle{
public:
  Rectangle(int x, int y, int width, int height, const Colour &c);
  void paint(Image *img)
};
```

**`Drawing`** arver fra **`Image`** og har dermed det som trengs for å lage et "punkt-grafikk" bilde. Linjer og rektangler som legges til skal tegnes opp slik at du f.eks. kan vise bildet på skjerm eller lagre til ei bildefil. Siden linje og rektangel må tegnes opp med forskjellig algoritme, har disse klassene hver sin **`paint`**-funksjon med **`Image*`** som parameter. Objektene skal med andre ord kunne tegne "seg selv" på et Image-objekt.

Figurene skal tegnes opp etter hvert som add-funksjonene kalles. Hvis det gjøres andre endringer må hele bildet tegnes opp på nytt, eksempelvis hvis du skifter bakgrunn. Vi kunne også hatt funksjoner som endret tegningen på andre måter f.eks. en funksjon for å fjerne spesifikke linjer og rektangler fra bildet.

**Fortsettelse neste side**

**a)** Implementer konstruktøren for **`Drawing`**-klassen.

**b)** **`Drawing`**-objektene må kunne iterere over alle linjer og rektangler som er lagt til og be disse om å tegne seg selv. Vis hvordan du kan bruke teknikken med arv, virtuelle funksjoner og polymorfi for å realisere dette. Vis også hvilken medlemsvariabel du vil deklarere i **`Drawing`**-klassen for å håndtere linjer og rektangler.

**c)** Implementer **`addRectangle`**-funksjonen i **`Drawing`**-klassen (se klassedeklarasjonen for parameterliste og returtype).

**d)** Implementer **`void paint(Image *img)`** for **`Rectangle`**-klassen. Den skal sørge for at rektangelet blir tegnet opp i **`img`**-objektet. Hele flaten til rektangelet skal settes til fargen som rektangelet har; dvs. du skal sette fargen til alle bildepunkter som rektangelet dekker.

**e)** Medlemsfunksjonen **`void update()`** i **`Drawing`**-klassen skal brukes til å oppdatere hele bildet når dette trengs, mens **`void setBackground(const Colour &c)`** skal brukes til å endre bakgrunnsfarge. Implementer begge funksjonene og legg vekt på "samspillet" mellom dem: dvs. at den ene bruker den andre på en fornuftig måte.

**f)** Hva skjer hvis en figur er helt eller delvis utenfor bildeflaten? Vis og forklar hvor i koden og hvordan du kan teste på slike situasjoner. Bruk unntaksmekanismen for å si fra om slike situasjoner (bruk en eksisterende unntakstype eller lag en egen type).

**g)** I **`Image`**-klassen er det en **`void resize()`** funksjon som brukes for å endre størrelsen på bildet. Er det behov for å redefinere denne i **`Drawing`**-klassen og hvordan vil du i såfall implementere denne (og forklar hvorfor/hvorfor ikke)?

**h)** Gitt den implementasjonen du har laget, trenger du å implementere en destruktør? Hvis ikke må du forklare hvorfor. Implementer destruktøren hvis du trenger en.

**i)** Vi har latt **`Drawing`** arve fra **`Image`** i denne oppgaven. Hvilket annet alternativ har vi for å la **`Drawing`**-klassen bruke **`Image`**? Diskuter kort fordeler og ulemper ved begge løsninger.

**j)** For å forenkle håndteringen av fargene i Drawing-klassen skal du bruke et "fargekart" som leses fra fil. Du finner et eksempel på ei slik fil i **`appendiks 1`**. I fila er hver farge beskrevet på en separat linje med et navn først, etterfulgt av verdiene for henholdsvis rød, grønn og blå. Det er brukt tabulator (whitespace) mellom verdiene.

1) Deklarer en egnet medlemsvariabel for fargekartet.
2) Implementer en medlemsfunksjon i Drawing-klassen for å lese inn fra fil: **`void loadColourMap(const char* filename)`**.
3) Implementer en overlagret versjon av en av funksjonene i **`Drawing`**-klassen (fritt valg) som bruker **`string`** (for fargenavn) i parameterlista i stedet for typen **`Colour`**.

## Appendiks 1

Eksempel på innhold i ei colormap fil (oppgave 4 j):

```
aliceblue       240  248  255
antiquewhite    250  235  215
aqua            0    255  255
aquamarine      127  255  212
azure           240  255  255
beige           245  245  220
bisque          255  228  196
black           0    0    0
blanchedalmond  255  235  205
blue            0    0    255
cornsilk        255  248  220
crimson         220  20   60
cyan            0    255  255
darkblue        0    0    139
darkcyan        0    139  139
darkslategray   47   79   79
deepskyblue     0    191  255
dimgray         105  105  105
white           255  255  255
red             255  0    0
lime            0    255  0
green           0    128  0
```